Systematic construction of efficient six-stage fifth-order explicit Runge–Kutta embedded pairs without standard simplifying assumptions

A Thesis Submitted to the
College of Graduate and Postdoctoral Studies
in Partial Fulfillment of the Requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Andrew Kroshko

Permission to Use

Abstract

This thesis examines methodologies and software to construct explicit Runge–Kutta (ERK) pairs for solving initial value problems (IVPs) by constructing efficient six-stage fifth-order ERK pairs without standard simplifying assumptions. The problem of whether efficient higher-order ERK pairs can be constructed algebraically without the standard simplifying assumptions dates back to at least the 1960s, with Cassity's complete solution of the six-stage fifth-order order conditions. Although RK methods based on the six-stage fifth-order order conditions have been widely studied and have continuing practical importance, prior to this thesis, the aforementioned complete solution to these order conditions has no published usage beyond the original series of publications by Cassity in the 1960s.

The complete solution of six-stage fifth-order ERK order conditions published by Cassity in 1969 is not in a formulation that can easily be used for practical purposes, such as a software implementation. However, it is shown in this thesis that when the order conditions are solved and formulated appropriately using a computer algebra system (CAS), the generated code can be used for practical purposes and the complete solution is readily extended to ERK pairs. The condensed matrix form of the order conditions introduced by Cassity in 1969 is shown to be an ideal methodology, which probably has wider applicability, for solving order conditions using a CAS. The software package `OCSage` developed for this thesis, in order to solve the order conditions and study the properties of the resulting methods, is built on top of the Sage CAS.

However, in order to effectively determine that the constructed ERK pairs without standard simplifying assumptions are in fact efficient by some well-defined criteria, the process of selecting the coefficients of ERK pairs is re-examined in conjunction with a sufficient amount of performance data. The `pythODE` software package developed for this thesis is used to generate a large amount of performance data from a large selection of candidate ERK pairs found using `OCSage`. In particular, it is shown that there is unlikely to be a well-defined methodology for selecting optimal pairs for general-purpose use, other than avoiding poor choices of certain properties and ensuring the error coefficients are as small as possible. However, for IVPs from celestial mechanics, there are obvious optimal pairs that have specific values of a small subset of the principal error coefficients (PECs). Statements seen in the literature that the best that can be done is treating all PECs equally do not necessarily apply to at least some broad classes of IVPs. By choosing ERK pairs based on specific values of individual PECs, not only are ERK pairs that are 20–30% more efficient than comparable published pairs found for test sets of IVPs from celestial mechanics, but the variation in performance between the best and worst ERK pairs that otherwise would seem to have similar properties is reduced from a factor of 2 down to as low as 15%. Based on observations of the small number of IVPs of other classes in common IVP test sets, there are other classes of IVPs that have different optimal values of the PECs. A more general contribution of this thesis is that it specifically demonstrates how specialized software tools and a larger amount of performance data than is typical can support novel empirical insights into numerical methods.

Acknowledgements

Searching for new insights straddling the boundary of mathematics and computer science was always going to be a long haul. Nothing like the study described in this thesis can ever be done alone. There are many people I would like to express my thanks to:

Dr. Raymond J. Spiteri, the supervisor of this thesis, for suggesting the many ideas that lead to the research direction of this thesis, in addition to the support and feedback I have received over the years. The many members of his lab that I have had the pleasure of knowing have also provided me with essential support. I would also like to extend thanks to the faculty and staff in the Computer Science department at the University of Saskatchewan, who also provided invaluable support.

A. Paul Kroshko, my father, who directly provided the inspiration for many of the things that are essential to my life today. My first exposure to the subject of this thesis, the numerical solution of differential equations, occurred while I was still a teenager and was helping him with software he wrote for his career of exploring for oil. Before that, he taught me typing and computer programming while still in grade school.

Dr. Joan E. Krochko, my mother, has provided invaluable support along the way. Especially from the wide-ranging experience she gained during a long and successful career of research into plant physiology and molecular biology. Without her continued support and mentoring over the years, this thesis would never have seen the light of day.

Kathy, Jeanette, Jenn, David, Thomas, Jessie, and Christine, the rest of my immediate family, have all helped me in important ways during this journey.

Dedication

To my parents. For all you showed me and all you've done.

Contents

List of Tables

List of Figures

List of Abbreviations

Cash–Karp 4(5) method	CK4(5)$_{6(6)}$
computer algebra system	CAS
condensed matrix form	CMF
Dormand–Prince RK5(4)7FC method	DP5(4)C$_{6(7)}$
Dormand–Prince RK5(4)7FM method	DP5(4)$_{6(7)}$
Dormand–Prince RK5(4)7FS method	DP5(4)S$_{6(7)}$
Dormand–Prince RK5(4)6M method	DP5(4)M$_{6(6)}$
error per step	EPS
error per unit step	EPUS
explicit Runge–Kutta	ERK
first same as last	FSAL
forward Euler method	FE
implicit Runge–Kutta	IRK
initial-value problem	IVP
integral (controller)	I
left-hand side	LHS
ordinary differential equation	ODE
fifth- and fourth-order pair by Papakostas and Papageorgiou	PP5(4)$_{6(7)}$
partial differential equation	PDE
proportional-integral (controller)	PI
proportional-integral-derivative (controller)	PID
principal error coefficient	PEC
normalized principal error coefficient	$\overline{\text{PEC}}$
relational database management system	RDBMS
right-hand side	RHS
Runge–Kutta–Fehlberg 4(5) method	RKF4(5)$_{6(6)}$
Runge–Kutta	RK
"the Runge–Kutta method" (particular four-stage fourth-order ERK method)	RK4
structured query language	SQL
six-stage fifth-order ERK embedded pair	5(4)$_6$ ERK pair
six-stage fifth-order and six-stage fourth-order ERK embedded pair	5(4)$_{6(6)}$ ERK pair
six-stage fifth-order and seven-stage fourth-order ERK embedded pair	5(4)$_{6(7)}$ ERK pair
six-stage fifth-order and six-stage fourth-order ERK embedded pairs with the C(2) simplifying assumptions (family of)	5(4)$_{6(6)\mathrm{C}(2)}$
six-stage fifth-order and seven-stage fourth-order ERK embedded pairs with the C(2) simplifying assumptions (family of)	5(4)$_{6(7)\mathrm{C}(2)}$

Chapter 1

Introduction and overview of the thesis

Numerical methods for solving differential equations are important tools for many applications in science and engineering, as well as a component of many pieces of software. Runge–Kutta methods for ordinary differential equations are particularly important for many application areas that require high-quality numerical solutions, due to their efficiency, adaptivity, and stability properties at higher orders, i.e., higher order numerical methods have a higher potential for accuracy. Specific applications are too many to enumerate but include: celestial mechanics [7][72, pgs.129–131][122, 159], fluid mechanics [29, 203], motion of rigid bodies [72, pg.244][73, pgs.8–11], chemical and biological systems [86], electronic circuits [13], and control systems [12].

The main research problem addressed by this study is the construction of new families of explicit Runge–Kutta methods, both without standard simplifying assumptions and with the minimal number of standard simplifying assumptions, and showing that they can be practical and efficient. Despite the importance of Runge–Kutta methods and their utilization for real-world problems, the full algebraic solution of the equations that must be satisfied to construct them, in particular, the order conditions solved without necessarily using standard simplifying assumptions [32], has no known prior usage for constructing efficient higher order Runge–Kutta methods [128]. In this study, efficient higher-order Runge–Kutta pairs without simplifying assumptions are constructed as part of the process of finding more general families of the well-known and well-studied six-stage fifth-order explicit Runge–Kutta pairs. In order to qualitatively distinguish efficient methods, new work involving more performance data than what is typically used to study numerical methods is used to evaluate the coefficient selection process. Examination of this performance data helps determine the properties that are often the principal cause of the largest differences in efficiency between explicit Runge–Kutta pairs from the same family, which otherwise would seem to have similar properties. The performance data in this study shows that explicit Runge–Kutta pairs, both with and without standard simplifying assumptions, constructed in a systematic manner can be 20–30% more efficient for solving some test sets of initial-value problems, specifically initial-value problems from celestial mechanics, than existing published methods that otherwise appear similar. This result has practical significance because the performance of explicit Runge–Kutta pairs on these test sets is also shown to be well correlated with more complex initial-value problems based on real-world celestial mechanics problems.

A second research problem, already mentioned briefly and that must be addressed in order to find a well-defined solution to the main research problem, is that the quantity and scope of experimental data presented in the literature for comparing different explicit Runge–Kutta methods may not yet be sufficient to clearly distinguish among the individual pairs that can be constructed from the multiple families that make up the solution to a particular set of explicit Runge–Kutta order conditions. In fact, with increasing amounts of performance data the families used and the coefficients selected for many published Runge–Kutta methods can start to appear rather arbitrary. Due to this, in order to draw quantitative, informative, and reproducible conclusions on the construction of explicit Runge–Kutta methods, a much larger amount of performance data than is typical for studies into numerical methods is generated by solving many initial-value problems with large numbers (thousands) of explicit Runge–Kutta methods. This study demonstrates that this can be readily accomplished by taking advantage of recent advances in available software tools and easily available computational resources. A further contribution from studying the coefficient selection process is that it shows how to eliminate poorly performing pairs in a systematic and well-defined way. These poorly performing pairs often appear similar to the best pairs when using the properties described in the literature. In particular, by systematically examining the properties responsible for the variation in performance between explicit Runge–Kutta pairs, the amount of variation between the best performing and worst performing pairs, that otherwise appear to have similar properties, can sometimes be reduced from a factor of 2 to as low as about 15%. This assists finding a solution to the main research problem by giving a quantitative assessment of which explicit Runge–Kutta methods are in fact most efficient.

These research problems are addressed with the help of data generated by two Python-based software packages, `OCSage` and `pythODE`, developed primarily by the author for this study. The Python language has become extremely popular for scientific computing because the NumPy package supports array operations with similar syntax and functionality to Matlab. Using Python for scientific computing also has the advantage, in comparison to some other platforms commonly used for scientific computing, of a complete standard library that easily supports a diverse array of common computing tasks, e.g., data structures, file operations, operating system functions, object serialization, networking, etc. The popularity of Python can be further explained because there are many mature and widely used Python-based libraries for many computing tasks that are either not available in or as alternatives to the Python standard library, e.g., libraries for building user interfaces or connecting to databases. Furthermore, the SciPy project [91], which NumPy is now a part of, provides free and open-source libraries that have much of the same basic functionality as commercial problem-solving environments for scientific computing, e.g., Matlab, but can easily take advantage of the voluminous and well-tested Python libraries for general-purpose computing. The Python-based Sage computer algebra system [57], which integrates SciPy [91] and many other popular mathematical libraries written in many different languages, is used as a platform for the software tools developed for this study. The `OCSage` package, developed for this study, is approximately 11,000 lines of Python code built upon Sage [57] for solving the Runge–Kutta order conditions and studying their solutions. The `pythODE` package, developed to its current form for this study, is approximately 13,000 lines of Python code built upon

Sage [91] for generating large amounts of performance data for initial-value problem methods. Furthermore, both of these software tools integrate a `PostgreSQL` relational database that turned out to be essential in managing the data from the many different experiments conducted for this study. Using a proper database is not common in the study of numerical methods and this study helps demonstrate the potential of using these otherwise ubiquitous tools to support studies into numerical methods.

The work presented within this study has importance beyond the detailed investigation of specific families of explicit Runge–Kutta methods. There are many variations on the types of differential equations and appropriate types of numerical methods to solve them found in the literature; these could be the subject of future studies similar to this one that require the complete solution of order conditions, substantial amounts of performance data, and that can make use of the software tools described herein.

1.1 Organization of this thesis

This thesis is organized in the following manner:

- This chapter provides a brief overview of the thesis and its contributions.
- Chapter 2 gives an overview of the mathematical theory of modern initial-value problem solvers for ordinary differential equations. Examples of the behaviour of ordinary differential equations and their numerical solutions are included near the beginning to motivate the theory presented in the remainder of Chapter 2. Chapter 2 concludes by demonstrating the methodologies used to construct published six-stage fifth-order explicit Runge–Kutta pairs, in order to motivate the material presented in Chapter 3 chapter.
- Chapter 3 gives the construction of six-stage fifth-order embedded Runge–Kutta pairs without standard simplifying assumptions. Although this was a subject of papers by Cassity in 1966 and 1969 [31, 32], the original paper was extremely terse and did not include sufficient details for a software implementation. The full details for the complete solution of the order conditions for six-stage fifth-order explicit Runge–Kutta methods are given in Chapter 3. Furthermore, Chapter 3 presents new work that extends the solution to embedded pairs, and in combination with finding efficient embedded pairs in later chapters gives the solution to a well-known open problem in the study of numerical methods. There are 17 families that characterize the complete solution to the six-stage fifth-order order conditions and 14 families that characterize the complete solution for embedded pairs based on the six-stage fifth-order explicit Runge–Kutta order conditions.
- Chapter 4 describes the `OCSage` software package that integrates and extends many of the ideas that have been presented in the literature for using software to study and solve Runge–Kutta order conditions (as opposed to solving by hand or standard usage of a computer algebra system) into a single software package built upon a single platform, i.e., the Sage package that incorporates a Python software

ecosystem in a computer algebra focused platform. Because these ideas were originally implemented in wide variety of languages, it is particularly important that they be incorporated into an integrated package built on a single platform. The `OCSage` package implements the solution of the order conditions described in Chapter 3 and generates PYTHON code that can be used to quickly narrow down the number of candidates for new explicit Runge–Kutta methods. Furthermore, the `OCSage` package allows searching the free parameters of a family and storing the results in a `PostgreSQL` database.

- Chapter 5 describes the `pythODE` software package that allows testing many candidate numerical methods on many initial-value problems. The advantages of using modern software technology such as PYTHON-based platforms and a `PostgreSQL` database for performance testing numerical methods are discussed in detail. The overall structure of the `pythODE` package is also described in detail. Extensive performance data is presented on both the existing methods discussed in Chapter 2 and the families presented in Chapter 3. Observations are given that show a well-defined construction procedure for explicit Runge–Kutta pairs that is specific to initial-value problems from celestial mechanics, but that can also be applied to other classes of initial-value problems. Finally, specific explicit Runge–Kutta pairs are presented to show that methods from the families constructed in Chapter 3, including those without simplifying assumptions, can be competitive and even improve on published pairs.

- Chapter 6 concludes the thesis and gives several directions for future work.

1.2 Contributions of this thesis

- In Chapter 3, the solution to the six-stage fifth-order explicit Runge–Kutta order conditions published by Cassity [31, 32] in 1966 and 1969 is explicitly described in an appropriate formulation that has enough detail to support software implementation. Cassity's original publications [31, 32] in 1966 and 1969 are extremely terse and left out many details important for a software implementation.

- In Chapter 3, this solution to the six-stage fifth-order explicit Runge–Kutta order conditions is further extended to embedded pairs. These are the first higher-order explicit Runge–Kutta pairs to be published without standard simplifying assumptions that can exactly satisfy the appropriate order conditions. The other claims by Tsitouras [182, 183] to higher-order explicit Runge–Kutta pairs that do not satisfy the standard simplifying assumptions are shown in Chapter 4 to only ever be able to approximately satisfy the order conditions, even when corrected from the originally published coefficients. In Chapter 5, it is further shown that Tsitouras' pairs can have unexpected performance issues at high accuracies when using them to solve certain initial-value problems.

- In Chapter 3, the importance of the condensed matrix form of the order conditions introduced by Cassity in 1969 is demonstrated through the solution of the six-stage fifth-order explicit Runge–Kutta order conditions without standard simplifying assumptions. The condensed matrix form has not seen any published usage since Cassity first presented it in 1969 [32]. However, the condensed matrix form is particularly suitable for studying and solving order conditions using contemporary computer algebra systems. In particular, the condensed matrix form has a great deal of potential for future studies that use computer algebra systems to give more complete solutions to Runge–Kutta order conditions.

- In Chapter 4, the `OCSage` package is described. This package generates code that provides a useful implementation of the solutions to the Runge–Kutta order conditions. The generated code includes implementations that calculate the many properties used to describe explicit Runge–Kutta pairs. In particular, `OCSage` combines several pieces of functionality, which were originally described across several publications [18, 19, 28, 58], and extends them into a single integrated software package built on top of the modern SAGE platform that includes integration with a `PostgreSQL` database. The design of `OCSage` is discussed in detail in order to help others build software tools that help tackle specific research questions in the study of numerical methods, which would otherwise be difficult with simple ad-hoc programs. An open-source release of the `OCSage` package is planned, ensuring that other researchers do not need to replicate the long development process that led to `OCSage`, but can instead use or modify it as they see fit. This will allow the many research problems that can be addressed by studies similar to this study to be more accessible to the community.

- In Chapter 4, the `OCSage` package is also used to search the free parameters of the new explicit Runge–Kutta families from Chapter 3. For the first time, this gives a clear picture of the limits of and tradeoffs between the different properties involved in Runge–Kutta method construction. This is shown by figures that give some representative data from the searches of the free parameters. Although other researchers may have had some idea of these tradeoffs, this is the first time data showing tradeoffs in the choice of free parameters for Runge–Kutta families has been published in detail.

- In Chapter 5, the `pythODE` package is described. This package uses the PYTHON-based scientific computing ecosystem and a `PostgreSQL` database to allow testing many explicit Runge–Kutta pairs on many initial-value problems. The `pythODE` package uses a `PostgreSQL` database that allows more sophisticated performance analysis of candidate explicit Runge–Kutta pairs than has been previously published in the literature. Like `OCSage`, many design details are discussed that will help others build software tools for tackling specific research questions in numerical methods that would be difficult with just simple ad hoc programs. There is a planned open-source release of the `pythODE` package, ensuring that other researchers do not need to replicate the long development process that led to `pythODE` but can instead use or modify it as they see fit. This will allow the many research problems that can be addressed by studies similar to this one to be more accessible to the community.

- In Chapter 5, contrary to what is often both implicitly and explicitly assumed in the literature, the magnitude of error coefficients of higher order than the leading error coefficient can be important. Although the importance of the higher order error coefficients has been mentioned in the literature [16], it is shown that guidelines more restrictive than those given by other authors are beneficial for performance. In particular, it is shown that the widely used Dormand–Prince pair performs as well as can be expected on a test set of IVPs given the magnitude of its higher-order error coefficients. However, by reducing the magnitude of the higher-order error coefficients, there are many pairs that are otherwise similar to the Dormand–Prince pair with up to 20% better performance.

- In Chapter 5, it is shown that properties other than the leading error coefficient, including the individual values of the principal error coefficients, are often more important for explaining performance differences between the explicit Runge–Kutta pairs. In fact, it is also shown that, contrary to what has been stated in the literature, specific principal error coefficients can be individually important and one of the most significant factors in determining the performance for solving some classes of IVP. This is especially the case for IVPs from celestial mechanics, where well-defined patterns emerge from the large number explicit Runge–Kutta pairs used to solve a wide variety of IVPs. However, it is shown that minimizing the leading error coefficient, often one of the only properties discussed with respect to method construction, does remain important. One family from Chapter 3 also allows the leading error coefficient to be minimized arbitrarily. When the leading error coefficients are aggressively minimized, pairs from this family they can perform extremely well in some cases. However, study of the principal error coefficients of explicit Runge–Kutta pairs with an aggressively minimized leading error coefficient shows performance can be extremely sensitive to the specific values selected for the method coefficients. Although the extremely small leading error coefficients found can seem to be advantageous during limited performance testing, until this sensitivity is better understood, it is difficult to recommend pairs with aggressively minimized leading error coefficients for practical usage.

- In Chapter 5, it is shown that some of the classic properties that have been presented in the literature to help predict the performance of explicit Runge–Kutta pairs do not account for the performance differences seen. Performance testing shows that these properties do not vary in a large enough range to be useful and are either not correlated with performance at all or not correlated enough to be a useful predictor of performance.

- In Chapter 5, new explicit Runge–Kutta pairs from the families found in Chapter 3 are presented, including ones constructed without standard simplifying assumptions. In particular, the new explicit Runge–Kutta pairs without standard simplifying assumptions are shown to be at least comparable to pairs with standard simplifying assumptions and often provide an unexpected competitive advantage for IVPs derived from celestial mechanics problems.

Chapter 2

Mathematical background

"It is difficult to keep a cool head when discussing the various derivatives..."
(S. Gill 1956, through Hairer et al. 1993)

This chapter introduces the mathematical background required to understand variable-stepsize ordinary differential equation solvers that use explicit Runge–Kutta embedded pairs. Section 2.1 defines the ordinary differential equations and the associated initial-value problems that are studied in this thesis. Section 2.2 gives examples of specific ordinary differential equations, and Section 2.3 gives two basic examples of numerical methods. Section 2.4 introduces the basic theory of the numerical methods used to solve initial-value problems. Section 2.5 introduces Runge–Kutta methods and the order conditions necessary for constructing them, which form the foundation for the new families of the explicit Runge–Kutta pairs constructed in Chapter 3. However, prior to that material being presented in Chapter 3, Section 2.6 introduces the additional theory required to extend explicit Runge–Kutta methods to embedded pairs that provide a cheap error estimate. Section 2.7 gives specific examples of the construction methodology used for classic explicit Runge–Kutta methods and pairs.

2.1 Overview of ODEs

An equation for an unknown function $\mathbf{y}$ in terms of its derivatives is known as a *differential equation* (DE). If the unknown function $\mathbf{y}$ depends on only one independent variable t, i.e., $\mathbf{y} = \mathbf{y}(t)$, then the equation is known as an *ordinary differential equation* (ODE) and has the general form

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}(t, \mathbf{y}(t)), \qquad t_0 < t < t_f, \tag{2.1a}$$

where $\mathbf{y}(t) : \mathbb{R} \to \mathbb{R}^m$ is a function with the independent variable time $t \in \mathbb{R}$ and $\mathbf{f} : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ is typically called the *right-hand side* (RHS) of the ODE. The reader should be aware that the abbreviation RHS is sometimes used to describe the right-hand side of algebraic equations (with LHS used to describe the left-hand side of the same equations). However, when this is the case it should always be clear by context, and in this thesis the term RHS without referring to a specific algebraic equation always describes the RHS of an ODE (2.1a). An *initial-value problem* (IVP) is an ODE (2.1a) together with specific additional information about the solution. This additional information is known as the *initial condition* and is given by

$$\mathbf{y}(t_0) = \mathbf{y}_0, \tag{2.1b}$$

where $t_0 \in \mathbb{R}$ is known as the *initial time* and $\mathbf{y}_0 \in \mathbb{R}^m$ is known as the *initial value* [72, pgs.2–3]. The form of the ODE given by (2.1a) is sometimes known as the *state-space form* [167, pg.88].

When the derivative of $\mathbf{y}$ explicitly depends only on t, then (2.1) becomes a *quadrature problem* with the form

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}(t), \quad \mathbf{y}(t_0) = \mathbf{y}_0, \qquad t_0 < t < t_f, \tag{2.2}$$

with the solution given by the integral

$$\mathbf{y}(t) = \mathbf{y}(t_0) + \int_{t_0}^{t_f} \mathbf{f}(t)dt.$$

In order to further simplify the analysis of (2.1a), the RHS can be converted into *autonomous form* by defining a new dependent variable, $t = \mathbf{Y}_{m+1}(t)$,

$$\mathbf{Y}(t) = \begin{pmatrix} \mathbf{y}(t) \\ t \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{Y}}(t) \\ \mathbf{Y}_{m+1}(t) \end{pmatrix},$$

where $\bar{\mathbf{Y}}(t)$ represents the first m components of $\mathbf{Y}(t)$. The autonomous form is an equivalent ODE that simplifies some analysis because the RHS does not explicitly contain t, i.e.,

$$\frac{d\mathbf{Y}}{dt}(t) = \mathbf{F}(\mathbf{Y}(t)),$$

where

$$\mathbf{F}(\mathbf{y}(t)) = \begin{pmatrix} \mathbf{f}(\mathbf{Y}_{m+1}(t), \bar{\mathbf{Y}}(t)) \\ 1 \end{pmatrix}.$$

Without loss of generality, the autonomous form of an ODE (2.1a) can be stated as

$$\frac{d\mathbf{y}}{dt}(t) = \mathbf{f}(\mathbf{y}(t)), \tag{2.3}$$

which is the notation used for the autonomous form of an ODE in this thesis with initial condition (2.1b) when (2.3) is part of an IVP.

2.1.1 PDEs

Another widely seen class of DEs are *partial differential equations* (PDEs), which are a common source of ODEs solved in practice because each PDE can be approximated with a system of ODEs. The PDEs that are solved using IVP methods in this thesis have partial derivatives in both space and time, rather than only derivatives with respect to a single independent variable (usually time) as with ODEs. In Section 2.2, two examples of PDEs that are implemented in `pythODE`, which is described in Chapter 5, are presented.

2.1.2 Second-order ODEs

Many first-order ODEs (2.1a) solved in practice are derived from second-order ODEs that can be defined as an IVP of the form

$$\frac{d^2\mathbf{y}(t)}{dt^2} = \mathbf{f}\left(t, \mathbf{y}(t), \frac{d\mathbf{y}(t)}{dt}\right), \qquad t_0 < t < t_f, \tag{2.4a}$$

$$\mathbf{y}(t_0) = \mathbf{y}_0, \tag{2.4b}$$

$$\frac{d\mathbf{y}(t_0)}{dt} = \left.\frac{d\mathbf{y}_0}{dt}\right|_{t=t_0}, \tag{2.4c}$$

where two initial conditions, i.e., (2.4b) and (2.4c), are now required for an IVP. However, in order to take advantage of the existing analysis and commonly available software for first-order ODEs, IVPs (2.4) based on second-order ODEs (2.4a) can easily be treated as IVPs based on first-order ODEs using the reformulation

$$\begin{pmatrix} \frac{d\mathbf{y}(t)}{dt} \\ \frac{d\bar{\mathbf{y}}(t)}{dt} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{y}}(t) \\ \mathbf{f}(t, \mathbf{y}(t), \frac{d\mathbf{y}(t)}{dt}) \end{pmatrix}, \tag{2.5a}$$

$$\begin{pmatrix} \mathbf{y}(t_0) \\ \bar{\mathbf{y}}(t_0) \end{pmatrix} = \begin{pmatrix} \mathbf{y}_0 \\ \left.\frac{d\mathbf{y}_0}{dt}\right|_{t=t_0} \end{pmatrix}, \tag{2.5b}$$

where (2.5a) can easily be seen to be a system of first-order ODEs that is equivalent to (2.4a) when $\mathbf{y}(t)$ and $\bar{\mathbf{y}}(t)$ correspond to $\mathbf{y}(t)$ and $\frac{d\mathbf{y}(t)}{dt}$, respectively, from (2.4a). It can also easily be seen that if the two initial conditions (2.5) correspond to the initial conditions $\mathbf{y}(t_0)$ (2.4b) and $\frac{d\mathbf{y}(t_0)}{dt}$ (2.4c), then (2.5) is a first-order IVP equivalent to (2.4). When second-order ODEs (2.4a) are described in this thesis, any analysis of first-order ODEs (2.1a) can be applied directly by using the first-order formulation (2.5) and for this study it can also be assumed that second-order IVPs (2.4) are implemented in software using the first-order formulation (2.5).

2.1.3 Existence of solutions

A function $\mathbf{f}(\mathbf{y}(t))$ is said to be *Lipschitz continuous* on $D = \{t_0 \leq t \leq t_f,\, t \in \mathbb{R},\, \mathbf{y}(t) \subseteq \mathbb{R}^m\}$ if a constant L can be chosen such that

$$||\mathbf{f}(\mathbf{y}_a(t_a)) - \mathbf{f}(\mathbf{y}_b(t_b))|| \leq L||\mathbf{y}_a(t) - \mathbf{y}_b(t)||, \tag{2.6}$$

for any $\mathbf{y}_a(t_a)$, $\mathbf{y}_b(t_b) \in D$. If $\mathbf{f}(\mathbf{y}(t))$ is Lipschitz continuous then the corresponding IVP is *well-posed*, i.e., a unique differentiable solution $\mathbf{y}(t)$ exists throughout the interval $t \in D$, and the solution varies continuously with varying initial conditions (2.1b) [10, pg.9; 27, pg.23; 72, pg.37; 108, pg.5; 171, pg.431]. The importance of Lipschitz continuity is that it does not require the first derivative, i.e., $\mathbf{f}(\mathbf{y}(t))$ for $\mathbf{y}(t)$ in (2.1a), to be continuous. This is important because many IVPs (2.1) have the RHS defined piecewise, and Lipschitz continuity can still guarantee a well-posed IVP. However, even if all derivatives of $\mathbf{y}(t)$ are continuous but unbounded, the normally strong condition of having all derivatives of $\mathbf{y}(t)$ continuous does not guarantee a well-posed IVP or that a solution exists on all of D [152, pg.7]. Methodologies that handle IVPs that are not well-posed, i.e., IVPs that are *ill-posed*, are beyond the scope of this thesis.

2.1.4 The Jacobian matrix and linearization

An important property of both ODEs (2.1a) and the numerical methods used to solve them is stability, which encompasses a large variety of different concepts [10, pg.23]. The specific definitions of the stability properties relevant to some applications of the IVP methods studied in this thesis are given in Section 2.5.4. However, a lack of stability can generally be thought of as extreme sensitivity to initial conditions and can occur even for well-posed IVPs. The stability of IVP solutions and the stability of the analogous numerical solution can also be qualitatively different from each other. For instance, there are IVPs, e.g., see (2.14) and (2.21a) below, that have extremely stable solutions that can in fact cause instability for the numerical methods studied in this thesis. This is the concept of stiffness that is demonstrated in an example in Section 2.3.2 and is discussed in Section 2.5.4.

To help study the stability of both ODEs (2.1a) and numerical methods (as well as to study a wide range of other properties), it is useful to consider the linear approximation of (2.1a) at t as a system of linear constant-coefficient ODEs [73, pg.15]. This linearized system is given by

$$\frac{d\overline{\mathbf{y}}}{dt}(t) = \mathbf{J_f}\left(\overline{\mathbf{y}}(t)\right)\overline{\mathbf{y}}(t), \tag{2.7}$$

where $\overline{\mathbf{y}}(t)$ is the solution to the linearized system of ODEs and $\mathbf{J_f} \in \mathbb{R}^{m \times m}$ is the *Jacobian matrix* given by

$$\mathbf{J_f} = \begin{bmatrix} \dfrac{\partial f_1}{\partial y_1} & \cdots & \dfrac{\partial f_1}{\partial y_m} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial y_1} & \cdots & \dfrac{\partial f_m}{\partial y_m} \end{bmatrix}. \tag{2.8}$$

For the purposes of stability analysis in this thesis, $\mathbf{J_f}$ is assumed to be diagonalizable; i.e., there exists an invertible matrix $\mathbf{P}$ such that $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m) = \mathbf{P}^{-1}\mathbf{J_f}\mathbf{P}$. The case where $\mathbf{J_f}$ is not diagonalizable is covered by Higham and Trefethen [78], but is beyond the scope of this thesis. A change of variables $\mathbf{u}(t) = \mathbf{P}^{-1}\bar{\mathbf{y}}(t)$ gives the ODE system

$$\frac{d\mathbf{u}}{dt}(t) = \mathbf{\Lambda}\mathbf{u}(t), \tag{2.9}$$

which is equivalent to (2.7). This change of variables decouples the ODEs and allows (2.3) to be considered as m *scalar test problems* of the form

$$\frac{du}{dt}(t) = \lambda u(t), \tag{2.10}$$

where $\lambda \in \mathbb{C}$ is the eigenvalue associated with a particular component of the ODE in (2.9). The general form of the scalar test problems (2.10) can also be known as the *scalar test equation* or just *test equation* [152, pg.65]. Despite the approximations that take place, the decoupled test equations (2.10) have been found useful for analytically studying stability and other behaviours over a single step of a numerical method. Some solutions of the test equation (2.10) are given in the next section along with other examples of the solutions of other IVPs.

2.2 Some simple examples of IVPs and their solutions

In this section some simple examples of specific IVPs are given for the benefit of the reader. The reader should be aware that some terminology in this section, although in common use for the study of numerical methods, might not yet be defined in this thesis. Therefore, the reader should refer back to this section for specific examples of IVPs that motivate the discussion in this thesis as it unfolds.

The generic definition of an ODE (2.1a) uses a vector $\mathbf{y}$ for the dependent variables and t (commonly corresponding to time) for the independent variable. However, when specific ODEs (2.1a) are defined, it is beneficial to use whichever variable names are commonly used to represent corresponding quantities in the physical model. In the examples that follow, it is obvious which variable names that a specific ODE uses correspond to t and $\mathbf{y}$ in the generic definition of an ODE (2.1a).

The test equation

The test equation (2.10) is already defined in Section 2.1.4, and using an initial condition of $y = 1$ at $t_0 = 0$ gives one of the simplest possible IVPs (2.1). Solutions for different $\lambda \in \mathbb{R}$ of (2.10) are shown in Figure 2.1, where it can clearly be seen that when $\lambda > 0$ the solution y grows exponentially with t and when $\lambda < 0$ the solution y decays exponentially with t. Although the numerical solution of IVPs defined using complex numbers are generally beyond the scope of this thesis, the solutions of the test equation (2.10) with $\lambda \in \mathbb{C}$ are important for analyzing the common situation where the Jacobian (2.8) has complex eigenvalues. With several representative $\lambda \in \mathbb{C}$ it can be seen in Figure 2.2 that solutions of (2.10) with $\lambda \in \mathbb{C}$ are oscillatory in the complex plane.

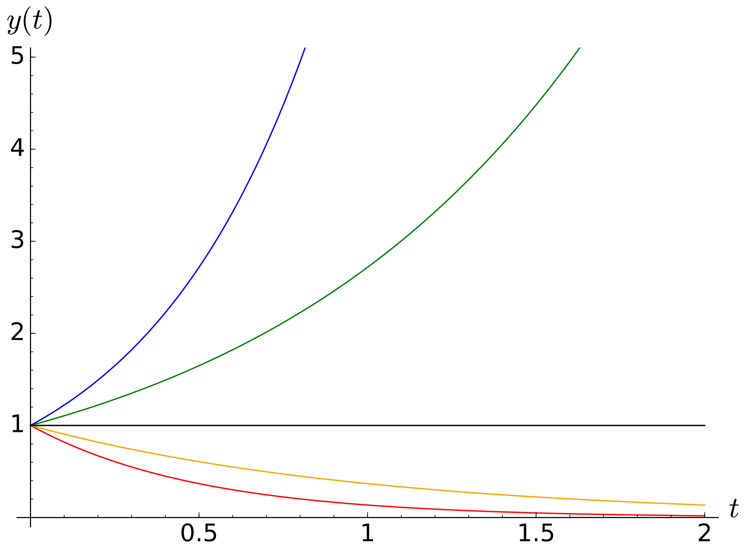
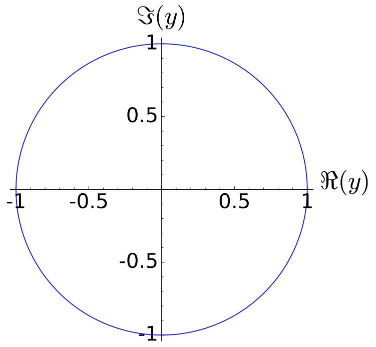
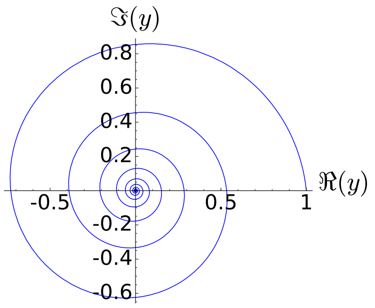


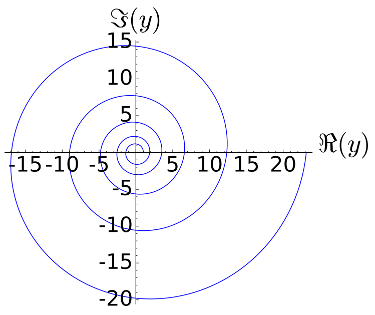
Figure 2.1: Solutions to the test equation (2.10) with real λ and $y(0) = 1$. (blue) $\frac{dy(t)}{dt} = 2y(t)$, $y(t) = e^{2t}$, (green) $\frac{dy(t)}{dt} = y(t)$, $y(t) = e^{t}$, (black) $\frac{dy(t)}{dt} = 0, y(t) = 1$, (orange) $\frac{dy(t)}{dt} = -y(t)$, $y(t) = e^{-t}$, (red) $\frac{dy(t)}{dt} = -2y(t)$, $y(t) = e^{-2t}$.



(a) $\frac{dy(t)}{dt} = iy(t)$, $y(t) = e^{it} = \cos(t) + i\sin(t)$ that gives an oscillatory solution.



(b) $\frac{dy(t)}{dt} = (i - 0.1)y(t)$, $y(t) = e^{(i-0.1)t}$ that gives a decaying oscillatory solution.



(c) $\frac{dy(t)}{dt} = (i + 0.1)y(t)$, $y(t) = e^{(i-0.1)t}$ that gives a growing oscillatory solution.

Figure 2.2: Solutions to the test equation (2.10) with complex λ.

The "Non-stiff A3" IVP, an extremely simple ODE

An IVP, which is also known as problem "A3" of the "non-stiff DE" test set (further described in Section 5.3.1) [83], is given by

$$\frac{dy(t)}{dt} = y\cos t, \quad y(0) = 1, \quad t_f = 20, \tag{2.11}$$

and has an analytic solution of $y(t) = e^{\sin(t)}$ [83] that is shown in Figure 2.3. The "A3" IVP (2.11) is a simple non-autonomous ODE (2.1a) that is ideal for testing and demonstrating IVP solvers, but it does not directly model a specific phenomenon.

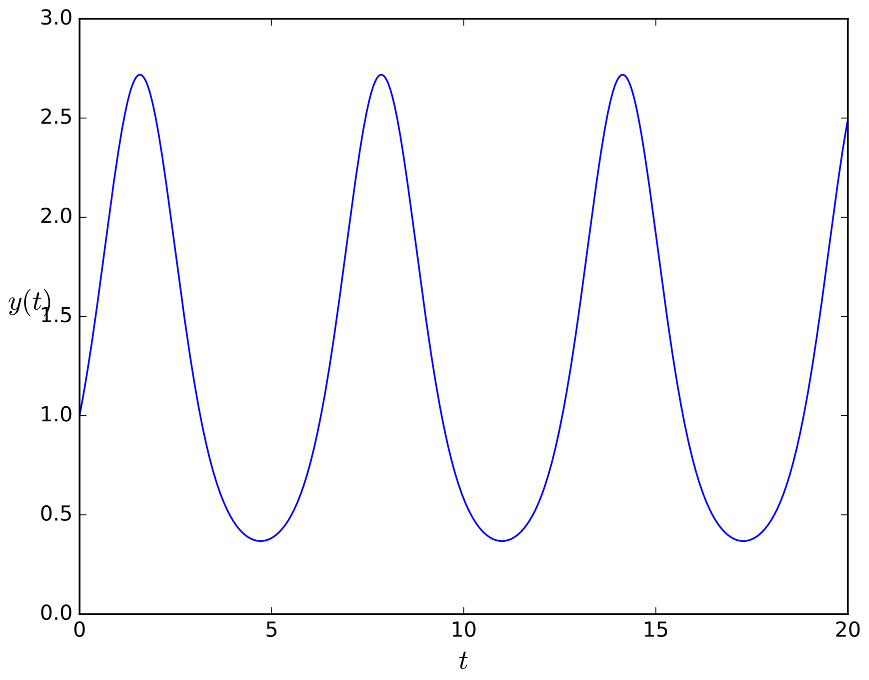


Figure 2.3: The solution of the "A3" IVP (2.11) from the "non-stiff DE" test set.

The "Stiff B1" IVP, a linear oscillatory constant coefficient ODE

Consider the linear constant coefficient IVP given by

$$\begin{aligned}
\frac{dy_1(t)}{dt} &= -y_1 + y_2, & y_1(0) &= 1,\\
\frac{dy_2(t)}{dt} &= -100y_1 + y_2, & y_2(0) &= 0,\\
\frac{dy_3(t)}{dt} &= -100y_3 + y_4, & y_3(0) &= 1,\\
\frac{dy_3(t)}{dt} &= -10000y_3 - 100y_4, & y_4(0) &= 0,\\
t_f &= 20, & & \qquad (2.12)
\end{aligned}$$

which is problem "B1" of the "stiff DE" test set (further described in Section 5.3.1) [55]. Due to (2.12) being a linear constant coefficient ODE, the Jacobian (2.8) of (2.12) is constant with constant eigenvalues of $-1 \pm 10i$, $-100 \pm 100i$. It can be seen that the solutions to (2.12) in Figure 2.4 are rapidly decaying oscillatory solutions, as expected from the representative solutions to the test equation (2.10) that are shown by Figures 2.1 and 2.2.

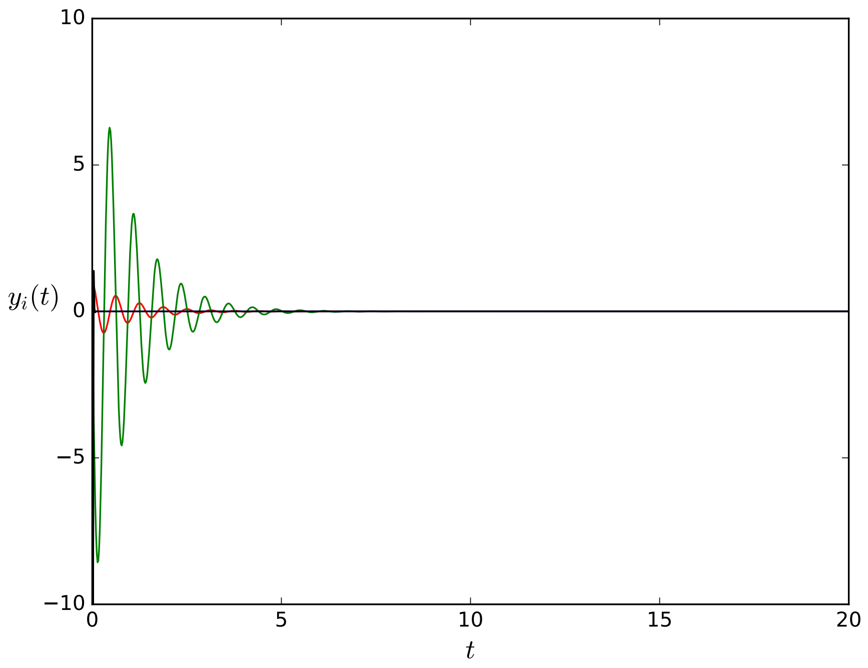


Figure 2.4: The solution of the "B1" IVP (2.12) of the "stiff DE" test set [55] with y_1 (red), y_2 (green), y_3 (blue), and y_4 (black). It can just be seen that y_3 and y_4 decay extremely quickly and are barely visible.

The "Non-stiff" "D1"–"D5" IVPs, 2-body celestial mechanics problems

Problems "D1" through "D5" of the "non-stiff DE" test set [83], which is further described in Section 5.3.1, can be represented by a ODE system that represents many 2-body celestial mechanics problems. This ODE system is given by

$$\begin{aligned}
\frac{dx(t)}{dt} &= \bar{x}(t), & x(0) &= 1-\epsilon,\\
\frac{dy(t)}{dt} &= \bar{y}(t), & y(0) &= 0,\\
\frac{d\bar{x}(t)}{dt} &= \frac{-x(t)}{(x(t)^2+y(t)^2)^{3/2}}, & \bar{x}(0) &= 0,\\
\frac{d\bar{y}(t)}{dt} &= \frac{-y(t)}{(x(t)^2+y(t)^2)^{3/2}}, & \bar{y}(0) &= \left(\frac{1+\epsilon}{1-\epsilon}\right),
\end{aligned} \tag{2.13}$$

where $x(t)$, $y(t)$ are functions representing spatial coordinates and $\bar{x}(t)$, $\bar{y}(t)$ are respectively their derivatives, i.e., velocities, and ϵ is the *orbital eccentricity*, with $\epsilon \in [0,1)$ representing a closed periodic orbit. Many celestial mechanics problems including (2.13) are naturally defined as second-order ODEs (2.5a). An eccentricity of $\epsilon = 0$ corresponds to circular orbit. The solutions of the "D1", "D3", and "D5" IVPs (2.13) from the "non-stiff DE" test set [83] with eccentricities of $\epsilon = 0.1$, 0.5, 0.9, respectively, are shown in Figure 2.5. The governing ODE is the same regardless of the eccentricity and the eccentricity is determined by the initial conditions only. The system of ODEs (2.13) is often known as the "Kepler problem" because if the bodies are assumed to be a Sun and a planet then Kepler's three classic laws of celestial mechanics can be derived directly from (2.13), although polar coordinates are usually used for this derivation in most references [174, pgs.47–60].

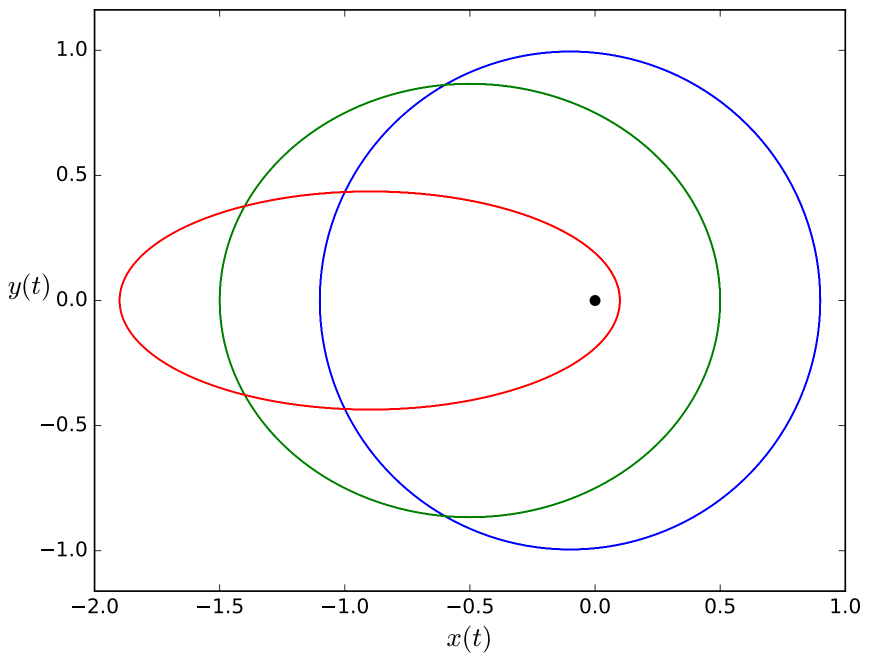


Figure 2.5: Solutions of the "D1" (blue), "D3" (green), and "D5" (red) IVPs (2.13) from the "non-stiff DE" test set [83], with $\epsilon = 0.1$, 0.5, 0.9 respectively. The curve can viewed as a satellite of negligible mass orbiting around a massive body indicated by the black circle.

The van der Pol IVP

An important non-linear ODE is the *van der Pol equation*, with a specific IVP given by

$$\begin{aligned} \frac{dy_1}{dt} &= y_2, & y_1(0) &= 2, \\ \frac{dy_2}{dt} &= \frac{1}{\epsilon}\left((1-y_1^2)y_2 - y_1\right), & y_2(0) &= 0, \end{aligned} \tag{2.14}$$

where ϵ is a non-physical parameter. The van der Pol IVP (2.14) has oscillatory solutions and complete cycles. The parameters $\epsilon = 1$ with $t_f = 8$ and $\epsilon = 0.001$ with $t_f = 2$ are illustrated by Figures 2.6a and 2.6b, respectively. The original importance of the van der Pol equation was that it provided a simple mathematical description of vacuum tube oscillations in the 1920s [115]. Since then, the van der Pol equation has become one of the most widely studied ODEs and found application as a simple system that can describe a wide range of oscillatory phenomena in many applications [117]. An important numerical property is that when $\epsilon \ll 1$ there can be stability issues with some numerical methods including those constructed in this thesis [73, pgs.4–6, 22–23]; this is the phenomena of stiffness that is demonstrated in Section 2.3 and defined in Section 2.5.4.

An Arenstorf orbit IVP

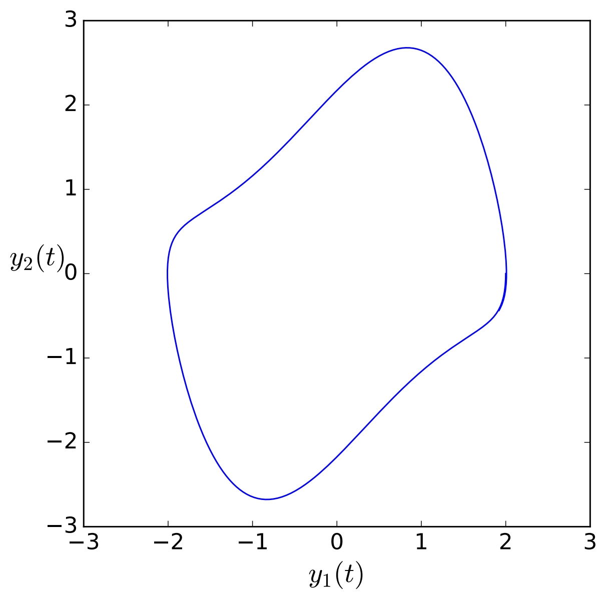
A particular family of 3-body celestial mechanics problems, with analytic solutions known as *Arenstorf orbits*, describe a body with negligible mass (such as a satellite) travelling in a periodic orbit around the centre of mass of two other bodies with non-negligible masses (such as a planet and a moon) [7]. A common example of an Arenstorf orbit has two bodies with the same ratio of masses as the Earth and the Moon but with a circular orbital motion [72, pgs.129–131][106][110, pgs.161–165][173, pgs.513–514]. A rotating coordinate system is chosen so that both the Earth-like and Moon-like bodies remain fixed, allowing the governing equations to be transformed to only describe the satellite [158]. The IVP describing this Arenstorf orbit is given by

$$\begin{aligned}
\frac{dx(t)}{dt} &= \bar{x}(t), \\
\frac{dy(t)}{dt} &= \bar{y}(t), \\
\frac{d\bar{x}(t)}{dt} &= x(t) + 2\bar{y}(t) - (1-\mu)\frac{x(t)+\mu}{D_1} - \mu\frac{x(t)-(1-\mu)}{D_2}, \\
\frac{d\bar{y}(t)}{dt} &= y(t) + 2\bar{x}(t) - (1-\mu)\frac{y(t)}{D_1} - \mu\frac{y(t)}{D_2}, \\
D_1 &= \left((x(t)+\mu)^2 + y(t)^2\right)^{3/2}, \quad D_2 = \left((x(t)+(1-\mu))^2 + y(t)^2\right)^{3/2}, \quad \mu = 0.012277471, \\
x(0) &= 0.994, \quad y(0) = 0, \quad \bar{x}(0) = 0, \quad \bar{y}(0) = -2.00158510637908252240537862224, \\
t_0 &= 0.0, \quad t_f = 17.0652165601579625588917206249,
\end{aligned} \tag{2.15}$$

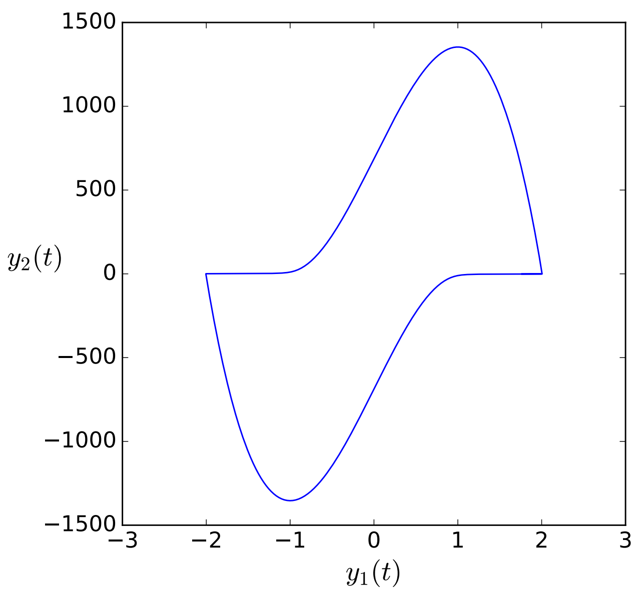
where $x(t)$, $y(t)$ are functions representing the spatial coordinates and $\bar{x}(t)$, $\bar{y}(t)$ respectively are their derivatives, i.e., their velocities, μ is the ratio of mass of the two massive bodies, i.e., the Moon/Earth mass ratio, and t_f is the time the satellite completes and orbit an returns to the initial condition. Note that the "Arenstorf orbit" (2.15) is scaled for analysis purposes and neither the units of mass, time, or space represent SI or other common systems of units. Although the initial conditions that give an Arentstorf orbit are derived by analytic methods, they do not have closed-form values. Therefore, $\bar{y}(0)$ and t_f are published as high-precision numeric values [72, pgs.129–131][110, pgs.161–165].

Due to the periodic solution and an extreme sensitivity to error, the "Arenstorf orbit" IVP (2.15) is a severe test of adaptive IVP solvers [37][110, pgs.161–165] that is used effectively by Hairer et al. for motivating the clear advantage to adaptivity [72, pgs.129–131] (their example is reproduced below in Figure 2.11). Because the periodic solution of the IVP (2.15) always returns to the initial conditions and small numerical difficulties quickly lead to large errors, it is easy to visualize the performance of numerical methods when solving (2.15), making it ideal for demonstrations such as shown by Figure 2.11 in Section 2.3.1.

Briefly described in Section 5.3.4 are 28 other IVPs that are also restricted three-body problems [158],



(a) A solution to the van der Pol IVP (2.14) with $\epsilon = 1$.



(b) A solution to the van der Pol IVP (2.14) with $\epsilon = 0.001$.

Figure 2.6: Solutions to the van der Pol IVP (2.14).

which were originally published by Sharp and are used for performance testing IVP methods in Chapter 5. A set of distinct IVPs that are derived from similar physical systems are ideal for performance testing numerical methods because it ensures that any conclusions reached can be confidently generalized beyond a single IVP.

The "Pleiades" IVP

Because variable-stepsize explicit Runge–Kutta methods are extremely important for solving celestial mechanics problems, numerous different types of IVPs of this class are used for performance testing in addition to (2.13) and (2.15). The "Pleiades" IVP was developed and named as such by Hairer et al. [72, pgs.245–246] because it simulates the motion due to gravity of seven stars and the real Pleiades star cluster is famous for its seven brightest stars. However, the "Pleiades" IVP should not be taken to represent the configuration of or be an actual model of the Pleiades star cluster. The stars in the "Pleiades" IVP have several close approaches (*quasi-collisions*) that result in rapidly varying derivatives that require an adaptive stepsize for an accurate and efficient solution [72, pgs.246]. Consider a plane with star $i \in \{1, 2, \ldots, 7\}$ at coordinate x_i, y_i and mass $m_i = i$, leading to the equations of motion

$$\frac{d^2 x_i}{dt^2} = \sum_{i \neq j} m_j (x_j - x_i)/r_{i,j}, \tag{2.16a}$$

$$\frac{d^2 y_i}{dt^2} = \sum_{i \neq j} m_j (y_j - y_i)/r_{i,j}, \tag{2.16b}$$

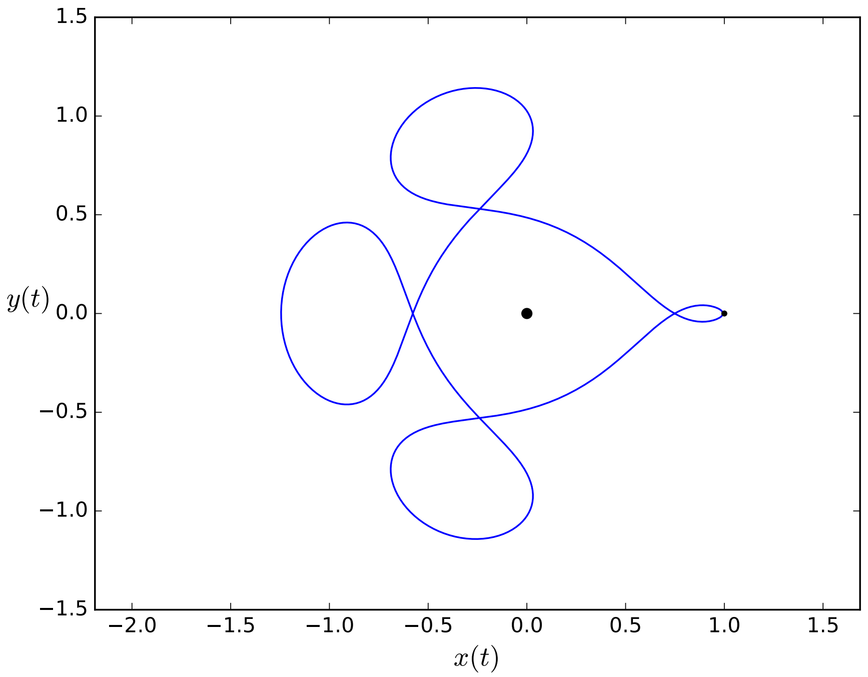


Figure 2.7: The solution of the "Arenstorf orbit" IVP (2.15). The curve can viewed as a massless body orbiting in-between two massive bodies represented by the large and small circles, where the massive bodies are orbiting about a common centre of gravity but a rotating coordinate transformation makes them appear stationary.

where

$$r_{i,j} = ((x_i - x_j)^2 + (y_i - y_j)^2)^{3/2}, \quad i, j = 1, \ldots, 7, \tag{2.16c}$$

with initial conditions

$$\begin{aligned}
&x_1(0) = 3, && x_2(0) = 3, && x_3(0) = -1, && x_4(0) = -3, && x_5(0) = 2, && x_6(0) = -2, && x_7(0) = 2,\\
&y_1(0) = 3, && y_2(0) = -3, && y_3(0) = 2, && y_4(0) = 0, && y_5(0) = 0, && y_6(0) = -4, && y_7(0) = 4,\\
&\frac{dx_6}{dt}(0) = 1.75, && \frac{dx_7}{dt}(0) = -1.5, && \frac{dy_4}{dt}(0) = -1.25, && \frac{dy_5}{dt}(0) = 1,
\end{aligned} \tag{2.16d}$$

and with the other $\dfrac{dx_i}{dt}(0) = 0$, $\dfrac{dy_i}{dt}(0) = 0$. The solution to (2.16) with $t_0 = 0$ and $t_f = 3$ is shown by Figure 2.8.

A semi-discretized PDE for the one-way wave equation

Systems of ODEs that are solved in practice are often those derived from PDEs that have spatial derivatives in addition to time derivatives. Systems of PDEs can be solved numerically by first approximating (*semi-discretizing*) each PDE with a system of ODEs and then numerically solving the ODEs, a procedure known

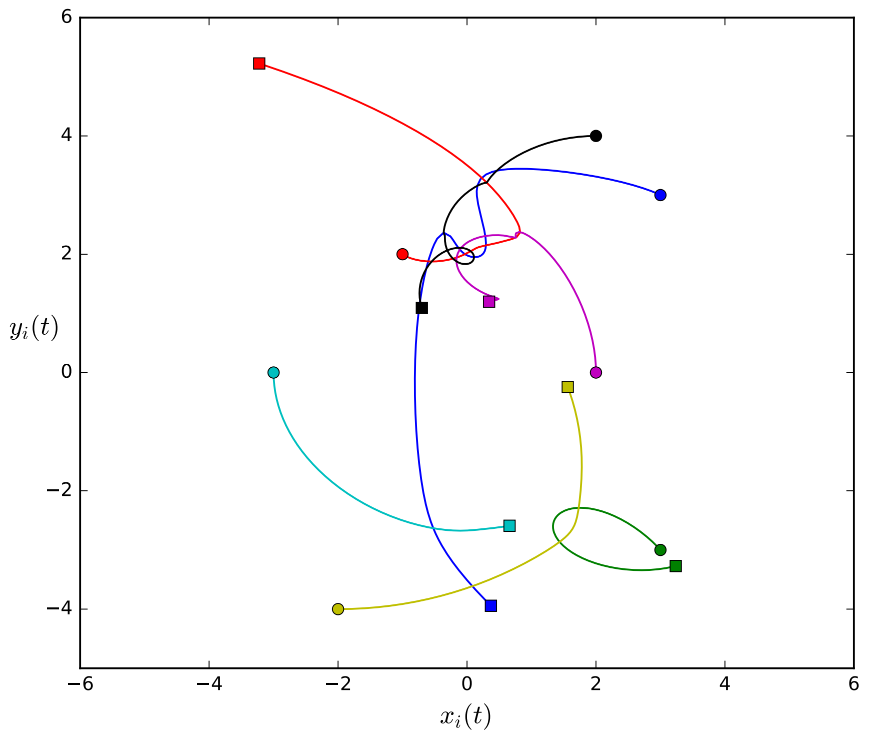


Figure 2.8: Solutions to the "Pleiades" IVP with star 1 x_1, y_1 (blue), star 2 x_2, y_2 (green), star 3 x_3, y_3 (red), star 4 x_4, y_4 (cyan), star 5 x_5, y_5 (magenta), star 6 x_6, y_6 (yellow), and star 7 x_7, y_7 (black). The start of each star path is indicated by a circle and the end is indicated by a square.

as the *method of lines* [86, pgs.94–95][143, pg.14]. A specific example of a PDE is a one-dimensional variable coefficient one-way wave (or advection) equation that is given by

$$\frac{\partial u(t,x)}{\partial t} + c(x)\frac{\partial u(t,x)}{\partial x} = 0, \quad c(x) = \frac{1}{5} + \sin^2(x-1), \tag{2.17}$$

where $x \in \mathbb{R} : 0 \leq x \leq 2\pi$ represents a single dimension in space and $t \in \mathbb{R} : 0 \leq t \leq 8$ represents time [152, pg.114][178, pg.24]. The unknown function $u(t,x)$ describes a "wave height", "concentration", or "velocity" depending on the specific physical manifestation of (2.17). A set of initial conditions commonly used specifically for (2.17) [152, pg.114][178, pg.24] is $u(x,0) = e^{-100(x-1)^2}$, in conjunction with the periodic spatial boundary conditions $u(t,0) = u(t,2\pi)$. The single PDE (2.17) can be approximated with a system of N ODEs by approximating the unknown function $u(t,x)$ at N spatial points that are uniformly spaced by Δx and the solution to each ODE, i.e., $y_0, y_1, \ldots, y_N$, are the approximations of each spatial point. This requires replacing the spatial derivative $\dfrac{\partial u(t,x)}{\partial x}$ with an approximation such as the divided difference formula

$$\frac{\partial u(t,x)}{\partial x} \approx \frac{y_n - y_{n-1}}{\Delta x}, \tag{2.18}$$

which specifically applies when $c(x) > 0$ and is one form of one of the most basic semi-discretization methods that is known as *first-order upwind finite differences* [86, pgs.52–53, 149]. A semi-discretization of $\frac{\partial u(t,x)}{\partial x}$ that is typically more accurate is

$$\frac{\partial u(t,x)}{\partial x} \approx \frac{2y_{n+1} + 3y_n - 6y_{n-1} + y_{n-2}}{6\Delta x}, \tag{2.19}$$

which also specifically applies when $c(x) > 0$ and analogously to the concepts that are defined for IVP methods in Section 2.4.3 converges spatially at a usually more accurate third-order rather than first-order as for (2.18) [86, pg.60]. In fact, advanced semi-discretization methods for PDEs can use many variations of linear divided differences similar to the semi-discretizations (2.18) and (2.19), non-linear semi-discretization formulae, or even techniques such as fast Fourier transforms [152, pg.114][178, pg.24]. These advanced semi-discretization techniques can be studied further in the references [86, 120, 143, 178]. The resulting system of ODEs from (2.17) using the semi-discretization (2.18) gives the IVP

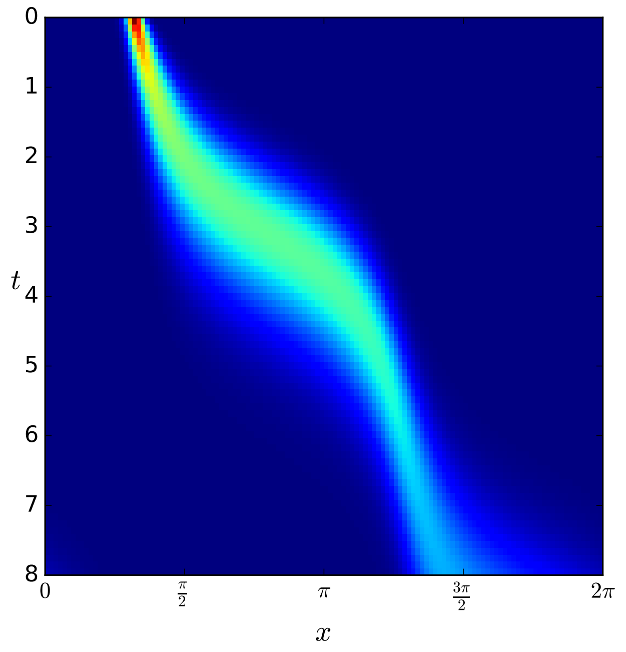
$$\frac{dy_i(t)}{dt} = -c(x)\frac{y_{i+1} - y_i}{\Delta x}, \qquad i \in 1, \ldots, N-1, \tag{2.20a}$$

$$\frac{dy_N(t)}{dt} = -c(x)\frac{y_1 - y_N}{\Delta x}, \tag{2.20b}$$

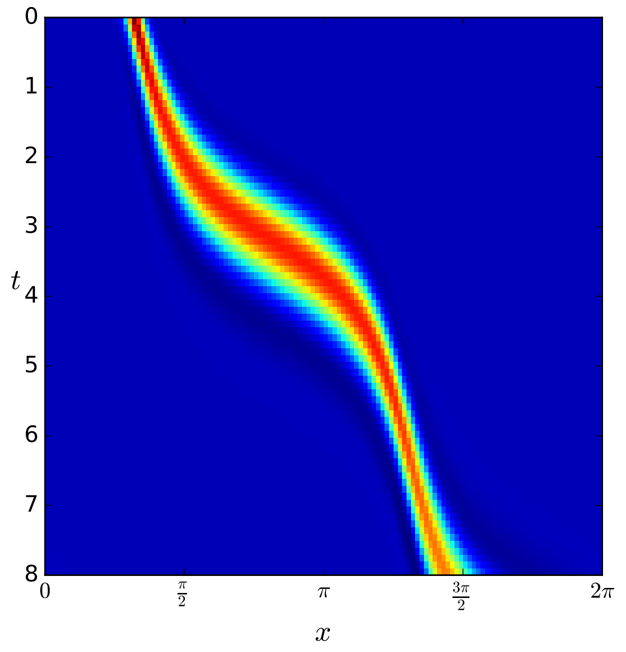
$$y_i(0) = \exp\left(-100\left(i\left(\frac{2\pi}{N}\right) - 1\right)^2\right), \qquad i \in 1, \ldots, N, \tag{2.20c}$$

where (2.20b) approximates the spatial boundary conditions of (2.17) and (2.20c) approximates the initial condition of (2.17). Using (2.19) for semi-discretization of (2.17) gives a similar set of IVPs to (2.20). The solutions to (2.17) with $N = 128$ using (2.18) is given by Figure 2.9a and using (2.19) is given by Figure 2.9b. Based on the solution given in the original reference [178], it can clearly be seen that the semi-discretization (2.19) is more accurate than the semi-discretization (2.18), which continues to be the case with $N = 128$ no matter how accurately the system of IVPs resulting from semi-discretization is solved. In order to achieve a similarly accurate solution to the original PDE (2.17) with the semi-discretization (2.18) as that obtained with the semi-discretization (2.19), $N \gg 128$ is required.

The class of PDEs with properties similar to (2.17) are known as *hyperbolic PDEs*, where simple examples can often be identified by inspection as PDEs that have only first-order spatial and time derivatives [143, pg.5]. Hyperbolic PDEs, which are widely used for mathematical modelling in application areas such as fluid flow and wave phenomena [112, pgs.1–3][120, pgs.2–3], often require specialized solution techniques. The explicit Runge–Kutta methods constructed in this thesis, as well as more sophisticated methods of which they form a component, continue to be extremely important for the solution of semi-discretized hyperbolic PDEs [2, 11, 94, 96, 129, 203]. It is often important to consider the class of underlying PDE and the specific semi-discretization method when selecting or designing numerical methods to solve IVPs derived from semi-discretized PDEs [11][86, pgs.149, 419][197].



(a) Solutions to the one-way wave equation (2.20) with (2.18) used for semi-discretization.



(b) Solutions to the one-way wave equation (2.20) with (2.19) used for semi-discretization.

Figure 2.9: Solutions to the one-way wave equation (2.20).

A semi-discretized PDE for the heat equation

Another common class of PDEs are *parabolic PDEs*, which often arise while modelling phenomena such as diffusion and heat conduction [120, pgs.7][143, pgs.1–2]. Parabolic PDEs have different numerical properties than hyperbolic PDEs. Therefore, they require different numerical methods for semi-discretization and solving the resulting IVPs. A specific example of a semi-discretized parabolic PDE, which was used by Fehlberg [61] for testing one of the original examples of the class of Runge–Kutta methods studied in this thesis (see (2.78) in Section 2.7), is

$$\frac{\partial u(t,x)}{\partial t} = \frac{1}{4}\frac{e^2}{2+x^2}e^{-u(x,t)}\frac{\partial^2 u(t,x)}{\partial x^2}, \tag{2.21a}$$

$$u(t=0,x) = 2\left(1-\log(2-x(t)^2)\right), \tag{2.21b}$$

$$u(t,x=0) : \frac{\partial u(t,x)}{\partial x} = 0, \tag{2.21c}$$

$$u(t,x=1) : u = 2+\log(1+t), \tag{2.21d}$$

with an analytic solution of

$$u(t,x) = 2+\log(1+t) - 2\log(2-x^2), \tag{2.22}$$

where the variables $t \in \mathbb{R}$ and $x \in \mathbb{R}$ correspond to time and a spatial dimension respectively as described for (2.17). Parabolic PDEs can sometimes be identified by inspection because simple examples have second-order spatial derivatives in combination with first-order time derivatives [143, pgs.5].

The unknown function $u(x, t)$ in (2.21) can often be seen as a "temperature" when modelling heat conduction or a "concentration" when modelling diffusion. The second-order spatial derivative in (2.21a) can be approximated (semi-discretized) by the divided difference formula

$$\frac{\partial^2 u(t,x)}{\partial x^2} \approx \frac{y_{n+1} - 2y_n + y_{n-1}}{\Delta x^2}, \tag{2.23}$$

where the unknown function $u(t, x)$ is approximated at time t by the variables y_0, y_1, $\ldots, y_N$, which correspond to N points in x that are uniformly spaced by Δx. The known analytic solution (2.22) to (2.21) can be used to evaluate how well the semi-discretization and IVP method in combination actually simulate the original PDE (2.21). Fehlberg [61] used higher-order semi-discretization methods that are often more accurate than the semi-discretization (2.23).

For testing IVP methods, semi-discretized parabolic PDEs, such as (2.21), can severely test the stability properties of an IVP method and impose severe stepsize restrictions in comparison to semi-discretized hyperbolic PDEs, such as (2.17). It is sometimes quite reasonable to solve the IVPs resulting from semi-discretization of parabolic PDEs to coarse spatial tolerances using the methods constructed in this thesis. However, when fine spatial tolerances are required (either for the parabolic PDE itself or another component of a more complex PDE) the stability restrictions can quickly become severe. This is further discussed as the phenomenon of stiffness in Section 2.5.4. The resulting system of ODEs from (2.21) using the semi-discretization (2.23) results in the IVP given by

$$\begin{aligned} y_0 &= y_1, \\ \frac{d(y_i(t))}{dt} &= \frac{1}{4}\frac{e^2}{\left(2 + (\frac{i}{N})^2\right)} e^{-y_i} \left(\frac{y_{i+1} - 2y_i + y_{i-1}}{\Delta x^2}\right), \quad i = 1, 2, \ldots N \\ y_N &= 2 + \log(1+t), \end{aligned} \tag{2.24}$$

where the solution of these IVPs with $N = 32$ is given by Figure 2.10 and where the analytic solution (2.22) to the original PDE (2.21) indicates a high accuracy is achieved.

2.3 Simple examples of numerical methods for IVPs

Specific examples of numerical methods that are in the same families as the new methods derived in this thesis are first introduced in Section 2.7. However, the introduction of those classes of numerical methods in Section 2.7 first benefits from the theory introduced prior to Section 2.7. To help motivate this mathematical theory as it is introduced, two simpler but commonly used numerical methods for IVPs are demonstrated

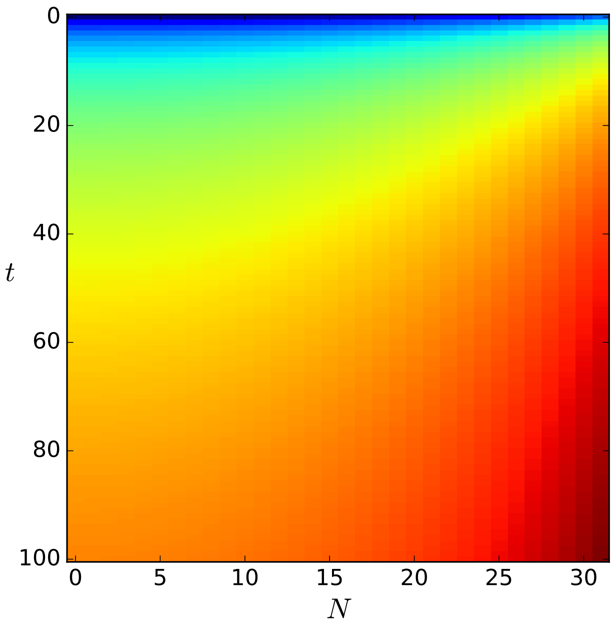


Figure 2.10: A solution to the semi-discretized PDE (2.22) using (2.23) for semi-discretization.

in this section. Like the last section, the reader should refer back to this section as this chapter unfolds if some terminology here is not understood at first. Both of the numerical methods introduced in this section are referenced throughout the thesis. Although both of these methods are also over 100 years old they continue to be used in practice. Sometimes this is because practitioners are not aware of or do not have the expertise and resources to implement better methods. However, sometimes practitioners use certain numerical methods because they have formed the standard practice in particular fields and are therefore the most trusted methods available.

The Forward Euler method

The *forward Euler* (FE) method [152, pg.49], also known as *Euler's method* [27, pg.45][86, pg.24] or *explicit Euler* [86, pg.35], takes a step in t using a simple linear approximation based on evaluating the RHS of the IVP (2.1). The formula for a step of size $\Delta t = t_{n+1} - t_n$ of FE giving a numerical approximation $\mathbf{y}_{n+1}$ at t_{n+1}, by using the previous numerical approximation $\mathbf{y}_n$ at t_n (or the initial condition (2.1b) if the step is the first), is given by

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t_n \mathbf{f}(t_n, \mathbf{y}_n). \tag{2.25}$$

Successive steps of (2.25) are evaluated until the time t_f is reached, with $\mathbf{y}_{n+1}$ at t_f taken as a numerical approximation of the solution (or a numerical solution) to the IVP (2.1) at t_f.

The classic four-stage fourth-order explicit Runge–Kutta method

A step in t with a stepsize of $\Delta t_n = t_{n+1} - t_n$ of a widely used numerical method for IVPs (2.1) is given by

$$\mathbf{k}_1 = \mathbf{f}(t_n,\, \mathbf{y}_n), \tag{2.26a}$$

$$\mathbf{k}_2 = \mathbf{f}(t_n + \frac{\Delta t}{2},\, \mathbf{y}_n + \frac{\Delta t}{2} k_1), \tag{2.26b}$$

$$\mathbf{k}_3 = \mathbf{f}(t_n + \frac{\Delta t}{2},\, \mathbf{y}_n + \frac{\Delta t}{2} k_2), \tag{2.26c}$$

$$\mathbf{k}_4 = \mathbf{f}(t_n + \Delta t,\, \mathbf{y}_n + \Delta t\, k_3), \tag{2.26d}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \frac{1}{6} \left(k_1 + 2k_2 + 2k_3 + k_4 \right), \tag{2.26e}$$

which is an obviously more complex method than FE (2.25) because it uses successive formulae (stages) to find the final numerical approximation $\mathbf{y}_{n+1}$ at t_{n+1}. The numerical method (2.26) is one of the Runge–Kutta methods originally derived by Kutta in 1901 [72, pg.138]. However, (2.26) is often the only Runge–Kutta method known to many practitioners and is sometimes presented as "the Runge–Kutta method" as a result. However, Runge–Kutta methods are actually a general class of numerical methods, which for this thesis are formally defined in Section 2.5. Furthermore, it becomes obvious as the thesis unfolds and in performance testing in Chapter 5 that thinking of (2.26) as "the Runge–Kutta method" is extremely limiting. In the remainder of this thesis, the specific method (2.26) is referred to as RK4.

A more precise definition of a Runge–Kutta method is given in Section 2.5. Runge–Kutta methods are a class of IVP methods with one or more stages, such as the four stages of the RK4 method (2.26). The FE method (2.25) is a Runge–Kutta method with only a single stage. The definition of the order of a numerical method is given in Section 2.4.3. However, using a given stepsize Δt a fourth-order method, such as RK4 (2.26), can often be significantly more accurate a first-order method, such as FE (2.25). In comparison to other classes of IVP methods, adding stages to Runge–Kutta methods can give many additional degrees of freedom in their construction. This means that many desirable properties, such as high order and certain stability properties, can occur in combinations with each other that are not possible with other classes of IVP methods, such as multi-step methods.

The RK4 method (2.26) starts out with an evaluation of the RHS at the beginning of the full step Δt that is used to give a solution at the midpoint of the full step Δt with a miniature FE (2.25) step that then allows another RHS evaluation (2.26b) at the midpoint of the full step Δt. It should be observed that this second RHS evaluation (2.26b) is based on a solution at the end of the second miniature step, and this can intuitively seem less correct than evaluating the RHS at the beginning of a (miniature) step as FE (2.25) does. However, a simple linear approximation based on an RHS evaluation at any t within the full step Δt is always a viable numerical method [27, pgs.86–87] and can often be used as a building block for more complex numerical methods. The third stage (2.26c) uses the RHS evaluation (2.26b) to find a third RHS evaluation that is also at the midpoint of the full step Δt. The last stage uses this latest RHS evaluation (2.26c) for a fourth RHS evaluation (2.26d) at the end of the full step Δt. Finally, a linear combination of these four stages, i.e. (2.26a), (2.26b), (2.26c), and (2.26d) is used to give the numerical approximation $\mathbf{y}_{n+1}$ at t_{n+1} using the RK4 method (2.26). Observe that a linear combination of several RHS evaluations, some of which seem "crude" and others more "refined" at first glance, is required to obtain the best properties for the Runge–Kutta method as a whole.

2.3.1 Examples of numerical solutions

In order to demonstrate the benefits of more complex numerical methods as well as adaptivity, the "Arenstorf orbit" IVP (2.15) is solved with three numerical methods. The solutions are shown in Figure 2.11, which is replicated (using the `pythODE` software package described in Chapter 5) from the classic demonstration by Hairer et al. [72, pgs.130–132]. Notice that although the solution using the FE method (2.25) with $\Delta t = 0.0004$ appears stable, it does not at all resemble the exact solution shown by Figure 2.7. In fact, with a slightly larger stepsize the solution using the FE method (2.25) loses stability and the $x(t)$, $y(t)$ coordinates of the numerical solution quickly (although not exponentially) blow up to $\gg 100$. The solution using the RK4 method (2.26) with $\Delta t = 0.004$ not only takes 2.5 times less work (the number of RHS evaluations are often a good proxy for computational work), it also gives a solution that starts to appear qualitatively correct.

The adaptive Dormand–Prince method (2.79) [47], which is one of the prototypical examples of the classes of new methods derived in this thesis only requires 96 steps for a solution with no apparent inaccuracies. This is about 34 times less computational work than the still obviously inaccurate solution using fixed-stepsize RK4 (2.26). In Figure 2.11 it can be observed that near the initial conditions (and solution after one orbit) is where the adaptive method needs to take small steps but away from this region it can be seen that much larger steps can be taken. There also exist adaptive solvers, e.g., the step-doubling error estimation described in Section 2.6.1, using FE (2.25) and RK4 (2.26) that require somewhat greater cost than the Dormand–Prince method (2.79) [47] but can still be much more economical than a fixed stepsize.

Even with a fixed stepsize of $\Delta t = 0.00001$, the simple FE (2.25) method still gives a noticeably inaccurate solution to (2.15) at t_f (not shown). With the more complex formula for the RK4 (2.26) method, $\Delta t = 0.001$ is all that is necessarily to eliminate any noticeable inaccuracies in the solution of (2.15) (also not shown). The order of convergence (order), introduced below in Section 2.4.3, is the most significant property from additional stages that leads to the increased accuracy seen. FE (2.25) has an order of one, RK4 (2.26) has an order of four, and the Dormand–Prince method (2.79) has an order five.

2.3.2 Examples of unstable numerical solutions

As an example of how the explicit Runge–Kutta methods constructed in this thesis can exhibit instability, consider the van der Pol IVP (2.14) solved with $\epsilon = 10^{-3}$ using RK4 (2.26) and several different stepsizes Δt in Figure 2.12. With a stepsize of $\Delta t = 0.00083$, the solution is stable and accurate although it appears a

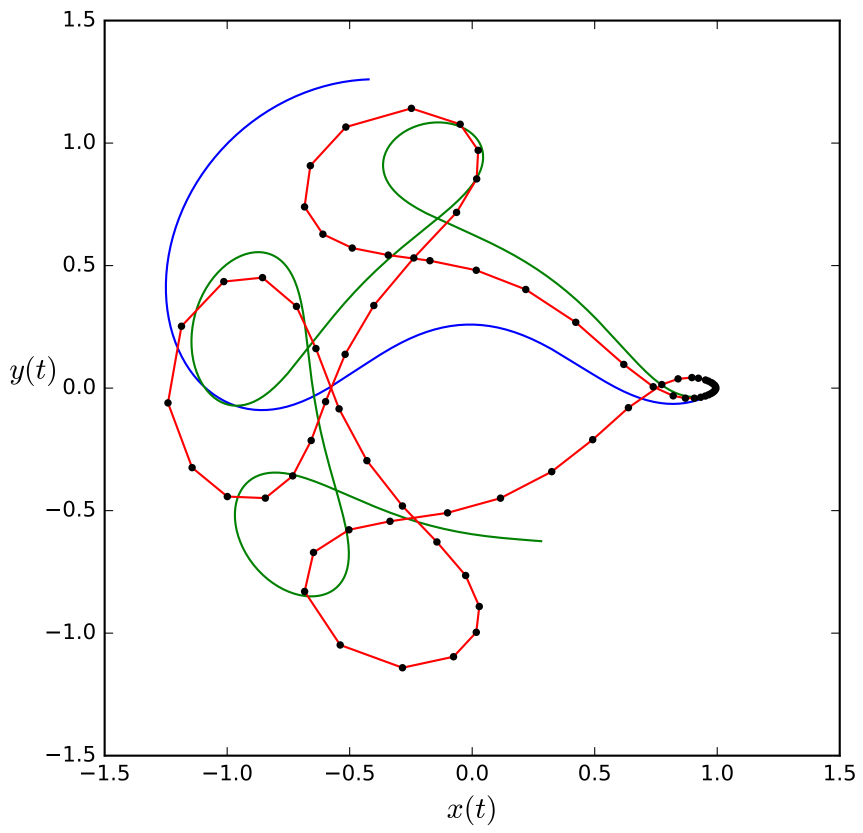


Figure 2.11: The "Arenstorf orbit" IVP (2.15) solved using FE (2.25) (blue, $\Delta t = 0.0004$) taking 42664 steps with 42664 evaluations of the RHS, RK4 (2.26) (green, $\Delta t = 0.004$) taking 4267 steps using 17068 evaluations of the RHS, and the adaptive Dormand–Prince method (2.79) that is introduced in Section 2.7.3 (red with each solution point a black dot) taking only 96 steps with 501 evaluations of the RHS. It is clearly seen that the adaptive Dormand–Prince method (2.79) method is by far the most accurate and requires the fewest steps.

bit jagged due to the large steps taken at certain portions of the solution. With a stepsize of $\Delta t = 0.00084$, the solution loses stability just before completing a cycle. With a stepsize of $\Delta t = 0.00085$, the solution loses stability much earlier before completing even half a cycle and the bottom-right of Figure 2.12 shows the same stepsize, i.e., $\Delta t = 0.00085$, but clearly demonstrates the catastrophic blowup to extremely large numbers $\approx 10^{100}$. This example shows how a numerical solution can be unstable even when the IVP being solved is extremely stable; this is the phenomena of stiffness that is discussed in detail in Section 2.5.4.

2.4 Numerical analysis of IVPs

2.4.1 Basic formulation of numerical methods

It is well-known that many ODEs and the IVPs based on them, such as most of the IVPs shown in the previous section, do not have closed-form solutions. Therefore, the only solutions that can often be computed

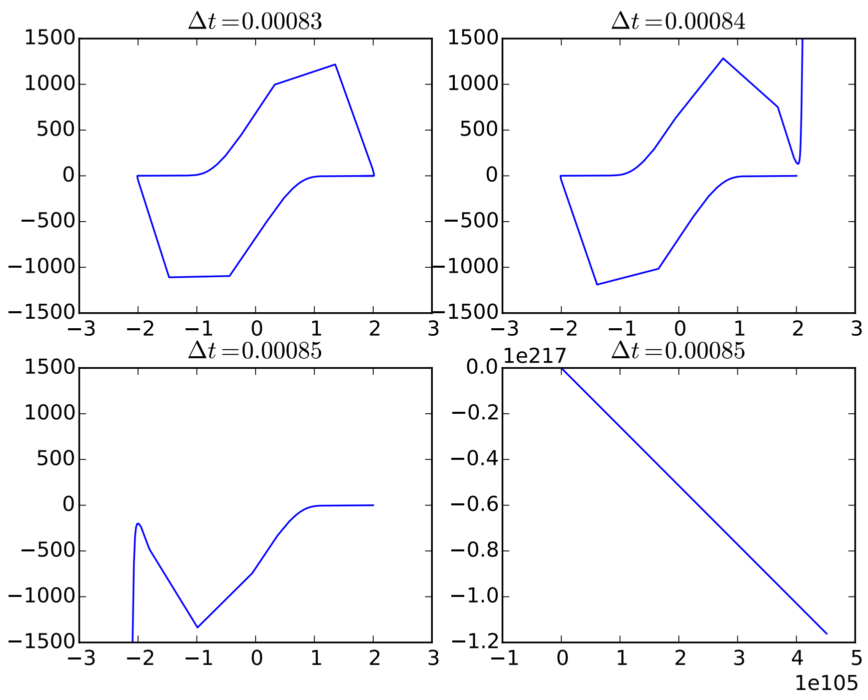


Figure 2.12: Examples of a stable numerical solution with $\Delta t = 0.00083$, as well unstable numerical solutions with $\Delta t = 0.00084$ and $\Delta t = 0.00085$. The full extent of the catastrophic blow-up with $\Delta t = 0.00085$ can be seen by the scale in the lower-right corner.

are approximations. In addition, because the exact solutions to ODEs are defined on a continuous domain, approximate solutions can only be computed at a finite number of discrete points using *numerical methods*.

A numerical method for an IVP (2.1) is a formula for a *numerical solution* $\mathbf{y}_n$ at time t_n in terms of the numerical solutions $\mathbf{y}_{n-1}, \mathbf{y}_{n-2}, \ldots, \mathbf{y}_1, \mathbf{y}_0$ at times $t_{n-1}, t_{n-2}, \ldots, t_1, t_0$ that is given by

$$\mathbf{y}_n = \Psi(\mathbf{y}_{n-1}, \mathbf{y}_{n-2}, \ldots, \mathbf{y}_1, \mathbf{y}_0), \tag{2.27}$$

where $\Psi : \mathbb{R}^n \to \mathbb{R}^n$ is the *numerical method* or *numerical formula* for one step. The process of applying the formula Ψ to obtain $\mathbf{y}_n$ from the previous solutions of $\mathbf{y}$ in (2.27) is known as a *timestep* with a *stepsize* of $\Delta t_n = t_{n+1} - t_n$ for a step taken from t_{n+1} to t_n. If only the previous solution at $\mathbf{y}_{n-1}$ is used in the formula $\Psi(\mathbf{y}_{n-1})$ then the IVP (2.1) method is known as a *single-step* method. Otherwise, if other previous solution points are used, the IVP method is known as a *multi-step* method. Although only *single-step* Runge–Kutta methods are studied in this thesis, the reader should be aware that *multi-step* methods provide strong competitors that could be a topic for future studies based on this thesis.

2.4.2 Series expansion of numerical solutions

A *local solution* $\tilde{\mathbf{y}}_n(t)$ is an exact solution of the ODE (2.1a) that satisfies a given solution to the ODE (2.1a) $\mathbf{y}_n$ at t_n (that can be an exact solution or numerical solution depending on the context) as the initial condition (2.1b) [154]. The *Taylor series* [72, pg.46] that gives the exact solution $\tilde{\mathbf{y}}_n(t_n + \Delta t)$ in terms of the known solution $\tilde{\mathbf{y}}_n(t_n)$ is given by

$$\tilde{\mathbf{y}}(t_n + \Delta t) = \tilde{\mathbf{y}}(t_n) + \sum_{i=1}^{\infty} \frac{(\Delta t)^i}{i!} \frac{d^{(i-1)}}{dt^{(i-1)}} \mathbf{f}(\tilde{\mathbf{y}}(t_n)), \tag{2.28}$$

which is presented as a formulation that is convenient for studying numerical methods for IVPs (2.1) and that can easily be seen to be equivalent to more commonly used formulations [72, pg.46] of the Taylor series because $\frac{d^{(i)}}{dt^{(i)}}\tilde{\mathbf{y}}(t_n) = \frac{d^{(i-1)}}{dt^{(i-1)}}\mathbf{f}(\tilde{\mathbf{y}}(t_n))$. The *derivative of a numerical method* for an IVP is found by taking the derivative of the numerical formula (2.27) at the beginning of the desired timestep by using *Leibniz' formula* [72, pg.144], i.e.,

$$(\Delta t \, \psi(\tilde{\mathbf{y}}_n))^{(q)} \,|_{\Delta t = 0} = q \; (\psi(\tilde{\mathbf{y}}_n))^{(q-1)} \,|_{\Delta t = 0},$$

to evaluate derivatives of terms of the numerical formula Ψ (2.27) that contain Δt and where $\Delta t \, \psi(\tilde{\mathbf{y}}_n)$ is a term of Ψ (2.27) with a factor of Δt. Numerical formulae exist with factors that are non-linear in Δt but these are beyond the scope of this study. As with any other mathematical expression, the higher derivatives of a numerical method are found by repeatedly taking first derivatives. This allows the *Taylor series of a numerical method* at $\mathbf{y}_n$ to be defined as the Taylor series (2.28) using the numerical formula (2.27) for the function $\tilde{\mathbf{y}}(t_n)$ and with the derivatives of the numerical method used as the derivatives of $\tilde{\mathbf{y}}(t_n)$ in (2.28). Using the formulation (2.28) ensures that the Taylor series expansion exactly reflects a timestep of the numerical method [72, pg.144].

A truncated Taylor series (2.28) can in fact be used as a numerical method, and these are known as *Taylor series methods* [27, pg.107][40], which can be used to give extremely accurate numerical solutions. However, practical implementations of Taylor series methods require computing high-order derivatives, which can lead to a much more difficult implementation than the IVP methods described in this thesis, require additional libraries for tasks such as automatic differentiation, and are not efficient for solving many IVPs in practice to the typical (often coarse) accuracies desired [138].

2.4.3 Errors in numerical solutions

The *global error* of a numerical solution of an IVP (2.1) at time t_{n+1} is defined as

$$\mathbf{E}_{\text{global},n+1} = \mathbf{y}(t_{n+1}) - \mathbf{y}_{n+1}, \tag{2.29}$$

where $\mathbf{y}(t_{n+1})$ is the exact solution of the IVP (2.1) at t_{n+1} given the initial condition (2.1b) at t_0, and $\mathbf{y}_{n+1}$ is the numerical solution at timestep $n+1$ with a time of t_{n+1} using the initial condition at time t_0 [72, pg.159].

The *local error* of the timestep from t_n to t_{n+1} with a stepsize Δt of the numerical solution $\mathbf{y}_{n+1}$ is given by

$$\mathbf{E}_{\text{local}} = \tilde{\mathbf{y}}_n(t_{n+1}) - \mathbf{y}_{n+1}, \tag{2.30}$$

where $\tilde{\mathbf{y}}_n(t_{n+1})$ is the local solution at t_{n+1} using the numerical solution $\mathbf{y}_n$ at t_n as an initial condition and $\mathbf{y}_{n+1}$ is the numerical solution at the end of timestep $n+1$ using the numerical solution $\mathbf{y}_n$ at t_n as an initial condition [72, pg.156]. In the context of IVPs, the global error and local error are often be referred to as the *global truncation error* [108, pg.57] and *local truncation error* [108, pgs.56,152], respectively. There can be several variants on the definition of local truncation error for different classes of IVP methods [108, pg.27] and to avoid confusion the term "truncation error" is not used further in this thesis.

The asymptotic behaviour of the local error is $\mathcal{O}(\Delta {t_n}^{p+1})$, where p is known as the *order of convergence* of the numerical method. The asymptotic behaviour of the global error is $\mathcal{O}(\Delta {t_n}^{p})$ because a numerical method takes $\mathcal{O}\left(\frac{t_f - t_0}{\Delta t}\right)$ steps in total over the solution interval assuming $t_f - t_0 = \mathcal{O}(1)$ [147]. When Δt is large, the order of convergence may not be an accurate description of the asymptotic behaviour of a numerical method even with a substantial reduction in Δt. However, when Δt is small enough that the order of convergence does accurately describe the behaviour of a numerical method, then the value of Δt is known as being in the *asymptotic region*. In general, numerical methods with a higher order of convergence have the potential for higher accuracy, as already shown by Figure 2.11. In addition, due to the faster convergence of higher-order methods, extremely inaccurate solutions such as those seen with FE (2.25) and RK4 (2.26) in Figure 2.11 can be often be diagnosed and eliminated with a minimal increase in computational cost. For example, for the numerical solutions shown by Figure 2.11 when using the first-order FE (2.25) a 400-fold increase in computational cost was required to mostly eliminate the visible inaccuracy, whereas when using the fourth-order RK4 method (2.26) only a 4-fold increase in computational cost eliminated any visible inaccuracy in the solution.

2.4.4 Convergence and consistency of numerical methods

To have any possibility of a viable numerical method for solving IVPs or to give any guarantees whatsoever that the numerical solution at discrete points is a useful approximation of the true solution defined on a continuous domain, a numerical method must have the property of *convergence*. A numerical method for an IVP (2.1) is called convergent if for any Lipschitz continuous (2.6) IVP, the numerical solution approaches the exact solution in the limit of the stepsize going to zero [108, pg.152]. This is shown by the relation

$$\lim_{\Delta t \to 0} \mathbf{y}(t_n) = \mathbf{y}_n, \forall\, t_n \in [t_0,\, t_f]. \tag{2.31}$$

However, the above definition of convergence is difficult to apply directly for determining whether a numerical method is convergent. A necessary condition for a numerical method to be convergent is that a numerical method must also be *consistent*, i.e., ensuring that the cumulative errors of each step approaches zero in the limit where the stepsize goes to zero [108, pgs.28,152]. Consistency can be ensured by the relation

$$\lim_{\Delta t \to 0} \frac{1}{\Delta t} \mathbf{E}_{\text{local}} = 0. \tag{2.32}$$

It may seem that removing the $\frac{1}{\Delta t}$ factor from (2.32) might give a viable numerical method because in the limit the local error at each step would still go to zero. However, this is not sufficient for convergence because on average $\mathbf{E}_{\text{local}}$ must converge faster than Δt. For the numerical methods studied in this thesis (Runge–Kutta methods), a specific consistency condition [108, pg.152] that can be found from (2.32) is always equivalent to convergence and is defined below in Section 2.5. However, for some classes of numerical methods, such as multistep methods, consistency (2.32) is not always a sufficient condition for convergence [108, pg.31].

This thesis is concerned with variable-stepsize methods, which can sometimes have complicated behaviour at coarse Δt. However, it can easily be seen that the above conditions for convergence and consistency still hold for variable stepsizes when the limit $\Delta t \to 0$ is replaced by $\max \Delta t \to 0$.

2.5 Runge–Kutta methods

Given a solution at t_n to an IVP (2.1), a *Runge–Kutta* (RK) method finds a numerical approximation at t_{n+1} by using linear combinations of $\mathbf{f}(t,\, \mathbf{y}(t))$ from (2.1a) evaluated at intermediate *abscissae* that generally lie within the timestep $t_n \le t \le t_{n+1}$. The formula for an RK method is given by

$$\mathbf{k}_i = \mathbf{f}\left(t_n + \Delta t_n c_i, \quad \mathbf{y}_n + \Delta t_n \sum_{j=1}^{s} a_{i,j}\, \mathbf{k}_j\right), \quad i = 1, 2, \ldots, s, \tag{2.33a}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t_n \sum_{i=1}^{s} b_i\, \mathbf{k}_i, \tag{2.33b}$$

where the $\mathbf{k}_i \in \mathbb{R}^m$ (2.33a) are the s intermediate *stages* of the RK method that are the derivatives $\mathbf{f}(t_n + \Delta t_n c_i,\, \mathbf{y})$ of a numerical approximation of $\mathbf{y}(t_n + \Delta t_n c_i)$ [27, pgs.86,123][72, pg.134][108, pg.149]. The *coefficients* defining the particular RK method (2.33) are the values of $a_{i,j}$, b_i, and c_i. The $a_{i,j}$ values are the elements of a matrix $\mathbf{A} \in \mathbb{R}^{s \times s}$. The values of b_i and c_i are elements of the vectors $\mathbf{b}$, $\mathbf{c} \in \mathbb{R}^s$, respectively. Every RK method can be used as a method for solving quadrature problems, which only requires using the $\mathbf{b}$ and $\mathbf{c}$ vectors (2.2) vectors [27, pgs.86–87][72, pg.132]. Therefore, the components of the $\mathbf{b}$ vector are sometimes known as the *quadrature weights* [27, pg.87] and the components of the $\mathbf{c}$ vector are sometimes known as the *quadrature points* [172, pg.238]. An important reason for relating the RK method (2.33) to the underlying method for solving quadrature problems (2.2) is that in some cases the order or other properties of the underlying quadrature method for solving a quadrature problem (2.2) are different from the RK method (2.33) applied to a non-quadrature IVP. This can cause issues for many ODEs (2.1a), such as those that have regions where they are “near quadrature” problems [27, pgs.192–193][50, 136].

Particular RK methods (2.33) where the summation in (2.33a) has an upper bound of $i - 1$ instead of

s are known as *explicit Runge–Kutta* (ERK) methods because each stage k_i only depends on the previous stages $k_j : j < i$, a property that allows each stage k_i to be explicitly computed. Otherwise, an RK method is known as an *implicit Runge–Kutta* (IRK) method, where some stages depend on other stages that cannot be explicitly computed before them and hence require the solution of implicit systems of equations to apply the numerical formula. This thesis is only concerned with ERK methods, but it should be clear from the context which statements apply to all RK methods and which statements apply only to certain classes of RK methods such as ERK methods. A specific RK method is usually specified as a *Butcher tableau*, given by

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|cccc} c_1 & a_{1,1} & a_{1,2} & \dots & a_{1,s} \\ c_2 & a_{2,1} & a_{2,2} & \dots & a_{2,s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s,1} & a_{s,2} & \dots & a_{s,s} \\ \hline & b_1 & b_2 & \dots & b_s \end{array} \tag{2.34}$$

where $\mathbf{A}$ is a strictly lower triangular matrix in the case of an ERK method [27, pgs.86,92,196][72, pgs.134–135, pg.205][73, pgs.71,118][108, pgs.149–151]. As an example, the Butcher tableaux (2.34) of the FE method (2.25) and RK4 method (2.26) are respectively given by

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \qquad \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

In order to simplify the construction of RK methods (2.33), the *row-sum* conditions [58, 168] given by

$$c_i = \sum_{i}^{s} a_{ij}, \tag{2.35}$$

are almost universally assumed in the literature and are always assumed in this thesis.

2.5.1 Elementary differentials and order conditions

The Taylor series (2.28) of a numerical method (2.27) for IVPs can be used to study the order of convergence and error of RK methods. The function $\mathbf{y}(t)$ representing the solution of an IVP (2.1) is unknown. However, the first derivative of $\mathbf{y}(t)$ is the known function $\mathbf{f}(t, \mathbf{y}(t))$, i.e., the RHS of the ODE (2.1), that can be used to explicitly find the higher derivatives of $\mathbf{y}(t)$. The expressions for the first and higher derivatives can be described using constructs known as *elementary differentials*, which is terminology first introduced by John Butcher [25][27, pgs.134–138][70]. The first derivative of the solution of $\mathbf{y}(t)$ is associated with an elementary differential contained in the set

$$\mathfrak{F}^1(\mathbf{f}(\mathbf{y}(t))) = \{\mathbf{f}(\mathbf{y}(t))\}. \tag{2.36a}$$

The elementary differentials associated with higher-order derivatives form a set of expressions. The set of elementary differentials associated with derivative q of $\mathbf{y}(t)$ is the set $\mathfrak{F}^q(\mathbf{y}(t))$ containing a representative of each equivalent expression in the set $\bar{\mathfrak{F}}^q(\mathbf{f}(\mathbf{y}(t)))$ that is defined as

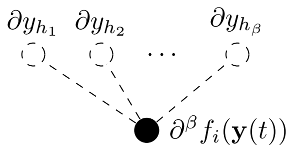
$$\bar{\mathfrak{F}}^q(\mathbf{f}(\mathbf{y}(t))) = \left\{ \begin{array}{c} \mathbf{F}^q(\mathbf{f}(\mathbf{y}(t))) = \sum_{i=1}^m \left(\sum_{h_1,h_2,\ldots,h_\beta=1}^m \dfrac{\partial^\beta f_i(\mathbf{y}(t))}{\partial y_{h_1}\partial y_{h_2}\ldots\partial y_{h_\beta}} (\mathbf{F}^{\delta_1})_{h_1} (\mathbf{F}^{\delta_2})_{h_2} \ldots (\mathbf{F}^{\delta_\beta})_{h_\beta} \right) \mathbf{e}_i \\ q = 1+\delta_1+\delta_2+\ldots+\delta_\beta \\ \beta \in 1,2,\ldots,q-1 \\ \delta_j \in 1,2,\ldots,q-1 \\ \mathbf{F}^{\delta_j} \in \mathfrak{F}^{\delta_j}(\mathbf{y}(t)) \end{array} \right\}, \tag{2.36b}$$

where $\mathbf{F}^q = \mathbf{F}^q(\mathbf{f}(\mathbf{y}(t))) : \mathbb{R}^m \to \mathbb{R}^m$ is a representative elementary differential of order q, y_{h_i} is component h_i of $\mathbf{y}(t)$, $(\mathbf{F}^{\delta_j})_{h_i}$ is component h_i of the elementary differential $\mathbf{F}^{\delta_j}$, and $\mathbf{e}_i \in \mathbb{R}^m, i \in 1,2,\ldots,m$, are the standard basis vectors [72, pgs.145–150][108, pgs.157–162].

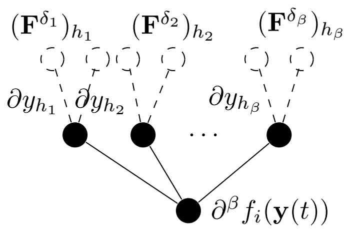
The indices of the elementary differential expressions (2.36b) occur in a regular and recursive pattern. Therefore, a more compact representation of each elementary differential can be found by considering each elementary differential (2.36) isomorphic to a *rooted tree* [27, pgs.88–90,134–138][72, pgs.147–149]. The inner summation of (2.36b) given by

$$\sum_{h_1,h_2,\ldots,h_\beta=1}^m \frac{\partial^\beta f_i(\mathbf{y}(t))}{\partial y_{h_1}\,\partial y_{h_2}\,\ldots\,\partial y_{h_\beta}} (\mathbf{F}^{\delta_1})_{h_1} (\mathbf{F}^{\delta_2})_{h_2} \ldots (\mathbf{F}^{\delta_\beta})_{h_\beta},$$

can be seen as having the $\partial^\beta f_i(\mathbf{y}(t))$ and ∂y_{h_i} expressions isomorphic to the root and the lowest branches of a rooted tree, respectively. This is shown by

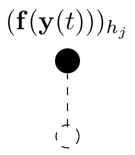


with each component $\mathbf{F}^{\delta_1}$, $\mathbf{F}^{\delta_2}$, ..., $\mathbf{F}^{\delta_\beta}$ of the elementary differential, i.e., $(\mathbf{F}^q)_j$ (2.36b), recursively attaching to the roots of the lowest branches as shown by

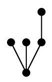


$(\mathbf{F}^{\delta_1})_{h_1}$
$(\mathbf{F}^{\delta_2})_{h_2}$
$(\mathbf{F}^{\delta_\beta})_{h_\beta}$
∂y_{h_1}
∂y_{h_2}
∂y_{h_β}
$\cdots$
$\partial^\beta f_i(\mathbf{y}(t))$

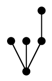
and using the elementary differential corresponding to (2.36a) for terminating a branch as shown by



The elementary differential (2.36a) corresponds to the single root $\bullet$. The rooted tree corresponding to an elementary differential (2.36) is often used to denote the one of the equivalent expressions from $\mathfrak{F}^q(\mathbf{y}(t))$ that have a structure corresponding to that tree. For example, the tree



corresponds to one of the equivalent expressions $\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))$, *s.t.* $q = 5$ from (2.36) that all have a structure corresponding to that same tree. This equivalence is shown for one of these expression in $\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))$, *s.t.* $q = 5$ by



$$\cong \sum_{i=1}^{m}\left(\sum_{h_1,h_2,h_3=1}^{m}\frac{\partial^3 f_i(\mathbf{y}(t))}{\partial y_{h_1}\,\partial y_{h_2}\,\partial y_{h_3}}\,(\mathbf{f}(\mathbf{y}(t)))_{h_1}\,(\mathbf{f}(\mathbf{y}(t)))_{h_2}\left(\sum_{j_1=1}^{m}\frac{\partial f_{h_3}(\mathbf{y}(t))}{\partial y_{j_1}}(\mathbf{f}(\mathbf{y}(t)))_{j_1}\right)\right)\mathbf{e}_i.$$

If all rooted trees with q vertices are enumerated [27, pgs.134–138][72, pgs.147–149], this corresponds exactly to all equivalent elementary differentials of order q, a property that is exploited heavily to generate code implementing elementary differentials and other related expressions in the `OCSage` software package described in Chapter 4.

The elementary differentials (2.36) of order q can be used to express derivative q of an unknown function $\mathbf{y}(t)$ (2.1a) by the equivalent expressions

$$\begin{aligned}\frac{d^q}{dt^q}\mathbf{y}(t) &= \sum_{\mathbf{F}\in\mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))}\alpha(\mathbf{F})\,\mathbf{F},\\ \frac{d^q}{dt^q}\mathbf{y}(t) &= \sum_{\mathbf{F}\in\mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))}\frac{q!}{\sigma(\mathbf{F})\,\gamma(\mathbf{F})}\,\mathbf{F},\end{aligned} \tag{2.37}$$

where $\alpha(\mathbf{F})$ is the number of distinct ways of labelling a tree, $\gamma(\mathbf{F})$ is known as the density of the tree, and

$\sigma(\mathbf{F}) = \frac{q!}{\alpha(\mathbf{F})\gamma(\mathbf{F})}$ is known as the order of the symmetry group of the tree, all of which are all explained and derived in detail by Butcher [27, pgs.123–131][24] and others [72, pgs.147–149], in addition to as implemented in `OCSage`. The Taylor series expansion (2.28) of the exact solution of $\mathbf{y}(t)$ (2.1a) in terms of $\mathbf{f}(t, y)$ can be given using the elementary differentials (2.36) [72, pgs.148–149] as one of the equivalent expansions

$$\mathbf{y}(t_n + \Delta t) = \mathbf{y}(t_n) + \Delta t\, \mathbf{f}(\mathbf{y}(t)) + \ldots + \frac{\Delta t^q}{q!} \sum_{\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))} \alpha(\mathbf{F})\, \mathbf{F} + \ldots,$$
$$\mathbf{y}(t_n + \Delta t) = \mathbf{y}(t_n) + \Delta t\, \mathbf{f}(\mathbf{y}(t)) + \ldots + \Delta t^q \sum_{\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))} \frac{1}{\sigma(\mathbf{F})\, \gamma(\mathbf{F})}\, \mathbf{F} + \ldots. \tag{2.38}$$

To use a particular elementary differential $\mathbf{F}$ (2.36) to find the derivative of the numerical solution of an RK method (2.33), an additional quantity known as the *scalar sum* $\Phi(\mathbf{F})$ is required and is given by

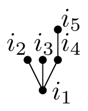
$$\Phi(\mathbf{F}) = \sum_{i=1}^{s} b_i\, \Phi_i(\mathbf{F}), \tag{2.39a}$$

where $\Phi_i(\mathbf{F})$ is defined as

$$\Phi_i(\mathbf{F}) = \sum_{j_1, j_2, \ldots, j_\varsigma = 1}^{s} a_{\varrho(j_1), j_1}\, a_{\varrho(j_2), j_2}\, \ldots\, a_{\varrho(j_\varsigma), j_\varsigma}, \tag{2.39b}$$

where each factor $a_{\varrho(j_k), j_k}$ corresponds to one of the $a_{i,j}$ coefficients of the RK method (2.33).

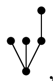
The set of edges of the rooted tree corresponding to $\mathbf{F}$ is given by $\{\varrho(j_k),\, j_k : j_k \in \{j_1,\, j_2,\, \ldots,\, j_\varsigma\}\}$ with each element being edge k, where its closest vertex to the root is given by $\varrho(j_k)$ and its furthest vertex from the root is j_k alone [72, pg.150]. This gives the scalar sum (2.39) a particular structure linking the subscripts between successive factors to form the product that composes each term of the summation. An example of a correspondence between a rooted tree and a scalar sum is



$$\cong \sum_{i_1=1}^{s} b_{i_1} \sum_{i_2, i_3, i_4, i_5 = 1}^{s} a_{\varrho(i_2), i_2} a_{\varrho(i_3), i_3} a_{\varrho(i_4), i_4} a_{\varrho(i_5), i_5}$$
$$= \sum_{i_1=1}^{s} b_{i_1} \sum_{i_2, i_3, i_4, i_5 = 1}^{s} a_{i_1, i_2} a_{i_1, i_3} a_{i_1, i_4} a_{i_4, i_5}$$
$$= \sum_{i_1=1}^{s} b_{i_1} \sum_{i_6=1}^{s} c_{i_1} c_{i_1} a_{i_1, i_6} c_{i_6},$$

where the symbol $\cong$ indicates the tree is isomorphic to the scalar sums and using the row-sum conditions (2.35) the $a_{i,j}$ corresponding to branch tips can be replaced by c_i. It can now be seen that in addition to being isomorphic to an elementary differential $\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))$ itself, depending on context, each rooted tree can also be seen as isomorphic to the scalar sum $\Phi(\mathbf{F})$ (2.39) derived from the elementary differential $\mathbf{F}$ [27, pgs.140–144][72, pgs.151–153].

Following Butcher [27, 124–125] and [108, 164–165], a more compact notation for rooted trees used within the text of this thesis is given by the example

$$[\tau^2[\tau]] \cong$$

$$, \tag{2.40}$$

where the symbol τ corresponds to vertices at the end of a branch, the elements in the outer square brackets are the elements branching off of the root of the tree, and each layer of square brackets corresponds to a branch of the tree. Multiple elements within a set of square brackets corresponds to a vertex with multiple branches extending from it. More examples of this square-bracket notation along with equivalent rooted trees and scalar sums can be found in Tables B.1, B.2, and B.3.

Although the row sum conditions (2.35) are generally assumed when specifying the scalar sums (2.39) and order conditions (2.43), they are not strictly required for low-order RK methods, e.g., see Hairer et al [72, pg.134]. It is still unknown whether the row-sum conditions (2.35) are required for higher-order RK methods. In addition, the row-sum conditions (2.35) are necessary for the intermediate numerical approximation of $\mathbf{y}$ at $t_n + \Delta t_n c_i$ for each stage of an RK method (2.33) to have an order of convergence of at least one, and therefore for the individual stages to be convergent and consistent. Whether practical and efficient RK methods without the row-sum conditions (2.35) could be constructed remains an open problem.

The derivative of the numerical solution of an RK method can now be given by the two equivalent expressions

$$\begin{aligned}
\frac{d^q}{dt^q}\mathbf{y}_n &= \sum_{\mathbf{F}\in\mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))} \alpha(\mathbf{F})\,\gamma(\mathbf{F})\,\Phi(\mathbf{F})\,\mathbf{F},\\
\frac{d^q}{dt^q}\mathbf{y}_n &= \sum_{\mathbf{F}\in\mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))} \frac{q!}{\sigma(\mathbf{F})}\,\Phi(\mathbf{F})\,\mathbf{F},
\end{aligned} \tag{2.41}$$

where $\alpha(\mathbf{F})$, $\gamma(\mathbf{F})$, and $\sigma(\mathbf{F})$ are the same functions on trees used in (2.38) [27, pg.145–146][72, pg.151]. These derivatives of the numerical solution allow the Taylor series of the numerical solution of an RK method to be given by the two equivalent expansions

$$\begin{aligned}
\mathbf{y}_n(t_n+\Delta_t) &= \mathbf{y}(t_n) + \Delta t\,\mathbf{f}(\mathbf{y}(t)) + \ldots + \frac{\Delta t^q}{q!}\sum_{\mathbf{F}\in\mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))} \alpha(\mathbf{F})\,\gamma(\mathbf{F})\,\Phi(\mathbf{F})\,\mathbf{F} + \ldots,\\
\mathbf{y}_n(t_n+\Delta_t) &= \mathbf{y}(t_n) + \Delta t\,\mathbf{f}(\mathbf{y}(t)) + \ldots + \Delta t^q\sum_{\mathbf{F}\in\mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))} \frac{1}{\sigma(\mathbf{F})}\,\Phi(\mathbf{F})\,\mathbf{F} + \ldots.
\end{aligned} \tag{2.42}$$

It can clearly be seen that the terms involving the elementary differential $\mathbf{F}$ in the Taylor series of the numerical solution (2.42) differs from the Taylor series of the exact solution (2.38) only by the extra factors of $\gamma(\mathbf{F})$ and the scalar sum $\Phi(\mathbf{F})$ in the Taylor series of the numerical solution (2.42) [27, pgs.147–148][72, pg.158]. Therefore, the terms corresponding to $\mathbf{F}$ of the Taylor series of the exact solution (2.38) and numerical solution (2.42) are only equal if $\gamma(\mathbf{F})\Phi(\mathbf{F}) = 1$, i.e., the *order condition* corresponding to $\mathbf{F}$ [27, pg.147][47][72, pg.153][94][108, pg.165], that is often given in the form

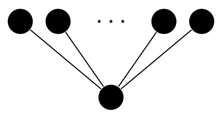
$$\Phi(\mathbf{F}) - \frac{1}{\gamma(\mathbf{F})} = 0, \quad \mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))). \tag{2.43}$$

If all of the order conditions (2.43) that correspond to all elementary differentials of the orders up to q are satisfied, then the RK method in question will be order of convergence q [27, pgs.147–148][72, pgs.153–154]. A summary of the number of trees for each order, total number of order conditions for each order, and minimum number of stages necessary for ERK methods of a particular order is given in Table 2.1. Some relatively narrow bounds on the minimum number of stages to achieve up to fourteenth-order are known because examples of ERK methods of up to fourteenth-order have been constructed while attempting to maintain a relatively minimal number of stages [59], only up to eighth-order are the bounds precisely known. In fact, ERK methods of arbitrarily order high can be constructed with $s = \frac{p^2}{4} + 1$ stages where order p is an even number [72, pgs.232]. However, it can be seen that even for eighth order, this would lead to a 17-stage ERK method, which is much higher than the minimum number of stages for this order as well as much higher than the number of stages used for practical eighth-order ERK methods.

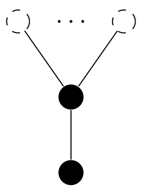
Table 2.1: Number of trees for each order, total number of order conditions for each order, and minimum number of stages $s_{\min}(q)$ required for ERK Methods of order q. OC = order conditions.

q	# trees	total # OC	$s_{\min}(q)$ [72, pg.179]
1	1	1	$q = 1$
2	1	2	$q = 2$
3	2	4	$q = 3$
4	4	8	$q = 4$
5	9	17	$q + 1 = 6$
6	20	37	$q + 1 = 7$
7	48	85	$q + 2 = 9$
8	115	200	$q + 3 = 11$
9	286	486	$12 \le s_{\min}(q) \le 17$
10	719	1205	$13 \le s_{\min}(q) \le 17$

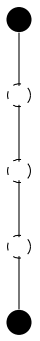
Like other expressions found from the elementary differentials (2.36), depending on the context in which a rooted tree is used, the rooted tree may be considered isomorphic to the order condition associated with an elementary differential $\mathbf{F}$ rather than the elementary differential $\mathbf{F}$ itself. Certain types of rooted trees and the corresponding order conditions have special names. The *quadrature conditions* [27, pg.175][72, pgs.175,208][73, pg.71] or *bushy trees* [168] refer to the trees of the form



because these are the only trees of order $\le q$ that must be satisfied for an RK method to solve quadrature problems (2.2) with order of convergence q. The *one-leg trees* [72, pg.175] are those of the form



Also important for this study are the *trees of height two*, which are any trees of height two, rather than height one like the bushy trees. Of particular importance to ERK method construction described in Chapter 3 are the *non-branching trees of height two*, which is terminology used exclusively in this thesis, that are those trees of height two that do not split into branches above the root. The *tall trees* are those of the form



with the expression for the corresponding elementary differential $\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))$ (2.36) only involving $\mathbf{f}(t, \mathbf{y}(t))$ and $\dfrac{d}{dt}\mathbf{f}(t, \mathbf{y}(t))$. Therefore, the tall trees (that also include the tree $\bullet$ (2.36a)) are the only non-zero components of derivatives of the RHS of (2.1a) when the RHS of (2.1a) is a linear function. In this study, it is sometimes convenient to simply call trees of height greater than two the *taller trees*, although this name is used in a way that is clear from context and not an exact definition.

Finally, the expressions for the elementary differentials (2.36) are just the components of an expression that gives for $\mathbf{y}(t)$ the order q *Fréchet derivative*, which is the derivative generalized to vector valued functions [108, pgs.158–161]. Furthermore, using the elementary differentials (2.36) the expressions for derivatives (2.37) and (2.41) are a special case of *Faà di Bruno's formula* (2.1a) [72, pgs.149–150], which is the chain rule generalized to higher derivatives and vector-valued functions.

2.5.2 Simplifying assumptions

Although construction of ERK methods is not discussed in detail until Section 2.7 and Chapter 3, it can be seen from Table 2.1 that when constructing higher-order RK methods, i.e., fifth-order or higher for ERK methods, that solution of the order conditions can become difficult. This is because with increasingly higher orders, ERK method construction involves simultaneously solving a rapidly increasing number of order conditions (multivariate polynomials) that are of increasingly high degree. Therefore, when constructing higher-order RK methods, additional expressions known as *simplifying assumptions* are almost always introduced that exploit patterns in the elementary differentials (2.36) and values of $\gamma(\mathbf{F})$ (2.41) to reduce the algebraic complexity of finding a solution to the order conditions [25][27, pgs.157–161][72, pgs.175–185]. It is shown below that the patterns the simplifying assumptions exploit can often be visualized using rooted trees.

At this point, it can be noted that one of the main contributions of this thesis is constructing higher-order ERK methods, specifically six-stage fifth-order ERK methods, using the complete algebraic formulation of the order conditions rather than the more limited algebraic formulation resulting from when the standard simplifying assumptions are used. The construction of ERK methods using the complete algebraic formulation of the order conditions is presented in Chapter 3. Before that, in Section 2.7 the construction of published six-stage fifth-order ERK methods using the standard simplifying assumptions is shown.

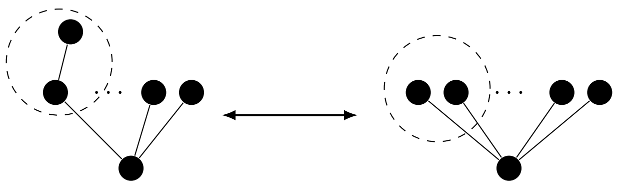
Although others are used, especially when constructing ERK methods higher than fifth order [72, pgs.181–185], the common simplifying assumptions important to this study are

$$B(p) : \sum_{i=1}^{s} b_i\, c_i^{q-1} = \frac{1}{q}, \qquad\qquad q = 1, 2, \ldots, p, \tag{2.44a}$$

$$C(\eta) : \sum_{j=1}^{s} a_{i,j}\, c_j^{q-1} = \frac{c_i^q}{q}, \qquad i = 1, 2, \ldots, s, \quad q = 1, 2, \ldots, \eta, \tag{2.44b}$$

$$D(\zeta) : \sum_{i=1}^{s} b_i\, c_i^{q-1}\, a_{i,j} = \frac{b_j}{q}\left(1 - c_j^q\right), \qquad j = 1, 2, \ldots, s, \quad q = 1, 2, \ldots, \zeta. \tag{2.44c}$$

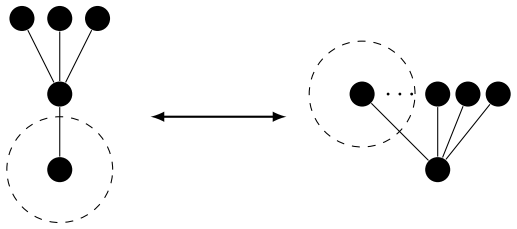
The expressions $B(p)$ (2.44a) are simply the quadrature conditions of order p that must be satisfied by any RK method of order p. However, $B(p)$ (2.44a) are commonly included as simplifying assumptions because when the appropriate instances of all three conditions given by (2.44) are taken together, they are often a major component of a *reduced system*, i.e., a set of equations that is a simplification of the original order conditions and is much easier to solve because the reduced system can have a much lower degree or even be linear in the unknowns [72, pgs.175–185,208–210][73, pgs.71–77]. The simplifying assumptions $C(\eta)$ (2.44b), which are sometimes called the *row simplifying assumptions* [94], allow all pairs of order conditions corresponding to trees with the forms



to be satisfied simultaneously when the pair of order conditions can be written in the form

$$\sum_{i,j} b_i\, a_{i,j}\, c_j \tilde{\Phi}_i = \frac{1}{\gamma_1}, \quad \sum_i b_i\, c_i^2\, \tilde{\Phi}_i = \frac{1}{\gamma_2}, \quad \gamma_1 = 2\,\gamma_2,$$

where the circled vertices are the only parts that are different between the two order conditions, the order condition corresponding to the particular scalar sum $\tilde{\Phi}_i$ is also satisfied, and $\tilde{\Phi}_i$ is an arbitrary subtree represented by the two non-circled vertices. If $C(\eta)$ (2.44b) is satisfied for a particular i this means that the quadrature conditions of order η are satisfied for stage i of the RK method. The value η from $C(\eta)$ (2.44b) is sometimes known as the *stage order*, of either a particular stage or of the method as a whole, if either all or a significant subset of stages satisfy the quadrature conditions up to η. Stage order is more important for IRK methods [73, pg.226] but also appears in the literature in the context of ERK methods [191, 193]. The simplifying assumptions $D(\zeta)$ (2.44c), which are sometimes called the *column simplifying assumptions* [94], allow all order conditions corresponding to pairs of one-leg trees and non-one-leg trees with the forms



to be satisfied simultaneously when the pair of order conditions can be written in the form

$$\sum_{i,j} b_i\, a_{i,j}\, \tilde{\Phi}_j = \frac{1}{\gamma_1}, \quad \sum_j b_j\, c_j\, \tilde{\Phi}_j = \frac{1}{\gamma_2}, \quad \gamma_1 = 4\,\gamma_2,$$

where the circled vertices are the only parts that are different between the two order conditions, the order condition corresponding to the particular scalar sum $\tilde{\Phi}_j$ is also satisfied, and $\tilde{\Phi}_j$ is an arbitrary subtree represented by the three non-circled vertices [27, pgs.157–161][72, pgs.175–185]. Specific examples of applying the standard simplifying assumptions to ERK method construction are given in Section 2.7. With an appropriate procedure, the standard simplifying assumptions are also used to help satisfy ERK order conditions where the groups of vertices analogous to the circled ones above do not attach to the root. This can be seen specifically in examples of ERK method construction that is shown in Sections 2.7.1 and 2.7.2.

2.5.3 Error of RK methods

Rigorous bounds on the local error (2.30) can be found for many numerical methods, including an expression for RK methods of order p given by

$$||\tilde{\mathbf{y}}(t_{n+1}) - \mathbf{y}_{n+1}|| \leq \Delta t^{p+1} \left(\frac{1}{(p+1)!} \max_{\zeta \in [0,1]} ||\mathbf{y}^{(p+1)}(t_n + \zeta \Delta t)|| + \frac{1}{p!} \sum_{i=1}^{s} |b_i| \max_{\zeta \in [0,1]} ||\mathbf{k}_i^{(p)}(\zeta \Delta t)|| \right), \qquad (2.45)$$

that holds when the IVP is Lipschitz continuous (2.6) and thus well-posed, all partial derivatives up to order p exist and are continuous, and $\zeta \in [0,1]$ indicates that the solution is bounded by the derivatives of the solution and stages within the step itself [72, pg.157]. Using the definitions of convergence (2.31) and consistency (2.32) from Section 2.4.4 with (2.45) it can be shown that any RK method that is at least first order is consistent and converges [108, pg.152]. In fact, the necessary and sufficient condition for an RK method to be consistent (which immediately implies convergence for RK methods) is simply the order condition (2.43) for first order [108, pg.152], i.e., (B.1a) for six-stage fifth-order ERK methods. With these properties known, RK methods of orders greater than or equal to one can now be studied freely throughout the rest of the thesis without additional concern as to whether the constructed methods converge and are consistent. Although expressions derived from (2.45) are often used to estimate local error in lower-order methods, the bound (2.45) quickly becomes too imprecise to use for higher-order methods [72, pg.158].

From (2.45) it is possible to estimate a bound on global error for an RK method [72, pg.160] of

$$||\mathbf{E}_{\text{global}}|| \leq \max(\Delta t)^p \frac{C}{L} \left(\exp(L(t_f - t_0) - 1\right), \tag{2.46}$$

where C is an arbitrary constant and the first-order partial derivatives have the bounds required for the Lipschitz condition L (2.6).

Analytic error bounds such as (2.46) and (2.45) often become exponentially less restrictive with increasing $t_f - t_0$ or Δt, respectively. Therefore, analytic bounds are usually of limited use in practice, especially for higher-order methods and large stepsizes. An example showing the limited utility of analytic bounds for practical purposes can be seen in the paper by Higham and Trefethen [78], where Figure 2 in their paper shows that the upper and lower analytic bounds on the numerical solution of a relatively simple IVP differ by over 20 orders of magnitude after a relatively small $t_f - t_0$. Further examples of the limited practicality of many analytic bounds on the numerical solutions of IVP methods are also discussed by Lambert [108, pgs.59–62]. However, an important observation to guide the construction of RK methods is that the first term inside the parenthesis is dependent on the properties of the whole ERK method and the second term is dependent on the properties of the individual stages [72, pg.157]. This justifies examining the error properties of the individual stages in some cases. Hairer et al. give an example where the individual stages of an ERK method are too far away from the solution, sometimes adversely affecting the performance of that ERK method [72, pg.254].

A practical methodology to help study both local and global error is to examine the *error terms*, i.e., the terms of the Taylor series of the local error that is the difference between the Taylor series of the local solution (2.38) and the Taylor series of the numerical solution (2.42). The *leading error term*, i.e., the lowest order error term, is often dominant, and its properties are often used alone to study the local error and compare different numerical formulae [16][72, pg.158]. In fact, Lambert raised the issue of authors studying the leading error term as the "local error" [108, pg.63]. Performance data in Chapter 5 shows that studying just the magnitude of the leading error term in isolation is rarely adequate and that studying the components of the error terms corresponding to the individual elementary differentials as well as higher-order error terms is often necessary.

For many first- and second-order numerical methods, the value of the leading error term can be accurately estimated because the magnitude of the appropriate term of the Taylor series of the local error is easily found using divided differences to calculate the appropriate derivatives [148]. For higher-order RK methods, finding an accurate estimate of the leading error term requires evaluating each of the many elementary differentials (2.36) in the corresponding derivative (2.41) of $\mathbf{y}_n$. Using divided differences to estimate these higher derivatives is generally considered impractical [72, pg.158][138].

The coefficient corresponding to each elementary differential, but excluding any powers of Δt, in the Taylor expansion of the local error is known as the *principal error coefficient* (PEC) [189, 190] corresponding to that elementary differential (2.36). The PEC is of course zero when the order condition (2.43) corresponding to a particular elementary differential (2.36) is satisfied. The PEC, sometimes given as $A(\mathbf{F})$ when referring to a specific $\mathbf{F}$, corresponding to an elementary differential $\mathbf{F}$ (2.36) [47, 94, 136] is

$$\text{PEC} = A(\mathbf{F}) = \frac{1}{\sigma(\mathbf{F})}\left(\Phi(\mathbf{F}) - \frac{1}{\gamma(\mathbf{F})}\right), \tag{2.47}$$

where $\gamma(\mathbf{F})$ and $\sigma(\mathbf{F})$ correspond to the functions on rooted trees introduced with (2.41) [94] and $\Phi(\mathbf{F})$ corresponds to the appropriate scalar sum (2.39). A definition [72, pg.158] normalizing the PEC corresponding to an elementary differential $\mathbf{F}$ (2.36) is

$$\overline{\text{PEC}} = \bar{A}(\mathbf{F}) = 1 - \gamma(\mathbf{F})\,\Phi(\mathbf{F}), \tag{2.48}$$

which is different in value from (2.47) because a factor of $\frac{\alpha(\mathbf{F})}{q!}$ has been removed, and originates from Hairer et al. [72, pg.158] but is not otherwise widely used in the literature. However, this study finds that the $\overline{\text{PEC}}$ definition (2.48) is extremely useful because it can easily be seen that $\bar{A}(\mathbf{F}) = 1$ implies that the component of the error corresponding to $\mathbf{F}$ is the same as if the Taylor series of the numerical method (2.42) were a truncation of the Taylor series of the local solution (2.38). If $\bar{A}(\mathbf{F}) = 0$, this implies that the component corresponding to $\mathbf{F}$ is identical in the Taylor series of the local solution (2.38) and the Taylor series of the numerical method (2.42), meaning that the error component corresponding to $\mathbf{F}$ has a PEC of zero. If $\bar{A}(\mathbf{F}) = -1$ then the error of the component corresponding to $\mathbf{F}$ is equal but opposite in magnitude of the truncation just described for $\bar{A}(\mathbf{F}) = 1$. If $|\bar{A}(\mathbf{F})| > 1$, this implies that the magnitude of the error from the PEC corresponding to $\mathbf{F}$ is greater than if the Taylor series of the numerical method (2.42) were a simple truncation of the Taylor series of the exact solution (2.38). The advantages of using the $\overline{\text{PEC}}$ definition (2.48) is further demonstrated in Chapter 5.

Without IVP- or solution-specific information, it is generally assumed that no elementary differential (2.36) or corresponding PEC can easily be argued to be more important than any other [15, 16][27, pg.151]. On the contrary, in Chapter 5 it can now shown that some PECs are more important for certain classes of IVPs. However, under the assumption that all PECs are of equal importance, a common measure of the magnitude of the error term of order q that has proven useful in practice is the root-mean square (RMS) or 2-norm of all of the individual PECs (2.47) of order q. This measure is generally known as the *error coefficient* of order q, with the *leading error coefficient* corresponding to the error coefficient of the leading error term [16, 47, 94]. Finally, because of the asymptotic behaviour of the global error with an increasing number of (smaller) steps, which is discussed in Section 2.4.3, an order p method satisfies the order condition up to order p and has a leading error coefficient that has an order of convergence for individual steps of $(p + 1)$, but a global order of convergence of p.

2.5.4 Stiffness and the stability of RK methods

Even when using stepsizes that are small enough for a desired accuracy for some IVPs, many classes of IVP methods, including explicit and therefore ERK methods, exhibit catastrophic instability, i.e., the numerical solution exponentially accumulates error even when the magnitude of the local errors are initially considered acceptable. When an IVP method is constrained by stability to use a stepsize that is excessively small relative to the accuracy required, it is said that the problem is *stiff* in the interval where this occurs. Stiffness has historically been seen as a phenomenon without a rigorous mathematical definition because it is a transient phenomena and cannot be characterized by limits such as $t \to \infty$ and $t \to t_0$ [78]. However, Söderlind has now found a rigorous mathematical definition of stiffness, which can be further examined in the references [165]. A good working definition of stiffness is that an implicit method designed for stiff IVPs, i.e., a *stiff method* that requires the relatively expensive solution of implicit systems of equations, outperforms IVP methods designed to be most efficient on non-stiff problems, i.e., *non-stiff methods* that are often but not necessarily explicit methods [108, pg.220].

Even for IVPs derived from relatively small systems of ODEs, the degree of stiffness can be so severe that an IVP that is readily solved in under a second with commonly used implicit IVP solvers can take over a week when using virtually any explicit IVP solver [27, 73, 108]. An example of a small IVP system from a practical application with stiffness this severe is a model of atmospheric pollution described by Hundsdorfer and Verwer [86, pg.6], a system of four ODEs that is not described in detail in this thesis because the stiffness precludes reasonably using it to test the performance of the new ERK methods constructed herein. A more reasonable demonstration of stiffness is the van der Pol IVP (2.14), where $\epsilon \ll 1$ results in an extremely stiff problem, as already demonstrated in Section 2.3.2 using $\epsilon = 10^{-3}$. As a more extreme example of stiffness, when solving the van der Pol IVP (2.14) with $\epsilon = 10^{-6}$ the best explicit methods described in this thesis take just over a million steps, whereas solving the same problem for a similar accuracy typically takes well under a thousand steps with most of the numerous implicit methods designed for stiff problems that are described by Mazzia and Magherini [115]. The severe stepsize restriction and large numbers of steps required by explicit methods when solving stiff IVPs are not enough to offset the cost of solving the implicit systems of nonlinear equations for the relatively small number of steps required by an implicit method when solving a stiff IVP.

Linear stability theory, i.e., the analysis of a step of the numerical method (2.27) using the test equation (2.10), has been found effective to characterize stiffness for many IVPs. The linearization and decoupling of the components of an IVP using the frozen Jacobian is already described in Section 2.1.4, and applying this procedure to the test equation (2.10) allows the study of the stability of a numerical method. It can easily be seen from the analytic solution to the test equation (2.10) (as well as in the examples of solutions to (2.10) in Figures 2.1 and 2.2) that if $\Re(\lambda) \leq 0$, then the exact solution of the test equation remains bounded as $t \to +\infty$. Therefore, it is desirable that any numerical method applied to (2.10) also produces a bounded solution when $\Re(\lambda) \le 0$. Applying an RK method to the test equation (2.10) yields

$$u_{n+1} = u_n + R(\lambda\,\Delta t_n), \tag{2.49}$$

where $R(\lambda\,\Delta t_n)$ is the *stability function* of a particular RK method (2.33) and corresponds to one step of that RK method applied to the test equation (2.10) with an initial condition $u_0 = 1$. It is easily seen that this numerical solution of the test equation (2.10) over one step (2.49) remains bounded as $t \to +\infty$ if and only if $R(\lambda\,\Delta t_n) \le 1$. The values of $\lambda\,\Delta t_n$, $\lambda \in \mathbb{C}$ where a numerical method remains bounded when solving (2.10) define the *region of absolute stability* in the complex plane that is also often called just the *stability region*. The case where $\mathbf{J_f}$ is not diagonalizable is not required for this thesis but can be studied as a similar but more sophisticated analysis of stiffness by Higham and Trefethen [78]. The stability function $R(z)$ of a specific RK method (2.33)[73, pg.41][94] with $z = \lambda\,\Delta t$ can be given using the coefficients from Butcher tableau (2.34) by

$$R(z) = \frac{\det(\mathbf{I} - z\mathbf{A} + z\mathbb{1}\mathbf{b})}{\det(\mathbf{I} - z\mathbf{A})} = \frac{P(z)}{Q(z)}. \tag{2.50}$$

For ERK methods the function $R(z)$ is a polynomial because $Q(z) = 1$ when $\mathbf{A}$ is strictly lower triangular and it can be shown that this gives a stability region that is always bounded [108, pg.200]. A bounded stability region means a large enough $|z|$ always leads to the numerical solution (2.49) of the test equation (2.10) becoming unbounded even when $\mathbb{R}(z) \le 0$ and the exact solution to the test equation (2.10) is bounded.

The stability functions for FE (2.25) and RK4 (2.26) are $1 + z$ and $1 + z + \frac{1}{2}z^2 + \frac{1}{6}z^3 + \frac{1}{24}z^4$, respectively. These can clearly be seen to be the Taylor series of the function e^z truncated to the appropriate order and representing the numerical solution to the test equation (2.10) from one step of the appropriate method. The regions of absolute stability for FE (2.25) and RK4 (2.26) are shown in Figures 2.13a and 2.13b, respectively. An important quantity to quickly characterize a stability region is its length along the negative real axis. This is because many stiff IVPs have eigenvalues that are relatively close to the negative real axis and some important IVP methods have their unbounded region only along the negative real axis. Another important quantity is the distance along the imaginary axis that the stability region extends. It can clearly be seen in Figure 2.13a that the stability region of FE (2.25) only touches the imaginary axis at the origin, whereas the stability region of RK4 (2.26) extends a significant distance along the imaginary axis. The stability along the imaginary axis for RK4 (2.26) is advantageous for many types of problems, such as oscillatory ODEs and semi-discretized hyperbolic PDEs, where it helps control *dispersion* and *dissipation* [82]. The imaginary stability is another benefit of RK4 (2.26) over FE (2.25), in addition to being higher order.

Other stability regions for specific ERK methods that will be introduced in Section 2.7 can be examined in Figures 2.14, 2.15, and 2.16. An interesting observation is that the size and shape of the stability region of RK methods depends only on the value of the elementary differentials (2.36) corresponding to the tall trees [73, pg.16]. This can easily be seen because the linearized RHS is always a linear constant-coefficient expression with its derivatives greater than the first always zero. Therefore, the expressions corresponding to the tall trees are the only ones that do not have derivatives greater than the first in the product corresponding to the elementary differentials (2.36).

IRK methods (and many other implicit methods) can have unbounded stability regions, i.e., with $\lim_{z\to\infty} R(z) = 0$ in some direction in the left half of the complex plane. Often the stability polynomials of IRK methods are the rational Padé approximations of e^z and can be further examined in the references [73, pgs.48–49][108, pgs.232–233]. Although not shown in this thesis, practical implicit methods often have just an arc around the negative real axis corresponding to the unbounded stability region and some have stability regions that include the whole left-hand side of the complex plane. Other IVP methods have stability regions that include parts of the right complex plane where the numerical solution remains bounded even though the solution to the test equation (2.10) does not. This adds dampening to the numerical solution that can sometimes be beneficial and sometimes not [108, pgs.229–231].

Bounded stability regions constrain many numerical methods to a small stepsize when solving stiff problems. If this relatively small step size occurs because the solution is not smooth or is rapidly changing, this small stepsize may be expected due to accuracy considerations, then the problem is not considered stiff in that interval. Stiffness can often be confusing for practitioners when solution components such as $e^{-1000\,t}$ are present that decay quickly and contribute little to the solution over timescales of $t \approx 1$, but can be the source of catastrophic instability and intractable restrictions on stepsize when using non-stiff methods [171, pg.489].

Discussions of stiffness by non-experts have often assumed $\lambda \gg \Delta t$ or large Lipschitz coefficients L (2.6), giving an imprecisely defined *stiff region*. However, several authors [78][108, pgs.213–224][165] have shown through extensive analysis and examples that guidelines based on eigenvalues and Lipschitz coefficients cannot be used to definitively define or diagnose stiffness. Due to the lack of a precise mathematical definition, there are always instances where some IVPs meeting particular guidelines for stiffness do not exhibit stiffness in the interval under study. Due to all of these issues and as mentioned before, the best working definition of stiffness is that an IVP is stiff in an interval if it can be solved most efficiently in that interval with an implicit method designed for stiff problems, i.e., one with an unbounded stability region [108, pg.220].

Despite the well-established utility, there are limitations to applying classic linear stability theory too precisely. This is because either or both of the steps involved in linearizing the ODE and then freezing these coefficients may not give a linear constant-coefficient ODE that reflects the behaviour of the original non-linear ODE [108, pgs.261–264]. In particular, Higham and Trefethen warn that non-diagonalizable matrices $\mathbf{\Lambda}$ may cause issues with standard linear stability theory and require additional theory beyond the scope of this thesis to resolve [78]. However, despite these limitations, linear stability theory has proven remarkably useful in diagnosing many stability issues with solving IVPs in order to design numerical methods and solvers that address these stability issues.

However, despite implicit methods being more efficient for stiff IVPs, it is important to study stiffness when constructing explicit methods such as ERK methods because they are often used when the restriction on Δt is only moderate, a condition known as *mild stiffness*. Robust ERK-based IVP solvers should be able to gracefully handle mild stiffness if it occurs unexpectedly, or if the practitioner accepts losing some computational efficiency by using a solver based on simpler-to-implement explicit methods as opposed to the much more complex implementations generally required of implicit methods. The IVPs from semi-discretized PDEs described in Section 2.2 represent mildly stiff IVPs that are commonly solved with ERK methods. In general, semi-discretized hyperbolic PDEs tend to be less stiff than semi-discretized parabolic PDEs.

2.6 Error estimation and variable-stepsize ERK methods

Many RK solvers in practice use variable stepsizes, and many IVPs actually require variable stepsizes to efficiently obtain a high-quality solution, e.g., the "Arenstorf orbit" IVP (2.15) as demonstrated by Figure 2.11, or similar issues that occur due to quasi-collisions in the Pleiades IVP (2.16). Variable-stepsize solvers using RK methods directly control only the local error (2.30) by changing the stepsize to take advantage of the asymptotic properties of a convergent numerical method. Therefore, although variable stepsize solvers do not directly control the global error (2.29) because the bounds on global error (2.29) also diminish asymptotically with decreasing bounds on the local error (2.30) due to (2.46), reducing the stepsize with the intention of reducing the local error (2.30) generally has the effect of reducing the global error (2.29) too [77, 164].

To provide an efficient error estimate for RK methods, an *embedded pair* is constructed by incorporating a second RK method (2.33) of generally lower order. This second RK method can usually be evaluated cheaply with no extra RHS evaluations and just a single additional matrix-vector multiplication because this second method uses the same $\mathbf{A}$ matrix and $\mathbf{c}$ vector and only differs in using a different $\mathbf{b}$ vector, i.e., $\hat{\mathbf{b}}$. The Butcher tableau that describes an ERK embedded pair is given by

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \\ & \hat{\mathbf{b}}^T \end{array} = \begin{array}{c|cccc} c_1 & 0 & 0 & \dots & 0 \\ c_2 & a_{2,1} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s,1} & a_{s,2} & \dots & 0 \\ \hline & b_1 & b_2 & \dots & b_s \\ & \hat{b}_1 & \hat{b}_2 & \dots & \hat{b}_s \end{array} \tag{2.51}$$

and is an extension of the definition of a Butcher tableau (2.34) for an ERK method.

The *embedded error estimate* for solution $\mathbf{y}_{n+1}$ found by taking the difference between the two component methods of an embedded pair is given by

$$\grave{\mathbf{E}}_{n+1} = \mathbf{y}_{n+1} - \hat{\mathbf{y}}_{n+1}, \tag{2.52}$$

where $\hat{\mathbf{y}}_{n+1}$ is the solution from the RK method using the $\hat{\mathbf{b}}$ vector (2.51) and $\mathbf{y}_{n+1}$ is the solution from the RK method using the $\mathbf{b}$ vector (2.51). The expression

$$\grave{E}_{n+1} = ||\hat{\mathbf{y}}_{n+1} - \mathbf{y}_{n+1}||, \tag{2.53}$$

is used when a scalar number representing the magnitude of the embedded error estimate is required. In this thesis, the use of a superscript circumflex "ˆ" denotes variables used to correspond to properties or solutions from the lower-order component of an embedded pair, e.g., $\hat{\mathbf{y}}$ [47, 136, 187]. Sometimes an error estimate $\grave{\mathbf{E}}$ might end up being an extremely poor estimate of the local error due to unknown properties of the ODE (2.1a). However, it can still form a good heuristic for step control [72, pg.168]. A value for the "error estimate", which may not be even remotely similar to the actual local error, is indicated by the grave accent "ˋ" in this thesis. The embedded error estimate from an embedded pair of orders $(p, p-1)$ is an $\mathcal{O}(\Delta t^{p+1})$ estimate of the local error (2.30) of the RK method of order $p-1$ for the embedded pair. The utility of $\grave{\mathbf{E}}_{n+1}$ and the related values described above for step control, even when inaccurate or inappropriate, is demonstrated by the popularity of *local extrapolation*, where the higher-order method (with the presumably more accurate solution) of an ERK embedded pair is used to advance the integration. Local extrapolation often works in practice even though $\grave{\mathbf{E}}_{n+1}$ only makes mathematical sense for estimating the error of the lower-order method of an ERK pair [72, pg.168].

In some cases, the constituent methods of an embedded pair can have different numbers of stages; this can easily be allowed by setting some components of the appropriate $\mathbf{b}$ and $\hat{\mathbf{b}}$ vectors to zero. An extension of embedded pairs consisting of three distinct ERK methods has been constructed by Bogacki and Shampine [16] in order to provide both a cheaper preliminary and more expensive refined error estimate. Methods consisting of three distinct ERK methods, i.e., eighth-, fifth-, and third-order ERK methods, that are all used together to provide a single high-quality error estimate have been constructed by Dormand and Prince and published by Hairer et al. in their book [72, pgs.254–255]. However, the construction of ERK methods with more than two components is not studied further in this thesis. In addition, although these extensions of embedded pairs might logically be called "embedded triples", there are also embedded pairs in conjunction with a high-order interpolant called "Runge–Kutta triples" [49]. Interpolants are not addressed further in this thesis either.

In order to guide a search for better ERK embedded pairs (the subject of Chapters 4 and 5), it is necessary to consider a number of error coefficients, not just the leading error coefficient of the method used to advance the integration. The error coefficient of order q is referred to using the notation

$$A^q = ||A(\mathbf{F}^q)||_2, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.54}$$

where $A(\mathbf{F}^q)$ is the PEC (2.47) for the elementary differential $\mathbf{F}$ (2.36) and $||\ \ ||_2$ corresponds to the standard 2-norm. Norms other than the standard 2-norm have occasionally been discussed for the error coefficients, especially by Verner [189, 190], and they are further examined in Chapter 5. The notation used for these other norms of the error coefficients is

$$|A^q|_1 = ||A(\mathbf{F}^q)||_1, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.55a}$$

$$|A^q|_2 = ||A(\mathbf{F}^q)||_2, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.55b}$$

$$|A^q|_\infty = ||A(\mathbf{F}^q)||_\infty, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.55c}$$

$$|\bar{A}^q|_1 = ||\bar{A}(\mathbf{F}^q)||_1, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.55d}$$

$$|\bar{A}^q|_2 = ||\bar{A}(\mathbf{F}^q)||_2, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.55e}$$

$$|\bar{A}^q|_\infty = ||\bar{A}(\mathbf{F}^q)||_\infty, \quad \mathbf{F}^q \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t))), \tag{2.55f}$$

where $|| \quad ||_1$ corresponds to the standard 1-norm and $|| \quad ||_\infty$ corresponds to the standard ∞-norm.

For an RK method of order p, A^p is the leading error coefficient. In order to help evaluate the suitability of the error estimate for embedded ERK pairs, the *characteristic numbers* [189, 190], in addition to A^q, that are commonly used are defined as

$$\mathrm{B} = \hat{A}^{(p+2)}/\hat{A}^{(p+1)}, \tag{2.56a}$$

$$\mathrm{C} = ||\hat{\tau}^{(p+2)} - \tau^{(p+2)}||_2/\hat{A}^{(p+1)}, \tag{2.56b}$$

$$\mathrm{D} = \max(|a_{ij}|, |b_i|, |\hat{b}_i|, |c_i|) \quad \forall \quad i, j \in \{1, 2, \ldots, s\}, \tag{2.56c}$$

$$\mathrm{E} = A^{(p+2)}/\hat{A}^{(p+1)}, \tag{2.56d}$$

$$S_R = \min\left(\mathrm{Real}(z)\right) \quad s.t. \quad |R(z)| < 1, \tag{2.56e}$$

$$\hat{S}_R = \min\left(\mathrm{Real}(z)\right) \quad s.t. \quad |\hat{R}(z)| < 1, \tag{2.56f}$$

where the B characteristic number measures the dominance of the leading term of the error estimate, the C characteristic number measures the degree to which the second error term of the error estimate vanishes thus ensuring the leading error term of the lower-order method remains relatively dominant, limiting the D characteristic number ensures the magnitude of the Butcher tableau (2.34) coefficients do not become too large and lead to unnecessary roundoff errors, the E characteristic number measures the relative magnitude of the leading error terms of both the lower-order and higher-order components of the embedded pair, S_R measures the length of the stability region along the negative real axis for the higher-order method of an embedded pair, and $\hat{S}_R$ measures the length of the stability region along the negative real axis for the lower-order method of an embedded pair. An examination of published guidelines for selecting these characteristic numbers along with supporting performance data is given in Section 5.13. However, the classic guidelines are that B and C characteristic numbers should be made as small as possible so they approach 1.0 [94, 96, 189, 190]. The E characteristic number is less widely mentioned in the literature than the A, B, C, and D characteristic numbers, but the classic guidelines based on observations by Shampine [146] is that it should be less than 1.0 and closer to 0.5. The unfortunately similar notation of the characteristic numbers (2.56) and the simplifying assumptions (2.44) using capital letters is standard notation and generally follows other notable publications [47, 72, 94, 136, 190].

2.6.1 Step-doubling error estimation

Step-doubling error estimation estimates the local error based on *Richardson extrapolation* [72]. It can be used with any numerical method for which the order of convergence p is known. It requires three steps to be taken from the current time t_n, i.e., two regular steps of size Δt_n to give the solution $\mathbf{y}_{n+1}$ and one double step of size $2\,\Delta t_n$ to give the solution $\widehat{\mathbf{y}}_{n+1}$. An estimate of the local error is using step-doubling is

$$\mathbf{y}(t_n + 2\,\Delta t_n) - \mathbf{y}_{n+1} = \frac{\mathbf{y}_{n+1} - \widehat{\mathbf{y}}_{n+1}}{2^p - 1} + \mathcal{O}(\Delta t_n^{p+2}), \tag{2.57}$$

with an order of convergence of $p + 1$ [72]. Step-doubling error estimation is not studied in detail in this thesis. However, it is implemented in the `pythODE` software package described in Chapter 5. Further details on step-doubling can be found in excellent expositions by Shampine [145], Hairer et al [72, pgs.164–165], and Stoer and Bulirsch [171, pgs.448–451].

2.6.2 Stepsize selection and control

In order to select the next stepsize using an error estimate from an embedded RK pair (2.52), heuristics have traditionally been used. However, these traditional heuristics can be extended and put on a sound theoretical basis using *linear feedback control theory* [66, 67, 164]. For determining whether an error estimate is acceptable and for the step-controller to respond appropriately, a *tolerance* value is needed to compare with the error estimate. However, a single fixed user-specified tolerance value rarely leads to robust step control even for simple IVPs. Therefore, an *absolute-relative norm* [66] is generally required. A common formula giving an absolute-relative norm for finding an acceptable tolerance value for the local error estimate is

$$T_{n+1,i} = T_{\text{rel},i} \max(y_{n,i},\, y_{n+1,i}) + T_{\text{abs},i}, \tag{2.58}$$

where $T_{\text{rel},i}$ and $T_{\text{abs},i}$ are respectively the user specified *relative tolerance* and *absolute tolerance* for component i of the IVP (2.1) solution $\mathbf{y}$ [72, pg.167]. However, somewhat different formulae for absolute-relative norms than (2.58) are also in use [66, 67][150, pg.234]. The relative tolerance is the primary means for the user to indirectly control the global accuracy of the solution, and the choice sometimes needs to take into consideration the sensitivity of the IVP to perturbations. An example of a sensitive IVP is the Pleiades IVP (2.16) where the quasi-collisions shown in Figure 2.8 may require tolerances at certain intermediate $t_0 \le t \le t_f$ that are much tighter than the tolerances that would be required for the same desired accuracy but for a slightly perturbed Pleiades IVP (2.16) without quasi-collisions. The relative tolerance should be less than 1.0 to be a viable way to control global error. Otherwise, the acceptable local error estimates would be greater than the solution value itself. The author's experience has shown that relative tolerances should be less than 0.1 for the numerical solution to start reflecting the true solution for at least some IVPs, and relative tolerances in an approximate range of 0.01 to 0.0001 are typical for finding a solution of "coarse" accuracy. If an exceedingly high accuracy is required, experience has shown that relative tolerances as low as 10^{-14} can be effective, although this value is probably too close to the unit roundoff for 64 bit floating-point numbers ($\approx 1.11 \times 10^{-16}$) when solving at least some IVPs. Therefore, relative tolerances in the range 10^{-8} to 10^{-12} are typical when an extremely accurate solution is needed without regard to computational cost. The absolute tolerance used for the absolute-relative norm ensures that the step controller does not make the stepsizes too small if $\max(y_{n,i}, y_{n+1,i})$ in (2.58) begins to vanish. The absolute tolerances are often smaller than the corresponding relative tolerances because they often act mostly as a safety measure when solution components vanish, and the relative tolerance should dominate the tolerance formula (2.58) for most steps of typical IVP solutions. However, when some IVPs are scaled to certain physical units, for example, if any of the examples of celestial mechanics problems in Section 2.3 were rescaled to meters or centimetres, some values of components that are $\gg 1.0$ would still be considered vanishing, and absolute tolerances could be required that are well above 1.0 even when solving to higher accuracies.

In many cases, The absolute and relative tolerances are uniform values across the components of an IVP, unless the user has a specific reason to use different tolerance values for different solution components. Examples of where different tolerances may be required for different solution components are IVPs based on chemistry problems, such as the stiff atmospheric pollution problem presented by Hundsdorfer and Verwer [86, pg.6], where different chemical components have widely different ranges.

Gustafsson [66, 67] was probably the first [73, pg.28] to formally use classic control theory to study stepsize control for numerical methods. A good general reference on linear control theory used in the course of this study is the book by Haidekker [68]. A good review of the application of control theory to numerical methods for IVPs is the paper by Söderlind [164]. Analogous to the convergence, consistency, and stability properties already discussed for the numerical integration method itself, there are several similar conditions that are necessary to ensure a variable stepsize IVP solver has the possibility of giving a viable solution [164]. Using T to represent a generic value for tolerance, these necessary conditions are:

- *Tolerance convergence*: $\lim_{\mathrm{T}\to 0} ||\mathbf{y}_n - \mathbf{y}(t_n)|| = 0$.
- *Computational stability*: A small change in tolerance produces only a small change in results.
- *Tolerance proportionality*: $c \cdot \mathrm{T} \leq ||\mathbf{y}_n - \mathbf{y}(t_n)|| \leq C \cdot \mathrm{T}$, hopefully with $\frac{C}{c} \lessapprox 10$.

Although these are conditions on the global error, they can in fact be satisfied by controlling a suitable local error estimates, because it can be shown if local error per unit time is kept below a tolerance, then global error can be bounded by a constant multiple of that tolerance [164]. Harder to quantify are the general assumptions that the norm used for the error estimate is smooth and reflects the actual behaviour of the local error [164].

To achieve the just-mentioned goals, using $\grave{\psi}_n$ as a function that allows examining the behaviour of the error estimate independent of changes in Δt_n, there are two assumptions made to model the integration process [164]:

- *Constant asymptotic behaviour of the error estimate*: $\grave{\psi}_n \Delta t_n^k = \grave{E}_{n+1}$.
- *Slow variation of the error estimate*: $||\grave{\psi}_{n+1}|| \approx ||\grave{\psi}_n||$.

Based on the assumptions used to model the integration process, the classic and still commonly used formula [72, pg.168][108, pg.146][152, pg.47][171, pg.448] to estimate the optimal stepsize is

$$\Delta t_{\text{opt}} = \Delta t_n \left(\frac{T_{n+1}}{\grave{E}_{n+1}} \right)^{(1/k)}, \tag{2.59}$$

where k is generally one greater than the minimum order of the two component methods in the ERK embedded pair, for example, if an embedded pair has methods of fourth- and fifth-order then $k = 5$ is used. If $\left(\frac{\grave{E}_{n+1}}{T_{n+1}}\right) \geq 1.0$ then the step is rejected and Δt_{opt} is used to retry the current step; otherwise the step is accepted and Δt_{opt} is used for the next step. If the error estimate were exact and the two assumptions modelling the integration process held exactly, then the standard step control formula (2.59) perfectly adjusts the next stepsize so $\left(\frac{\grave{E}_{n+2}}{T_{n+2}}\right)$ would be 1.0 when the error estimate is computed for this next step, without either excessive local error with $\left(\frac{\grave{E}_{n+2}}{T_{n+2}}\right) > 1.0$ or excessive cost with $\left(\frac{\grave{E}_{n+2}}{T_{n+2}}\right) < 1.0$.

However, the assumptions modelling the integration process only hold roughly for the solution of IVPs, i.e., the norm used for the error estimate may not be smooth, and the local dynamics of the IVP may not always match the ideal process assumptions. Especially outside of the asymptotic region, the differential system can have unanticipated behaviour [72, pg.168]. Consider a step where local conditions mean that the error is unexpectedly large or small. Therefore, directly applying any formula, such as (2.59), based on the expected asymptotic behaviour could cause extreme step changes that would require multiple rejected steps to reduce the stepsize back to a more appropriate value or extra undersized steps to increase the stepsize back to the ideal value. At worst, extreme step changes could unexpectedly cause a loss in the stability of the step controller and a completely incorrect solution. A safer and less aggressive approach to using only (2.59) is to use *limiters* and *safety factors* at the possible cost of a small number of extra steps due to slower changes when extreme step changes would actually be most efficient [72, pg.168]. Although the specific heuristics used as limiters and their implementation details can vary between software packages, an example the heuristics used by default in `pythODE` to find a new stepsize from the optimal stepsize (2.59) is

$$\Delta t_{\text{new}} = \alpha \min(a_{\max}\Delta t_n, \max(\Delta t_{\text{opt}}, a_{\min}\Delta t_n)), \tag{2.60}$$

where $a_{\max}$ and $a_{\min}$ are values representing the maximum amounts by which the stepsize can increase and decrease, respectively, and $\alpha \leq 1.0$ is a safety factor. Typical values are $\alpha = 0.9$, $a_{\min} = 0.2$, and $a_{\max} = 5.0$ for an accepted step. The value of $a_{\max} = 1.0$ is often used after a rejected step [72, pg.167] in order to avoid a stepsize increase after a step is rejected and to ensure repeated rejections lead to a decreasing stepsize sequence (the stepsize for a retry after a rejected step will never be larger than $\alpha\Delta t_n$). Using heuristics such as (2.60) ensures that extreme step changes due to unanticipated behaviour are limited so that a variable stepsize IVP solver uses as few extra steps as possible while avoiding loss of stability.

Although (2.59) was originally a heuristic that had been validated with practical experience, it can be seen as one form of an *integrating controller* (or I controller) [164]. An integrating controller is known in the context of numerical methods as an *integrating step controller* or *I step control* [73, pg.28][164]. In order to study the behaviour of a step controller (2.59), it is convenient to restate (2.59) by using a fixed tolerance T and taking logarithms in order to give the linear difference equation

$$\log(\Delta t_{n+1}) = \log(\Delta t_n) + \frac{1}{k}(\log(T) - \log(\grave{r}_{n+1})), \tag{2.61}$$

where $\log(T) - \log(\grave{r}_{n+1})$ is the *control error*, $\log(T)$ is the *setpoint*, and $\log(\Delta t)$ is the *control* (the quantity controlled) itself. *Discrete linear feedback control* for RK methods works because the *controller* attempts to set the control error (corresponding to the difference between the tolerance and error estimate, with the setpoint being where the control error is zero) for a *process* (corresponding to the integration itself (2.27)) equal to zero by changing a control (corresponding to the stepsize). The difference equation (2.61) can be solved to give

$$\log(\Delta t_n) = \log(\Delta t_0) + \frac{1}{k}\sum_{n=1}^{n}(\log(T) - \log(\grave{r}_n)), \tag{2.62}$$

where the origin of the name "integrating controller" becomes apparent because the summation corresponds to an integral that would appear in an analogous continuous controller. However, continuous controllers are not relevant to this study and not discussed further. What can also be observed is that the summation in the gain of an integrating controller (2.62) is an attempt to keep the total sum of all past control errors as zero. Linear feedback step controllers are known as "linear" because scaling the control errors (or summing control errors in more complex controllers) causes a proportional change in the control itself (the proportional change is actually with respect to the control given by $\log(\Delta t)$ on the LHS of (2.62)) [68, 164].

Solving the difference equation of a more general version [164] of an integrating step controller using the same logarithmic form as (2.61) gives

$$\log(\Delta t_n) = (1 - k\,k_I)^n \log(\Delta t_0) + k_I \sum_{m=1}^{n}(1 - k\,k_I)^{(n-m)}(\log(T) - \log(\grave{\psi}_{m-1})), \tag{2.63}$$

where k_I is known as the *integral gain* of the integrating controller and k describes the behaviour of the asymptotic process model as before. It can clearly be seen that the classic step controller (2.62) is actually a version of (2.63) with an integral gain k_I of $\frac{1}{k}$. The classic step controller (2.62) with $k\,k_I = 1.0$ (2.63) is also known as a *deadbeat controller*, i.e., it attempts to completely eliminate the control error for the next step [164]. A gain higher than that of a deadbeat control results in *fast oscillatory control* where if $k\,k_I > 2.0$ the controller is unstable. If $k\,k_I < 1.0$, then the lower gain results in *slow smooth control* [164]. Therefore, rather than (2.59) being the only possible way of selecting the next step given a particular asymptotic behaviour, there is a tradeoff between response time of the controller and stability [164]. The strategy described for (2.59) is sometimes known as *error per step* (EPS). Another classic although less commonly used stepsize strategy, is limiting *error per unit step* (EPUS) with k one lower than is described for (2.59). For example, if using EPUS with an embedded pair consisting of fourth- and fifth-order methods, then (2.59) would have $k = 4$ rather than $k = 5$. However, rather than a specific choice, EPUS can now be seen as one specific instance of slow smooth control [164].

It is important for the assumption on the specific asymptotic behaviour to be at least approximately correct. For instance, the ERK pairs primarily studied later in this thesis have a fifth-order error estimate, i.e., the expected asymptotic behaviour of the process is $k = 5$. However, if the solver is in a region of an ODE where the error estimate is converging at a rate greater than tenth-order, the step controller would actually be unstable. If a fast response is desired, the potential for instability can be worse if the IVP method does not exhibit the correct asymptotic behaviour. Therefore, although it is not possible to eliminate instability entirely, it is desirable to design numerical methods as carefully as possible to increase the chances of correct asymptotic behaviour. More advanced linear controllers that extend the ideas behind I controllers are also available. Advanced linear controllers commonly seen in IVP software are known as *proportional-integral* (PI) and *proportional-integral-derivative* (PID), and may provide some advantages for difficult IVPs but are not examined in this study, but can be studied further in the references [68, 164].

In implementations of embedded RK pairs where local extrapolation is used, the actual value of the error estimate no longer has the pretense of resembling the local error of the higher-order component of the embedded pair that is used to advance the solution. However, the error estimate from an embedded pair used in local extrapolation mode remains a useful quantity for step-size selection because it is expected that the higher-order method provides a more accurate solution than the lower-order method, or at least locally it can be bounded by some constant multiple of the error of the lower-order method. Practical experience has shown that well-designed embedded pairs using local extrapolation can give superior performance, although several examples are given in Section 5.6 that show that local extrapolation may not always be superior.

2.7 Prologue: Classic six-stage fifth-order ERK pairs using standard simplifying assumptions

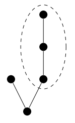
The families of ERK embedded pairs that are studied in depth consist of a six-stage fifth-order ERK method in combination with either a six-stage fourth-order ERK method or a seven-stage fourth-order ERK method, with the resulting embedded pair referred to with the shorthand *5(4)*$_6$ *ERK pair* regardless of whether the fourth-order method has six stages or seven stages. These variations are studied together because most of the complexity of constructing a $5(4)_6$ ERK pair comes from the six-stage fifth-order ERK method, and it is clearly seen in Chapter 3 that there is only a small difference between adding either a six- or seven-stage fourth-order ERK method to form a $5(4)_6$ ERK pair. When referring specifically to a six-stage fifth-order method that forms a pair with a six-stage fourth-order method, the shorthand is *5(4)*$_{6(6)}$ *ERK pair*. When referring specifically to a six-stage fifth-order method that forms a pair with a seven-stage fourth-order method the shorthand is *5(4)*$_{6(7)}$ *ERK pair*.

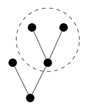
The methodologies for finding solutions to the systems of order conditions (2.43) are extensive; for instance, see the numerous examples in the references [25, 26, 155, 161, 187, 188, 189, 190, 191, 194]. For higher-order methods, the order conditions are rarely solved directly from the form given by (2.43). Often the simplifying assumptions (2.44) (or other possible algebraic simplifications) are used to find a more restrictive but more easily solved algebraic system, i.e., a reduced system. In most cases, the number of unknowns is greater than the number of equations, and the algebraic system or order conditions can be solved explicitly in terms of a chosen set of the unknowns that are called *free parameters*. Once the order conditions are solved as expressions explicitly for the other unknowns in terms of the free parameters, then a corresponding Butcher tableau (2.34) can quickly be found for any set of specific values of the free parameters. For each reduced system that is solved, one or more specific Butcher tableaux (2.51) are usually published in addition to the details and expressions defining a more general *family* of RK methods. When referencing a published family by naming it after a specific method, unless stated otherwise, the name used in this thesis is the family in the original publication where the specific method was originally published. For instance, the "RKF4(5)$_{6(6)}$ family" (see Section 2.7.2 for more details and construction of the RKF4(5)$_{6(6)}$ method (2.78)) refers to the family of $5(4)_{6(6)}$ ERK pairs originally published by Fehlberg in 1969 [61].

For the ERK method construction described in this section, the families are simple enough that they can readily be constructed and described in detail using hand computations (rather being forced to rely on a computer algebra system) from using the simplifying assumptions (2.44). However, in Chapter 3, the ERK construction methodologies presented that do not reduce the complexity of the algebraic system of the order conditions or require a reduction in the number of free parameters; these methodologies are made possible because of the sophisticated software tools that are presented in full detail in Chapter 4. It is seen in Section 3.3 that some classic methods given in this section belong to more general families that have more free parameters available because these general families do not require the C(3) simplifying assumptions (2.44b) that were originally used. However, the simplifying assumptions (2.44) do not necessarily decrease the number of free parameters available for an ERK method of a particular order with a particular number of stages. In general, it is best to think of the beneficial effect of simplifying assumptions as replacing more complex higher-degree order conditions with usually simpler equations that are often the simplifying assumptions themselves.

2.7.1 A six-stage fifth-order ERK method based on John Butcher's ideas

In the same paper showing the non-existence of fifth-order five-stage ERK methods [25], Butcher presented the construction of a seven-stage sixth-order ERK method. Hairer et al. [72, pg.175] used these ideas to show the construction of a six-stage fifth-order ERK method using the C(2) simplifying assumptions (2.44b) and D(1) simplifying assumptions (2.44c). A complication to using the C(2) simplifying assumptions (2.44b) is that it can easily be seen that the C(2) simplifying assumptions (2.44b), or in fact any C(η) simplifying assumptions (2.44b) with $\eta \geq 2$, cannot be satisfied for any ERK method when $i = 2$. This restriction on using the C(2) simplifying assumptions (2.44b) is because the LHS of (2.44b) is always zero when $i = 2$ (because the only terms of the summation in (2.44b) with any non-zero values in the case of the C(2) simplifying assumptions (2.44b) are $a_{2,1}\, c_1$ and $a_{2,2}\, c_2$ that both evaluate to zero for ERK methods) but that the RHS of (2.44b) must be non-zero (because $c_2 = 0$ would reduce the number of effective stages). This is remedied by the constraint $b_2 = 0$ and not requiring the C(2) simplifying assumptions (2.44b) to hold when $i = 2$. Having $b_2 = 0$ still allows standard reductions for the C(2) simplifying assumptions (2.44b) for subtrees attached to the root exactly as shown in Section 2.5.2. Taking into account this limitation to using the C(2) simplifying assumptions (2.44b), the only remaining order conditions left after applying the C(2) simplifying assumptions (2.44b) and the D(1) simplifying assumptions (2.44c) are the order conditions corresponding to the trees


(2.64a)


(2.64b)

that cannot immediately be reduced with the C(2) simplifying assumptions (2.44b). This is because $b_2 = 0$ only remedies the $i = 2$ issue with the C(2) simplifying assumptions (2.44b) for sub-trees attached to the root and not sub-trees above the root such as the ones circled. It is important to note that these circled subtrees for (2.64a) and (2.64b) can be equivalent when using the C(2) simplifying assumptions (2.44b) to construct IRK methods because the issue with $i = 2$ no longer applies. IRK method construction using the standard simplifying assumptions (2.44) can be studied further in the references [72, pg.208]. However, the limitation that prevents using the C(2) simplifying assumptions (2.44b) when $i = 2$ can be overcome to find an appropriate reduced system for ERK method construction. This is done by multiplying (2.64b) by $\frac{1}{2}$ and subtracting it from (2.64a) to give the expression

$$\sum_{i,j} b_i\, c_i\, a_{i,j}\, \left(a_{j,k} - c_j^2/2\right) = 0, \tag{2.65}$$

where the C(2) simplifying assumptions (2.44b) make the expression inside of the parenthesis vanish except when $j = 2$ in (2.65). When $j = 2$ for (2.65) the other factor in the summation in (2.65) must vanish to ensure the order conditions (2.64a) and (2.64b) are satisfied. Combined with the C(2) simplifying assumptions (2.44b), this reduces (2.65) to the condition

$$\sum_{i=3}^{6} b_i\, c_i\, a_{i,2} = 0, \tag{2.66}$$

that along with the C(2) simplifying assumptions (2.44b) and (2.64b) is now sufficient to satisfy (2.64a). Finally, to satisfy the order condition corresponding to the tree $[\tau[\tau^2]]$ (2.64b), subtracting the order conditions corresponding to the trees $[[\tau^2]]$ and c_3 times $[[[\tau]]]$ from $[\tau[\tau^2]]$ (2.64b) gives

$$\sum_{i,j} b_i(1-c_i)a_{i,j}c_j(c_j - c_3) = \frac{1}{60} - \frac{c_3}{24}, \tag{2.67}$$

which is an expression that has fewer terms and is more convenient to use than (2.64b) because $c_6 = 1$ means the terms involving $a_{6,i}$ to vanish.

The resulting reduced system consisting of the appropriate simplifying assumptions (2.44) along with (2.66) and (2.67) is now easily solved as a straightforward sequence of linear systems, despite the extreme non-linearities in the original order conditions. The quadrature conditions (2.44a) can be solved for the **b** vector by solving the linear system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & c_2 & c_3 & c_4 & c_5 & c_6 \\ 0 & c_2^2 & c_3^2 & c_4^2 & c_5^2 & c_6^2 \\ 0 & c_2^3 & c_3^3 & c_4^3 & c_5^3 & c_6^3 \\ 0 & c_2^4 & c_3^4 & c_4^4 & c_5^4 & c_6^4 \end{pmatrix} \begin{pmatrix} b_1 \\ 0 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \end{pmatrix},$$

for b_1, b_3, b_4, b_5, b_6 where it can be noted that $b_2 = 0$ due to the previously discussed restrictions on the C(2) simplifying assumptions (2.44b).

In each of the following steps, quantities already computed are substituted back into the conditions still to be solved, and terms involving only known quantities are moved to the RHS. Taking $a_{4,2}$ as an arbitrary parameter and using (2.44b) with $i = 3, 4$, $a_{3,2}$, $a_{4,3}$ can be found by solving the system

$$\begin{pmatrix} c_2 & 0 \\ 0 & c_3 \end{pmatrix} \begin{pmatrix} a_{3,2} \\ a_{4,3} \end{pmatrix} = \begin{pmatrix} \frac{c_3^2}{2} \\ \frac{c_4^2}{2} - a_{4,2}\, c_2 \end{pmatrix}.$$

Using (2.66) with $j = 2$, $a_{5,2}$, $a_{6,2}$ can be found by solving the system

$$\begin{pmatrix} b_5 & b_6 \\ b_5c_5 & b_6c_6 \end{pmatrix} \begin{pmatrix} a_{5,2} \\ a_{6,2} \end{pmatrix} = \begin{pmatrix} -\left(\sum_{i=3}^{4} b_i a_{i,2}\right) + b_2(1 - c_2) \\ -\sum_{i=3}^{4} b_i\, c_i\, a_{i,2} \end{pmatrix},$$

using the now-known $a_{i,j}$ quantities for the RHS. Using (2.67) and (2.44b) with $i = 5$ allows $a_{5,3}$, $a_{5,4}$ to be found by solving the system

$$\begin{pmatrix} 0 & b_5(1 - c_5)c_4(c_4 - c_3) \\ c_3 & c_4 \end{pmatrix} \begin{pmatrix} a_{5,3} \\ a_{5,4} \end{pmatrix} = \begin{pmatrix} -\sum_{i=3}^{5} b_i(1 - c_i)a_{i,2}\, c_2(c_2 - c_3) + \frac{1}{60} - \frac{c_3}{24} \\ -(a_{5,2}\, c_2) + \frac{c_5^2}{2} \end{pmatrix},$$

using the now-known $a_{i,j}$ quantities for the RHS and where it can now clearly be seen that the form of (2.67) is advantageous because all terms involving $a_{6,i}$ vanish. However, as mentioned above, directly using (2.64b) instead would have terms involving $a_{6,i}$ that would be more complicated to solve than the straightforward solution of a linear system. Using the D(1) simplifying assumptions (2.44c) with $j = 3, 4, 5$ allows $a_{6,3}$, $a_{6,4}$, $a_{6,5}$ to be found by solving the system

$$\begin{pmatrix} b_6 & 0 & 0 \\ 0 & b_6 & 0 \\ 0 & 0 & b_6 \end{pmatrix} \begin{pmatrix} a_{6,3} \\ a_{6,4} \\ a_{6,5} \end{pmatrix} = \begin{pmatrix} -\left(\sum_{i=1}^{5} b_i\, a_{i,3}\right) + b_3(1 - c_3) \\ -\left(\sum_{i=1}^{5} b_i\, a_{i,4}\right) + b_4(1 - c_4) \\ -\left(\sum_{i=1}^{5} b_i\, a_{i,5}\right) + b_5(1 - c_5) \end{pmatrix},$$

using the now-known $a_{i,j}$ quantities for the RHS. Finally, using the row-sum condition (2.35) allows $a_{2,1}$, $a_{3,1}$, $a_{4,1}$, $a_{5,1}$, $a_{6,1}$ to be found by solving the linear system

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{2,1} \\ a_{3,1} \\ a_{4,1} \\ a_{5,1} \\ a_{6,1} \end{pmatrix} = \begin{pmatrix} c_2 \\ -a_{3,2} + c_3 \\ -\sum_{i=2}^{3} a_{4,i} + c_4 \\ -\sum_{i=2}^{4} a_{5,i} + c_5 \\ -\sum_{i=2}^{6} a_{6,i} + c_6 \end{pmatrix}.$$

because all the $a_{i,j}$ quantities outside of the first column of the $\mathbf{A}$ matrix are now known.

2.7.2 The Runge–Kutta–Fehlberg $5(4)_{6(6)}$ ERK pair

The *Runge–Kutta–Fehlberg 4(5) method* (RKF4$(5)_{6(6)}$), which was first constructed by Fehlberg in 1969 [61], was one of the first widely used higher-order ERK embedded pairs. Although replaced in MATLAB by the Dormand–Prince method (see the next subsection) in the 1990s [153][1], RKF4$(5)_{6(6)}$ continues to be a widely used ERK method. The specific procedure given for constructing RKF4$(5)_{6(6)}$ in this subsection is based

[1]https://blogs.mathworks.com/cleve/2014/05/26/ordinary-differential-equation-solvers-ode23-and-ode45/

on the original work by Fehlberg [61] where the specifics of construction are described by Hairer et al. [72, pg.175].

In contrast to the six-stage fifth-order ERK method from Section 2.7.1 that is not constructed as an embedded pair, in order to construct find the RKF4(5)$_{6(6)}$ family the C(2) and C(3) simplifying assumptions (2.44b) are used. The presence of $\mathbf{b}$ vector components in the expressions for the D(1) simplifying assumptions (2.44c) introduces difficulties constructing embedded pairs. Therefore, this different set of simplifying assumptions than those used in Section 2.7.1 is typically used to find a reduced system.

As already discussed in Section 2.7.1, when performing reductions using the C(2) and C(3) simplifying assumptions (2.44b) for ERK methods, the constraint $b_2 = 0$ is required. Similar to the procedure used to find (2.65), reductions involving subtrees above the root no longer work for the C(η) simplifying assumptions (2.44b) with $\eta \geq 2$, $i = 2$. Following a similar procedure to the one that led to (2.65), the conditions

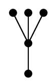
$$\sum_{i}^{s} b_i a_{i,2} = 0, \tag{2.68a}$$

$$\sum_{i}^{s} b_i c_i a_{i,2} = 0, \tag{2.68b}$$

$$\sum_{i,j}^{s} b_i a_{i,j} a_{j,2} = 0, \tag{2.68c}$$

now allow the C(2) and C(3) simplifying assumptions (2.44b) to be used for the appropriate subtrees not attached to the root in a similar manner to (2.65).

After using the C(2) and C(3) simplifying assumptions (2.44b), the reduced system consists of the C(2) and C(3) simplifying assumptions (2.44b), the quadrature conditions (2.44a), the relations (2.68a) and (2.68b), and the order condition corresponding to the tree


(2.69)

The order condition (2.69) is easy to incorporate into a reduced system because coefficients from the $\mathbf{A}$ matrix only appear linearly. The remaining condition (2.68c) is addressed below because it is automatically satisfied by the rest of the reduced system for the RKF4(5)$_{6(6)}$ family.

The quadrature conditions (2.44a) can be solved for the **b** vector by solving the linear system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & c_2 & c_3 & c_4 & c_5 & c_6 \\ 0 & c_2^2 & c_3^2 & c_4^2 & c_5^2 & c_6^2 \\ 0 & c_2^3 & c_3^3 & c_4^3 & c_5^3 & c_6^3 \\ 0 & c_2^4 & c_3^4 & c_4^4 & c_5^4 & c_6^4 \end{pmatrix} \begin{pmatrix} b_1 \\ 0 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \end{pmatrix}, \tag{2.70}$$

in an identical procedure to that described in Section 2.7.1. To simultaneously satisfy the simplifying assumptions C(2) and C(3) (2.44b) with $i = 3$, the necessary constraint to be found when both assumptions must be satisfied simultaneously is found by the procedure

$$\begin{aligned} a_{3,2}\, c_2 &= \frac{c_3^2}{2}, \quad a_{3,2}\, c_2^2 = \frac{c_3^3}{3}, \\ a_{3,2}\, c_2^2 &= \frac{c_3^2 c_2}{2}, \quad a_{3,2}\, c_2^2 = \frac{c_3^3}{3}, \\ \frac{c_3^2 c_2}{2} &= \frac{c_3^3}{3}, \\ \frac{c_3^3}{3} &= \frac{c_3^2 c_2}{2}, \\ 0 &= \frac{c_3^3}{3} - \frac{c_3^2 c_2}{2}, \\ 0 &= c_3^2 \left(\frac{c_3}{3} - \frac{c_2}{2} \right), \\ c_3 &= \frac{3}{2} c_2. \end{aligned} \tag{2.71}$$

The expression (2.71) allows immediately determining $a_{3,2}$, $a_{4,2}$, $a_{4,3}$ because it ensures consistency of the linear system

$$\begin{pmatrix} c_2 & 0 & 0 \\ c_2^2 & 0 & 0 \\ 0 & c_2 & c_3 \\ 0 & c_2^2 & c_3^2 \end{pmatrix} \begin{pmatrix} a_{3,2} \\ a_{4,2} \\ a_{4,3} \end{pmatrix} = \begin{pmatrix} \frac{c_3^2}{2} \\ \frac{c_3^3}{3} \\ \frac{c_4^2}{2} \\ \frac{c_4^3}{3} \end{pmatrix}. \tag{2.72}$$

The C(2) and C(3) simplifying assumptions (2.44b) with $i = 4, 5$ along with the relations (2.68a) and (2.68b) give the linear system

$$\begin{pmatrix} c_2 & c_3 & c_4 & 0 \\ c_2^2 & c_3^2 & c_4^2 & 0 \\ b_5 & 0 & 0 & b_6 \\ b_5 c_5 & 0 & 0 & b_6 c_6 \end{pmatrix} \begin{pmatrix} a_{5,2} \\ a_{5,3} \\ a_{5,4} \\ a_{6,2} \end{pmatrix} = \begin{pmatrix} \frac{c_5^2}{2} \\ \frac{c_5^3}{3} \\ -\sum_{i=3}^{4} a_{i,2} \\ -\sum_{i=3}^{4} c_i a_{i,2} \end{pmatrix}, \tag{2.73}$$

that can be solved for $a_{5,2}$, $a_{5,3}$, $a_{5,4}$, $a_{6,2}$ using the $a_{i,j}$ quantities of the RHS that are now known. The C(2) and C(3) simplifying assumptions (2.44b) with $i = 6$ along with the remaining order condition (2.69) give the linear system

$$\begin{pmatrix} c_3 & c_4 & c_5 \\ c_3^2 & c_4^2 & c_5^2 \\ b_6 c_3^3 & b_6 c_4^3 & b_6 c_5^3 \end{pmatrix} \begin{pmatrix} a_{6,3} \\ a_{6,4} \\ a_{6,5} \end{pmatrix} = \begin{pmatrix} -a_{6,2}c_2 + \frac{c_6^2}{2} \\ -a_{6,2}c_2^2 + \frac{c_6^3}{3} \\ -\sum_{i=3,j}^{5} b_i a_{i,j}\, c_j^3 - b_6 a_{6,2}\, c_2^3 + \frac{1}{20} \end{pmatrix},$$

that can be solved for $a_{6,3}$, $a_{6,4}$, $a_{6,5}$ using the $a_{i,j}$ quantities of the RHS that are now known. Finally, using (2.35), $a_{2,1}$, $a_{3,1}$, $a_{4,1}$, $a_{5,1}$, $a_{6,1}$ can be found by solving the linear system

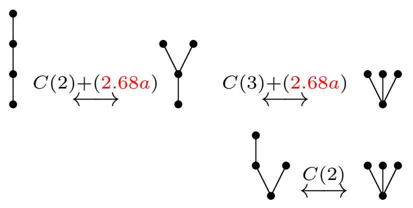
$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{2,1} \\ a_{3,1} \\ a_{4,1} \\ a_{5,1} \\ a_{6,1} \end{pmatrix} = \begin{pmatrix} c_2 \\ -a_{3,2} + c_3 \\ -\sum_{i=2}^{3} a_{4,i} + c_4 \\ -\sum_{i=2}^{4} a_{5,i} + c_5 \\ -\sum_{i=2}^{6} a_{6,i} + c_6 \end{pmatrix},$$

because all the $a_{i,j}$ quantities outside of the first column of the $\mathbf{A}$ matrix are now known.

The following discussion shows how to find the constraints required for an embedded pair. This discussion is the solution to exercise II.5.1 of Hairer et al. [72, pgs.185–186] and is a good introduction to the methodologies used for constructing ERK pairs in Chapter 3. In order to find a distinct $\hat{\mathbf{b}}$ vector that only satisfies the fourth-order ERK order conditions (but not the fifth-order ERK order conditions) when using the C(2) and C(3) simplifying assumptions (2.44b), a $\hat{\mathbf{b}}$ vector satisfying the fourth-order conditions can be found by solving the linear system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & c_2 & c_3 & c_4 & c_5 & c_6 \\ 0 & c_2^2 & c_3^2 & c_4^2 & c_5^2 & c_6^2 \\ 0 & c_2^3 & c_3^3 & c_4^3 & c_5^3 & c_6^3 \\ 0 & 0 & a_{3,2} & a_{4,2} & a_{5,2} & a_{6,2} \end{pmatrix} \begin{pmatrix} \hat{b}_1 \\ 0 \\ \hat{b}_3 \\ \hat{b}_4 \\ \hat{b}_5 \\ \hat{b}_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ 0 \end{pmatrix}, \tag{2.74}$$

which are the required conditions for a fourth-order ERK method because satisfying (2.68a) (in the fifth row) allows the simplifications



Although when (2.74) is non-singular the $\hat{\mathbf{b}}$ vector found is a unique solution, any unique solution of (2.74) for the $\hat{\mathbf{b}}$ vector would be identical to the $\mathbf{b}$ vector because the conditions given by (2.74) for fourth-order ERK are a subset of the conditions for fifth-order ERK. Therefore, to find a $\hat{\mathbf{b}}$ vector distinct from the $\mathbf{b}$ vector, the system (2.74) must be singular.

Consider the sub-matrix of rows 2–5 and columns 2–6 of (2.74) along with using the hint from exercise II.5.1 in Hairer et al. [72, pgs.185–186]. This hint is to find arbitrary multipliers α, β, γ that lead to a linear dependency in order to ensure (2.74) is singular and consistent. The linear dependency suggested by Hairer et al.'s hint is given by the system

$$\begin{pmatrix} c_3 & c_3^2 & c_3^3 \\ c_4 & c_4^2 & c_4^3 \\ c_5 & c_5^2 & c_5^3 \\ c_6 & c_6^2 & c_6^3 \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} a_{3,2} \\ a_{4,2} \\ a_{5,2} \\ a_{6,2} \end{pmatrix}. \tag{2.75}$$

To find a consistent solution for (2.75), it is helpful to use linear combinations of rows to find a simpler equivalent linear system, such as

$$\begin{pmatrix} c_3 & c_3^2 & c_3^3 \\ c_4 & c_4^2 & c_4^3 \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \end{pmatrix} \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} a_{3,2} \\ a_{4,2} \\ 0 \\ 0 \end{pmatrix}. \tag{2.76}$$

The linear combination giving a new row 5 of (2.76) can be found by multiplying each row i of (2.75) by b_{i+2} and summing these to form the linear combination that is the new row 5. It can clearly be seen that the resulting sums in the first, second, and third columns now correspond to the second-, third-, and fourth-order quadrature conditions. The RHS of the summation giving the new row 5 corresponds to (2.68a), which evaluates to zero. The linear combination giving a new row 6 of (2.76) can be found by multiplying row i of (2.75) by $b_{i+2}\, c_i$ and summing these to form the linear combination that is the new row 6. It can clearly be seen that the resulting sums in the first, second, and third columns correspond to the second-, third-, and fourth-order quadrature conditions. The RHS of the summation giving the new row 6 corresponds to (2.68b), which evaluates to zero.

Specific expressions for $a_{3,2}$, $a_{4,2}$ (these would have been found while solving (2.72)) give

$$a_{3,2} = \frac{3}{4} c_3,$$
$$a_{4,2} = -\frac{(3c_3^2 c_4^2 - 2c_3 c_4^3)}{6(c_2^2 c_3 - c_2 c_3^2)}.$$

Because the augmented matrix for (2.76) must have the same rank as the coefficient matrix, in order to have rank 3 or lower the determinant must be zero, i.e.,

$$\begin{vmatrix} c_3 & c_3^2 & c_3^3 & a_{3,2} \\ c_4 & c_4^2 & c_4^3 & a_{4,2} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & 0 \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & 0 \end{vmatrix} = 0,$$

where expanding and using known values for $a_{3,2}$, $a_{4,2}$, c_3 (using the SAGE computer algebra system [57]) gives

$$(45c_2^2c_4 - 24c_2c_4 - 3c_2 + 4c_4)(3c_2 - 2c_4)c_4 = 0, \tag{2.77}$$

where if $c_3 \neq c_4$ this leads directly to a constraint on c_4 of

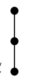
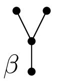
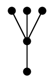
$$c_4 = \frac{3c_2}{4 - 24c_2 + 45c_2^2},$$

with the multipliers from (2.76) given by

$$\alpha = \frac{\frac{3}{2}c_2}{15\,c_2^3 - 12\,c_2^2 + 2\,c_2}, \quad \beta = \frac{-6c_2}{15\,c_2^3 - 12\,c_2^2 + 2\,c_2}, \quad \gamma = \frac{5\,c_2}{15\,c_2^3 - 12\,c_2^2 + 2\,c_2}.$$

In Section 3.3, it is shown that if $c_6 \neq 1$, then (2.77) is a necessary condition for six-stage fifth-order ERK methods that satisfy the C(2) simplifying assumption (2.44b) even when not part of a $5(4)_{6(6)}$ ERK pair.

The final condition (2.68c) required for the fifth-order method still remains to be satisfied, but the second part of exercise II.5.1 in Hairer et al [72, pgs.185–186] is to show that it is already satisfied by the rest of the reduced system used for the RKF$4(5)_{6(6)}$ family. Observe that the trees $[[\tau]], [[\tau^2]], [[\tau^3]]$ only differ from (2.68c) by substituting the factors c_j, c_j^2, c_j^3 for the factor $a_{j,2}$. Using SAGE [57] it can be verified that

α  $+\beta$  $+\gamma$  $= \alpha\frac{1}{6} + \beta\frac{1}{12} + \gamma\frac{1}{20} = 0,$

which can be factored to give

$$\sum_{i,j} b_i a_{i,j}(\alpha c_j + \beta c_j^2 + \gamma c_j^3) = 0,$$

where from (2.75) it is known that $\alpha c_j + \beta c_j^2 + \gamma c_j^3 = a_{j,2}$ and with substitutions leads directly to the condition (2.68c).

The free parameters used for the RKF$4(5)_{6(6)}$ family are c_2, c_5, c_6, demonstrating that the construction of an embedded pair comes at the cost of free parameters. Based on the method construction just described, the Butcher tableau published by Fehlberg [61] for the RKF4(5)$_{6(6)}$ method is given by

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & 0 & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
\mathbf{b} & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55} \\
\hline
\hat{\mathbf{b}} & \frac{25}{216} & 0 & \frac{1408}{2565} & \frac{2197}{4108} & -\frac{1}{5} & 0
\end{array} \tag{2.78}$$

2.7.3 The Dormand–Prince 5(4)$_{6(7)}$ ERK pair

The *Dormand–Prince 5(4) pair*, with the notation DP5(4)$_{6(7)}$ used for it in this thesis, was first published in 1980 by Dormand and Prince [47]. It continues to be widely used, including as one of the explicit IVP methods in the popular commercial software package MATLAB [153].

The DP5(4)$_{6(7)}$ method is a 5(4)$_{6(7)}$ ERK pair where the extra stage for the lower-order method of the embedded pair allows implementations to use a procedure known as *first same as last* (FSAL) [72, pg.167]. This FSAL property allows seven stages to be used with a minimal amount of extra work for the fourth-order method when the six-stage fifth-order method is used to advance the integration. This is because if a step is not rejected, the evaluation of the RHS (2.1a) for the seventh stage can be used as the initial RHS evaluation for the next step. Because the seventh stage is used as the first stage of the next step and would have to be evaluated anyways, when a step is not rejected, the effective number of stages of the DP5(4)$_{6(7)}$ method is six.

The construction of DP5(4)$_{6(7)}$ presented here is based on the original paper by Dormand and Prince [47] and this discussion in Hairer et al. [72, pgs.178–179]. Unlike the case for RKF4(5)$_{6(6)}$ where finding $\hat{\mathbf{b}} \neq \mathbf{b}$ required careful analysis, the $\hat{\mathbf{b}}$ vector for the seven-stage fourth-order method is always distinct from the $\mathbf{b}$ vector for the six-stage fifth-order method because $b_7 = 0$ and $\hat{b}_7 \neq 0$, making the method construction more straightforward. This easier construction procedure also allows the explicit use of the D(1) simplifying assumptions (2.44c) along with the C(2) and C(3) simplifying assumptions (2.44b). The quadrature conditions (2.44a) can be solved for the $\mathbf{b}$ vector by solving the linear system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & c_2 & c_3 & c_4 & c_5 & c_6 \\ 0 & c_2^2 & c_3^2 & c_4^2 & c_5^2 & c_6^2 \\ 0 & c_2^3 & c_3^3 & c_4^3 & c_5^3 & c_6^3 \\ 0 & c_2^4 & c_3^4 & c_4^4 & c_5^4 & c_6^4 \end{pmatrix} \begin{pmatrix} b_1 \\ 0 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \end{pmatrix},$$

in an identical procedure to that described in Section 2.7.1. Like the RKF4(5)$_{6(6)}$ method (2.78), satisfying the C(2) and C(3) simplifying assumptions (2.44b) simultaneously requires $c_3 = \frac{3}{2}c_2$ (2.71), and this allows immediately finding $a_{3,2}$, $a_{4,2}$, $a_{4,3}$ from

$$\begin{pmatrix} c_2 & 0 & 0 \\ c_2^2 & 0 & 0 \\ 0 & c_2 & c_3 \\ 0 & c_2^2 & c_3^2 \end{pmatrix} \begin{pmatrix} a_{3,2} \\ a_{4,2} \\ a_{4,3} \end{pmatrix} = \begin{pmatrix} \frac{c_3^2}{2} \\ \frac{c_3^3}{2} \\ \frac{c_4^2}{2} \\ \frac{c_4^3}{3} \end{pmatrix}.$$

Using (2.66) with $j = 2$ allows $a_{5,2}$, $a_{6,2}$ to be found by solving

$$\begin{pmatrix} b_5 & b_6 \\ b_5c_5 & b_6c_6 \end{pmatrix} \begin{pmatrix} a_{5,2} \\ a_{6,2} \end{pmatrix} = \begin{pmatrix} -\left(\sum_{i=3}^{4} b_i a_{i,2}\right) + b_2(1-c_2) \\ -\sum_{i=3}^{4} b_i c_i a_{i,2} \end{pmatrix},$$

using the now-known $a_{i,j}$ quantities for the RHS. Using (2.67) and (2.44b) with $i = 5$ allows $a_{5,3}$, $a_{5,4}$ to be found by solving

$$\begin{pmatrix} 0 & b_5(1-c_5)c_4(c_4-c_3) \\ c_3 & c_4 \end{pmatrix} \begin{pmatrix} a_{5,3} \\ a_{5,4} \end{pmatrix} = \begin{pmatrix} -\sum_{i=3}^{5} b_i(1-c_i)a_{i,2}c_2(c_2-c_3) + \frac{1}{60} - \frac{c_3}{24} \\ -(a_{5,2}c_2) + \frac{c_5^2}{2} \end{pmatrix},$$

using the now-known $a_{i,j}$ quantities for the RHS. Using (2.44c) with $j = 3, 4, 5$ allows $a_{6,3}$, $a_{6,4}$, $a_{6,5}$ to be found by solving

$$\begin{pmatrix} b_6 & 0 & 0 \\ 0 & b_6 & 0 \\ 0 & 0 & b_6 \end{pmatrix} \begin{pmatrix} a_{6,3} \\ a_{6,4} \\ a_{6,5} \end{pmatrix} = \begin{pmatrix} -\left(\sum_{i=1}^{5} b_i a_{i,3}\right) + b_3(1-c_3) \\ -\left(\sum_{i=1}^{5} b_i a_{i,4}\right) + b_4(1-c_4) \\ -\left(\sum_{i=1}^{5} b_i a_{i,5}\right) + b_5(1-c_5) \end{pmatrix},$$

using the now-known $a_{i,j}$ quantities for the RHS. Finally using (2.35) allows $a_{2,1}$, $a_{3,1}$, $a_{4,1}$, $a_{5,1}$, $a_{6,1}$ to be found by solving

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{2,1} \\ a_{3,1} \\ a_{4,1} \\ a_{5,1} \\ a_{6,1} \end{pmatrix} = \begin{pmatrix} c_2 \\ -a_{3,2} + c_3 \\ -\sum_{i=2}^{3} a_{4,i} + c_4 \\ -\sum_{i=2}^{4} a_{5,i} + c_5 \\ -\sum_{i=2}^{6} a_{6,i} + c_6 \end{pmatrix},$$

because all the $a_{i,j}$ quantities outside of the first column of the $\mathbf{A}$ matrix are now known.

The published Butcher tableau for the well-known DP5(4)$_{6(7)}$ ERK method is given by

$$\begin{array}{c|ccccccc} 0 & & & & & & & \\ \frac{1}{5} & \frac{1}{5} & & & & & & \\ \frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\ \frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\ \frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\ 1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\ 1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\ \hline \mathbf{b} & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 \\ \hline \hat{\mathbf{b}} & \frac{5179}{57600} & 0 & \frac{7561}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40} \end{array} \tag{2.79}$$

for which Dormand and Prince use the notation RK5(4)7FM. The free parameters for the family described in this subsection, including (2.79) and the two additional published pairs described in the next subsection, are c_2, c_4, c_5 along with $\hat{b}_7 \neq 0$ as an additional free parameter for the fourth-order method.

2.7.4 Other ERK pairs in the DP5(4)$_{6(7)}$ family

The DP5(4)$_{6(7)}$ method is not the only specific Butcher tableau (2.51) that Dormand and Prince published using the DP5(4)$_{6(7)}$ family. In the same paper [47] that the DP5(4)$_{6(7)}$ method was presented, Dormand and Prince gave another 5(4)$_{6(7)}$ ERK pair with a Butcher tableau of

$$\begin{array}{c|ccccccc} 0 & & & & & & & \\ \frac{2}{9} & \frac{2}{9} & & & & & & \\ \frac{1}{3} & \frac{1}{12} & \frac{1}{4} & & & & & \\ \frac{5}{9} & \frac{55}{324} & -\frac{25}{108} & \frac{50}{81} & & & & \\ \frac{2}{3} & \frac{83}{330} & -\frac{13}{22} & \frac{61}{60} & \frac{9}{110} & & & \\ 1 & -\frac{19}{28} & \frac{9}{4} & \frac{1}{7} & -\frac{27}{7} & \frac{22}{7} & & \\ 1 & \frac{19}{200} & 0 & \frac{3}{5} & -\frac{243}{400} & \frac{33}{40} & \frac{7}{80} & \\ \hline \mathbf{b} & \frac{19}{200} & 0 & \frac{3}{5} & -\frac{243}{400} & \frac{33}{40} & \frac{7}{80} & 0 \\ \hline \hat{\mathbf{b}} & \frac{431}{5000} & 0 & \frac{333}{500} & -\frac{7857}{10000} & \frac{957}{1000} & \frac{193}{2000} & -\frac{1}{50} \end{array} \tag{2.80}$$

for which they use the notation RK5(4)7FS [47] but which in this thesis is referred to as the DP5(4)S$_{6(7)}$ pair. The stability region of DP5(4)S$_{6(7)}$ (2.80) is given in Figure 2.15, and is larger than the one for DP5(4)$_{6(7)}$ (2.79) given by Figure 2.14.

In a followup paper in 1986 [48], Dormand and Prince revisited their coefficient selection process and gave another 5(4)$_{6(7)}$ ERK pair with a Butcher tableau of

$$\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{6}{13} & \frac{264}{2197} & -\frac{90}{2197} & \frac{840}{2197} & & & & \\
\frac{2}{3} & \frac{932}{3645} & -\frac{14}{27} & \frac{3256}{5103} & \frac{7436}{25515} & & & \\
1 & -\frac{367}{513} & \frac{30}{19} & \frac{9940}{5643} & -\frac{29575}{8208} & \frac{6615}{3344} & & \\
1 & \frac{35}{432} & 0 & \frac{8500}{14553} & -\frac{28561}{84672} & \frac{405}{704} & \frac{19}{196} & \\
\hline
\mathbf{b} & \frac{35}{342} & 0 & \frac{8500}{14553} & \frac{28561}{84672} & \frac{405}{704} & \frac{19}{196} & 0 \\
\hline
\hat{\mathbf{b}} & \frac{11}{108} & 0 & \frac{6250}{14553} & -\frac{2197}{21168} & \frac{81}{176} & \frac{171}{1960} & \frac{1}{40}
\end{array} \tag{2.81}$$

for which they use the notation RK5(4)7FC [48] but which in this thesis is referred to as the DP5(4)C$_{6(7)}$ pair. The stability region of DP5(4)C$_{6(7)}$ (2.81) is given by Figure 2.81 and represents a compromise between the stability region size of DP5(4)$_{6(7)}$ (2.79) and DP5(4)S$_{6(7)}$ (2.80). Dormand and Prince found that DP5(4)C$_{6(7)}$ (2.81) often gave better overall performance despite a leading error coefficient (2.54) nearly four times the size of the DP5(4)$_{6(7)}$ pair (2.79) [47, 48].

2.7.5 A Dormand–Prince 5(4)$_{6(6)}$ ERK pair

The DP5(4)M$_{6(6)}$ pair by Dormand and Prince [47] is given by the Butcher tableau

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & & & \\
\frac{2}{3} & \frac{226}{729} & -\frac{25}{27} & \frac{880}{729} & \frac{55}{729} & & \\
1 & -\frac{181}{270} & \frac{5}{2} & -\frac{266}{297} & -\frac{91}{27} & \frac{189}{55} & \\
\hline
\mathbf{b} & \frac{19}{216} & 0 & \frac{1000}{2079} & -\frac{125}{216} & \frac{81}{88} & \frac{5}{56} \\
\hline
\hat{\mathbf{b}} & \frac{31}{540} & 0 & \frac{190}{297} & -\frac{145}{108} & \frac{351}{220} & \frac{1}{20}
\end{array} \tag{2.82}$$

which was originally published in the same paper as DP5(4)$_{6(7)}$ (2.79). The original notation used by Dormand and Prince for the DP5(4)M$_{6(6)}$ method (2.82) is RK5(4)6M. Like the DP5(4)$_{6(7)}$ method (2.79), the DP5(4)M$_{6(6)}$ method (2.82) explicitly uses the C(2) and C(3) simplifying assumptions (2.44b) as well as the D(1) simplifying assumption (2.44c). The only difference in method construction for a 5(4)$_{6(6)}$ ERK pair such as the DP5(4)M$_{6(6)}$ method (2.82) as compared to a 5(4)$_{6(7)}$ ERK pair, such as the DP5(4)$_{6(7)}$ method (2.79), is that the former requires a condition for singularity of a linear system that is similar to that seen constructing RKF4(5)$_{6(6)}$ (2.78), which ensures that the system (2.74) is singular. The specifics of the construction of the DP5(4)M$_{6(6)}$ method (2.82) can be found in the original paper that also presented the DP5(4)$_{6(7)}$ method (2.79) [47].

2.7.6 The Cash–Karp 5(4)$_{6(6)}$ ERK pair

The CK4(5)$_{6(6)}$ pair by Cash and Karp [30] is given by the Butcher tableau

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & \\
\frac{3}{5} & \frac{3}{10} & -\frac{9}{10} & \frac{6}{5} & & & \\
1 & -\frac{11}{54} & \frac{5}{2} & \frac{70}{27} & \frac{35}{27} & & \\
\frac{7}{8} & \frac{1631}{55296} & \frac{175}{512} & \frac{575}{13824} & \frac{44275}{110592} & \frac{253}{4096} & \\
\hline
\mathbf{b} & \frac{37}{378} & 0 & \frac{250}{621} & \frac{125}{594} & 0 & \frac{512}{1771} \\
\hline
\hat{\mathbf{b}} & \frac{2825}{27648} & 0 & \frac{18575}{48384} & \frac{13525}{55296} & \frac{277}{14336} & \frac{1}{4}
\end{array} \tag{2.83}$$

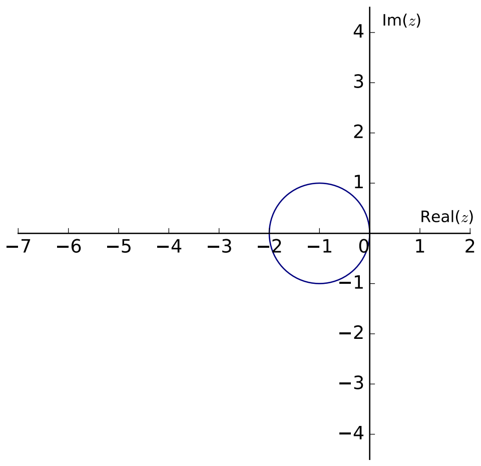
The CK4(5)$_{6(6)}$ pair (2.83) is used by the commercial software package MAPLE as well as well IVP solvers used by the popular open-source statistics and scientific computing platform R [114]. Although the construction procedure was not explicitly detailed by Cash and Karp, a quick calculation shows that the CK4(5)$_{6(6)}$ pair satisfies the C(2) and C(3) simplifying assumptions (2.44b) in its construction as well as the D(1) simplifying assumption (2.44c) with $i \in \{3, 4, 5\}$.

2.7.7 The 5(4)$_{6(7)}$ ERK pair from Papakostas and Papageorgiou

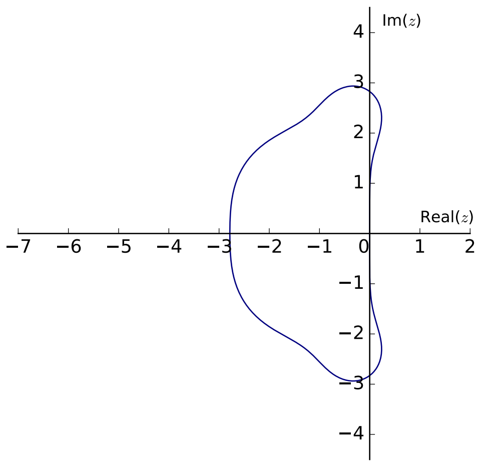
The PP5(4)$_{6(7)}$ pair by Papakostas and Papageorgiou [128] is given by the Butcher tableau

$$\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{9}{40} & \frac{9}{40} & & & & & & \\
\frac{21}{64} & \frac{91}{1024} & \frac{245}{1024} & & & & & \\
\frac{17}{18} & \frac{2512481}{1928934} & -\frac{752845}{137781} & \frac{1641520}{321489} & & & & \\
\frac{90}{91} & \frac{167600779485}{95414145736} & -\frac{1480997775}{200449886} & \frac{17446962744}{2621673509} & -\frac{4711141359}{138253149944} & & & \\
1 & \frac{502734007}{269217270} & \frac{6511090}{829521} & \frac{977303027168}{139196025045} & -\frac{31502187}{18516316} & -\frac{18516316}{12851752535} & & \\
1 & \frac{47641}{481950} & 0 & \frac{9183428608}{18507820275} & \frac{8673642}{2202775} & -\frac{2605848518}{189659475} & -\frac{4389}{430} & \\
\hline
\mathbf{b} & \frac{47641}{481950} & 0 & \frac{9183428608}{18507820275} & \frac{8673642}{2202775} & -\frac{2605848518}{189659475} & -\frac{4389}{430} & 0 \\
\hline
\hat{\mathbf{b}} & \frac{41590501}{460262250} & 0 & \frac{9282227273728}{17674968362625} & \frac{4486060422}{2103650125} & \frac{1016614753973}{181124798625} & \frac{3133053}{181124798625} & \frac{1}{20}
\end{array} \tag{2.84}$$

The PP5(4)$_{6(7)}$ pair (2.84) is notable because it is a classically constructed 5(4)$_6$ ERK pair (as opposed to using the new construction procedures introduced in Chapter 3, which probably require a computer algebra system) that satisfies only the C(2) simplifying assumptions (2.44b) but not the C(3) simplifying assumptions (2.44b). Also note that the values $c_4 = \frac{17}{18}$ and $c_5 = \frac{90}{91}$ are extremely close to each other as well as to $c_6 = c_7 = 1$; this type of nearness is suspected to sometimes cause performance issues from roundoff error [47], despite the PP5(4)$_{6(7)}$ pair (2.84) having an extremely small error leading error coefficient of $A^6 = 0.0000655$ (2.54) that is nearly six times smaller than the DP5(4)$_{6(7)}$ pair (2.79) [47].



(a) The stability region for the FE method (2.25).



(b) The stability region for the RK4 method (2.26).

Figure 2.13: Some stability regions for simple classic ERK methods.

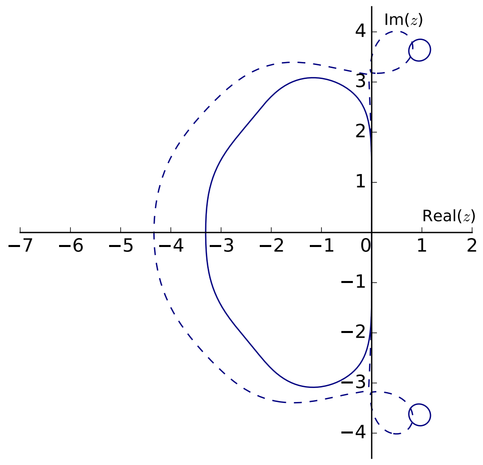


Figure 2.14: The stability regions for the component methods of the DP5(4)$_{6(7)}$ pair (2.79). The fifth-order method is the solid line and fourth-order method is the dashed line.

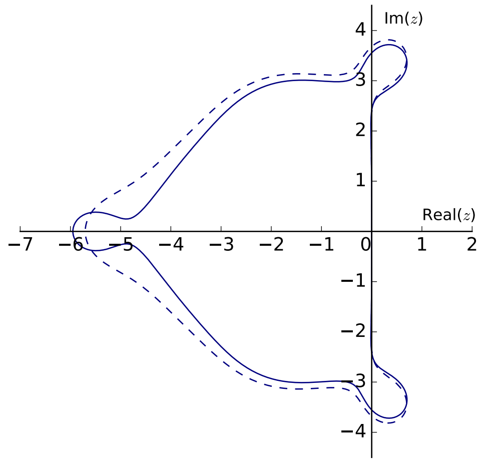


Figure 2.15: The stability regions for the component methods of the DP5(4)S$_{6(7)}$ pair (2.80). The fifth-order method is the solid line and fourth-order method is the dashed line.

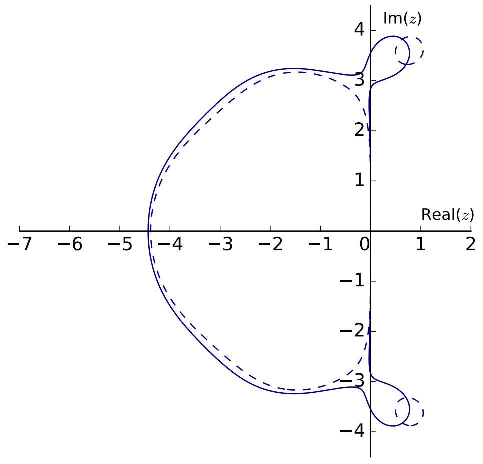


Figure 2.16: The stability regions for the component methods of the DP5(4)C$_{6(7)}$ pair (2.81). The fifth-order method is the solid line and fourth-order method is the dashed line.

Chapter 3

The complete solution of the order conditions for six-stage fifth-order ERK methods

"The traditional problem of choosing the coefficients leads to a nonlinear algebraic jungle, to which civilization and order were brought in the pioneering work of J.C.Butcher..."
(R. Alexander 1977)

In this chapter, solutions to the order conditions for six-stage fifth-order ERK methods and $5(4)_6$ ERK pairs are found, including solutions that do not use common simplifying assumptions, especially the C(2) simplifying assumptions (2.44b). As demonstrated in Section 2.7, the vast majority of higher-order (generally considered to mean fifth-order or greater) ERK methods, and many other RK methods in general, have been constructed using certain common simplifying assumptions (that generally means the C(2) and C(3) simplifying assumptions (2.44b), see discussions in the references [47, 61, 72, 128, 190, 192, 193, 195] and elsewhere). It has already been mentioned in Section 2.5.2 that it is shown by Cassity [32] (as well as in this chapter) that the commonly used D(1) simplifying assumptions (2.44c) must always hold for six-stage fifth-order ERK methods when $c_6 = 1$. By contrast, in papers published in 1966 and 1969, Cassity [31, 32] showed that solutions exist to the order conditions for six-stage fifth-order ERK methods that in fact do not require the C(2) simplifying assumptions (2.44b). In this chapter, Cassity's derivation of the complete algebraic solution for six-stage fifth-order ERK methods [31, 32] is examined, clarified, and replicated in detail. This includes the derivation of the constraints just described in a new explicit formulation that directly supports implementation in software such as *computer algebra systems* (CASs). Note that like Cassity, this thesis uses the terminology *complete algebraic solution* rather than *general algebraic solution*. This indicates that all cases of the full algebraic formulation are solved and not reduced in complexity by using simplifying assumptions (2.44), except as an additional special case. However, unlike a general solution, the complete solution does not usually address the cases where many of the linear systems solved can be singular. Instead of the standard simplifying assumptions, there are additional constraints specific to each case that must be used for the families of six-stage fifth-order ERK methods and $5(4)_6$ ERK pairs derived from Cassity's work [31, 32] and these are presented in Section 3.4.

Cassity's work [32] is the only known description of the complete algebraic solution of the six-stage fifth-order ERK order conditions or in fact of a complete solution of any set of order conditions fifth order and higher. Therefore, the use of this or any other higher-order complete algebraic solution to construct practical higher-order ERK methods and pairs without the common simplifying assumptions is a known open problem in the study of numerical methods. The nature of the open problem is stated explicitly by Papakostas and Papageorgiou [128] in the following passage:

> "A complete characterization of the solution of the 17 order conditions for a fifth-order method is given by Cassity [32]. However, the results of his study have not found any practical implementation. No analogous study exists for fifth-order pairs. All methods and pairs of orders five or higher are constructed according to certain types of simplifying assumptions, applied to the original system of order conditions."

Constructing practical $5(4)_6$ ERK pairs from Cassity's work [32], including the analogous study for fifth-order pairs, is the specific open problem this thesis addresses.

Finding the complete solution to the six-stage fifth-order ERK order (B.1)–(B.5), including the explicit expressions required for a fourth-order embedded method, is made possible because with appropriate variable substitutions these order conditions can be viewed as a sequence of linear systems (with a single quadratic equation in several cases) despite being highly non-linear. This alternate formulation of the six-stage fifth-order ERK order conditions that can be solved as a sequence of linear systems was a significant finding of Cassity in 1969 [32]. In addition, it is explicitly shown how to use Cassity's formulation of the order conditions to construct specific ERK methods, because the explicit expressions for the ERK coefficients in terms of the free parameters and the associated construction procedure are not fully described in Cassity's work or addressed elsewhere in the literature. This explicit construction procedure is further extended to embedded ERK methods and this represents the new work presented in this chapter that is not found elsewhere in the literature.

It is also shown in Chapter 4 that the particular formulation of the order conditions as linear systems that is introduced by Cassity and used in this thesis is well-suited to thoroughly solving and studying the order conditions using CASs such as Sage [57]. This is in comparison to the more ad-hoc procedures that can be found in many journal articles and textbooks such as those by Butcher [27] and Hairer et al. [72], that allow the order conditions to be easily solved by hand and fully presented in limited space. The software tool `OCSage`, which is built upon Sage [57] to study and utilizes the complete solution of these order conditions, is further described in Chapter 4. However, the mathematics can be fully described before presenting the software tools that are used for this study.

3.1 Review of ERK method construction without standard simplifying assumptions

Cassity provides only high-level information supporting his published mathematical expressions as the actual complete solution [32] of the six-stage fifth-order ERK order conditions, with the only specific ERK Butcher tableau (2.34) without the C(2) simplifying assumptions (2.44b) he published being from a prior partial solution in his 1966 paper [31]. As well, there are only scant details on the procedure that might be used to construct an actual ERK method from the mathematical expressions provided. However, there is a 1968 abstract for a conference talk by Cassity and Steinkopff with the title "Optimization of parameters for the complete solution of the fifth-order Runge–Kutta equations" [33] indicating that the specific procedure for construction was known but not published. Using `OCSage`, the published mathematical expressions from Cassity's work [32] have been verified and found identical to the expressions derived in this chapter. However, in Section 3.4.1, small extensions are found to Cassity's published complete solution where a quadratic irrationality (a square root) occurs in the situations where Cassity mentioned he purposely limited his solutions so that the ERK coefficients would be rational expressions in the free parameters. Using modern CASs, it is no longer necessary to be restricted to much simpler rational expressions. These small extensions are part of the contribution this thesis makes and are not contained in Cassity's published expressions [32] or elsewhere in the literature.

Cassity's work on the complete solution [32] is known to have been cited by at least 15 other studies (from a search on Google Scholar in early 2018), and none of those citing studies constructed new ERK methods with, or otherwise utilized, any part of his complete algebraic solution. In the time since Papakostas and Papageorgiou [128] published the comment given above, two other studies that claim to give numerical solutions to higher-order ERK order conditions that do not use the C(2) simplifying assumptions (2.44b) have appeared [99, 183]; however neither are based on either Cassity's work [31, 32] or another complete algebraic solution of the ERK order conditions. Cassity's partial algebraic solution and the two other studies are described in the following papers:

- Cassity [31] in 1966 gave a complete solution to a restricted version of the six-stage fifth-order ERK order conditions with $b_2 = 0$ that is derived in full in Section 3.3. This restricted solution with $b_2 = 0$ encompasses six-stage fifth-order ERK methods constructed using the C(2) simplifying assumptions (2.44b) plus an additional family without the C(2) simplifying assumptions (2.44b) that also has $b_2 = 0$. Cassity also gave a partial solution to the six-stage fifth-order ERK order conditions with $b_2 \neq 0$ and presented a single ERK method with $b_2 \neq 0$ that does not use the C(2) simplifying assumptions (2.44b) based on this partial solution. Note that Verner does present this single tableau with $b_2 \neq 0$ in a technical report [191], but it is taken verbatim from Cassity's 1966 paper [31] and not used further. However, this ERK method with $b_2 \neq 0$ was not subject to the refinements necessary for practical usage and it was not indicated that it was ever used to solve any IVPs. The author has not found the form of this partial complete solution to the six-stage fifth-order order conditions useful for constructing practical ERK methods (it involves solving unnecessarily difficult equations that are found from the expansion of determinants into degree-sixteen polynomials, where it is not obvious how to explicitly find the real (non-complex) solutions [31]). The formulation of the complete solution in Section 3.4 from Cassity's 1969 paper [32] means it is not necessary to consider this partial complete solution any further.
- Tsitouras [182, 183] uses an evolutionary optimization method (differential evolution) along with a symbolic/numeric solution process to solve the order conditions and generate many candidate $5(4)_6$

ERK pairs. However, there are few details, such as supporting characteristic numbers, as to why the two specific embedded pairs presented were selected, the extent to which the complete solution space of the system solved was explored using his symbolic/numeric techniques, or how to build upon the research.

In addition, the published coefficients of the embedded ERK pairs in Tsitouras' 2011 journal article [183] do not satisfy the appropriate order conditions at all, and thus are incorrect and unusable. However, it is highly likely this is just a transcription error. Coefficients that satisfy the appropriate order conditions to about the magnitude of 64-bit unit roundoff appear for a different ERK pair of the same family in a 2009 conference paper also published by Tsitouras [182]. Corrected and higher precision coefficients for the ERK pair presented Tsitouras' 2011 paper [183] have been published online by Peter Stone.[1] It should be further noted that in the course of this thesis work, the author (Andrew Kroshko) has found the pairs given by Tsitouras (both the original coefficients and those corrected by Peter Stone) cannot exist with exact coefficients (in terms of rational numbers, algebraic numbers, or any other analytic expressions); this is despite satisfying the order conditions (2.43) to at least approximately the magnitude of 64-bit unit roundoff, the specifics of these issues are further addressed in Section 4.15.1. In fact, the residuals of the order conditions (and other properties, such as determinants that must be zero) are significantly greater than the values of similar residuals from when the coefficients of methods such as $\text{DP5}(4)_{6(7)}$ (2.78) and $\text{RKF4}(5)_{6(6)}$ (2.79) are converted to floating-point numbers. A more detailed discussion of how Tsitouras' pairs compare to the families constructed in this chapter is given in Section 4.15.1.

- Khashin [99] uses a symbolic/numeric approach to solve ERK order conditions and constructs higher-order ERK methods that do not use the C(2) simplifying assumptions (2.44b). However, no embedded pairs are constructed, the fifth-order ERK case is not addressed, the study is limited to ERK methods of orders higher than but not including fifth order, and no discussion of the performance of how the new methods performed solving IVPs was included. Due to a similar use of numerical optimization in a symbolic/numeric solution process to Tsitouras' work [182, 183], it is likely that embedded pairs based on Khashin's ERK methods would have similar issues to Tsitouras' ERK methods.

Although the overall approach to constructing ERK methods used in this thesis could be called a symbolic/numeric approach, it is different from the symbolic/numeric approaches described and named as such by Tsitouras [182, 183] and Khashin [99]. In this thesis, the order conditions are solved with a purely symbolic (algebraic) approach, giving all calculated coefficients as explicit expressions of the free parameters (a process described in this chapter), after which a search of the coefficient space of the free parameters is done using numerical computation (described in Chapter 4) to find good candidate formulae for a particular family.

[1]Peter Stone was a mathematics professor at Vancouver Island University who has since retired. He has not published his calculations related to Tsitouras' methods (and many others) beyond his personal website. The author of this thesis (Andrew Kroshko) did similar calculations for both correcting and finding Tsitouras' pairs to higher precision, but Peter Stone's calculations give much lower residuals.

The advantage of the process used in this thesis is that it allows complete and systematic searches that can completely characterize the space of free parameters to whatever precision is desired (at least for $5(4)_6$ ERK pairs), and every solution, even if it must be approximated by floating-point numbers for practical implementations, can be made as near as floating-point tolerance allows to exactly satisfying the order conditions. Arbitrary-precision arithmetic provided by Sage is used for the final coefficients of the methods presented in this thesis. This ensures that all coefficients are found as accurately as possible and that residuals from checking the order conditions are typically much smaller than unit roundoff for 64-bit floating-point numbers.

3.2 Variable substitutions and condensed matrix form

The mathematical notation used throughout this thesis is chosen to be consistent with the modern literature on RK methods, which in turn is based on notation commonly attributed to Butcher [25]. Much of the notation in Cassity's work [31, 32] on the complete solution of the order conditions for six-stage fifth-order ERK methods differs from modern notation. For example, Cassity refers to the **A** matrix from the Butcher tableau (2.34) as **c**, the **b** vector from the Butcher tableau (2.34) as **R**, and the **c** vector from the Butcher tableau (2.34) as **a**. However, where no equivalent standard notation for a concept exists other than Cassity's original work and there are no conflicts, the original notation has been preserved for clarity and continuity with the original work, e.g., the multipliers given by (3.28).

3.2.1 Variable substitutions

Common variable substitutions used when finding the complete solution of the order conditions for six-stage fifth-order ERK methods are given by

$$p_i = \sum_{j=1}^{s} a_{i,j}\, c_j, \tag{3.1a}$$

$$q_i = \sum_{j=1}^{s} a_{i,j}\, c_j^2, \tag{3.1b}$$

$$r_i = \sum_{j=1}^{s} b_j\, a_{j,i}, \tag{3.1c}$$

$$s_i = \sum_{j=1}^{s} b_j\, c_j\, a_{j,i},^{[2]} \tag{3.1d}$$

[2]The **s** vector with s_i components is a different variable from the number of stages s.

with additional corresponding substitutions given by

$$P_i = \sum_{j=1}^{s} a_{i,j}\, c_j - \frac{1}{2} c_i, \tag{3.2a}$$

$$Q_i = \sum_{j=1}^{s} a_{i,j}\, c_j^2 - \frac{1}{3} c_i^2, \tag{3.2b}$$

$$R_i = \sum_{j=1}^{s} b_j\, a_{j,i} - b_i(1 - c_i), \tag{3.2c}$$

$$S_i = \sum_{j=1}^{s} b_j\, c_j\, a_{j,i} - \frac{1}{2} b_i(1 - c_i^2). \tag{3.2d}$$

The variable substitutions given by (3.1) and (3.2) were originally those used by Butcher to prove the non-existence of a five-stage fifth-order ERK method [25]. The variable substitutions (3.1) and (3.2), as well as some special cases of them, are the ones used by Cassity [31, 32] and in this chapter to eliminate the extreme non-linearity of the six-stage fifth-order ERK order conditions and find the complete solution. As mentioned in Section 2.5.2, the row-sum conditions given by (2.35) are always assumed in this thesis as well.

In this chapter, the notation $\tilde{\mathbf{b}} = \{b_3, b_4, b_5, b_6\}$, $\tilde{\mathbf{c}} = \{c_3, c_4, c_5, c_6\}$, $\tilde{\mathbf{p}} = \{p_3, p_4, p_5, p_6\}$, and $\tilde{\mathbf{P}} = \{P_3, P_4, P_5, P_6\}$ are used for certain subsets of vectors in the Butcher tableau (2.34) or the substitutions given by (3.1) and (3.2).

3.2.2 Condensed matrix form

As already mentioned, although the order conditions for fifth-order ERK (B.1)–(B.5) are highly non-linear and difficult to solve as explicit families, an observation commonly made in the literature (see discussions in the references [25, 47, 61, 128, 190, 192, 193, 195] and elsewhere) including popular texts [27, 72, 73], with statements such as the "non-linear algebraic jungle" used in the context of closely related IRK methods (see the references [1][73, pgs.91][108, pgs.179–182]). Part of a decomposition of the solution of multi-variate polynomials can be represented as the solution of single-variable polynomials, and if these single variable polynomials are greater than degree five, the classic Abel–Ruffini theorem indicates that there may be solutions that are not expressible in radicals, much less as rational functions [41, pgs.28–29]. Therefore, given the importance of RK order conditions, it is fortunate they have a solution in radicals even at higher orders and it is even more fortunate a complete solution is available to an important system such as the six-stage fifth-order ERK order conditions.

Despite the complexities of solving general multi-variate systems of polynomials, the procedure outlined by Cassity [32] along with appropriate variable substitutions (3.1) allows repeated expressions to be collected. For six-stage fifth-order ERK methods, the solution given by Cassity allows all coefficients to be given either as rational functions of the free parameters or for 3 of the 6 solution cases described in Section 3.4.1 requires a single quadratic irrationality (square root). In Section 3.4.1, new work gives the complete solution of two of the three solution cases requiring quadratic irrationality that were given by Cassity [32] only as restricted cases that had a solution in terms of rational functions. Cassity mentions knowingly applying restrictions to simplify these cases [32]. However, this makes his 1969 paper on the complete solution of the six-stage fifth-order order conditions incomplete.

In order to illustrate the patterns in the order conditions that allow their complete solution, with the variable substitutions used by Luther and Konen [113] that were also used by Cassity [31] in his 1966 paper to reformulate the order conditions, observe the patterns in the order conditions (B.1)–(B.5) for six-stage fifth-order ERK methods when written out using the substitutions (3.1). These reformulations and other similar reformulations of the order conditions are critical to finding the complete solution to the six-stage fifth-order ERK order conditions.

Taking advantage of the patterns in the order conditions, a formulation is used that is known as the *condensed matrix form* (CMF), which is a term currently used exclusively by Cassity [32] and in this thesis. CMF allows the order conditions to be analyzed and solved as a sequence of linear systems, which easily accommodates the cases requiring a quadratic irrationality (square root) mentioned above. Due to the well-known and easily applied results on the existence and uniqueness of solutions for linear systems, this allows the complete solution (and by extension the general solution if desired) to be found. Because the concept of CMF is not formally defined outside of the extremely limited body of work mentioned, all equations involving matrices given in this chapter are in CMF, and the particulars are introduced in the context of the derivation of the complete solution for ERK methods as this chapter unfolds. A more formal definition must wait for future work on more systems of order conditions other than just the six-stage fifth-order ERK methods discussed in this thesis.

3.3 Complete solution of a restricted system of six-stage fifth-order ERK order conditions with $b_2 = 0$

The complete solution of the six-stage fifth-order ERK order conditions with the restriction $b_2 = 0$ was first described by Cassity in 1966 [31]. This section follows Cassity's treatment closely, except that the term CMF was introduced by Cassity only in 1969 [32]; thus the term CMF is not used in the original work on the restricted system from 1966 [31] but is used in this section where it would have been appropriate in the original work. At the end of this section, new work is presented that extends this complete solution of the restricted system with $b_2 = 0$ to embedded pairs and includes families that are generalizations of the families upon which the popular RKF4(5)$_{6(6)}$ method (2.78) and DP5(4) method (2.79) are based.

A summary of the constructed families with $b_2 = 0$, free parameters, and examples of Butcher tableaux (2.34) are given in Table 3.1 for single ERK methods and Table 3.2 for embedded pairs. For comparison purposes, the constructed families are again summarized in the analogous tables for the complete solution with $b_2 \neq 0$ that is covered in Section 3.4, with Table 3.3 for single ERK methods and Table 3.4 for embedded pairs.

Although the unrestricted complete solution with $b_2 \neq 0$ is discussed in the next section, the C(2) simplifying assumptions (2.44b) and other solutions with $b_2 = 0$ occur as a singularity of the unrestricted complete solution covered in Section 3.4. This requires a special case of the more complicated Cases V and VI (see Section 3.4.1) of the complete solution with $b_2 \neq 0$. Because the restricted case with $b_2 = 0$ appears as an immediately obvious special case that was published first, has lower complexity, and due to the continued importance of RK methods constructed with the C(2) simplifying assumptions (2.44b), the restricted system with $b_2 = 0$ is addressed separately and before the complete solution.

3.3.1 Consistency of the quadrature conditions and non-branching conditions of height two

Completely solving the six-stage fifth-order order conditions with $b_2 = 0$ requires analyzing and solving several linear systems in sequence, which helps motivate the CMF required for finding the complete solution with $b_2 \neq 0$ in the next section. The condition $b_2 = 0$ is shown in Section 2.7.1 to be required by the C(2) simplifying assumptions, but following Cassity's paper [31] in this section it is shown the converse is not true, i.e., $b_2 = 0$ does not require the C(2) simplifying assumptions (2.44b).

Consider the quadrature conditions (B.1), i.e., τ, $[\tau]$, $[\tau^2]$, $[\tau^3]$, $[\tau^4]$, required for six-stage fifth-order ERK with $b_2 = 0$ that are given by the linear system

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & c_3 & c_4 & c_5 & c_6 \\ 0 & c_3^2 & c_4^2 & c_5^2 & c_6^2 \\ 0 & c_3^3 & c_4^3 & c_5^3 & c_6^3 \\ 0 & c_3^4 & c_4^4 & c_5^4 & c_6^4 \end{pmatrix} \begin{pmatrix} b_1 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \end{pmatrix}, \tag{3.3}$$

where the first row is obviously linearly independent of the other rows and can be largely ignored for analyzing ERK construction based on the restricted system. In this thesis, the components of the $\mathbf{c}$ vector are always assumed to be distinct, and this implies that the rows corresponding to the quadrature conditions are linearly independent; observe that the determinant given by (C.4) is for a matrix that matches the lower four rows of (3.3) and that this determinant must be non-zero when the components of the $\tilde{\mathbf{c}}$ vector are assumed to be distinct and non-zero. This non-zero determinant implies that the $\mathbf{b}$ vector is completely determined by the five equations in five variables of (3.3) when the components of the $\mathbf{c}$ vector are distinct. The existence of six-stage fifth-order ERK methods with non-distinct $\mathbf{c}$ components has been established by Konen and Luther [102]. However, no specific ERK methods with non-distinct $\mathbf{c}$ components are known from either their work or the general literature, and the case of constructing actual six-stage fifth-order ERK methods with non-distinct $\mathbf{c}$ components is not considered further in this thesis and remains an open problem.

Many of the explicit expressions and constraints that arise from the construction of ERK methods follow patterns involving rational functions of algebraic expressions, such as *symmetric polynomials* and *Vander-*

monde determinants (C.2) in the free parameters (that are mainly but not exclusively the components of the $\mathbf{c}$ vector). In the case of (3.3), explicit expressions for the $\tilde{\mathbf{b}}$ vector follow this form. Methodologies suitable for finding the appropriate rational functions in terms of components of the $\mathbf{c}$ vector are to either use *Cramer's rule* [6, pgs.111] with the relations (C.5) to calculate determinants of the form (C.6), or *Gaussian elimination* [6, pg.8] in order to solve for the $\tilde{\mathbf{b}}$ vector from rows 2–5 of (3.3). Either methodology yields

$$\begin{aligned}
b_3 &= \left(\frac{1}{2}c_4c_5c_6 - \frac{1}{3}(c_4c_5 + c_4c_6 + c_5c_6) + \frac{1}{4}(c_4 + c_5 + c_6) - \frac{1}{5}\right)\frac{\Delta_{456}}{\Delta_{3456}},\\
b_4 &= \left(\frac{1}{2}c_3c_5c_6 - \frac{1}{3}(c_3c_5 + c_3c_6 + c_5c_6) + \frac{1}{4}(c_3 + c_5 + c_6) - \frac{1}{5}\right)\frac{\Delta_{356}}{\Delta_{3456}},\\
b_5 &= \left(\frac{1}{2}c_3c_4c_6 - \frac{1}{3}(c_3c_4 + c_4c_5 + c_4c_6) + \frac{1}{4}(c_3 + c_4 + c_6) - \frac{1}{5}\right)\frac{\Delta_{346}}{\Delta_{3456}},\\
b_6 &= \left(\frac{1}{2}c_3c_4c_5 - \frac{1}{3}(c_3c_4 + c_3c_5 + c_4c_5) + \frac{1}{4}(c_3 + c_4 + c_5) - \frac{1}{5}\right)\frac{\Delta_{345}}{\Delta_{3456}},
\end{aligned} \tag{3.4}$$

where Δ_{ijk} and Δ_{ijkl} are Vandermonde determinants defined in Appendix C by (C.7) that can be used to simplify the expression $\frac{\Delta_{345}}{\Delta_{3456}}$ to match the form of the expressions given by Cassity [31]. The values of $\tilde{\mathbf{b}}$ from (3.4) now uniquely determine b_1 from the order condition corresponding to τ.

Now that the quadrature conditions (3.3) have been satisfied, consider the linear system with the quadrature conditions (3.3) as the upper sub-matrix and the non-branching conditions of height two (B.2), i.e., $[\tau]$, $[\tau^2]$, $[\tau^3]$, $[\tau^4]$, $[[\tau]]$, $[\tau[\tau]]$, $[\tau^2[\tau]]$, $[[\tau]^2]$, given by

$$\left(\begin{array}{c|cccc}
0 & c_3 & c_4 & c_5 & c_6\\
0 & c_3^2 & c_4^2 & c_5^2 & c_6^2\\
0 & c_3^3 & c_4^3 & c_5^3 & c_6^3\\
0 & c_3^4 & c_4^4 & c_5^4 & c_6^4\\
\hline
0 & p_3 & p_4 & p_5 & p_6\\
0 & c_3p_3 & c_4p_4 & c_5p_5 & c_6p_6\\
0 & c_3^2p_3 & c_4^2p_4 & c_5^2p_5 & c_6^2p_6\\
0 & p_3^2 & p_4^2 & p_5^2 & p_6^2
\end{array}\right)
\begin{pmatrix} b_3\\ b_4\\ b_5\\ b_6\end{pmatrix}
=
\left(\begin{array}{c}
\frac{1}{2}\\ \frac{1}{3}\\ \frac{1}{4}\\ \frac{1}{5}\\
\hline
\frac{1}{6}\\ \frac{1}{8}\\ \frac{1}{10}\\ \frac{1}{20}
\end{array}\right), \tag{3.5}$$

where it was just seen that the $\tilde{\mathbf{b}}$ vector is completely determined by the upper sub-matrix of (3.5), and this allows the consistency of the lower sub-matrix of (3.5) to be examined separately. Consider the equations represented by the lower sub-matrix of (3.5) with the now-known values of the $\tilde{\mathbf{b}}$ vector multiplied with the RHS coefficient matrix and with the $\tilde{\mathbf{p}}$ vector taken out as the vector of unknowns. This gives the linear system

$$\begin{pmatrix} b_3 & b_4 & b_5 & b_6 \\ b_3 c_3 & b_4 c_4 & b_5 c_5 & b_6 c_6 \\ b_3 c_3^2 & b_4 c_4^2 & b_5 c_5^2 & b_6 c_6^2 \\ b_3 p_3 & b_4 p_4 & b_5 p_5 & b_6 p_6 \end{pmatrix} \begin{pmatrix} p_3 \\ p_4 \\ p_5 \\ p_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{8} \\ \frac{1}{10} \\ \frac{1}{20} \end{pmatrix}. \tag{3.6}$$

An issue with using (3.6) directly to determine the consistency of (3.5) is that $\tilde{\mathbf{p}}$ appears linearly only in the first three equations but quadratically in the last equation. Therefore, (3.6) is not yet a linear system with $\tilde{\mathbf{p}}$ as the unknown vector. To address this, adding linear combinations of the other rows of (3.5) (multiplied out with the already-determined $\tilde{\mathbf{b}}$ vector) to rows of the lower sub-matrix of (3.5) (also multiplied with the already-determined $\tilde{\mathbf{b}}$ vector) can be used to find an equivalent homogeneous linear system with the desirable unknowns (the $\tilde{\mathbf{p}}$ vector) only appearing linearly. To find this homogeneous linear system by using (3.5), subtract $\frac{1}{2}$ times rows 3–5 from rows 6–9 respectively and $\frac{1}{4}$ times row 5 from row 10 to yield an equivalent system to (3.6) in terms of the $\tilde{\mathbf{P}}$ vector defined in (3.2) that can be simplified to

$$\begin{pmatrix} b_3 & b_4 & b_5 & b_6 \\ b_3\, c_3 & b_4\, c_4 & b_5\, c_5 & b_6\, c_6 \\ b_3\, c_3^2 & b_4\, c_4^2 & b_5\, c_5^2 & b_6\, c_6^2 \\ b_3\,(P_3 + c_3^2) & b_4\,(P_4 + c_4^2) & b_5 (P_5 + c_5^2) & b_6 (P_6 + c_6^2) \end{pmatrix} \begin{pmatrix} P_3 \\ P_4 \\ P_5 \\ P_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.7}$$

where further subtracting the third row from the fourth row of (3.7) and factoring out the $\tilde{\mathbf{P}}$ vector gives

$$\begin{pmatrix} b_3 & b_4 & b_5 & b_6 \\ b_3\, c_3 & b_4\, c_4 & b_5\, c_5 & b_6\, c_6 \\ b_3\, c_3^2 & b_4\, c_4^2 & b_5\, c_5^2 & b_6\, c_6^2 \\ b_3\, p_3 & b_4\, p_4 & b_5\, p_5 & b_6\, p_6 \end{pmatrix} \begin{pmatrix} P_3 \\ P_4 \\ P_5 \\ P_6 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.8}$$

where for this homogeneous linear system to be consistent, the conditions that must hold is either (or both of):

1. The determinant of the RHS coefficient matrix of (3.8) vanishes, allowing $\tilde{\mathbf{P}} \neq \mathbf{0}$; hence the C(2) simplifying assumptions (2.44b) are not satisfied. With the determinant of the RHS vanishing but $\tilde{\mathbf{P}} \neq \mathbf{0}$, this is the "nontrivial" solution [163, pg.60] of the homogeneous linear system (3.8).

2. The homogeneous linear system (3.8) is satisfied by the "trivial" solution [163, pg.43], i.e., $\tilde{\mathbf{P}} = \mathbf{0}$, which can easily be seen from (3.2) to imply the C(2) simplifying assumptions (2.44b). Note that the determinant of the RHS coefficient matrix is also always zero in this case because the C(2) simplifying assumptions (2.44b) imply row 4 is $\frac{1}{2}$ times row 3 of (3.8) and hence linearly dependent. The C(2) simplifying assumptions (2.44b) can therefore be seen as the "trivial" solution [163, pg.43] of the

homogeneous linear system (3.8).

Considering the first case (without the C(2) simplifying assumptions (2.44b) satisfied) and with the determinant of (3.8) as zero, factoring the $\tilde{\mathbf{b}}$ vector out of the columns of (3.8) and writing this determinant as co-factors gives an equivalent relation of

$$p_3\Delta_{456} - p_4\Delta_{356} + p_5\Delta_{346} - p_6\Delta_{345} = 0, \tag{3.9}$$

which is linear in the $\tilde{\mathbf{p}}$ vector. Note that the expression (3.9) is also satisfied when the C(2) simplifying assumptions (2.44b) hold; this can easily be shown by substituting $p_i = \frac{1}{2}\,c_i^2$ and simplifying.

With a distinct $\mathbf{c}$ vector, the first three rows of (3.8) are linearly independent. Note the similarity to the rows of the quadrature conditions (3.3) (see the form of the determinant (C.4)), except for the presence of the $\tilde{\mathbf{b}}$ vector in each column that can be factored out. Therefore, the condition (3.9) for a zero determinant of (3.8) implies the last row of (3.5) is linearly dependent on some or all of the other three rows. This result also holds for the linear system (3.6) because only matrix operations that result in an equivalent system are used to get to (3.8). Hence, due to the linear dependence of the last row of (3.5) and (3.6), this row and the corresponding order condition $[[\tau]^2]$ can be replaced by (3.9) giving the linear system

$$\begin{pmatrix} b_3 & b_4 & b_5 & b_6 \\ b_3c_3 & b_4c_4 & b_5c_5 & b_6c_6 \\ b_3c_3^2 & b_4c_4^2 & b_5c_5^2 & b_6c_6^2 \\ \Delta_{456} & -\Delta_{356} & \Delta_{346} & -\Delta_{345} \end{pmatrix} \begin{pmatrix} p_3 \\ p_4 \\ p_5 \\ p_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{6} \\ \frac{1}{8} \\ \frac{1}{10} \\ 0 \end{pmatrix}, \tag{3.10}$$

as equivalent to (3.6). The $\tilde{\mathbf{p}}$ vector now appears linearly in all equations of (3.10), where if the determinant of the RHS coefficient matrix is nonzero and $\tilde{\mathbf{p}}$ is solved for, the solution corresponds to $\tilde{\mathbf{P}} = \mathbf{0}$, i.e., the C(2) simplifying assumptions (2.44b). However, if the determinant of the coefficient matrix of (3.10) is zero, i.e., the *rank* of the coefficient matrix is 3, this allows one of the four components of the $\tilde{\mathbf{p}}$ vector to be a free parameter independent of the components of the $\tilde{\mathbf{c}}$ vector. With the $\tilde{\mathbf{b}}$ vector known explicitly in terms of the $\tilde{\mathbf{c}}$ vector (3.4), this condition for a zero determinant of the coefficient matrix of (3.10) can be found and with some simplification is given as

$$\begin{aligned} &\frac{8}{9}c_3c_4c_5c_6 \\ &-\frac{1}{2}(c_3c_4c_5 + c_3c_4c_6 + c_3c_5c_6 + c_4c_5c_6) \\ &+\frac{1}{3}(c_3c_4 + c_3c_5 + c_3c_6 + c_4c_5 + c_4c_6 + c_5c_6) \\ &-\frac{1}{4}(c_3 + c_4 + c_5 + c_6) + \frac{1}{5} = 0. \end{aligned} \tag{3.11}$$

Note that this condition is generally not satisfied when the C(2) simplifying assumptions (2.44b) hold because the RHS coefficient matrices are different between (3.8) (which always has a zero determinant even for the trivial solution) and (3.10) (which only requires a zero determinant when the non-trivial solution of (3.8) is considered). Note also that either solution gives the same number of free parameters; the expression (3.11) requires a constraint on one of c_3, c_4, c_5, c_6 but allows one of p_3, p_4, p_5, p_6 to be a free parameter.

3.3.2 Solving the taller trees

After the consistency of the quadrature conditions and non-branching conditions of height two (3.5) is established, continuing to follow the original work by Cassity [31] allows the explicit solution of the remaining order conditions. Consider the specific variable substitutions given by

$$r_3^* = b_4\, a_{4,3} + b_5\, a_{5,3} + b_6\, a_{6,3}, \tag{3.12a}$$

$$s_3^* = b_4 c_4 a_{4,3} + b_5\, c_5\, a_{5,3} + b_6\, c_6\, a_{6,3}, \tag{3.12b}$$

$$r_4^* = b_5 a_{5,4}, \tag{3.12c}$$

$$r_4^\dagger = b_6 a_{6,4}, \tag{3.12d}$$

$$r_5^* = b_6 a_{6,5}, \tag{3.12e}$$

which are the variable substitutions used by Cassity [31], but using notation that follows the more general-purpose variable substitutions defined by (3.1).

Consider the CMF consisting of the order conditions from (B.3) and (B.4) given by

$$\begin{pmatrix} c_3(c_2-c_3) & 0 & c_4(c_2-c_4) & c_4(c_2-c_4) & c_5(c_2-c_5) \\ c_3^2(c_2-c_3) & 0 & c_4^2(c_2-c_4) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) \\ 0 & c_3(c_2-c_3) & c_5c_4(c_2-c_4) & c_6c_4(c_2-c_4) & c_5c_6(c_2-c_5) \\ p_3 & 0 & p_4 & p_4 & p_5 \\ 0 & p_3 & p_4c_5 & p_4c_6 & p_5c_6 \end{pmatrix} \begin{pmatrix} r_3^* \\ s_3^* \\ r_4^* \\ r_4^\dagger \\ r_5^* \end{pmatrix} = \begin{pmatrix} \frac{1}{6}c_2 - \frac{1}{12} \\ \frac{1}{12}c_2 - \frac{1}{20} \\ \frac{1}{8}c_2 - \frac{1}{15} \\ \frac{1}{24} \\ \frac{1}{30} \end{pmatrix}, \tag{3.13}$$

where row 1 is derived from subtracting (B.3a) from c_2 times (B.2a) to get an expression $c_2[[\tau]] - [[\tau^2]]$ equivalent to (B.3a), row 2 is derived from subtracting (B.4d) from c_2 times (B.3a) to get an expression $c_2[[\tau^2]] - [[\tau^3]]$ equivalent to (B.4d), row 3 is derived from subtracting (B.3b) from c_2 times (B.2b) to get an expression $c_2[\tau[\tau]] - [\tau[\tau^2]]$ equivalent to (B.3b), row 4 is (B.4a) $[[[\tau]]]$, and row 5 is (B.4b) $[\tau[[\tau]]]$.

It can now be observed from (3.13) that the "condensed" in CMF is likely named as such because subtracting order conditions and multiplying by c_2 to find equivalent order conditions eliminates certain columns in the resulting linear system (by eliminating an extra column on the left of (3.13) that would be present if this multiplying and subtracting with c_2 was not done); this is seen to be particularly important in the next section where $b_2 \neq 0$.

Cassity originally used the sum of the order conditions $[\tau[[\tau]]] + [[\tau[\tau^2]]] = \frac{7}{120}$ for row 5 when presenting the analogous system to (3.13) [31]. This is because many authors from the era Cassity published in often only derived the order conditions for RK methods when used to solve a single ODE rather than a system of ODEs (2.1a) [31, 102, 113]. This is understandable because working with only single ODEs gives expressions for elementary differentials that are much simpler to work with than daunting expressions such as (2.36) that arise from finding the elementary differentials for systems of ODEs. Analyzing single ODEs gives the correct order conditions for systems of ODEs up to order four [27, pgs.148–149]. However, for a single ODE, the fifth-order conditions $[\tau[[\tau]]]$ and $[[\tau[\tau^2]]]$ are just permutations of identical expressions that allows them to be treated as a single order condition as Cassity did in his 1966 paper [31], whereas for systems of ODEs they are distinct expressions [27, pgs.148–149]. However, when $c_6 = 1$, the order condition $[[\tau[\tau^2]]]$ is satisfied when $[\tau^2[\tau]]$ from row 7 of (3.5) is also satisfied, because it will be shown in the next section that the D(1) simplifying assumptions (2.44c) must hold for six-stage fifth-order ERK methods when $c_6 = 1$. Cassity also notes at the end of his 1969 paper [32], which only uses the order conditions suitable for systems of ODEs, that one of the six-stage fifth-order ERK methods that he derived with $c_6 \neq 1$ in his 1966 paper [31] is only valid for solving a single ODE rather than a system.

Even if only the two elementary differentials corresponding to $[\tau[[\tau]]]$ and $[[\tau[\tau^2]]]$ require considering systems of ODEs to give correct order conditions for fifth-order RK methods, higher-order elementary differentials are important for finding the error coefficients (2.47) and other characteristic numbers (2.56) required to construct efficient RK methods. The set of elementary differentials from a single ODE and systems of ODEs become increasingly divergent with higher order [27, pgs.148–149]. Therefore, analyzing elementary differentials for systems of ODEs is extremely important despite having to work with daunting expressions such as (2.36) if accurate values of higher-order error coefficients are to be found. In line with modern practice, in this thesis only elementary differentials and RK methods suitable for systems of ODEs are considered.

Because the $\mathbf{p}$ vector is already known from (3.5), the equations (3.13) can be solved for the variables defined by (3.12), i.e., $r_3^*, s_3^*, r_4^*, r_4^\dagger, r_5^*$. Excluding the first column of the $\mathbf{A}$ matrix of the Butcher tableau (2.34) that is solved afterwards by applying (2.35), the 10 remaining non-zero components of $\mathbf{A}$ can now be given by 9 equations derived by using the definition of the four non-zero components of the $\mathbf{p}$ vector (3.1) and the expressions for the special variable substitutions (3.13). The necessary values of the coefficients $a_{5,4}, a_{6,4}, a_{6,5}$ can be found immediately from solving the last three equations of (3.13). However, there are still two order conditions remaining, $[[[\tau^2]]]$ and $[[[[\tau]]]]$. The condition $[[[\tau^2]]]$ has an equivalent expression that is a linear equation in the still-unknown values of $\mathbf{A}$ when it is reformulated as the linear combination of $c_2[[[\tau]]] - [[[\tau^2]]]$ to give the expression

$$a_{4,3}\, c_3\, (c_2 - c_3)(b_5\, a_{5,4} + b_6\, a_{6,4}) + (a_{5,3}\, c_3\, (c_2 - c_3) + a_{5,4}\, c_4\, (c_2 - c_4))\, a_{6,5}\, b_6 = \frac{1}{24} c_2 - \frac{1}{60}. \tag{3.14}$$

This idea of reformulating order conditions as linear combinations with other conditions is used extensively in the next section for the complete formulation with $b_2 \neq 0$.

Using the SAGE CAS it can be found that both $c_2[[[\tau]]] - [[[\tau^2]]]$ and $[[[[\tau]]]]$ contain the factor $(c_6 - 1)$, therefore both conditions are immediately satisfied when $c_6 = 1$. However, without the benefit of a modern CAS, Cassity used linear combinations of $[[[[\tau]]]]$ (3.14) and other order conditions to rewrite $[[[[\tau]]]]$ as

$$(p_4\, c_3\,(c_2 - c_3) - p_3\, c_4\,(c_2 - c_4))\, b_6\, a_{6,5}\, a_{5,4} = \frac{1}{120} c_3\,(c_2 - c_3) - p_3 \left(\frac{1}{24} c_2 - \frac{1}{60}\right), \tag{3.15}$$

that is also a linear equation in the still unknown values of the $\mathbf{A}$ matrix. When $c_6 \neq 1$ but the C(2) simplifying assumptions (2.44b) hold, then (3.15) simplifies to

$$(10c_3^2\, c_4 - 8c_3\, c_4 - c_3 + 2c_4)(c_6 - 1) = 0, \tag{3.16}$$

where the left-hand factor is best chosen as a constraint on c_4 if $c_6 \neq 1$. However, for the case with $b_2 = 0$ where $c_6 \neq 1$ and the C(2) simplifying assumptions (2.44b) do not hold, there results a different expression than (3.16) that is examined shortly. Substituting the expression for c_4 (3.16) into (3.14) and solving the resulting expression gives a constraint on the $a_{6,3}$ coefficient. This constraint on $a_{6,3}$ for $c_6 \neq 1$ is exactly the constraint on $a_{6,3}$ for subsequently finding a $5(4)_{6(6)}$ ERK pair (the constraint for the $5(4)_{6(6)}$ ERK pair is regardless of whether $c_6 = 1$ or $c_6 \neq 1$), which is given below by (3.21), where it can also be seen that the constraint on c_4 required for a $5(4)_{6(6)}$ ERK pair regardless of the value of c_6 is identical to (3.16) when $c_6 \neq 1$.

Using linear combinations of the already-satisfied and still-unsatisfied order conditions to find equivalent relations that are linear in some still-unknown coefficients is an important idea to solving the highly non-linear system of order conditions as a sequence of linear systems. The author found that at least some of the simplification and rewriting done by Cassity to find relatively simple constraints to satisfy the order conditions continues to be necessary for the order conditions to be practically solvable, even when using a CAS. However, there are many potential variations on the specific formulation of the expressions that he found that would work equally well. When the expressions for the order conditions are used directly without simplifications, even when using a CAS, simple but necessary conditions such as (3.16) are rarely obvious. Otherwise, the output of the CAS usually remains as high-degree multi-variate systems of polynomial equations rather than giving explicit expressions for the Butcher tableau coefficients. Despite the now-straightforward appearance of the solution procedure, even given Cassity's papers [31, 32] to work from, finding the sequence of steps and expressions used to completely solve the six-stage fifth-order order conditions in this thesis was the result of an iterative research process where the first attempts often ended with seemly unsolvable high-degree multi-variate polynomials. Complete solutions of more complicated systems of order conditions than studied in this thesis (for instance, for ERK order conditions higher than fifth-order) should also be possible using CMF and CASs. However, the complexity of these higher-order order conditions will make insights analogous to those Cassity had leading to the explicit solution of order conditions more important than ever.

The special case of $b_2 = 0$ and $c_6 \neq 1$ without the C(2) simplifying assumptions

When $b_2 = 0$ and $c_6 \neq 1$ without the C(2) simplifying assumptions (2.44b), the condition (3.16) can be simplified to

$$\begin{aligned}-(&680c_2^2c_3^2c_4 - 120c_2^2c_3^2 - 480c_2^2c_3c_4 - 480c_2c_3^2c_4 + 30c_2^2c_3 + 30c_2c_3^2 + 90c_2^2c_4 + 348c_2c_3c_4 + 90c_3^2c_4 + \\ &36c_2c_3 - 72c_2c_4 - 72c_3c_4 - 9c_2 - 9c_3 + 18c_4)(c_2 - c_3)c_2(c_6 - 1) = 0, \end{aligned} \tag{3.17}$$

where the first factor can be solved to give a constraint on c_4. This constraint on c_4 can then be back-substituted into an expression for $a_{6,3}$ found from (3.14) that is different from the expression (3.21) when the C(2) simplifying assumptions (2.44b) are satisfied. It can be noted at this point that Cassity's original work [31] introduced the condition (3.11) as a second case for $b_2 = 0$ for when the C(2) simplifying assumptions (2.44b) are not satisfied, but did not give (3.17) as one of the constraints or indicate that a separate constraint was necessary.

3.3.3 Solving for the Butcher tableau A matrix

With the now-known values of $\tilde{\mathbf{p}}$ from finding the consistency conditions for (3.5), the expressions (3.12), the condition (3.14), and using $a_{6,3}$ as a free parameter, the other 9 non-zero values of the $\mathbf{A}$ matrix outside the first column can be found from the linear systems

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 & 0 \\ 0 & 0 & b_5 & 0 & 0 \\ 0 & 0 & 0 & b_6 & 0 \\ 0 & 0 & 0 & 0 & b_6 \end{pmatrix} \begin{pmatrix} a_{2,1} \\ a_{3,2} \\ a_{5,4} \\ a_{6,4} \\ a_{6,5} \end{pmatrix} = \begin{pmatrix} c_2 \\ p_3 \\ r_4^* \\ r_4^\dagger \\ r_5^* \end{pmatrix},$$

$$\left(\begin{array}{cccccc} c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_2 & c_3 \\ \hline 0 & b_4 & 0 & b_5 & 0 & b_6 \\ 0 & b_4c_4 & 0 & b_5c_5 & 0 & b_6c_6 \end{array}\right) \begin{pmatrix} a_{4,2} \\ a_{4,3} \\ a_{5,2} \\ a_{5,3} \\ a_{6,2} \\ a_{6,3} \end{pmatrix} = \left(\begin{array}{c} p_4 \\ p_5 - a_{5,4}c_4 \\ p_6 - a_{6,5}c_5 - a_{6,4}c_4 \\ \hline r_3^* \\ s_3^* \end{array}\right).$$

Finally, the first column is found from the row-sum conditions (2.35) given by

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{3,1} \\ a_{4,1} \\ a_{5,1} \\ a_{6,1} \end{pmatrix} = \begin{pmatrix} c_3 - a_{3,2} \\ c_4 - a_{4,2} - a_{4,3} \\ c_5 - a_{5,2} - a_{5,3} - a_{5,4} \\ c_6 - a_{6,2} - a_{6,3} - a_{6,4} - a_{6,5} \end{pmatrix},$$

because all the $a_{i,j}$ quantities outside of the first column of the $\mathbf{A}$ matrix are now known.

Table 3.1: Cases and free parameters for the restricted solution for six-stage fifth-order ERK methods.

Case	Free parameters	Subcases	Example
$c_6 = 1, b_2 = 0$ with C(2)	$a_{6,3}$, c_2, c_3, c_4, c_5	1	(A.1)
$c_6 \neq 1, b_2 = 0$ with C(2)	c_2, c_3, c_5, c_6	1	(A.2)
$c_6 = 1, b_2 = 0$ w/o C(2)	$a_{6,3}$, c_2, c_3, c_4, c_5	1	(A.3)
$c_6 \neq 1, b_2 = 0$ w/o C(2)	c_2, c_3, c_5, c_6	1	(A.4)

3.3.4 Constructing the $5(4)_{6(6)\mathrm{C}(2)}$ family

The material in this subsection is part of the new work presented in this thesis and not found in Cassity's original work [31, 32] or elsewhere in the literature.

The classic ERK pairs that are included in the more general family discussed in this section, e.g., the RKF4(5)$_{6(6)}$ method (2.78) from Section 2.7.2 and the RK5(4)6M method (2.82) described by Dormand and Prince [47] in the same paper as DP5(4)$_{6(7)}$ (2.79), all use the C(3) simplifying assumptions (2.44b) in addition to the C(2) simplifying assumptions (2.44b). This is because neither the C(2) or C(3) simplifying assumptions (2.44b) involve the $\mathbf{b}$ and $\hat{\mathbf{b}}$ vectors but in combination are sufficient to simplify the order conditions enough to be readily solved. As already discussed in Section 2.7, simplifying assumptions that involve both the $\mathbf{b}$ and $\hat{\mathbf{b}}$ vectors would greatly complicate the classic approach to solving order conditions. However, by using CMF in combination with a CAS it is possible to find the complete solution for $5(4)_6$ ERK pairs.

Beginning with the fourth-order quadrature conditions and non-branching order conditions of height two, the fourth-order method of a $5(4)_{6(6)}$ ERK pair only requires the order conditions corresponding to rows 1–4 and rows 6,7 of (3.5). The CMF using the order conditions up to fourth-order of τ, $[\tau]$, $[\tau^2]$, $[\tau^3]$, $[[\tau]]$, $[\tau[\tau]]$

is given by

$$\left(\begin{array}{c|cccc} 1 & 1 & 1 & 1 & 1 \\ \hline 0 & c_3 & c_4 & c_5 & c_6 \\ 0 & c_3^2 & c_4^2 & c_5^2 & c_6^2 \\ 0 & c_3^3 & c_4^3 & c_5^3 & c_6^3 \\ \hline 0 & p_3 & p_4 & p_5 & p_6 \\ 0 & c_3 p_3 & c_4 p_4 & c_5 p_5 & c_6 p_6 \end{array}\right) \left(\begin{array}{c} \hat{b}_1 \\ \hline \hat{b}_3 \\ \hat{b}_4 \\ \hat{b}_5 \\ \hat{b}_6 \end{array}\right) = \left(\begin{array}{c} 1 \\ \hline \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ \hline \frac{1}{6} \\ \frac{1}{8} \end{array}\right), \tag{3.18}$$

where the $\hat{\mathbf{b}}$ vector must be distinct from the $\mathbf{b}$ vector for an effective embedded pair. It is easily seen that the last two rows of (3.18) are automatically satisfied when the C(2) simplifying assumptions (2.44b) hold and that this results in the LHS coefficient matrix of (3.18) being rank 4 with 5 unknowns. Therefore, one of the components of the $\hat{\mathbf{b}}$ vector is a free parameter, arbitrarily chosen as $\hat{b}_6$ in this subsection.

After satisfying the order conditions contained in (3.18), there are only two order conditions remaining, i.e., $[[\tau^2]]$ and $[[[\tau]]]$, and subtracting similarly to (3.13) to give equivalent conditions, i.e., $c_2[[\tau]] - [[\tau^2]]$ and $2[[[\tau]]]$, gives a CMF of

$$\begin{pmatrix} (c_2 - c_3)c_3 & (c_2 - c_4)c_4 & (c_2 - c_5)c_5 \\ 2\,p_3 & 2\,p_4 & 2\,p_5 \end{pmatrix} \begin{pmatrix} \hat{r}_3 \\ \hat{r}_4 \\ \hat{r}_5 \end{pmatrix} = \begin{pmatrix} \frac{1}{6}\,c_2 - \frac{1}{12} \\ \frac{1}{12} \end{pmatrix}, \tag{3.19a}$$

where

$$\left\{\begin{array}{lll} \hat{b}_4\,a_{4,3} & +\hat{b}_5\,a_{5,3} & +\hat{b}_6\,a_{6,3} \\ & +\hat{b}_5\,a_{5,4} & +\hat{b}_6\,a_{6,4} \\ & & +\hat{b}_6\,a_{6,5} \end{array}\right\} = \hat{\mathbf{r}}_i, \quad i \in \{3,4,5\}, \tag{3.19b}$$

that are a set of variable substitutions introduced fully in the next section but that are also useful for constructing $5(4)_{6(6)}$ ERK pairs in the context of the restricted system with $b_2 = 0$.

All quantities in the definition of the $\hat{\mathbf{r}}_i$ vector (3.19b) are known values from the $\mathbf{A}$ matrix and the solution to (3.18). Therefore, it is possible to multiply out the equations for (3.19a) (working with the resulting equations significantly benefits from a CAS and are too large to include here) to get two conditions that are linear in the free parameters $a_{6,3}$ (a free parameter of the fifth-order method) and $\hat{b}_6$. However, it can be seen that the two equations represented by (3.19a) are nearly identical except for substitution of corresponding terms in the RHS coefficient matrix, i.e., swapping $(c_2 - c_i)c_i$ for $2p_i$, and the addition of one term, i.e., $\frac{1}{6}c_2$, for the LHS of the first equation. Due to this similarity between the two conditions and that the terms in the $\hat{\mathbf{r}}$ vector add the majority of the "complexity", many terms cancel out when simultaneously satisfying both equations. If the second equation of (3.19b) is solved for $a_{6,3}$ and this expression for $a_{6,3}$ is substituted into the first equation, after simplification using SAGE [57], the resulting condition to satisfy the first equation of (3.19a) is given by

$$\begin{aligned}
&\Big(60\hat{b}_6c_3c_4c_5c_6 - 60\hat{b}_6c_3c_4c_6^2 - 60\hat{b}_6c_3c_5c_6^2 - 60\hat{b}_6c_4c_5c_6^2 + 60\hat{b}_6c_3c_6^3 + 60\hat{b}_6c_4c_6^3 + 60\hat{b}_6c_5c_6^3 - \\
&60\hat{b}_6c_6^4 - 30c_3c_4c_5 + 20c_3c_4 + 20c_3c_5 + 20c_4c_5 - 15c_3 - 15c_4 - 15c_5 + 12\Big) && \text{(3.20a)}\\
&\left(10\,c_3^2c_4 - 8\,c_3c_4 - c_3 + 2\,c_4\right) && \text{(3.20b)}\\
&(30c_3c_5c_6 - 20c_3c_5 - 20c_3c_6 - 20c_5c_6 + 15c_3 + 15c_5 + 15c_6 - 12) && \text{(3.20c)}\\
&c_2(c_3 - c_4)(c_3 - c_5)(c_3 - c_6)c_6 && \text{(3.20d)}\\
&= 0,
\end{aligned}$$

where this is a single equation with the factors grouped for clarity and at least one that must vanish. Examining the factors of the constraint (3.20):

- The term (3.20a) vanishing implies $\hat{b}_6 = b_6$; hence this violates the requirement the components of an embedded pair be distinct for an effective error estimate. However, this is the only term of (3.20) that contains $\hat{b}_6$, and if the other factors are the ones that vanish, this allows $\hat{b}_6$ to be a free parameter for the fourth-order method of the embedded pair.

- The term (3.20b) vanishing is the condition already known by the first factor of (3.16) and actually constitutes a necessary condition for a $5(4)_{6(6)}$ ERK pair when $b_2 = 0$ (even when $c_6 = 1$, unlike the first factor of (3.16) that is only required for the fifth-order method when $c_6 \neq 1$).

- The term (3.20c) cannot vanish because the expression appears in the denominator of many other expressions for the coefficients.

- The terms (3.20d) cannot vanish because that would imply that some components of the **c** vector are equal. However, these particular conditions cannot hold because Konen and Luther [102] have shown these particular **c** vector components cannot be equal in a six-stage fifth-order ERK method. Also note that c_2 or c_6 cannot vanish because this would imply an ERK method with only five stages.

After the expression (3.20) is satisfied (by the factor (3.20b)) the constraint on c_4 can be substituted into the expression for $a_{6,3}$ found from the second equation of (3.19a) to provide a second restriction on $a_{6,3}$ that gives the necessary conditions for an embedded pair. These two additional restrictions to free parameters of six-stage fifth-order ERK method with $b_2 \neq 0$ are given explicitly as

$$
\begin{aligned}
c_4 =& \frac{\frac{1}{2}c_3}{(5c_3^2 - 4c_3 + 1)}, \\
a_{6,3} =& \frac{N(c_3 - c_6)c_6}{2(10c_3^2 - 8c_3 + 1)(10c_3^2 - 12c_3 + 3)(20c_3c_5 - 15c_3 - 10c_5 + 8)(c_3 - c_5)c_3^2}, \\
N =& \big(600c_3^5c_5c_6 - 1600c_3^4c_5^2c_6 + 1000c_3^5c_6^2 - 400c_3^5c_5 + 400c_3^4c_5^2 - 1200c_3^5c_6 + 920c_3^4c_5c_6 + 2680c_3^3c_5^2c_6 \\
& -2400c_3^4c_6^2 + 300c_3^5 + 320c_3^4c_5 - 520c_3^3c_5^2 + 1950c_3^4c_6 - 2730c_3^3c_5c_6 - 1580c_3^2c_5^2c_6 + 2260c_3^3c_6^2 \\
& -400c_3^4 + 60c_3^3c_5 + 180c_3^2c_5^2 - 1072c_3^3c_6 + 1896c_3^2c_5c_6 + 400c_3c_5^2c_6 - 1064c_3^2c_6^2 + 143c_3^3 - 55c_3^2c_5 \\
& -10c_3c_5^2 + 238c_3^2c_6 - 528c_3c_5c_6 - 40c_5^2c_6 + 252c_3c_6^2 - 8c_3^2 - 4c_3c_5 - 20c_3c_6 + 56c_5c_6 - 24c_6^2\big),
\end{aligned}
\tag{3.21}
$$

where the expression for $a_{6,3}$ is also greatly simplified in comparison to the expanded second equation of (3.19a) because the factor (3.20a) involving $\hat{b}_6$ that would imply $\hat{\mathbf{b}} = \mathbf{b}$ is eliminated. By substituting in the appropriate value for $c_3 = \frac{3}{2}c_2$ to accommodate the C(3) simplifying assumptions (2.44b) and using the appropriate value for the free parameter $\hat{b}_6$, the appropriate expressions in this section in fact recover the coefficients of both the RKF4(5)$_{6(6)}$ method (2.78) and the RK5(4)6M method (2.82).

Therefore, it has been shown that the classic condition given by (3.16) is a necessary condition to construct a 5(4)$_{6(6)}$ ERK pair with the C(2) simplifying assumptions (2.44b). However, by using CMF it is not necessary to incorporate the C(3) simplifying assumptions (2.44b), and it is possible to solve the order conditions in a less restricted way than when the original family containing the RKF4(5)$_{6(6)}$ method (2.78) was constructed. This family just constructed is referred to as the *5(4)$_{6(6)\mathrm{C}(2)}$ family*. The free parameters in comparison to the classic RKF4(5)$_{6(6)}$ method (2.78) are given by the following tableaux:

F = RKF4(5)$_{6(6)}$ (2.78) free parameters (uses C(2) and C(3) simplifying assumptions (2.44b))

C = Additional free parameters for the complete solution with only the C(2) simplifying assumptions (2.44b)

X = Computed parameter

$$
\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F & X & 0 & 0 & 0 & 0 & 0 \\
C & X & X & 0 & 0 & 0 & 0 \\
X & X & X & X & 0 & 0 & 0 \\
F & X & X & X & X & 0 & 0 \\
F & X & X & X & X & X & 0 \\
\hline
 & X & 0 & X & X & X & X \\
\hline
 & X & 0 & X & X & X & C
\end{array}
$$

The implications of the additional free parameters are discussed when performance testing in the searches in Chapter 4 and Chapter 5. An example of a Butcher tableau without the C(3) simplifying assumptions (2.44b) satisfied, in order to contrast with the RKF4(5)$_{6(6)}$ method (2.78) is given by (A.5), where it can easily be seen that $c_3 \neq \frac{3}{2}c_2$, implying that the C(3) simplifying assumptions (2.44b) do not hold; see (2.71).

3.3.5 Constructing the $5(4)_{6(7)\mathrm{C}(2)}$ family

The material in this subsection is part of the new work presented in this thesis and not found in Cassity's original work [31] or elsewhere in the literature.

The classic ERK pairs that are included in the more general family discussed in this section are the family found by Dormand and Prince discussed in Section 2.7.3 that contains DP5(4)$_{6(7)}$ [47] and the family found by Papakostas and Papageorgiou [128] containing the PP5(4)$_{6(7)}$ pair (2.84). The family found by Papakostas and Papageorgiou [128] does not use the C(3) simplifying assumptions (2.44b). However, it has one less free parameter than the family found in this subsection. This indicates that an additional advantage of using CMF is also that it allows the maximum number of free parameters, and not just eliminating the dependence on simplifying assumptions for construction.

The construction of the general family of $5(4)_{6(7)}$ ERK pairs with the C(2) simplifying assumptions (2.44b) follows a similar procedure to that just used to construct the family of $5(4)_{6(6)}$ ERK pairs in the previous subsection. The CMF for the quadrature conditions and non-branching conditions of height two up to fourth order τ, $[\tau]$, $[\tau^2]$, $[\tau^3]$, $[[\tau]]$, $[\tau[\tau]]$ is given by

$$\left(\begin{array}{c|ccccc} 1 & 1 & 1 & 1 & 1 & 1 \\ \hline 0 & c_3 & c_4 & c_5 & c_6 & c_7 \\ 0 & c_3^2 & c_4^2 & c_5^2 & c_6^2 & c_7^2 \\ 0 & c_3^3 & c_4^3 & c_5^3 & c_6^3 & c_7^3 \\ \hline 0 & p_3 & p_4 & p_5 & p_6 & p_7 \\ 0 & c_3p_3 & c_4p_4 & c_5p_5 & c_6p_6 & c_7p_7 \end{array}\right) \begin{pmatrix} \hat{b}_1 \\ \hat{b}_3 \\ \hat{b}_4 \\ \hat{b}_5 \\ \hat{b}_6 \\ \hat{b}_7 \end{pmatrix} = \left(\begin{array}{c} 1 \\ \hline \frac{1}{2} \\ \frac{1}{3} \\ \frac{1}{4} \\ \hline \frac{1}{6} \\ \frac{1}{8} \end{array}\right), \tag{3.22}$$

where $c_7 = 1$ and $p_7 = \sum b_i c_i = \frac{1}{2}$ because the definition of p_7 (3.1) for the seventh row of the $\mathbf{A}$ matrix for a $5(4)_{6(7)}$ ERK pair corresponds to the $[\tau]$ order condition. The last two equations of (3.22) are automatically satisfied because of the C(2) simplifying assumptions (2.44b), and this leaves a rank-4 linear system with 6 equations and 6 unknowns. Therefore, both $\hat{b}_6$ and $\hat{b}_7$ can be free parameters when only satisfying the system (3.22).

Following the same procedure from Section 3.3.4, after satisfying the equations from (3.22), there are only two order conditions left, i.e., $[[\tau^2]]$ and $[[[\tau]]]$, and subtracting similarly to (3.13) to give equivalent conditions, i.e., $c_2[[\tau]] - [[\tau^2]]$ and $2[[[\tau]]]$, gives a CMF of

$$\begin{pmatrix} (c_2-c_3)c_3 & (c_2-c_4)c_4 & (c_2-c_5)c_5 & (c_2-c_6)c_6 \\ 2\,p_3 & 2\,p_4 & 2\,p_5 & 2\,p_6 \end{pmatrix} \begin{pmatrix} \hat{r}_3 \\ \hat{r}_4 \\ \hat{r}_5 \\ \hat{r}_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{6}\,c_2 - \frac{1}{12} \\ \frac{1}{12} \end{pmatrix}, \tag{3.23a}$$

with

$$\left\{\begin{matrix} \hat{b}_4\,a_{4,3} & +\hat{b}_5\,a_{5,3} & +\hat{b}_6\,a_{6,3} & +\hat{b}_7\,a_{7,3} \\ & +\hat{b}_5\,a_{5,4} & +\hat{b}_6\,a_{6,4} & +\hat{b}_7\,a_{7,4} \\ & & +\hat{b}_6\,a_{6,5} & +\hat{b}_7\,a_{7,5} \\ & & & +\hat{b}_7\,a_{7,6} \end{matrix}\right\} = \hat{r}_i, \quad i \in \{3,4,5,6\}, \tag{3.23b}$$

where all coefficients in the definition of the $\hat{\mathbf{r}}_i$ vector are known expressions. Therefore, it is possible to multiply out the equations for (3.23) (the resulting expressions are too large to include here) to get two equations that must vanish and are linear in the free parameter $a_{6,3}$ (a free parameter of the fifth-order method) and containing terms involving $\hat{b}_6$ and $\hat{b}_7$. Following similar reasoning discussed for the analogous case in Section 3.3.4, if the second equation of (3.23) is solved for $\hat{b}_6$ and this expression for $\hat{b}_6$ is substituted into the first equation, after simplification using SAGE [57], the resulting condition to satisfy this equation from the first row of (3.23) is given by

$$\begin{aligned}
&\big(60\,a_{6,3}c_3^5c_4c_5 - 60\,a_{6,3}c_3^4c_4^2c_5 - 60\,a_{6,3}c_3^4c_4c_5^2 + 60\,a_{6,3}c_3^3c_4^2c_5^2 - 40\,a_{6,3}c_3^5c_4 + 40\,a_{6,3}c_3^4c_4^2 - 40\,a_{6,3}c_3^5c_5 \\
&+40\,a_{6,3}c_3^4c_4c_5 + 40\,a_{6,3}c_3^4c_5^2 - 40\,a_{6,3}c_3^2c_4^2c_5^2 + 30\,a_{6,3}c_3^5 - 30\,a_{6,3}c_3^3c_4^2 - 30\,a_{6,3}c_3^3c_4c_5 + 20\,c_3^4c_4c_5 \\
&+30\,a_{6,3}c_3^2c_4^2c_5 - 20\,c_3^3c_4^2c_5 - 30\,a_{6,3}c_3^3c_5^2 + 30\,a_{6,3}c_3^2c_4c_5^2 - 20\,c_3^3c_4c_5^2 - 24\,a_{6,3}c_3^4 + 24\,a_{6,3}c_3^3c_4 - 10\,c_3^4c_4 \\
&+15\,c_3^3c_4^2 + 24\,a_{6,3}c_3^3c_5 - 24\,a_{6,3}c_3^2c_4c_5 - 20\,c_3^3c_4c_5 + 45\,c_3^2c_4^2c_5 + 60\,c_3^2c_4c_5^2 - 10\,c_3c_4^2c_5^2 + 9\,c_3^3c_4 - 31\,c_3^2c_4^2 \\
&-10\,c_3^2c_5^2 - 40\,c_3c_4c_5^2 + 10\,c_4^2c_5^2 + 2\,c_3^3 + 17\,c_3^2c_4 + 16\,c_3c_4^2 + 14\,c_3^2c_5 + 45\,c_3c_4c_5 - 6\,c_4^2c_5 + 10\,c_3c_5^2 - 8\,c_3^2 \\
&-45\,c_3^2c_4c_5 - 19\,c_3c_4^2c_5 - 16\,c_3c_4 - 14\,c_3c_5 + 6\,c_3\big) && (3.24a) \\
&(30\,c_3c_4c_5 - 20\,c_3c_4 - 20\,c_3c_5 - 20\,c_4c_5 + 15\,c_3 + 15\,c_4 + 15\,c_5 - 12) && (3.24b) \\
&(10\,c_3c_4 - 5\,c_3 - 5\,c_4 + 3) && (3.24c) \\
&(10\,c_3c_5 - 5\,c_3 - 5\,c_5 + 3) && (3.24d) \\
&\hat{b}_7 && (3.24e) \\
&c_2(c_3-c_4)(c_3-c_5)(c_3-1)c_3 && (3.24f) \\
&= 0,
\end{aligned}$$

where similar to the constraint (3.20) described for $5(4)_{6(6)}$ ERK pair from Section 3.3.4, the factors are grouped for clarity and at least one factor must vanish. Examining the factors of the constraint (3.24):

- The term (3.24a) vanishing allows solving for a constraint on the free parameter $a_{6,3}$, which only appears linearly in (3.24). This is the only term of (3.24) that can vanish and is a necessary condition.
- The terms (3.24b), (3.24c), and (3.24d) appear in the denominator of some expressions when the components of the **c** vector are distinct and therefore cannot vanish.
- The term (3.24e) cannot vanish because $\hat{b}_7 = 0$ would imply the two methods of the embedded pair are not distinct, or at least that the fourth-order method has six stages rather than seven. This case of a fourth-order method with six stages was already covered above in Section 3.3.4.
- The term (3.24f) cannot vanish because this would imply that some components of the **c** vector are equal. Note also that c_2 cannot vanish because this would imply the fifth-order ERK method has only five effective stages.

The free parameters in comparison to the classic DP5(4)$_{6(7)}$ method (2.79) are given by the following tableaux:
D = Free parameters for DP5(4)$_{6(7)}$ (2.79) (uses the C(2) and C(3) simplifying assumptions (2.44b))
C = Additional free parameters for the complete solution with only the C(2) simplifying assumptions (2.44b)
X = Computed parameter

$$\begin{array}{c|ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
D & X & 0 & 0 & 0 & 0 & 0 & 0 \\
C & X & X & 0 & 0 & 0 & 0 & 0 \\
D & X & X & X & 0 & 0 & 0 & 0 \\
D & X & X & X & X & 0 & 0 & 0 \\
X & X & X & X & X & X & 0 & 0 \\
X & X & 0 & X & X & X & X & 0 \\
\hline
 & X & 0 & X & X & X & X & 0 \\
\hline
 & X & 0 & X & X & X & X & D
\end{array}$$

$$\begin{aligned}
a_{6,3} &= -\frac{N_1(c_3-1)}{2(30c_3c_4c_5 - 20c_3c_4 - 20c_3c_5 - 20c_4c_5 + 15c_3 + 15c_4 + 15c_5 - 12)(c_3-c_4)(c_3-c_5)c_3^2}, \\
N_1 &= \big(20c_3^3c_4c_5 - 20c_3^2c_4^2c_5 - 20c_3^2c_4c_5^2 - 10c_3^3c_4 + 15c_3^2c_4^2 + 25c_3c_4^2c_5 + 40c_3c_4c_5^2 - 10c_4^2c_5^2 \\
&\quad - c_3^2c_4 - 16c_3c_4^2 - 45c_3c_4c_5 + 6c_4^2c_5 - 10c_3c_5^2 + 2c_3^2 + 16c_3c_4 + 14c_3c_5 - 6c_3\big), \\
\hat{b}_6 &= -\frac{N_2}{60(10c_3^2c_4 - 8c_3c_4 - c_3 + 2c_4)(c_3-1)(c_4-1)(c_5-1)}, \\
N_2 &= \Big(60\hat{b}_7c_3^2c_4 - 56\hat{b}_7c_3c_4 - 10c_3^2c_4 - 8\hat{b}_7c_3 + 16\hat{b}_7c_4 + 8c_3c_4 + c_3 - 2c_4\Big) \\
&\quad (30c_3c_4c_5 - 20c_3c_4 - 20c_3c_5 - 20c_4c_5 + 15c_3 + 15c_4 + 15c_5 - 12),
\end{aligned}$$

It can be seen that using the CMF allows an additional free parameter over the family used to originally construct DP5(4)$_{6(7)}$ (2.79). This family just constructed is referred to as the *5(4)$_{6(7)\mathrm{C}(2)}$ family*. An example of a Butcher tableau without the C(3) simplifying assumptions (2.44b) satisfied, that can be contrasted with DP5(4)$_{6(7)}$ (2.79) is given by (A.7), where it can easily be seen that $c_3 \neq \frac{3}{2}c_2$, implying that the C(3) simplifying assumptions (2.44b) do not hold.

3.3.6 Lack of embedded pairs without C(2) simplifying assumptions and $b_2 = 0$

Six-stage fifth-order ERK methods with $b_2 = 0$ and without the C(2) simplifying assumptions (2.44b), i.e., the expression (3.11) holds, do not allow 5(4)$_{6(6)}$ ERK pairs or 5(4)$_{6(7)}$ ERK pairs. This is because when $b_2 = 0$ and the C(2) simplifying assumptions (2.44b) do not hold, the only possible way to satisfy the last two conditions of (3.18) and (3.22) is with the solutions $\hat{b}_6 = b_6$ and $\hat{b}_6 = b_6$ with $\hat{b}_7 = 0$, respectively. In Section 3.5 below, it is seen that allowing $b_2 \neq 0$ gives enough freedom to allow 5(4)$_{6(6)}$ ERK pairs and 5(4)$_{6(7)}$ ERK pairs without the C(2) simplifying assumptions (2.44b).

Table 3.2: Cases and free parameters for the restricted solution for six-stage fifth-order ERK embedded pairs.

Case	6(6) free params	Subcases	6(7) free params	Subcases	Example
$c_6 = 1, b_2 = 0$ with C(2)	$c_2, c_3, c_5, \hat{b}_6$	1	$c_2, c_3, c_4, c_5, \hat{b}_7$	1	(A.5) (A.7)
$c_6 \neq 1, b_2 = 0$ with C(2)	$c_2, c_3, c_5, c_6, \hat{b}_6$	1	No	N/A	(A.6)

3.4 Complete solution of six-stage fifth-order ERK order conditions

The complete solution for the six-stage fifth-order ERK conditions described in this section follows as closely as possible the solution described by Cassity in 1969 [32], except for changes to reflect contemporary notation, reformulations advantageous for using a CAS and to support implementations in software, and for additions to the mathematics already mentioned that are noted where they occur. Cassity's paper [32] has few details on how the constraints he gives are found, especially the relatively complicated expressions required as constraints for the quadrature conditions and non-branching conditions of height two (3.27a) that are described fully in Section 3.4.1. Also important is that Cassity's paper [32] does not give the complete solution in an explicit form suitable for computer implementation, whereas this section and the supporting material for this thesis do. Cassity [32] does indicate that finding the singular solutions that would be necessary for a general solution would be straightforward. However, as already mentioned, the additional singularities in the linear systems to be solved are not covered in this thesis.

A summary of the constructed families with $b_2 \neq 0$ and free parameters are given in Table 3.3 for single ERK methods. For comparison purposes, the families with $b_2 = 0$ constructed in Section 3.3 are also summarized in this table.

Several special variable substitutions, already used when defining (3.19b) and (3.23b) in Section 3.3, based on the general ones (3.1) are given by

$$\left\{\begin{array}{lll} b_4\, a_{4,3} & +b_5\, a_{5,3} & +b_6\, a_{6,3} \\ & +b_5\, a_{5,4} & +b_6\, a_{6,4} \\ & & +b_6\, a_{6,5} \end{array}\right\} = \tilde{\mathbf{r}}_i, \quad i \in \{3,4,5\} = \tilde{\mathbf{r}}, \tag{3.25a}$$

and

$$\left\{\begin{array}{lll} b_4\, c_4\, a_{4,3} & +b_5\, c_5\, a_{5,3} & +b_6\, c_6\, a_{6,3} \\ & +b_5\, c_5\, a_{5,4} & +b_6\, c_6\, a_{6,4} \\ & & +b_6\, c_6\, a_{6,5} \end{array}\right\} = \tilde{\mathbf{s}}_i, \quad i \in \{3,4,5\} = \tilde{\mathbf{s}}. \tag{3.25b}$$

A six-stage fifth-order ERK method has order conditions for orders 1 and 2, i.e., $\tau, [\tau]$, that are given by

$$\sum_{1}^{6} b_i = 1, \tag{3.26a}$$

$$\sum_{i=2}^{6} b_i\, c_i = \frac{1}{2}, \tag{3.26b}$$

which can be largely ignored while analyzing the complete solution, analogous to the situation with the first row of (3.3) for the restricted system with $b_2 = 0$ from Section 3.3.

The linear system for $\tilde{\mathbf{b}}$ that describes conditions equivalent to the quadrature conditions (B.1) and non-branching conditions of height two (B.2), i.e., $[\tau^2], [\tau^3], [\tau^4], [[\tau]], [\tau[\tau]], [\tau^2[\tau]], [[\tau]^2]$, is given in CMF by the linear system

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & c_6(c_2-c_6) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & c_6^2(c_2-c_6) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & c_6^3(c_2-c_6) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \\ 2c_3p_3 & 2c_4p_4 & 2c_5p_5 & 2c_6p_6 \\ 2c_3^2p_3 & 2c_4^2p_4 & 2c_5^2p_5 & 2c_6^2p_6 \\ \hline 4p_3^2 & 4p_4^2 & 4p_5^2 & 4p_6^2 \end{array}\right) \tilde{\mathbf{b}} = \left(\begin{array}{c} \frac{1}{2}\, c_2 - \frac{1}{3} \\ \frac{1}{3}\, c_2 - \frac{1}{4} \\ \frac{1}{4}\, c_2 - \frac{1}{5} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \\ \frac{1}{5} \end{array}\right), \tag{3.27a}$$

where similar to previously seen equations such as (3.13), (3.19), and (3.23); row 1 is $c_2[\tau] - [\tau^2]$, row 2 is $c_2[\tau^2] - [\tau^3]$, row 3 is $c_2[\tau^3] - [\tau^4]$, row 4 is $2[[\tau]]$, row 5 is $2[\tau[\tau]]$, row 6 is $2[\tau^2[\tau]]$, and row 7 is $4[[\tau]^2]$.

The order conditions $[[\tau^2]]$, $[[\tau^3]]$, $[[[\tau]]]$, $[[\tau[\tau^2]]]$ are constructed by adding a leg to create a one-legged order condition from two quadrature conditions and two non-branching conditions of height two that are rows 1, 2, 4, 5 of (3.27a). In the context of CMF this is done by substituting $\tilde{\mathbf{r}}$ for $\tilde{\mathbf{b}}$ in order to give the linear system

$$\left(\begin{array}{ccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) \\ \hline 2p_3 & 2p_4 & 2p_5 \\ 2c_3p_3 & 2c_4p_4 & 2c_5p_5 \end{array}\right) \tilde{\mathbf{r}} = \begin{pmatrix} \frac{1}{6}\,c_2 - \frac{1}{12} \\ \frac{1}{12}\,c_2 - \frac{1}{20} \\ \frac{1}{12} \\ \frac{1}{20} \end{pmatrix}, \tag{3.27b}$$

where row 1 corresponds to $c_2[[\tau]] - [[\tau^2]]$, row 2 corresponds to $c_2[[\tau^2]] - [[\tau^3]]$, rows 3 corresponds to $2[[[\tau]]]$, and row 4 corresponds to $2[[\tau[\tau^2]]]$. The CMF for the order conditions $[\tau[\tau^2]]$ and $[\tau[[\tau]]]$ is given by the linear system

$$\left(\begin{array}{ccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) \\ \hline 2p_3 & 2p_4 & 2p_5 \end{array}\right) \tilde{\mathbf{s}} = \frac{1}{2}\begin{pmatrix} \frac{1}{4}c_2 - \frac{2}{15} \\ \frac{2}{15} \end{pmatrix}, \tag{3.27c}$$

which corresponds to additional order conditions where row 1 of (3.27c) is $c_2[\tau[\tau]] - [\tau[\tau^2]]$ and row 2 is $2[\tau[[\tau]]]$.

The final order conditions, i.e., $[[[\tau^2]]]$ and $[[[[\tau]]]]$, are given by the system

$$\left(\begin{array}{cc} c_3(c_2-c_3) & c_4(c_2-c_4) \\ \hline 2p_3 & 2p_4 \end{array}\right) \begin{pmatrix} r_4\,a_{4,3} & +r_5\,a_{5,3} \\ & +r_5\,a_{5,4} \end{pmatrix} = \frac{1}{2}\begin{pmatrix} \frac{1}{12}\,c_2 - \frac{1}{30} \\ \frac{1}{30} \end{pmatrix}, \tag{3.27d}$$

where row 1 of (3.27d) is $c_2[[[\tau]]] - [[[\tau^2]]]$ and row 2 is $2[[[[\tau]]]]$.

The particular partitioning of the order conditions into CMF by Cassity [32] given by (3.27) makes a well-defined solution procedure possible for the complete solution of the order conditions (B.1)–(B.5) for six-stage fifth-order ERK. In Cassity's first attempt from 1966 [31], which is already discussed in Section 3.1, he found that a similar procedure to that given for the restricted system in Section 3.3 could be used. However, as already discussed in Section 3.1 it does not easily lead to explicit expressions for ERK method construction and is not discussed further in this thesis.

3.4.1 Solving quadrature conditions and non-branching conditions of height two

To start finding the complete solution, notice that (3.27a) represents seven equations with four unknowns and that directly determining the consistency conditions required is much less straightforward than for the analogous linear system (3.5) from the restricted system with $b_2 = 0$. This is because with $b_2 \neq 0$ it is no longer possible to eliminate RHS values of the lower sub-matrix with integer multiples of rows from the upper sub-matrix in order to find a homogeneous linear system that determines consistency. However, (3.27b) is four equations with three unknowns; therefore examining (3.27b) first leads to a more straightforward procedure (because it is rank deficient by one rather than three like the linear system (3.27a)) for determining consistency conditions. Directly following the procedure described by Cassity [32], the multipliers $\lambda : \mu : \nu : \omega$ are defined and the linear system

$$\begin{pmatrix} c_3(c_2-c_3) & c_3^2(c_2-c_3) & 2p_3 & 2c_3p_3 \\ c_4(c_2-c_4) & c_4^2(c_2-c_4) & 2p_4 & 2c_4p_4 \\ c_5(c_2-c_5) & c_5^2(c_2-c_5) & 2p_5 & 2c_5p_5 \\ \frac{1}{6}c_2-\frac{1}{12} & \frac{1}{12}c_2-\frac{1}{20} & \frac{1}{12} & \frac{1}{20} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \nu \\ \omega \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.28}$$

that is required for the consistency of (3.27b). The last row of (3.28) gives the expression

$$G_2 \equiv \left(\frac{1}{6}c_2-\frac{1}{12}\right)\lambda+\left(\frac{1}{12}c_2-\frac{1}{20}\right)\mu+\frac{1}{12}\nu+\frac{1}{20}\nu=0. \tag{3.29a}$$

Now observe that the first three rows of (3.28) have the same entries as the corresponding first three columns of rows 1, 2, 4, 5 of (3.27a). Using this observation with (3.27a), consider the linear combination

$$\lambda\,\mathrm{row}_1+\mu\,\mathrm{row}_2+\nu\,\mathrm{row}_4+\omega\,\mathrm{row}_5.$$

From this linear combination, it can be seen that the first three rows of (3.28) imply that the first three columns of rows 1, 2, 4, 5 of (3.27a) must sum to zero (with a few minor considerations described below when one of the multipliers vanishes). Consider the same linear combination applied to the fourth column and the RHS of (3.27a) that respectively leads to the expressions

$$F_6 \equiv c_6(c_2-c_6)(\lambda+\mu c_6)+2p_6(\nu+\omega c_6), \tag{3.29b}$$

$$G_3 \equiv F_6 b_6 = \left(\frac{1}{2}c_2-\frac{1}{3}\right)\lambda+\left(\frac{1}{3}c_2-\frac{1}{4}\right)\mu+\frac{1}{3}\nu+\frac{1}{4}\omega, \tag{3.29c}$$

which do not necessarily vanish. The resulting reduced version of (3.27a) that can usually be considered is

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & c_6(c_2-c_6) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & c_6^2(c_2-c_6) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & c_6^3(c_2-c_6) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \\ 0 & 0 & 0 & F_6 \\ 0 & 0 & 0 & 0 \\ \hline 4p_3^2 & 4p_4^2 & 4p_5^2 & 4p_6^2 \end{array}\right) \tilde{\mathbf{b}} = \begin{pmatrix} \frac{1}{2}c_2-\frac{1}{3} \\ \frac{1}{3}c_2-\frac{1}{4} \\ \frac{1}{4}c_2-\frac{1}{5} \\ \frac{1}{3} \\ G_3 \\ 0 \\ \frac{1}{5} \end{pmatrix}, \tag{3.30}$$

where row 5 reduces to $F_6\,b_6 = G_3$ due to the just described linear combination of the first three columns of rows 1, 2, 4, 5 vanishing, and only the expressions (3.29b) and (3.29c) remain for the fourth column and RHS, respectively. Additionally, row 6 is linearly dependent on the other rows, as can be seen by considering the linear combination from (3.27a) of

$$\lambda\,(\mathrm{row}_2 - \mathrm{row}_1) + \mu\,(\mathrm{row}_3 - \mathrm{row}_2) + \nu\,(\mathrm{row}_5 - \mathrm{row}_4) + \omega\,(\mathrm{row}_6 - \mathrm{row}_5).$$

This linear combination leads to the linear system

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & c_6^2(c_2-c_6) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & c_6^2(c_2-c_6) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \\ 2c_3p_3 & 2c_4p_4 & 2c_5p_5 & 2c_6p_6 \end{array}\right) \begin{pmatrix} b_3(1-c_3) \\ b_4(1-c_4) \\ b_5(1-c_5) \\ b_6(1-c_6) \end{pmatrix} = \begin{pmatrix} \frac{1}{6}c_2 - \frac{1}{12} \\ \frac{1}{12}c_2 - \frac{1}{20} \\ \frac{1}{12} \\ \frac{1}{20} \end{pmatrix}, \tag{3.31}$$

where the RHS and first three columns are identical to (3.27b), meaning that consistency with $b_6 \neq 0$ requires $F_6(1-c_6) = G_2 = 0$. This can be seen because using the multipliers from (3.28) implies linear combinations of both the first three columns of the coefficient matrix and RHS of (3.31) all sum to zero. A similar linear combination of the fourth column of the coefficient matrix of (3.31) sums to $F_6(1-c_6)$, which must vanish because the RHS and all other columns sum to zero. Therefore, either or both of $F_6 = 0$ or $(1-c_6) = 0$ imply that $c_6 = 1$ is required. The linear dependence of row 6 on combinations of rows 1–5 of (3.27a) also means that the order condition corresponding to row 6 is generally satisfied when the conditions from rows 1–5 are satisfied. The exceptions to this are addressed when the specific cases are described in detail below.

This leads to the defining conditions for each of the six major cases of the complete solution of the six-stage fifth-order order conditions. Note that (3.28) can still be rank 3 even when only three of its four rows are linearly dependent. When one of the multipliers $\lambda : \mu : \nu : \omega$ are zero but G_3 and F_6 do not vanish, this requires $c_6 = 1$ and leads to cases I–IV, corresponding to Case I with $\lambda = 0$, Case II with $\mu = 0$, Case III with $\nu = 0$, and Case IV with $\omega = 0$. When $G_3 = F_6 = 0$ with $c_6 = 1$ and none of $\lambda : \mu : \nu : \omega$ are required to be zero, this leads to Case V. Case VI is identical to Case V in regards to satisfying (3.27a), except that $c_6 \neq 1$. The C(2) simplifying assumptions (2.44b) occur when $\lambda = 0$ leads to singularities for Case V and VI [32], which are best handled using the restricted system with $b_2 = 0$ from Section 3.3 that is discussed further in the context of the complete solution in Section 3.4.5.

When $c_6 = 1$, subtracting consecutive rows of (3.27a) as described above leads to the linear system (3.31), where the RHS is identical to (3.27b) and the coefficient matrix has three identical columns to (3.27b). Therefore, in order for the two linear systems (3.27a) (which implies (3.31) as a constraint) and (3.27b) to be consistent with each other when $c_6 = 1$, their unknowns must also be equal, i.e.,

$$b_i(1-c_i) = r_i, \quad i \in \{3,4,5\}. \tag{3.32}$$

This is equivalent to the D(1) simplifying assumptions (2.44c) with $i \in \{3,4,5\}$. From (3.32), it can be shown that when $c_6 = 1$, the D(1) simplifying assumptions (2.44c) for all i always hold. However, when $c_6 \neq 1$ it was observed that none of the D(1) simplifying assumptions (2.44c) generally hold.

An additional condition, which leads to specific expressions for each case, comes directly from (3.28) being at most rank 3, implying that the determinant of the augmented matrix of (3.27b) must vanish for (3.28) to be consistent. The simplified and factored expression for this determinant is

$$-\frac{N_1(\mu\nu-\lambda\omega)(c_2-c_3)(c_2-c_4)(c_2-c_5)(c_3-c_4)(c_3-c_5)c_3(c_4-c_5)c_4c_5}{60\,(c_3\omega+\nu)(c_4\omega+\nu)(c_5\omega+\nu)}=0,$$
$$N_1=(10\,c_2\lambda+5\,c_2\mu-5\,\lambda-3\,\mu+5\,\nu+3\,\omega),$$

that is only valid for one factor and leads to

$$(10c_2\lambda+5c_2\mu-5\lambda-3\mu+5\nu+3\omega)=0, \tag{3.33}$$

because this is the only factor that can vanish with distinct components of the **c** vector because the other factors either imply non-distinct components of the **c** vector or appear in the denominators of other expressions required for the complete solution. The particular form that the expression (3.33) assumes with any case-specific substitutions is important and is described along with each individual case below.

Case I:

The material outlining the non-restricted solutions of this case is part of the new work presented in this thesis and not found in Cassity's original work [31] or elsewhere in the literature.

The expressions particular to the solution of Case I given by Cassity [32] are:

$$c_6=1,\quad c_2=\frac{3}{5},\quad 2\,p_6=\frac{8}{5},\quad F_6\,b_6=G_3\neq 0, \tag{3.34a}$$
$$\lambda:\mu:\nu:\omega=0:\mu:-\frac{3}{5}\omega:\omega, \tag{3.34b}$$
$$5c_3c_4c_5(5\mu^2+41\mu\omega-100\omega^2)-60(c_3c_4+c_3c_5+c_4c_5)(2\mu-5\omega)\omega+$$
$$45(c_3+c_4+c_5)(\mu-4\omega)\omega+108\omega^2=0. \tag{3.34c}$$

The particular form of the homogeneous system (3.28) for Case I is given by

$$\begin{pmatrix} c_3^2(c_2-c_3) & 2p_3 & 2c_3p_3\\ c_4^2(c_2-c_4) & 2p_4 & 2c_4p_4\\ c_5^2(c_2-c_5) & 2p_5 & 2c_5p_5\\ \frac{1}{12}c_2-\frac{1}{20} & \frac{1}{12} & \frac{1}{20}\end{pmatrix}\begin{pmatrix}\mu\\ \nu\\ \omega\end{pmatrix}=\begin{pmatrix}0\\0\\0\\0\end{pmatrix}, \tag{3.35}$$

with the particular expressions (3.29) for Case I given by

$$G_2 = \left(\frac{1}{12}c_2 - \frac{1}{20}\right)\mu + \frac{1}{12}\nu + \frac{1}{20}\omega = 0, \tag{3.36a}$$

$$F_6 = \mu(c_2 - 1) + 2p_6(\nu + \omega) \neq 0, \tag{3.36b}$$

$$G_3 = F_6\, b_6 = \left(\frac{1}{3}c_2 - \frac{1}{4}\right)\mu + \frac{1}{3}\nu + \frac{1}{4}\omega \neq 0. \tag{3.36c}$$

Solving for p_i, $i \in \{3, 4, 5\}$ from (3.35) gives

$$\begin{aligned} -2\, p_i\,(\nu + c_i\omega) &= c_i^2(c_2 - c_i)\mu, \\ p_i &= -\frac{c_i^2(c_2 - c_i)\mu}{2\,(\nu + c_i\omega)}, \end{aligned} \tag{3.37}$$

where if the values for c_2 and ν from Cassity's solution (3.34) are used, it is easily seen that the expressions for p_i, $i \in \{3, 4, 5\}$ (3.37) reduce to

$$p_i = \frac{c_i^2\mu}{2\omega}. \tag{3.38}$$

However, there is a more complete solution for Case I than the one given by Cassity in 1969 (3.34) [32], and it is now convenient to use because of the capabilities of modern CASs. This more complete solution means it is not necessary to consider other possible simplifications similar to those of $c_2 = \frac{3}{5}$ and $\nu = -\frac{3}{5}\omega$ from Cassity's solution (3.34b) that lead to convenient forms of the p_i, such as (3.38).

Before looking at the more complete solution, using the values (3.38) reduces (3.27a) to

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\ \hline c_3^2\frac{\mu}{\omega} & c_4^2\frac{\mu}{\omega} & c_5^2\frac{\mu}{\omega} & 2p_6 \\ c_3^3\frac{\mu}{\omega} & c_4^3\frac{\mu}{\omega} & c_5^3\frac{\mu}{\omega} & 2p_6 \\ c_3^4\frac{\mu}{\omega} & c_4^4\frac{\mu}{\omega} & c_5^4\frac{\mu}{\omega} & 2p_6 \\ \hline c_3^4\frac{\mu^2}{\omega^2} & c_4^4\frac{\mu^2}{\omega^2} & c_5^4\frac{\mu^2}{\omega^2} & 4p_6^2 \end{array}\right) \begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\,c_2 - \frac{1}{3} \\ \frac{1}{3}\,c_2 - \frac{1}{4} \\ \frac{1}{4}\,c_2 - \frac{1}{5} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \\ \frac{1}{5} \end{pmatrix},$$

where subtracting $\frac{\mu}{\omega}$ times row 6 from row 7 and incorporating the results from the discussion earlier in this section (see the discussion about how the system (3.30) is formed) gives

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\ \hline c_3^2\frac{\mu}{\omega} & c_4^2\frac{\mu}{\omega} & c_5^2\frac{\mu}{\omega} & 2p_6 \\ 0 & 0 & 0 & F_6 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 4p_6^2 - 2\frac{\mu}{\omega}p_6 \end{array}\right) \begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}c_2 - \frac{1}{3} \\ \frac{1}{3}c_2 - \frac{1}{4} \\ \frac{1}{4}c_2 - \frac{1}{5} \\ \frac{1}{3} \\ G_3 \\ 0 \\ \frac{1}{5} - \frac{\mu}{\omega}\frac{1}{5} \end{pmatrix}, \tag{3.39}$$

where rows 5 and 7 can be solved to compute the value of b_6 along with finding the constraint $2\,p_6 = \frac{8}{5}$, and using now-known values of $\tilde{\mathbf{b}}$ allows row 4 to be reduced to the constraint given by Cassity (3.34c), which was verified using SAGE [57]. The simplifications that Cassity used, i.e., $c_2 = \frac{3}{5}$ and $\nu = -\frac{3}{5}\omega$, along with the calculated values for b_6, p_6, and the constraint from row 4, now give all of the expressions for Cassity's solution [32]. One of the c_3, c_4, c_5 needs to be fixed to satisfy the expression from row 4 of (3.39); in this thesis, c_5 is arbitrarily selected. Although c_3 or c_4 could be fixed as well, choosing any of c_3, c_4, c_5 results in the same family of ERK methods, e.g., having c_3 and c_4 as arbitrary parameters and solving for c_5 gives the same set of coefficients as having c_3 and c_5 as arbitrary parameters and solving for c_4.

However, as mentioned above, despite purporting to give a complete solution, Cassity [32] was satisfied to find simplifications that would ensure the calculated coefficients would be rational expressions in terms of the free parameters. Hence the statement in his 1969 paper [32] of

> "In spite of extreme non-linearities in the defining equations, it is found that solutions for all but one case can be expressed rationally in terms of properly selected parameters; one case requires a single quadratic irrationality."

that indicates finding easier to use rational expressions at the cost of the full complete solution was acceptable, because from the above passage it is apparent that he knew that he was not fully describing the complete solution of the six-stage fifth-order ERK order conditions (B.1)–(B.5). However, given the size and complexity that any quadratic irrationality generally gives to expressions for the calculated coefficients in terms of the free parameters, it is understandable why it would be desirable to use the simplifications whenever possible for demonstrating the existence of solutions in 1969, before computer algebra was in widespread use. Although computer algebra was first seen in the 1960s [121], it was not until the 1980s that powerful CASs, such as MAPLE, were in widespread use [93].

The entire complete solution of Case I, which allows c_2 to be an extra free parameter because it no longer needs to be fixed as $c_2 = \frac{3}{5}$, can now be presented. In order to ensure consistency of the one-legged order conditions (3.27b), a constraint on ν can be found using (3.33) to give

$$\nu = -\frac{(5c_2\mu - 3\mu + 3\omega)}{5},$$

which is more general than (but is also satisfied by) Cassity's simplifications for c_2 and ν for Case I (3.34). With the general expressions for p_i (3.37), the equivalent linear system to (3.27a) without the simplifications can now be given as

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \\ 0 & 0 & 0 & F_6 \\ 0 & 0 & 0 & 0 \\ \hline 4p_3^2 & 4p_4^2 & 2p_5^2 & 2p_6^2 \end{array}\right) \begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}c_2 - \frac{1}{3} \\ \frac{1}{3}c_2 - \frac{1}{4} \\ \frac{1}{4}c_2 - \frac{1}{5} \\ \frac{1}{3} \\ G_3 \\ 0 \\ \frac{1}{5} \end{pmatrix}, \tag{3.40}$$

where the SAGE CAS easily handles directly substituting the expressions for $\tilde{\mathbf{p}}$ (3.37) without simplifications similar to those used leading to (3.39). The conditions corresponding to rows 4, 5, 7 of (3.40) can be solved together in terms of the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$ (the expressions satisfying the conditions for (3.27a) that do not incorporate the $a_{6,3}$ free parameter), with b_6, p_6, c_5 being chosen as additional constraints to ensure the over-determined system (3.40) is consistent. The procedure used in software to satisfy the expressions (3.40) in terms of only the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$ for this case is given by the following steps.

- Intermediate expressions for $\tilde{\mathbf{b}}$ that still explicitly contain p_6 and c_5 are found from the first three rows of (3.40).

- An intermediate expression for b_6 is found from row 5 of (3.40), which still explicitly contains c_5 and p_6.

- An intermediate expression for p_6 is found from row 4 of (3.40), which still explicitly contains c_5.

- Substitute these intermediate expressions for $\tilde{\mathbf{b}}$, b_6, p_6 into row 7 of (3.40) and simplify to a quadratic equation in terms of the constraint c_5 that is only in terms of the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$. An expression for the constraint on c_5 requires solving this quadratic equation, and this means Case I is one of the three cases requiring a quadratic irrationality (square root) for the full complete solution.

- This expression for c_5 can now be substituted back into the intermediate expression for p_6 to obtain an expression for p_6 in terms of only the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$ that satisfies the condition corresponding to row 4 of (3.40).

- Row 5 of (3.40) is satisfied by b_6 and the expressions for c_5 and p_6 are used to obtain expressions in terms of the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$ for b_6.

- Finally, the expression for b_6 in terms of the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$ are further used to obtain expressions in terms of these free parameters for $\tilde{\mathbf{b}}$.

When this procedure is applied, explicit expressions are available for $\tilde{\mathbf{b}}$, b_6, p_6, c_5 in terms of only the free parameters c_2, c_3, c_4, $\frac{\mu}{\omega}$.

The constraint on c_5 contains a square root from applying the quadratic formula. Because the radical in the quadratic formula has a plus/minus sign in front of it, there are two sub-cases, named Ia and Ib, which are considered separately for computer implementation.

Case II:

The material outlining the non-restricted solutions of this case is part of the new work presented in this thesis and not found in Cassity's original work [31] or elsewhere in the literature.

The expressions particular to the solution of Case II given by Cassity [32] are:

$$c_6 = 1, \quad 2p_6(10c_2^2 - 12c_2 + 3) = 3(c_2 - 1)(3c_2 - 1), \quad F_6\, b_6 = G_3 \neq 0, \tag{3.41a}$$

$$\lambda : \mu : \nu : \omega = 5c_2 - 3 : 0 : -5c_2(2c_2 - 1) : 5(2c_2 - 1),$$

$$\begin{vmatrix} 1 & 1 & 1 & \omega[-\frac{1}{3}(\lambda - 2p_6\omega) + (\frac{1}{4}\lambda - \frac{1}{5}\omega)] \\ c_3 & c_4 & c_5 & \omega[-\frac{1}{4}(\lambda - 2p_6\omega) + (\frac{1}{4}\lambda - \frac{1}{5}\omega)] \\ c_3^2 & c_4^2 & c_5^2 & \omega[-\frac{1}{5}(\lambda - 2p_6\omega) + (\frac{1}{4}\lambda - \frac{1}{5}\omega)] \\ c_3^3 & c_4^3 & c_5^3 & [(\frac{1}{4}c_2 - \frac{1}{5})\lambda + \frac{1}{5}\omega](\lambda - 2p_6\omega) - \lambda G_3 - \nu(\frac{1}{4}\lambda - \frac{1}{5}\omega) \end{vmatrix} = 0. \tag{3.41b}$$

The particular form of the homogeneous system (3.28) for Case II is given by

$$\begin{pmatrix} c_3(c_2 - c_3) & 2p_3 & 2c_3p_3 \\ c_4(c_2 - c_4) & 2p_4 & 2c_4p_4 \\ c_5(c_2 - c_5) & 2p_5 & 2c_5p_5 \\ \frac{1}{6}c_2 - \frac{1}{12} & \frac{1}{12} & \frac{1}{20} \end{pmatrix} \begin{pmatrix} \lambda \\ \nu \\ \omega \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.42}$$

with particular expressions (3.29) for Case II given by

$$G_2 = \left(\frac{1}{6}c_2 - \frac{1}{12}\right)\lambda + \nu\frac{1}{12} + \omega\frac{1}{20} = 0, \tag{3.43a}$$

$$F_6 = (c_2 - 1)\lambda + 2p_6(\nu + \omega) \neq 0, \tag{3.43b}$$

$$G_3 = \left(\frac{1}{2}c_2 - \frac{1}{3}\right)\lambda + \frac{1}{3}\nu + \frac{1}{4}\omega \neq 0. \tag{3.43c}$$

Solving for p_i from (3.42) gives

$$-2p_i(\nu + c_i\,\omega) = c_i(c_2 - c_i)\,\lambda,$$

$$p_i = -\frac{c_i(c_2 - c_i)\,\lambda}{2(\nu + c_i\,\omega)}. \tag{3.44}$$

If the values from Cassity's solution (3.41) are used, it is easily seen that the expressions for p_i (3.44) reduce to

$$2p_i = c_i \frac{\lambda}{\omega}. \tag{3.45}$$

However, following the same discussion from Case I, there is a more general version of the complete solution. Before looking at this more general version of the complete solution, using the values (3.45) reduces (3.27a) to

$$\left(\begin{array}{cccc}
c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\
c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\
c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\
\hline
c_3\frac{\lambda}{\omega} & c_4\frac{\lambda}{\omega} & c_5\frac{\lambda}{\omega} & 2p_6 \\
c_3^2\frac{\lambda}{\omega} & c_4^2\frac{\lambda}{\omega} & c_5^2\frac{\lambda}{\omega} & 2p_6 \\
c_3^3\frac{\lambda}{\omega} & c_4^3\frac{\lambda}{\omega} & c_5^3\frac{\lambda}{\omega} & 2p_6 \\
\hline
c_3^2\left(\frac{\lambda}{\omega}\right)^2 & c_4^2\left(\frac{\lambda}{\omega}\right)^2 & c_5^2\left(\frac{\lambda}{\omega}\right)^2 & 4p_6^2
\end{array}\right)
\begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix}
=
\begin{pmatrix}
\frac{1}{2}c_2-\frac{1}{3} \\
\frac{1}{3}c_2-\frac{1}{4} \\
\frac{1}{4}c_2-\frac{1}{5} \\
\frac{1}{3} \\
\frac{1}{4} \\
\frac{1}{5} \\
\frac{1}{5}
\end{pmatrix},$$

where subtracting $\frac{\lambda}{\omega}$ times row 5 from 7 and incorporating the results from the discussion earlier in this section (see the discussion about how the system (3.30) is formed) gives

$$\left(\begin{array}{cccc}
c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\
c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\
c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\
\hline
c_3\frac{\lambda}{\omega} & c_4\frac{\lambda}{\omega} & c_5\frac{\lambda}{\omega} & 2p_6 \\
0 & 0 & 0 & F_6 \\
0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 4p_6^2-2p_6\frac{\lambda}{\omega}
\end{array}\right)
\begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix}
=
\begin{pmatrix}
\frac{1}{2}c_2-\frac{1}{3} \\
\frac{1}{3}c_2-\frac{1}{4} \\
\frac{1}{4}c_2-\frac{1}{5} \\
\frac{1}{3} \\
G_3 \\
0 \\
\frac{1}{5}-\frac{1}{4}\frac{\lambda}{\omega}
\end{pmatrix}, \tag{3.46}$$

where, similar to the restricted solution of Case I, rows 5 and 7 can be solved to compute the value of b_6 along with the value of p_6 from Cassity's solution (3.41a) and c_5 can be chosen without loss of generality as the constraint to ensure row 4 is satisfied; see the discussion for Case I for further details.

The more complete solution of Case II has an extra free parameter because the multipliers no longer depend exclusively on the choice of c_2. In order to ensure consistency of the one-legged order conditions (3.27b), ν can be found using (3.33) to give

$$\nu = -\frac{10c_2\lambda - 5\lambda + 3\omega}{5},$$

which is also satisfied by the simplifications for Cassity's solution for Case II (3.41). With the expressions for p_i (3.44) without the simplifications, the linear system (3.27a) now becomes the equivalent linear system (3.27a) given by

$$\begin{pmatrix} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \\ 0 & 0 & 0 & F_6 \\ 0 & 0 & 0 & 0 \\ \hline 4p_3^2 & 4p_4^2 & 4p_5^2 & 4p_6^2 \end{pmatrix} \begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}\,c_2 - \frac{1}{3} \\ \frac{1}{3}\,c_2 - \frac{1}{4} \\ \frac{1}{4}\,c_2 - \frac{1}{5} \\ \frac{1}{3} \\ G_3 \\ 0 \\ \frac{1}{5} \end{pmatrix}, \tag{3.47}$$

where the SAGE CAS easily handles directly substituting the expressions for $\tilde{\mathbf{p}}$ (3.44) without reductions similar to those used leading to (3.46). The conditions corresponding to rows 4, 5, 7 of (3.47) can be solved together in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ (the expressions satisfying the conditions for (3.27a) do not incorporate the $a_{6,3}$ free parameter) with b_6, p_6, c_5 being chosen as additional constraints to ensure that the over-determined system (3.47) is consistent. The procedure used in software to satisfy the expressions (3.47) in terms of only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ for this case is given by:

- Intermediate expressions for $\tilde{\mathbf{b}}$ that still explicitly contain p_6 and c_5 are found from the first three rows of (3.47).

- An intermediate expression for b_6 is found from row 5 of (3.47), which still explicitly contains c_5 and p_6.

- An intermediate expression for p_6 is found from row 4 of (3.47), which still explicitly contains c_5.

- Substitute these intermediate expressions for $\tilde{\mathbf{b}}$, b_6, p_6 into row 7 of (3.47) and simplify to a quadratic equation in terms of the constraint c_5 that contains only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ as coefficients. An expression for the constraint on c_5 requires solving this quadratic equation, and this means Case II is one of the three cases requiring a quadratic irrationality (square root) for the full complete solution.

- This expression for c_5 can now be substituted back into the intermediate expression for p_6 to obtain an expression for p_6 in terms of only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ that satisfies the condition corresponding to row 4 of (3.47).

- Row 5 of (3.47) is satisfied by b_6 and the expressions for c_5 and p_6 are used to obtain expressions in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ for b_6.

- Finally the expression for b_6 in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ are further used to obtain expressions in terms of these free parameters for $\tilde{\mathbf{b}}$.

When this procedure is applied, explicit expressions are available for $\tilde{\mathbf{b}}$, b_6, p_6, c_5 only in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$. The constraint on c_5 contains a square root from applying the quadratic formula.

Because the radical in the quadratic formula has a plus/minus sign in front of it, there are two sub-cases that are named IIa and IIb, which are are easiest to consider separately for computer implementation.

Case III:

The expressions particular to the solution of Case III given by Cassity [32] are:

$$c_6 = 1, \quad 2p_6 = 1, \quad F_6\, b_6 = G_3 \neq 0, \tag{3.48a}$$

$$\lambda : \mu : \nu : \omega = \lambda : \mu : 0 : -\frac{5}{3}\,(2c_2 - 1)\,\lambda - \left(\frac{5}{3}c_2 - 1\right)\mu, \tag{3.48b}$$

$$\begin{vmatrix} 1 & 1 & 1 & 5[5(2c_2-1)(16c_2-5)\lambda^2 + (40c_2-23)c_2\lambda\mu - 2c_2^2\mu^2] \\ c_3 & c_4 & c_5 & 15[5(2c_2-1)(3c_2-1)\lambda + (9c_2-5)c_2\mu]\lambda \\ c_3^2 & c_4^2 & c_5^2 & 15[(2c_2-1)(8c_2-3)\lambda + (5c_2-5)c_2\mu]\lambda \\ c_3^3 & c_4^3 & c_5^3 & 9[(15c_2^2-13c_2+3)\lambda + (5c_2-3)c_2\mu]\lambda \end{vmatrix} = 0. \tag{3.48c}$$

The particular form of the homogeneous system (3.28) for Case III is given by

$$\begin{pmatrix} c_3(c_2-c_3) & c_3^2(c_2-c_3) & 2c_3p_3 \\ c_4(c_2-c_4) & c_4^2(c_2-c_4) & 2c_4p_4 \\ c_5(c_2-c_5) & c_5^2(c_2-c_5) & 2c_5p_5 \\ (\frac{1}{6}c_2 - \frac{1}{12}) & (\frac{1}{12}c_2 - \frac{1}{20}) & \frac{1}{20} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \omega \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.49}$$

with the particular expressions (3.29) for Case III given by

$$G_2 = \left(\frac{1}{6}c_2 - \frac{1}{12}\right)\lambda + \left(\frac{1}{12}c_2 - \frac{1}{20}\right)\mu + \frac{1}{20}\omega = 0, \tag{3.50a}$$

$$F_6 = (c_2-1)\lambda + (c_2-1)\mu + 2p_6\omega \neq 0, \tag{3.50b}$$

$$G_3 = \left(\frac{1}{2}c_2 - \frac{1}{3}\right)\lambda + \left(\frac{1}{3}c_2 - \frac{1}{4}\right)\mu + \frac{1}{4}\omega \neq 0. \tag{3.50c}$$

Solving for p_i, $i \in \{3, 4, 5\}$ from (3.49) gives

$$\begin{aligned} -2\,\omega\, c_i\, p_i &= c_i(c_2 - c_i)(\lambda + c_i\mu), \\ p_i &= -\frac{(c_2 - c_i)(\lambda + c_i\mu)}{2\omega}. \end{aligned} \tag{3.51}$$

However, unlike Case I and Case II above, the solution given by Cassity is in fact the complete solution for Case III. This can be seen from the expression (3.33) to ensure consistency of the one-legged order conditions (3.27b), where ω can be found using (3.33) to give

$$\omega = -\frac{10c_2\lambda + 5c_2\mu - 5\lambda - 3\mu}{3}, \tag{3.52}$$

which is exactly the expression for ω given by Cassity's solution (3.48b). Using the values (3.51) this reduces (3.27a) to

$$\left(\begin{array}{cccc}
c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-c_6) \\
c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-c_6) \\
c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-c_6) \\
\hline
-\frac{(c_2-c_3)(\lambda+c_3\mu)}{\omega} & -\frac{(c_2-c_4)(\lambda+c_4\mu)}{\omega} & -\frac{(c_2-c_5)(\lambda+c_5\mu)}{\omega} & 2p_6 \\
-c_3\frac{(c_2-c_3)(\lambda+c_3\mu)}{\omega} & -c_4\frac{(c_2-c_4)(\lambda+c_4\mu)}{\omega} & -c_5\frac{(c_2-c_5)(\lambda+c_5\mu)}{\omega} & 2p_6 \\
-c_3^2\frac{(c_2-c_3)(\lambda+c_3\mu)}{\omega} & -c_4^2\frac{(c_2-c_4)(\lambda+c_4\mu)}{\omega} & -c_5^2\frac{(c_2-c_5)(\lambda+c_5\mu)}{\omega} & 2p_6 \\
\hline
\frac{(c_2-c_3)^2(\lambda+c_3\mu)^2}{\omega^2} & \frac{(c_2-c_4)^2(\lambda+c_4\mu)^2}{\omega^2} & \frac{(c_2-c_5)^2(\lambda+c_5\mu)^2}{\omega^2} & 4p_6^2
\end{array}\right)
\begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix}
=
\begin{pmatrix} \frac{1}{2}c_2-\frac{1}{3} \\ \frac{1}{3}c_2-\frac{1}{4} \\ \frac{1}{4}c_2-\frac{1}{5} \\ \frac{1}{3} \\ \frac{1}{4} \\ \frac{1}{5} \\ \frac{1}{5} \end{pmatrix},$$

where incorporating the results from the discussion earlier in this section (see the discussion about how the system (3.30) is formed) gives

$$\left(\begin{array}{cccc}
c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-c_6) \\
c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-c_6) \\
c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-c_6) \\
\hline
-\frac{(c_2-c_3)(\lambda+c_3\mu)}{\omega} & -\frac{(c_2-c_4)(\lambda+c_4\mu)}{\omega} & -\frac{(c_2-c_5)(\lambda+c_5\mu)}{\omega} & 2p_6 \\
0 & 0 & 0 & F_6 \\
0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 4p_6^2+\frac{2p_6(c_2\lambda+(c_2\mu-\lambda)-\mu)}{\omega}
\end{array}\right)
\begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix}$$

$$=\begin{pmatrix} \frac{1}{2}c_2-\frac{1}{3} \\ \frac{1}{3}c_2-\frac{1}{4} \\ \frac{1}{4}c_2-\frac{1}{5} \\ \frac{1}{3} \\ G_3 \\ 0 \\ \frac{1}{5}+\frac{\left(\frac{1}{3}c_2\lambda+\frac{1}{4}(c_2\mu-\lambda)-\frac{1}{5}\mu\right)}{\omega} \end{pmatrix},$$

where in this case the new row 7 is found from the linear combination of

$$\mathrm{row}_7=\mathrm{row}_7+c_2\lambda\,\mathrm{row}_4-(c_2\mu+\lambda)\,\mathrm{row}_5+\mu\,\mathrm{row}_6.$$

In (3.53a), similar to the restricted solutions of Case I and Case II, rows 5 and 7 can be solved to compute the value of b_6 along with the value of p_6 from Cassity's solution (3.41a). Note that in this case, a quadratic irrationality (square root) is not required and all calculated coefficients are rational expressions of the free parameters. As well, similar to Cases I and II, c_5 can be chosen without loss of generality as the constraint to ensure row 4 of (3.53a) is satisfied.

Case IV:

The expressions particular to the solution of Case IV given by Cassity [32] are:

$$c_6 = 1, \quad F_6\, b_6 = G_3 \neq 0, \tag{3.53a}$$

$$\lambda : \mu : \nu : \omega = \lambda : \mu : -(2c_2 - 1)\,\lambda - \left(c_2 - \frac{3}{5}\right)\mu : 0, \tag{3.53b}$$

$$\begin{vmatrix} 1 & 1 & 1 & (\frac{1}{3}F_6 - 2p_6G_3)\mu \\ c_3 & c_4 & c_5 & (\frac{1}{4}F_6 - 2p_6G_3)\mu \\ c_3^2 & c_4^2 & c_5^2 & (\frac{1}{5}F_6 - 2p_6G_3)\mu \\ c_3^3 & c_4^3 & c_5^3 & [(\frac{1}{4}c_2 - \frac{1}{5})\lambda + \frac{1}{5}c_2\mu + \frac{1}{5}\nu]F_6 - 2p_6G_3[(c_2-1)\lambda + c_2\mu + 2p_6\nu] \end{vmatrix} = 0, \tag{3.53c}$$

$$\begin{vmatrix} 1 & 1 & 1 & -[(\frac{1}{2}c_2 - \frac{1}{3})F_6 - (c_2-1)G_3]\mu^3 \\ c_3 & c_4 & c_5 & [(\frac{1}{2}c_2 - \frac{1}{3})F_6 - (c_2-1)G_3]\lambda\mu^2 + (\frac{1}{3}F_6 - 2p_6G_3)\mu^2\nu \\ c_3^2 & c_4^2 & c_5^2 & -[(\frac{1}{2}c_2 - \frac{1}{3})F_6 - (c_2-1)G_3]\lambda^2\mu - (\frac{1}{3}F_6 - 2p_6G_3)\lambda\mu\nu + (\frac{1}{4}F_6 - 2p_6G_3)\mu^2\nu \\ c_3^3 & c_4^3 & c_5^3 & -[(\frac{1}{2}c_2 - \frac{1}{3})F_6 - (c_2-1)G_3]\lambda^3 + (\frac{1}{3}F_6 - 2p_6G_3)\lambda^2\nu - (\frac{1}{4}F_6 - 2p_6G_3)\lambda\mu\nu + (\frac{1}{5}F_6 - 2p_6G_3)\mu^2\nu \end{vmatrix} = 0. \tag{3.53d}$$

The particular form of the homogeneous system (3.28) for Case IV is given by

$$\begin{pmatrix} c_3(c_2 - c_3) & c_3^2(c_2 - c_3) & 2p_3 \\ c_4(c_2 - c_4) & c_4^2(c_2 - c_4) & 2p_4 \\ c_5(c_2 - c_5) & c_5^2(c_2 - c_5) & 2p_5 \\ (\frac{1}{6}c_2 - \frac{1}{12}) & (\frac{1}{12}c_2 - \frac{1}{20}) & \frac{1}{12} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \nu \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.54}$$

with the particular expressions (3.29) for Case IV being given by

$$\begin{aligned} G_2 &= (\frac{1}{6}c_2 - \frac{1}{12})\lambda + (\frac{1}{12}c_2 - \frac{1}{20})\mu + \frac{1}{12}\nu = 0, \\ F_6 &= (c_2 - 1)\lambda + (c_2 - 1)\mu + 2p_6\nu \neq 0, \\ G_3 &= \left(\frac{1}{2}c_2 - \frac{1}{3}\right)\lambda + \left(\frac{1}{3}c_2 - \frac{1}{4}\right)\mu + \frac{1}{3}\nu \neq 0. \end{aligned}$$

Solving for p_i from (3.54) gives

$$\begin{aligned} -2\,\nu\, p_i &= c_i(c_2 - c_i)(\lambda + c_i\mu), \\ 2p_i &= -c_i\frac{(c_2 - c_i)(\lambda + c_i\mu)}{\nu}. \end{aligned} \tag{3.55}$$

However, unlike Case I and Case II above but like Case III, the solution given by Cassity is in fact the complete solution for Case IV that also requires a quadratic irrationality (square root). This is likely because Cassity could not find a simplification to (3.55) that functioned like (3.38) and (3.45) for Cases I and II, respectively. This can be seen because in order to ensure consistency of the one-legged order conditions (3.27b), ν can be found using (3.33) to give

$$\nu = -\frac{10c_2\lambda + 5c_2\mu - 5\lambda - 3\mu}{5}, \tag{3.56}$$

which is exactly the expression for ν given by Cassity's solution (3.53b). With the expressions for p_i (3.55) that are particular to Case IV, incorporating the results from the discussion earlier in this section, noting that $\omega = 0$ means rows 4 and 5 are reduced similarly to how rows 5 and 6 are reduced respectively in Cases I, II, III (this is a variant mentioned in the discussion of the procedure used to form the system (3.30)), gives

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) \\ c_3^3(c_2-c_3) & c_4^3(c_2-c_4) & c_5^3(c_2-c_5) & (c_2-1) \\ \hline 0 & 0 & 0 & F_6 \\ 0 & 0 & 0 & 0 \\ 2c_3^2p_3 & 2c_4^2p_4 & 2c_5^2p_5 & 2p_6 \\ \hline 4p_3^2 & 4p_4^2 & 4p_5^2 & 4p_6^2 \end{array}\right) \begin{pmatrix} b_3 \\ b_4 \\ b_5 \\ b_6 \end{pmatrix} = \begin{pmatrix} \frac{1}{2}c_2 - \frac{1}{3} \\ \frac{1}{3}c_2 - \frac{1}{4} \\ \frac{1}{4}c_2 - \frac{1}{5} \\ G_3 \\ 0 \\ \frac{1}{5} \\ \frac{1}{5} \end{pmatrix}, \tag{3.57}$$

where the SAGE CAS easily handles directly substituting the expressions for $\tilde{\mathbf{p}}$ (3.55). The conditions corresponding to rows 4, 5, 7 of (3.57) can be solved together in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ (the expressions satisfying the conditions for (3.27a) do not incorporate the $a_{6,3}$ free parameter) with b_6, p_6, c_5 being chosen as additional constraints to ensure the over-determined system (3.57) is consistent. The procedure used in software to satisfy the expressions (3.57) in terms of only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ for this case is given by:

- Intermediate expressions for $\tilde{\mathbf{b}}$ that still explicitly contain p_6 and c_5 are found from the first three rows of (3.57).
- An intermediate expression for b_6 is found from row 4 of (3.57), which still explicitly contains c_5 and p_6.
- An intermediate expression for p_6 is found from row 6 of (3.57), which still explicitly contains c_5.
- Substitute these intermediate expressions for $\tilde{\mathbf{b}}$, b_6, p_6 into row 7 of (3.57) and simplify to a quadratic equation in terms of the constraint c_5 that contains only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ as coefficients. An expression for the constraint on c_5 requires solving this quadratic equation. Therefore, this means Case IV is one of the three cases requiring a quadratic irrationality (square root) for the full complete solution.
- This expression for c_5 is now substituted back into the intermediate expression for p_6 to obtain an expression for p_6 in terms of only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ that satisfies the condition corresponding to row 6 of (3.57).

- Row 4 of (3.57) is satisfied by b_6 and the expressions for c_5 and p_6 are used to obtain expressions in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ for b_6.

- Finally the expression for b_6 in terms of the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$ can be further used to obtain expressions in terms of these free parameters for $\tilde{\mathbf{b}}$.

When this procedure is applied, explicit expressions are now available for $\tilde{\mathbf{b}}$, b_6, p_6, c_5 only in terms of only the free parameters c_2, c_3, c_4, $\frac{\lambda}{\omega}$.

Rows 6 and 7 of (3.57) correspond to (3.53c) and (3.53d) respectively of Cassity's solution; this was verified by expanding his published expressions [32] and comparing terms with the generated expressions in `OCSage`. The formulation of these constraints as determinants is not required when using a CAS because explicit constraints can be solved for directly.

The constraint on c_5 contains a square root from applying the quadratic formula. Because the radical in the quadratic formula has a plus/minus sign in front of it, there are two sub-cases that are named IVa and IVb, which are considered separately for computer implementation.

Cases V and VI:

The solutions for Case V and Case VI are identical except that Case V requires $c_6 = 1$ and Case VI requires $c_6 \neq 1$ with two additional conditions given in Section 3.4.2.

The expressions particular to the solution of Case V/VI given by Cassity [32] are:

$$c_6 = 1, \quad F_6 = G_3 = 0, \tag{3.58a}$$

$$\lambda : \mu : \nu : \omega = \lambda : \mu : -(4c_2 - 1)\lambda - c_2\mu : \frac{10}{3}c_2\lambda + \mu, \tag{3.58b}$$

$$\lambda\nu \left[\begin{vmatrix} X_3 & X_4 & X_5 & X_6 \\ c_3X_3 & c_4X_4 & c_5X_5 & c_6X_6 \\ c_3^2X_3 & c_4^2X_4 & c_5^2X_5 & c_6^2X_6 \\ Y_3 & Y_4 & Y_5 & Y_6 \end{vmatrix} c_2 - \begin{vmatrix} X_3 & X_4 & X_5 & X_6 \\ c_3X_3 & c_4X_4 & c_5X_5 & c_6X_6 \\ c_3^2X_3 & c_4^2X_4 & c_5^2X_5 & c_6^2X_6 \\ c_3Y_3 & c_4Y_4 & c_5Y_5 & c_6Y_6 \end{vmatrix} \right] = 0, \tag{3.58c}$$

$$X_i = 10c_i^2 - 12c_i + 3, \quad i = 3,4,5,6,$$

$$Y_i = (4 - 5c_i)(\lambda + \mu c_i)^2, \quad i = 3,4,5,6. \tag{3.58d}$$

The particular form of the homogeneous system (3.28) for Case V/VI is given by

$$\begin{pmatrix} c_3(c_2 - c_3) & c_3^2(c_2 - c_3) & 2p_3 & 2c_3p_3 \\ c_4(c_2 - c_4) & c_4^2(c_2 - c_4) & 2p_4 & 2c_4p_4 \\ c_5(c_2 - c_5) & c_5^2(c_2 - c_5) & 2p_5 & 2c_5p_5 \\ \frac{1}{6}c_2 - \frac{1}{12} & \frac{1}{12}c_2 - \frac{1}{20} & \frac{1}{12} & \frac{1}{20} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \nu \\ \omega \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{3.59}$$

with particular expressions (3.29) for Case V/VI given by

$$
\begin{aligned}
G_2 &= \left(\frac{1}{6}c_2 - \frac{1}{12}\right)\lambda + \left(\frac{1}{12}c_2 - \frac{1}{20}\right)\mu + \frac{1}{12}\nu + \frac{1}{20}\nu = 0,\\
F_6 &= c_6(c_2 - c_6)(\lambda + \mu c_6) + 2p_6(\nu + \omega c_6) = 0,\\
G_3 = F_6\, b_6 &= \left(\frac{1}{2}c_2 - \frac{1}{3}\right)\lambda + \left(\frac{1}{3}c_2 - \frac{1}{4}\right)\mu + \frac{1}{3}\nu + \frac{1}{4}\omega = 0.
\end{aligned}
$$

Solving for p_i from (3.28) gives

$$
\begin{aligned}
2p_i(-\nu - c_i\,\omega) &= c_i(c_2 - c_i)(\lambda + c_i\,\mu),\\
2p_i &= -\frac{c_i(c_2 - c_i)(\lambda + \mu\, c_i)}{(\nu + c_i\,\omega)}.
\end{aligned}
\tag{3.60}
$$

However, unlike Case I and Case II above but like Cases III and IV, the solution given by Cassity is in fact the full complete solution for Cases V and VI. The multipliers ν and ω can be found by simultaneously solving for ν, ω, and p_6 from the three equations

$$
\begin{aligned}
10c_2\lambda + 5c_2\mu - 5\lambda - 3\mu + 5\nu + 3\omega &= 0,\\
F_6 = c_6(c_2 - c_6)(\lambda + \mu c_6) + 2p_6(\nu + \omega c_6) &= 0,\\
G_3 = F_6\, b_6 = \left(\frac{1}{2}c_2 - \frac{1}{3}\right)\lambda + \left(\frac{1}{3}c_2 - \frac{1}{4}\right)\mu + \frac{1}{3}\nu + \frac{1}{4}\omega &= 0,
\end{aligned}
$$

where the solution for ν and ω are exactly that of Cassity's solution in (3.58b), and the solution for p_6 (not explicitly given by Cassity [32]) is

$$
p_6 = \frac{3(c_2 - 1)(\lambda + \mu)}{2(3c_2\lambda + 3c_2\mu - 3\lambda - 3\mu)}.
$$

With the expressions for p_i (3.60) that are particular to Case V, the linear system (3.27a) now becomes

$$
\left(\begin{array}{cccc}
c_3(c_2 - c_3) & c_4(c_2 - c_4) & c_5(c_2 - c_5) & c_6(c_2 - c_6)\\
c_3^2(c_2 - c_3) & c_4^2(c_2 - c_4) & c_5^2(c_2 - c_5) & c_6^2(c_2 - c_6)\\
c_3^3(c_2 - c_3) & c_4^3(c_2 - c_4) & c_5^3(c_2 - c_5) & c_6^3(c_2 - c_6)\\
\hline
2\frac{c_3(c_2-c_3)(\lambda+\mu c_3)}{(-\nu-\omega c_3)} & 2\frac{c_4(c_2-c_4)(\lambda+\mu c_4)}{(-\nu-\omega c_4)} & 2\frac{c_5(c_2-c_5)(\lambda+\mu c_5)}{(-\nu-\omega c_5)} & 2c_6\, p_6\\
0 & 0 & 0 & 0\\
0 & 0 & 0 & 0\\
\hline
4\frac{c_3^2(c_2-c_3)^2(\lambda+\mu c_3)^2}{(-\nu-\omega c_3)^2} & 4\frac{c_4^2(c_2-c_4)^2(\lambda+\mu c_4)^2}{(-\nu-\omega c_4)^2} & 4\frac{c_5^2(c_2-c_5)^2(\lambda+\mu c_5)^2}{(-\nu-\omega c_5)^2} & 4p_6^2
\end{array}\right)
\begin{pmatrix} b_3\\ b_4\\ b_5\\ b_6 \end{pmatrix}
=
\begin{pmatrix} \frac{1}{2}\,c_2 - \frac{1}{3}\\ \frac{1}{3}\,c_2 - \frac{1}{4}\\ \frac{1}{4}\,c_2 - \frac{1}{5}\\ \frac{1}{3}\\ 0\\ 0\\ \frac{1}{5} \end{pmatrix},
\tag{3.61}
$$

that comes from incorporating the results from the discussion earlier in this section (see the discussion about how the system (3.30) is formed). Because $F_6 = G_3 = 0$, rows 5 and 6 now both vanish because they are linear combinations of other rows. The first four rows of (3.61) can be solved for $\tilde{\mathbf{b}}$, therefore leaving only row 7 as an additional constraint. Row 7 of (3.61) is exactly the constraint (3.58c) given by Cassity after both are expanded and compared using Sage [57]. Case VI involves exactly the solution to either subcase of Case V, except $c_6 \neq 1$ and two additional constraints are required; they are described below in Section 3.4.2 because they do not involve quadrature conditions and non-branching conditions of height two. However, it can clearly be seen from (3.58c) that, despite the complicated-looking determinant, with c_2 as a constraint, it can be factored into two sub-cases that each give a different possible constraint on c_2. The explicit constraints on c_2 are

$$\begin{aligned}
c_2 =& \frac{N}{D}, \qquad (3.62)\\
N =& 2000c_3c_4c_5c_6\lambda^2 + 1800c_3c_4c_5c_6\lambda\mu + 480c_3c_4c_5c_6\mu^2 - 1500c_3c_4c_5\lambda^2 - 1500c_3c_4c_6\lambda^2 - 1500c_3c_5c_6\lambda^2 -\\
& 1500c_4c_5c_6\lambda^2 - 1200c_3c_4c_5\lambda\mu - 1200c_3c_4c_6\lambda\mu - 1200c_3c_5c_6\lambda\mu - 1200c_4c_5c_6\lambda\mu - 270c_3c_4c_5\mu^2 -\\
& 270c_3c_4c_6\mu^2 - 270c_3c_5c_6\mu^2 - 270c_4c_5c_6\mu^2 + 1200c_3c_4\lambda^2 + 1200c_3c_5\lambda^2 + 1200c_4c_5\lambda^2 + 1200c_3c_6\lambda^2 +\\
& 900c_3c_5\lambda\mu + 900c_4c_5\lambda\mu + 900c_3c_6\lambda\mu + 900c_4c_6\lambda\mu + 900c_5c_6\lambda\mu + 180c_3c_4\mu^2 + 180c_3c_5\mu^2 + 180c_4c_5\mu^2 +\\
& 180c_3c_6\mu^2 + 1200c_4c_6\lambda^2 + 1200c_5c_6\lambda^2 + 900c_3c_4\lambda\mu + 180c_4c_6\mu^2 + 180c_5c_6\mu^2 - 990c_3\lambda^2 - 990c_4\lambda^2 -\\
& 990c_5\lambda^2 - 990c_6\lambda^2 - 720c_3\lambda\mu - 720c_4\lambda\mu - 720c_5\lambda\mu - 720c_6\lambda\mu - 135c_3\mu^2 - 135c_4\mu^2 - 135c_5\mu^2 -\\
& 135c_6\mu^2 + 828\lambda^2 + 594\lambda\mu + 108\mu^2,\\
D =& 5000c_3c_4c_5c_6\lambda^2 + 4000c_3c_4c_5c_6\lambda\mu + 900c_3c_4c_5c_6\mu^2 - 4000c_3c_4c_5\lambda^2 - 4000c_3c_4c_6\lambda^2 - 4000c_3c_5c_6\lambda^2 -\\
& 4000c_4c_5c_6\lambda^2 - 3000c_3c_4c_5\lambda\mu - 3000c_3c_4c_6\lambda\mu - 3000c_3c_5c_6\lambda\mu - 3000c_4c_5c_6\lambda\mu - 600c_3c_4c_5\mu^2 -\\
& 600c_3c_4c_6\mu^2 - 600c_3c_5c_6\mu^2 - 600c_4c_5c_6\mu^2 + 3300c_3c_4\lambda^2 + 3300c_3c_5\lambda^2 + 3300c_4c_5\lambda^2 + 3300c_3c_6\lambda^2 +\\
& 3300c_4c_6\lambda^2 + 3300c_5c_6\lambda^2 + 2400c_3c_4\lambda\mu + 2400c_3c_5\lambda\mu + 2400c_4c_5\lambda\mu + 2400c_3c_6\lambda\mu + 2400c_4c_6\lambda\mu +\\
& 2400c_5c_6\lambda\mu + 450c_3c_4\mu^2 + 450c_3c_5\mu^2 + 450c_4c_5\mu^2 + 450c_3c_6\mu^2 + 450c_4c_6\mu^2 + 450c_5c_6\mu^2 - 2760c_3\lambda^2 -\\
& 2760c_4\lambda^2 - 2760c_5\lambda^2 - 2760c_6\lambda^2 - 1980c_3\lambda\mu - 1980c_4\lambda\mu - 1980c_5\lambda\mu - 1980c_6\lambda\mu - 360c_3\mu^2 -\\
& 360c_4\mu^2 - 360c_5\mu^2 - 360c_6\mu^2 + 2322\lambda^2 + 1656\lambda\mu + 297\mu^2,
\end{aligned}$$

leading to Case VIa, where substituting $c_6 = 1$ gives

$$\begin{aligned}
c_2 =& \frac{N}{D}, \qquad (3.63)\\
N =& 500\,c_3c_4c_5\lambda^2 + 600\,c_3c_4c_5\lambda\mu + 210\,c_3c_4c_5\mu^2 - 300\,c_3c_4\lambda^2 - 300\,c_3c_5\lambda^2 - 300\,c_4c_5\lambda^2 - 300\,c_3c_4\lambda\mu -\\
& 300\,c_3c_5\lambda\mu - 300\,c_4c_5\lambda\mu - 90\,c_3c_4\mu^2 - 90\,c_3c_5\mu^2 - 90\,c_4c_5\mu^2 + 210\,c_3\lambda^2 + 210\,c_4\lambda^2 + 210\,c_5\lambda^2 +\\
& 180\,c_3\lambda\mu + 180\,c_4\lambda\mu + 180\,c_5\lambda\mu + 45\,c_3\mu^2 + 45\,c_4\mu^2 + 45\,c_5\mu^2 - 162\,\lambda^2 - 126\,\lambda\mu - 27\,\mu^2,\\
D =& 1000\,c_3c_4c_5\lambda^2 + 1000\,c_3c_4c_5\lambda\mu + 300\,c_3c_4c_5\mu^2 - 700\,c_3c_4\lambda^2 - 700\,c_3c_5\lambda^2 - 700\,c_4c_5\lambda^2 - 600\,c_3c_4\lambda\mu -\\
& 600\,c_3c_5\lambda\mu - 600\,c_4c_5\lambda\mu - 150\,c_3c_4\mu^2 - 150\,c_3c_5\mu^2 - 150\,c_4c_5\mu^2 + 540\,c_3\lambda^2 + 540\,c_4\lambda^2 + 540\,c_5\lambda^2 +\\
& 420\,c_3\lambda\mu + 420\,c_4\lambda\mu + 420\,c_5\lambda\mu + 90\,c_3\mu^2 + 90\,c_4\mu^2 + 90\,c_5\mu^2 - 438\,\lambda^2 - 324\,\lambda\mu - 63\,\mu^2,
\end{aligned}$$

that leads to Case Va and

$$c_2 = \frac{\lambda}{4\,\lambda + \mu}, \qquad (3.64)$$

which corresponds to $\nu = 0$ from (3.58c) and leads to Cases Vb and VIb. For Case V when $c_6 = 1$ is substituted directly into the condition corresponding to the bottom row of (3.61), the SAGE CAS easily simplifies and factors this expression into (3.63) and (3.64). However, for Case VI with $c_6 \neq 1$ as a free parameter, a significant amount of manual algebraic manipulation and simplification is required before the SAGE CAS can simplify and factor the bottom row of (3.61) into (3.62) and (3.64). This clearly demonstrates that careful analysis of the order conditions is still necessary despite contemporary access to powerful software for computer algebra.

3.4.2 Solving the taller trees

Continuing to follow Cassity's described solution procedure [32], once the quadrature conditions and non-branching conditions of height two (3.27a) as well as the one-leg order conditions (3.27b) are satisfied, a straightforward procedure ensures that the other order conditions are satisfied (similar to how relatively straightforward solving (3.13) for the restricted system with $b_2 = 0$ is in Section 3.3).

The linear system (3.27c) is under-determined with two equations and three unknowns. Furthermore, the two rows in (3.27c) are identical to rows 1 and 3 of (3.27b). Therefore, (3.27c) must be solved considering the solutions to (3.27b) to ensure that all solutions are consistent with both. Following Cassity's procedure [32], the solutions of the over-determined system (3.27b) can be used to find the unique consistent solution to the under-determined system (3.27c) by subtracting (3.27c) from c_6 times (3.27b) to eliminate b_6 and allowing the vector $(\tilde{\mathbf{r}}c_6 - \tilde{\mathbf{s}})$ of size 2 to be solved for. This subtraction gives the system

$$\left(\begin{array}{ccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) \\ \hline 2p_3 & 2p_4 & 2p_5 \end{array}\right)(\tilde{\mathbf{r}}c_6 - \tilde{\mathbf{s}}) = \begin{pmatrix} \frac{1}{6}c_2 - \frac{1}{12} \\ \frac{1}{12} \end{pmatrix} c_6 - \frac{1}{2}\begin{pmatrix} \frac{1}{4}c_2 - \frac{2}{15} \\ \frac{2}{15} \end{pmatrix},$$

that simplifies to

$$\left(\begin{array}{cc} c_3(c_2-c_3) & c_4(c_2-c_4) \\ \hline 2p_3 & 2p_4 \end{array}\right)\begin{pmatrix} (r_3c_6 - s_3) \\ (r_4c_6 - s_4) \end{pmatrix} = \begin{pmatrix} \frac{1}{6}c_2 - \frac{1}{12} \\ \frac{1}{12} \end{pmatrix} c_6 - \frac{1}{2}\begin{pmatrix} \frac{1}{4}c_2 - \frac{2}{15} \\ \frac{2}{15} \end{pmatrix}, \tag{3.65}$$

because the substitutions (3.25) give $s_5 = r_5c_6$, eliminating the third column and third unknown s_5 from immediate consideration. When $c_6 = 1$, a consistent solution results in $s_5 = r_5$. When $c_6 \neq 1$ for Case VI, a consistent solution results in $s_5 = r_5c_6$.

When $c_6 = 1$, applying the variable substitutions (3.25) and the D(1) simplifying assumptions (2.44c), which always hold for six-stage fifth-order ERK when $c_6 = 1$ (see the discussion concerning (3.32)), then $(r_3 - s_3) = b_4(1-c_4)\,a_{4,3} + b_5(1-c_5)\,a_{5,3} = r_4\,a_{4,3} + r_5\,a_{4,3}$ and $(r_4 - s_5) = b_5(1-c_5)\,a_{5,4} = r_5\,a_{5,4}$. These are exactly the unknown expressions for the final remaining linear system (3.27d). This shows that when $c_6 = 1$ this final remaining linear system (3.27d) is equivalent to the already satisfied linear systems (3.27b) and (3.27c).

When $c_6 = 1$, all order conditions in (3.27) are now satisfied. However, if $c_6 \neq 1$, the final remaining linear system (3.27d) requires additional consideration.

Case VI with $c_6 \neq 1$

When $c_6 \neq 1$, (3.27b) and (3.27c) are no longer equivalent to the remaining linear system (3.27d). Therefore with $c_6 \neq 1$, (3.27d) becomes two further conditions that must be satisfied. The c_6 coefficient becomes a new free parameter, and two other free parameters must be used to ensure both conditions represented by (3.27d) are satisfied.

Similar to when $c_6 \neq 1$ for the restricted case with $b_2 = 0$ described in Section 3.3.2, the free parameters best used to satisfy (3.27d) are c_4 and $a_{6,3}$ because the constraints are rational expressions. Both conditions of (3.27d) are linear in $a_{6,3}$. Arbitrarily choosing the second condition of (3.27d), this expression can be solved for $a_{6,3}$ and substituted into the first condition of (3.27d). This first condition of (3.27d) then simplifies to

$$\begin{aligned}&\big(\lambda^2(200c_3^2c_4 - 240c_3c_4 + 90c_4 + 30c_3 - 36)\\ &+\lambda\mu(240c_3^2c_4 - 228c_3c_4 + 72c_4 + 30c_3^2 - 36c_3 - 9)\\ &+\mu^2(90c_3^2c_4 - 72c_3c_4 + 18c_4 - 9c_3)\big)\\ &c_2(c_3 - c_4)(c_6 - 1) = 0,\end{aligned} \tag{3.66}$$

where the first factor is exactly the constraint given by Cassity, $c_2 = 0$ and $(c_3 - c_4) = 0$ do not give valid six-stage fifth-order ERK methods for reasons already discussed, and $(c_6 - 1) = 0$ corresponds to Case V. The first factor of (3.66) is best solved with c_4 as a constraint and leaving c_3 as a free parameter, because c_4 only appears linearly. This expression for c_4 can be back-substituted into the expression for $a_{6,3}$ found from the second condition of (3.27d) to obtain an expression for $a_{6,3}$ only in terms of the free parameters. Note that deriving the expression (3.66) for Case VI is one of the more challenging expressions to find with the Sage CAS. Although there was little manipulation required, the final procedure still required more than an hour of CPU time to do the appropriate simplification and factoring to obtain the expression (3.66).

3.4.3 Solving for the Butcher tableau A matrix

All the solutions of the order conditions in CMF (3.27) that are required to find the **A** matrix are now known, i.e., $\tilde{\mathbf{b}}$, $\tilde{\mathbf{r}}$, and $\tilde{\mathbf{s}}$, that give the linear systems

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & c_2 & 0 & 0 \\ 0 & 0 & b_5(c_6 - c_5) & 0 \\ 0 & 0 & 0 & b_6 \end{pmatrix} \begin{pmatrix} a_{2,1} \\ a_{3,2} \\ a_{5,4} \\ a_{6,5} \end{pmatrix} = \begin{pmatrix} c_2 \\ p_3 \\ r_4c_6 - s_4 \\ r_5 \end{pmatrix},$$

$$\begin{pmatrix} c_2 & c_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & c_2 & c_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & c_2 & c_4 \\ 0 & b_4 & 0 & b_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & b_6 \\ 0 & b_4c_4 & 0 & b_5c_5 & 0 & 0 \end{pmatrix} \begin{pmatrix} a_{4,2} \\ a_{4,3} \\ a_{5,2} \\ a_{5,3} \\ a_{6,2} \\ a_{6,4} \end{pmatrix} = \begin{pmatrix} p_4 \\ p_5 - a_{5,4}c_4 \\ p_6 - a_{6,5}c_5 - a_{6,3}c_3 \\ r_3 - a_{6,3}b_6 \\ r_4 - b_5a_{5,4} \\ s_3 - a_{6,3}b_6c_6 \end{pmatrix},$$

where $a_{6,3}$ is chosen as a free parameter in the **A** matrix, except for Case VI where it is determined from the procedure described above. Similarly to Section 3.3.3, the row-sum conditions (2.35) give the system

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_{3,1} \\ a_{4,1} \\ a_{5,1} \\ a_{6,1} \end{pmatrix} = \begin{pmatrix} c_3 - a_{3,2} \\ c_4 - a_{4,2} - a_{4,3} \\ c_5 - a_{5,2} - a_{5,3} - a_{5,4} \\ c_6 - a_{6,2} - a_{6,3} - a_{6,4} - a_{6,5} \end{pmatrix},$$

that can be solved for the first column because all the $a_{i,j}$ quantities outside of the first column of the **A** matrix are now known.

3.4.4 Procedure for finding the complete solution

A summary of the procedure for finding the complete solution is as follows:

1. Choose the appropriate free parameters for the particular case according to Table 3.3.
2. Solve for the p_3, p_4, p_5 using the appropriate formula for the particular case from Section 3.4.1.
3. Solve for the fourth column of (3.27a).
   (a) For Cases I (restricted), II (restricted), III: solve for b_6, p_6 for the appropriate case as described in Section 3.4.1.
   (b) For Cases I, II, IV: solve for b_6 as described in Section 3.4.1.
   (c) For Case V, VI: solve for p_6 as described in Section 3.4.1.
4. Solve for b_i, $i \in \{3, 4, 5\}$.
   (a) For Cases I, II, III, IV: solve for b_i, $i \in \{3, 4, 5\}$ using the first three rows of the quadrature conditions from (3.27a).
   (b) For Cases V, VI: solve for b_i, $i \in \{3, 4, 5\}$ using the first four rows of the quadrature conditions from (3.27a).
5. Solve for the more complex constraints from (3.27a) if they are not precomputed expressions, and back-substitute where necessary.

   (a) For Cases I, II, IV: solve for p_6, c_5 as described in Section 3.4.1.

   (b) For Cases I (restricted), II (restricted), III: solve for c_5 as described in Section 3.4.1.

   (c) For Cases V, VI: solve for c_2 as described in Section 3.4.1.

6. Solve for $\tilde{\mathbf{r}}$ from three of the rows of the now consistent linear system of the one-legged trees (3.27b).

   (a) For Cases I, II, III, V, VI: rows 1, 2, 3 of (3.27b) are used because it can easily be seen that $\omega \neq 0$ means that the last row is always linearly dependent on the first three rows.

   (b) For Case IV: rows 1, 3, and 4 of (3.27b) are used in order to incorporate row 4 because when $\omega = 0$, row 4 is not linearly dependent on the other rows and must be considered.

7. Solve for $(\tilde{\mathbf{r}}c_6 - \tilde{\mathbf{s}})$ from (3.65) in order to satisfy (3.27c).

   - Additionally for Case VI: find constraints for c_4 and $a_{6,3}$ as described in Section 3.4.2.

8. Solve for the $\mathbf{A}$ matrix of the Butcher tableau (2.34) using the linear systems in Section 3.4.3.

9. Solve for b_1 and b_2 using (3.26).

Table 3.3: Cases and free parameters for the complete solution to six-stage fifth-order ERK methods. N/S means that the associated tableaux constructed by the author took up too much space to include in this thesis.

Case	Free parameters	Subcases	Example
I (Cassity's restricted)	$a_{6,3}$, c_3, c_4, $\frac{\lambda}{\omega}$	1	(A.8)
I (full)	$a_{6,3}$, c_2, c_3, c_4, $\frac{\lambda}{\omega}$	2	N/S
II (Cassity's restricted)	$a_{6,3}$, c_2, c_3, c_4	1	(A.9)
II (full)	$a_{6,3}$, c_2, c_3, c_4, $\frac{\mu}{\omega}$	2	N/S
III	$a_{6,3}$, c_2, c_3, c_4, $\frac{\lambda}{\mu}$	1	(A.10)
IV	$a_{6,3}$, c_2, c_3, c_4, $\frac{\lambda}{\mu}$	2	N/S
V	$a_{6,3}$, c_2, c_3, c_5, $\frac{\lambda}{\mu}$	2	(A.11) (A.12)
VI	c_2, c_3, c_6, $\frac{\lambda}{\mu}$	2	(A.13) (A.14)
V $b_2 = 0$ with C(2) from Section 3.3	$a_{6,3}$, c_2, c_3, c_4, c_5	1	(A.1)
VI $b_2 = 0$ with C(2) from Section 3.3	c_2, c_3, c_5, c_6	1	(A.2)
V $b_2 = 0$ w/o C(2) from Section 3.3	$a_{6,3}$, c_2, c_3, c_4, c_5	1	(A.3)
VI $b_2 = 0$ w/o C(2) from Section 3.3	c_2, c_3, c_5, c_6	1	(A.4)

3.4.5 The complete solution and standard simplifying assumptions

It can clearly be seen that for Cases V and VI that if $\lambda = 0$ and the expressions for ν and ω are substituted into (3.60), then it immediately simplifies to $p_i = \frac{c_i^2}{2}$, $i \in \{3,4,5\}$, i.e., the C(2) simplifying assumptions (2.44b). Based on the discussion in Section 2.7.1, this leads immediately to $b_2 = 0$, which requires considering the restricted case that is presented in Section 3.3. If $b_2 = 0$, the linear systems giving the order conditions in CMF (3.27) for $b_2 \neq 0$ do not have the same ranks that are assumed for the procedure given in this section. Therefore, the CMF used for the special case of $b_2 = 0$ in Section 3.3 is required.

3.5 $5(4)_6$ ERK pairs without C(2) simplifying assumptions

The embedded pairs without the C(2) simplifying assumptions (2.44b) that are derived in this section are not part of Cassity's work [31, 32] at all or found anywhere else in the current literature. Constructing an embedded pair using the complete solution of six-stage fifth-order ERK order conditions without the C(2) simplifying assumptions (2.44b), as already mentioned at the beginning of this chapter, was a known open problem in the study of numerical methods [128].

A summary of the constructed families with $b_2 \neq 0$, free parameters and examples of Butcher tableaux are given in Table 3.4 for embedded pairs. For comparison purposes, the constructed families with $b_2 = 0$ are also summarized in these tables just mentioned.

Analogous to when constructing the six-stage fifth-order ERK method, when constructing the fourth-order method of a corresponding pair this Section employs the notation $\tilde{\hat{\mathbf{b}}} = \{\hat{b}_3, \hat{b}_4, \hat{b}_5, \hat{b}_6\}$ for the six-stage fourth-order component of an embedded pair or $\tilde{\hat{\mathbf{b}}} = \{\hat{b}_3,\ \hat{b}_4,\ \hat{b}_5,\ \hat{b}_6,\ \hat{b}_7\}$ for the seven-stage fourth-order component of an embedded pair. The corresponding variable substitution to $\tilde{\mathbf{r}}$ when constructing the six-stage fourth-order component method of a $5(4)_{6(6)}$ ERK pair is

$$\left\{\begin{array}{lll} \hat{b}_4\, a_{4,3} & +\hat{b}_5\, a_{5,3} & +\hat{b}_6\, a_{6,3} \\ & +\hat{b}_5\, a_{5,4} & +\hat{b}_6\, a_{6,4} \\ & & +\hat{b}_6\, a_{6,5} \end{array}\right\} = r_i, \quad i \in \{3,4,5\} = \tilde{\hat{\mathbf{r}}}.$$

The corresponding variable substitution for $\tilde{\mathbf{r}}$ for the seven-stage fourth-order component of a $5(4)_{6(7)}$ ERK pair is

$$\left\{\begin{array}{llll} \hat{b}_4\, a_{4,3} & +\hat{b}_5\, a_{5,3} & +\hat{b}_6\, a_{6,3} & +\hat{b}_7\, b_3 \\ & +\hat{b}_5\, a_{5,4} & +\hat{b}_6\, a_{6,4} & +\hat{b}_6\, b_4 \\ & & +\hat{b}_6\, a_{6,5} & +\hat{b}_6\, b_5 \\ & & & +\hat{b}_7\, b_6 \end{array}\right\} = r_i, \quad i \in \{3,4,5,6\} = \tilde{\hat{\mathbf{r}}}.$$

These two possible definitions of $\tilde{\hat{\mathbf{r}}}$ for either six- or seven-stage fourth-order components of $5(4)_6$ ERK pairs can always be distinguished by context.

3.5.1 $5(4)_{6(6)}$ ERK pairs without the C(2) simplifying assumptions

An effective $5(4)_{6(6)}$ ERK pair requires finding $\hat{\mathbf{b}} \neq \mathbf{b}$ from the linear system consisting of the order conditions $[\tau^2]$, $[\tau^3]$, $[[\tau]]$, $[\tau[\tau]]$ that are given in CMF by

$$\left(\begin{array}{cccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & c_6(c_2-c_6) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & c_6^2(c_2-c_6) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \\ 2c_3p_3 & 2c_4p_4 & 2c_5p_5 & 2c_6p_6 \end{array}\right) \tilde{\hat{\mathbf{b}}} = \begin{pmatrix} \frac{1}{2}c_2 - \frac{1}{3} \\ \frac{1}{3}c_2 - \frac{1}{4} \\ \frac{1}{3} \\ \frac{1}{4} \end{pmatrix}, \tag{3.67}$$

where the rows come from the same subtractions and multiplications used for the corresponding order conditions to give (3.27a). The linear system (3.67) is only singular for Cases V and VI, and therefore only gives a valid embedded pair with $\hat{\mathbf{b}} \neq \mathbf{b}$ for these cases because it can easily be seen that $F_6 = G_3 = 0$ is a necessary condition for the coefficient matrix of (3.67) to have a determinant that is zero, and therefore be singular. For Cases I, II, III, IV, the non-singular nature of the coefficient matrix of (3.67), which is because none of the rows are linearly dependent due to distinct $\mathbf{c}$ components with $F_6 \neq 0$ and $G_3 \neq 0$, means that any $\hat{\mathbf{b}}$ vector satisfying (3.67) must equal the already computed $\mathbf{b}$ vector and will not meet the requirements for an effective embedded pair. Considering Cases V and VI, taking one of the components of $\tilde{\hat{\mathbf{b}}}$ as a free parameter that is distinct from the corresponding component of the $\mathbf{b}$ vector is necessary to ensure the $5(4)_{6(6)}$ ERK pair consists of two distinct methods. It can easily be seen that appropriate expressions to satisfy τ and $[\tau]$ are given by

$$\sum_{1}^{6} \hat{b}_i = 1, \quad \sum_{i=2}^{6} \hat{b}_i\, c_i = \frac{1}{2}, \tag{3.68}$$

and, analogous to the construction already seen of single fifth-order methods, can be disregarded until the end of method construction.

Case V with $c_6 = 1$

There are still two taller fourth-order trees left after (3.67) and (3.68) are satisfied. These correspond to the trees $[[\tau^2]]$, $[[[\tau]]]$ given in CMF by the linear system

$$\left(\begin{array}{ccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) \\ \hline 2p_3 & 2p_4 & 2p_5 \end{array}\right) \tilde{\hat{\mathbf{r}}} = \begin{pmatrix} \frac{1}{6}c_2 - \frac{1}{12} \\ \frac{1}{12} \end{pmatrix}, \tag{3.69}$$

where the equation corresponding to each row is formed from the same subtractions and multiplications used for the corresponding order conditions (3.27b) for the fifth-order component. The only solution of (3.69) found consistent with (3.27d) for Case VI is $\hat{b}_6 = 0$. Notice that, similar to the corresponding situation constructing embedded pairs in Section 3.3.4, most of the complexity in expanding (3.69) comes from the $\hat{\mathbf{r}}$ vector because it incorporates expressions for coefficients of the $\mathbf{A}$ matrix in terms of the free parameters.

For Case V when $c_6 = 1$, solving the equation corresponding to the second row of (3.69) for $a_{6,3}$, which appears linearly in the equations corresponding to both rows, gives one constraint on the $a_{6,3}$ free parameter of the fifth-order method (this expression is not given here due to its size). Using this value of $a_{6,3}$ and substituting it into the first row of (3.69), then factoring using SAGE [57], gives an expression equivalent to the first row of (3.69) of

$$
\begin{aligned}
&\Big(1800\,\hat{b}_6c_2c_3c_4c_5\lambda^2 + 2160\,\hat{b}_6c_2c_3c_4c_5\lambda\mu + 540\,\hat{b}_6c_2c_3c_4c_5\mu^2 - 1800\,\hat{b}_6c_2c_3c_4\lambda^2 - 1800\,\hat{b}_6c_2c_3c_5\lambda^2 - \\
&1800\,\hat{b}_6c_2c_4c_5\lambda^2 - 1800\,\hat{b}_6c_3c_4c_5\lambda^2 - 400\,c_2c_3c_4c_5\lambda^2 - 2160\,\hat{b}_6c_2c_3c_4\lambda\mu - 2160\,\hat{b}_6c_2c_3c_5\lambda\mu - \\
&2160\,\hat{b}_6c_2c_4c_5\lambda\mu - 2160\,\hat{b}_6c_3c_4c_5\lambda\mu - 780\,c_2c_3c_4c_5\lambda\mu - 540\,\hat{b}_6c_2c_3c_4\mu^2 - 540\,\hat{b}_6c_2c_3c_5\mu^2 - 540\,\hat{b}_6c_2c_4c_5\mu^2 - \\
&540\,\hat{b}_6c_3c_4c_5\mu^2 - 270\,c_2c_3c_4c_5\mu^2 + 1800\,\hat{b}_6c_2c_3\lambda^2 + 1800\,\hat{b}_6c_2c_4\lambda^2 + 1800\,\hat{b}_6c_3c_4\lambda^2 + 300\,c_2c_3c_4\lambda^2 + \\
&1800\,\hat{b}_6c_2c_5\lambda^2 + 1800\,\hat{b}_6c_3c_5\lambda^2 + 300\,c_2c_3c_5\lambda^2 + 1800\,\hat{b}_6c_4c_5\lambda^2 + 300\,c_2c_4c_5\lambda^2 + 600\,c_3c_4c_5\lambda^2 + \\
&2160\,\hat{b}_6c_2c_3\lambda\mu + 2160\,\hat{b}_6c_2c_4\lambda\mu + 2160\,\hat{b}_6c_3c_4\lambda\mu + 570\,c_2c_3c_4\lambda\mu + 2160\,\hat{b}_6c_2c_5\lambda\mu + 2160\,\hat{b}_6c_3c_5\lambda\mu + \\
&570\,c_2c_3c_5\lambda\mu + 2160\,\hat{b}_6c_4c_5\lambda\mu + 570\,c_2c_4c_5\lambda\mu + 870\,c_3c_4c_5\lambda\mu + 540\,\hat{b}_6c_2c_3\mu^2 + 540\,\hat{b}_6c_2c_4\mu^2 + 540\,\hat{b}_6c_3c_4\mu^2 + \\
&180\,c_2c_3c_4\mu^2 + 540\,\hat{b}_6c_2c_5\mu^2 + 540\,\hat{b}_6c_3c_5\mu^2 + 180\,c_2c_3c_5\mu^2 + 540\,\hat{b}_6c_4c_5\mu^2 + 180\,c_2c_4c_5\mu^2 + 270\,c_3c_4c_5\mu^2 - \\
&1800\,\hat{b}_6c_2\lambda^2 - 1800\,\hat{b}_6c_3\lambda^2 - 240\,c_2c_3\lambda^2 - 1800\,\hat{b}_6c_4\lambda^2 - 240\,c_2c_4\lambda^2 - 450\,c_3c_4\lambda^2 - 450\,1800\,\hat{b}_6c_5\lambda^2 - \\
&240\,c_2c_5\lambda^2 - c_3c_5\lambda^2 - 450\,c_4c_5\lambda^2 - 2160\,\hat{b}_6c_2\lambda\mu - 2160\,\hat{b}_6c_3\lambda\mu - 450\,c_2c_3\lambda\mu - 2160\,\hat{b}_6c_4\lambda\mu - 450\,c_2c_4\lambda\mu - \\
&630\,c_3c_4\lambda\mu - 2160\,\hat{b}_6c_5\lambda\mu - 450\,c_2c_5\lambda\mu - 630\,c_3c_5\lambda\mu - 630\,c_4c_5\lambda\mu - 540\,\hat{b}_6c_2\mu^2 - 540\,\hat{b}_6c_3\mu^2 - 135\,c_2c_3\mu^2 - \\
&540\,\hat{b}_6c_4\mu^2 - 135\,c_2c_4\mu^2 - 180\,c_3c_4\mu^2 - 540\,\hat{b}_6c_5\mu^2 - 135\,c_2c_5\mu^2 - 180\,c_3c_5\mu^2 - 180\,c_4c_5\mu^2 + 1800\,\hat{b}_6\lambda^2 + \\
&198\,c_2\lambda^2 + 360\,c_3\lambda^2 + 360\,c_4\lambda^2 + 360\,c_5\lambda^2 + 2160\,\hat{b}_6\lambda\mu + 369\,c_2\lambda\mu + 495\,c_3\lambda\mu + 495\,c_4\lambda\mu + 495\,c_5\lambda\mu + \\
&540\,\hat{b}_6\mu^2 + 108\,c_2\mu^2 + 135\,c_3\mu^2 + 135\,c_4\mu^2 + 135\,c_5\mu^2 - 297\,\lambda^2 - 405\,\lambda\mu - 108\,\mu^2\Big) \qquad \text{(3.70a)} \\
&\big(200\,c_3^2c_4\lambda^2 + 240\,c_3^2c_4\lambda\mu + 90\,c_3^2c_4\mu^2 - 240\,c_3c_4\lambda^2 + 30\,c_3^2\lambda\mu - 228\,c_3c_4\lambda\mu - 72\,c_3c_4\mu^2 + 30\,c_3\lambda^2 + 90\,c_4\lambda^2 - \\
&36\,c_3\lambda\mu + 72\,c_4\lambda\mu - 9\,c_3\mu^2 + 18\,c_4\mu^2 - 36\,\lambda^2 - 9\,\lambda\mu\big)\,c_2 \qquad \text{(3.70b)} \\
&= 0,
\end{aligned}
$$

where the factor (3.70a) is the only one involving $\hat{b}_6$, and solving for $\hat{b}_6$ when this factor vanishes implies $b_6 = \hat{b}_6$. The other factor (3.70b) (the value in the parenthesis, but excluding the c_2 outside the parenthesis because c_2 cannot be vanish for reasons already discussed) contains c_4 linearly and leads to a constraint on c_4 of

$$
c_4 = \frac{-3(10c_3\lambda - 12\lambda - 3\mu)(c_3\mu + \lambda)}{2(100c_3^2\lambda^2 + 120c_3^2\lambda\mu + 45c_3^2\mu^2 - 120c_3\lambda^2 - 114c_3\lambda\mu - 36c_3\mu^2 + 45\lambda^2 + 36\lambda\mu + 9\mu^2)},
$$

that along with the already mentioned expression for $a_{6,3}$ gives the necessary conditions for an embedded pair for Case V without the C(2) simplifying assumptions (2.44b).

The resulting constraints are more complex than for the corresponding embedded pair from the restricted system with the C(2) simplifying assumptions (2.44b) in Section 3.3. However, analogous to the case in Section 3.3, factoring out the expressions involving $\hat{b}_6$ similarly allows finding an embedded pair.

Case VI with $c_6 \neq 1$

The only consistent fourth-order component of a pair for Case VI with $c_6 \neq 1$ and without the C(2) simplifying assumptions (2.44b) is when $\hat{b}_6 = 0$. The implications of not having a free parameter for the fourth-order component of an ERK pair are further discussed in Chapters 4 and 5.

3.5.2 $5(4)_{6(7)}$ ERK pairs without the C(2) simplifying assumptions

The complete solution of the six-stage fifth-order ERK order conditions also allows for constructing $5(4)_{6(7)}$ ERK pairs without the C(2) simplifying assumptions (2.44b) when using cases I, II, III, VI from Section 3.4.1. The FSAL operating mode requires $c_6 = c_7 = 1$ and a distinct $\hat{\mathbf{b}} \neq \mathbf{b}$ vector that satisfies the linear system consisting of the order conditions $[\tau^2]$, $[\tau^3]$, $[[\tau]]$, $[\tau[\tau]]$ given in CMF by

$$\left(\begin{array}{ccccc} c_3(c_2-c_3) & c_4(c_2-c_4) & c_5(c_2-c_5) & (c_2-1) & (c_2-1) \\ c_3^2(c_2-c_3) & c_4^2(c_2-c_4) & c_5^2(c_2-c_5) & (c_2-1) & (c_2-1) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 & 2p_7 \\ 2c_3p_3 & 2c_4p_4 & 2c_5p_5 & 2p_6 & 2p_7 \end{array}\right) \tilde{\hat{\mathbf{b}}} = \begin{pmatrix} \frac{1}{2}c_2 - \frac{1}{3} \\ \frac{1}{3}c_2 - \frac{1}{4} \\ \frac{1}{3} \\ \frac{1}{4} \end{pmatrix}, \tag{3.71}$$

where for Case V (there are no Case VI $5(4)_{6(7)}$ ERK pairs because $c_6 = 1$ is required for the FSAL operating mode) it appears there is the possibility of two free parameters. However, for (3.71) to be consistent, the augmented matrix must be the same rank as the coefficient matrix [163, pg.62], which for Case V is rank 3 without the C(2) simplifying assumptions (2.44b) and rank 2 with the C(2) simplifying assumptions (2.44b). It can be easily seen that the seventh row of the Butcher tableau is the $\mathbf{b}$ vector for a $5(4)_{6(7)}$ ERK pair, consequently $p_7 = \sum b_i c_i = \frac{1}{2}$ because this expression for the seventh row corresponds exactly to the $[\tau]$ order condition. Determining the rank of the augmented matrix of the linear system (3.71) requires looking at determinants known as *basis minors*; see the references [163, pgs.58–59] for more information. The determinants of all combinations of four columns of the augmented matrix of (3.71) must vanish if the augmented matrix is to have rank 3 and the linear system (3.71) is to be consistent. If these determinants do not vanish, then augmented matrix has rank 4 and the linear system (3.71) will not be consistent. Combinations of columns lead to determinants with a common form, an example is the determinant with columns 1, 2, 3, 5 that is given by

$$-3\left(10\,\lambda^2 + 12\,\lambda\mu + 3\,\mu^2\right)\hat{b}_7(c_2-c_3)(c_2-c_4)(c_2-c_5)c_2^2(c_3-c_4)(c_3-c_5)c_3(c_4-c_5)c_4c_5\lambda = 0, \tag{3.72}$$

where c_i, $i > 2$, cannot be zero because this reduces the effective number of stages and the $\mathbf{c}$ vector is assumed distinct; ruling out factors that are differences of the c_i, $\hat{b}_7$ corresponds to a six-stage embedded pair that is addressed in Section 3.5.1, and the factor $\left(10\,\lambda^2 + 12\,\lambda\mu + 3\,\mu^2\right)$ appears on the denominators of the expressions for the $\mathbf{b}$ vector, and therefore it cannot vanish either. All of the other determinants lead to similar conditions that cannot hold except for when $\lambda = 0$. Therefore, to construct a $5(4)_{6(7)}$ ERK pair from Case V requires $\lambda = 0$ that leads to the C(2) simplifying assumptions (2.44b) as a consistency condition, a special case already addressed in Section 3.3.5.

In contrast, the situation with Cases I, II, III, IV is different. Because the RHS coefficient matrix of (3.71) is already rank 4, the augmented matrix is also rank 4, implying that there is no additional consistency condition for (3.71) to obtain a different $\tilde{\hat{\mathbf{b}}}$ vector from the different RHS when $\hat{b}_7$ is a free parameter. Cases I, II, III, IV are covered in this subsection because they allow construction of $5(4)_{6(7)}$ ERK pairs without the C(2) simplifying assumptions (2.44b). Unlike where the C(2) simplifying assumptions (2.44b) are used in Section 3.3.5, in these cases the linear system (3.71) only admits one free parameter for the fourth-order method, and that is taken as $\hat{b}_7$. It can easily be seen that the expressions to satisfy τ and $[\tau]$ for the seven-stage fourth-order embedded pair are given by

$$\sum_{1}^{7} \hat{b}_i = 1, \quad \sum_{i=2}^{7} \hat{b}_i\, c_i = \frac{1}{2}. \tag{3.73}$$

Analogous to the $5(4)_{6(6)}$ ERK pair addressed in Section 3.5, there are still two taller fourth-order trees left after (3.71) and (3.73) are satisfied, corresponding to the trees $[[\tau^2]]$ and $[[[\tau]]]$ given by the linear system

$$\begin{pmatrix} c_3(c_2 - c_3) & c_4(c_2 - c_4) & c_5(c_2 - c_5) & c_6(c_2 - 1) \\ \hline 2p_3 & 2p_4 & 2p_5 & 2p_6 \end{pmatrix} \hat{\mathbf{r}} = \begin{pmatrix} \frac{1}{6}\,c_2 - \frac{1}{12} \\ \frac{1}{12} \end{pmatrix}, \tag{3.74}$$

where the rows are constructed using the same subtractions used for the corresponding order conditions for (3.27b). The specifics of finding a family of $5(4)_{6(7)}$ ERK pairs for each of cases I, II, III, and IV can now be addressed individually.

Case I:

Cassity's restricted Case I using the simplifications leading to (3.38) does allow finding two families of $5(4)_{6(7)}$ ERK pairs. However, experimentation shows that this family of $5(4)_{6(7)}$ ERK pairs does not give methods with even moderately small error coefficients or other desirable properties. Therefore, only the full complete solution of Case I introduced in this thesis is used.

Similar to procedures already described in Sections 3.3.4, 3.3.5, and 3.5, the equation corresponding to the second row of (3.69) can be solved for $a_{6,3}$. Then substituting this expression for $a_{6,3}$ in the equation corresponding to the first row of (3.69) and simplifying gives

$$\left(25c_2c_3\mu - 10c_2\mu - 10c_3\mu + 10c_3\omega + 6\mu - 6\omega\right)\hat{b}_7c_2c_4 = 0,$$

where, as already discussed extensively, $\hat{b}_7 = 0$, $c_2 = 0$, and $c_4 = 0$ do not give effective embedded pairs. Because c_4 cannot be used as a constraint such as it is in most other embedded pair construction in this thesis, arbitrarily choosing to constrain c_3 to satisfy the condition for a $5(4)_{6(7)}$ ERK pair gives the expression

$$c_3 = \frac{2}{5}\frac{((5c_2-3)\mu + 3\omega)}{((5c_2-2)\mu + 2\omega)}.$$

Similar to the construction of other families of $5(4)_{6(7)}$ ERK pairs, the expression for $a_{6,3}$ is too complex to give here.

The seven-stage fourth-order component of the pair does not satisfy any fifth-order conditions. However, the smallest value of the leading error coefficient A^6 is much larger than in other families, making this case difficult to recommend for practical use.

Case II:

Cassity's restricted Case II using the simplifications leading to (3.45) does allow for finding a family of $5(4)_{6(7)}$ ERK pairs. However, experimentation shows that this family of $5(4)_{6(7)}$ ERK pairs does not give methods with relatively small error coefficients or other desirable properties. Therefore, only the full complete solution of Case II introduced in this thesis is used.

Similar to procedures already described in Sections 3.3.4, 3.3.5, and 3.5, the equation corresponding to the second row of (3.69) can be solved for $a_{6,3}$. Then substituting this expression for $a_{6,3}$ in the equation corresponding to the first row of (3.69) and simplifying gives

$$\left(20c_2c_3c_4\lambda\omega - 10c_3^2c_4\omega^2 + 10c_2^2\lambda^2 - 5c_2c_3\lambda\omega + 6c_3c_4\omega^2 + 3c_2\lambda\omega\right)\hat{b}_7c_2 = 0,$$

where, for reasons already discussed, $\hat{b}_7 = 0$ and $c_2 = 0$ do not give effective embedded pairs. Arbitrarily choosing to constrain c_4 (without loss of generality as already described) to satisfy the condition for a $5(4)_{6(7)}$ ERK pair gives the expression

$$c_4 = -\frac{(10c_2^2\lambda^2 - (5c_2c_3 - 3c_2)\lambda\omega)}{(5(4c_2c_3 - 3c_2)\lambda\omega - 2(5c_3^2 - 3c_3)\omega^2)}.$$

Similar to other construction of other families of $5(4)_{6(7)}$ ERK pairs, the expression for $a_{6,3}$ is too complex to give here.

The seven-stage fourth-order component of the pair satisfies the $[[\tau]^2]$ fifth-order condition, which can cause numerical issues, making this case difficult to recommend for practical usage.

Case III:

Case III also allows construction of a family of $5(4)_{6(7)}$ ERK pairs. Similar to procedures already described in Sections 3.3.4, 3.3.5, and 3.5, the equation corresponding to the second row of (3.69) can be solved for $a_{6,3}$. Then substituting this expression for $a_{6,3}$ in the equation corresponding to the first row of (3.69) and simplifying gives

$$\begin{aligned}
&-(200c_2^2c_3^2c_4\lambda^3 + 200c_2^2c_3^2c_4\lambda^2\mu + 20c_2^2c_3^2c_4\lambda\mu^2 - 15c_2^2c_3^2c_4\mu^3 - 200c_2c_3^2c_4\lambda^3 - 30c_2^2c_3^2\lambda^2\mu - 120c_2^2c_3c_4\lambda^2\mu - \\
&\quad 160c_2c_3^2c_4\lambda^2\mu - 15c_2^2c_3^2\lambda\mu^2 - 60c_2^2c_3c_4\lambda\mu^2 - 15c_2c_3^2c_4\lambda\mu^2 - 30c_2^2c_3\lambda^3 - 90c_2^2c_4\lambda^3 + 60c_2c_3c_4\lambda^3 + \\
&\quad 50c_3^2c_4\lambda^3 - 15c_2^2c_3\lambda^2\mu + 15c_2c_3^2\lambda^2\mu - 45c_2^2c_4\lambda^2\mu + 90c_2c_3c_4\lambda^2\mu + 30c_3^2c_4\lambda^2\mu + 18c_2c_3c_4\lambda\mu^2 + 15c_2c_3\lambda^3 + \\
&\quad 45c_2c_4\lambda^3 - 30c_3c_4\lambda^3 - 9c_2c_3\lambda^2\mu + 18c_2c_4\lambda^2\mu - 18c_3c_4\lambda^2\mu - 9c_2\lambda^3) \\
&\quad (800c_2^2c_3^2c_4\lambda^2 + 200c_2^2c_3^2c_4\lambda\mu - 10c_2^2c_3^2c_4\mu^2 - 450c_2^2c_3^2\lambda^2 - 900c_2^2c_3c_4\lambda^2 - 650c_2c_3^2c_4\lambda^2 - 135c_2^2c_3^2\lambda\mu - \\
&\quad 270c_2^2c_3c_4\lambda\mu - 115c_2c_3^2c_4\lambda\mu + 480c_2^2c_3\lambda^2 + 375c_2c_3^2\lambda^2 + 240c_2^2c_4\lambda^2 + 750c_2c_3c_4\lambda^2 + 125c_3^2c_4\lambda^2 + \\
&\quad 150c_2^2c_3\lambda\mu + 75c_2c_3^2\lambda\mu + 75c_2^2c_4\lambda\mu + 150c_2c_3c_4\lambda\mu - 135c_2^2\lambda^2 - 420c_2c_3\lambda^2 - 75c_3^2\lambda^2 - 210c_2c_4\lambda^2 - \\
&\quad 150c_3c_4\lambda^2 - 45c_2^2\lambda\mu - 90c_2c_3\lambda\mu - 45c_2c_4\lambda\mu + 117c_2\lambda^2 + 90c_3\lambda^2 + 45c_4\lambda^2 + 27c_2\lambda\mu - 27\lambda^2) \\
&\quad \hat{b}_7c_2(c_3 - c_4)\lambda = 0,
\end{aligned}$$

where, for reasons already discussed, $\hat{b}_7 = 0$ and $c_2 = 0$ do not give effective embedded pairs, and the second factor appears on the denominator of the expressions for $\tilde{\mathbf{b}}$. Therefore, only the first factor can vanish. Arbitrarily choosing to constrain c_4 (without loss of generality, as already described) to ensure that this first factor vanishes gives the expression

$$\begin{aligned}
c_4 =& \frac{3(10c_2c_3\lambda + 5c_2c_3\mu - 5c_3\lambda + 3\lambda)(c_3\mu + \lambda)c_2\lambda}{D}, \\
D = & (200c_2^2c_3^2\lambda^3 + 200c_2^2c_3^2\lambda^2\mu + 20c_2^2c_3^2\lambda\mu^2 - 15c_2^2c_3^2\mu^3 - 200c_2c_3^2\lambda^3 - 120c_2^2c_3\lambda^2\mu - 160c_2c_3^2\lambda^2\mu - \\
& 60c_2^2c_3\lambda\mu^2 - 15c_2c_3^2\lambda\mu^2 - 90c_2^2\lambda^3 + 60c_2c_3\lambda^3 + 50c_3^2\lambda^3 - 45c_2^2\lambda^2\mu + 90c_2c_3\lambda^2\mu + 30c_3^2\lambda^2\mu + \\
& 18c_2c_3\lambda\mu^2 + 45c_2\lambda^3 - 30c_3\lambda^3 + 18c_2\lambda^2\mu - 18c_3\lambda^2\mu).
\end{aligned}$$

Similar to other construction of other families of $5(4)_{6(7)}$ ERK pairs, the expression for $a_{6,3}$ is too complex to give here.

The seven-stage fourth-order component of the pair satisfies the $[\tau^4]$, $[\tau^2[\tau]]$, $[\tau[\tau^2]]$, $[\tau[[\tau]]]$, $[[\tau]^2]$ fifth-order conditions, which can cause numerical issues making this case difficult to recommend for practical usage.

Case IV:

Similar to procedures already described in Sections 3.3.4, 3.3.5, and 3.5, the equation corresponding to the second row of (3.69) can be solved for $a_{6,3}$. Then substituting this expression for $a_{6,3}$ in the equation corresponding to the first row of (3.69) and simplifying gives

$$-(200c_2^2c_3c_4\lambda^2\mu+100c_2^2c_3c_4\lambda\mu^2+100c_2c_3^2c_4\lambda\mu^2+75c_2c_3^2c_4\mu^3+50c_2^2c_3\lambda^2\mu-100c_2^2c_4\lambda^2\mu-100c_2c_3c_4\lambda^2\mu+$$
$$25c_2^2c_3\lambda\mu^2+25c_2c_3^2\lambda\mu^2-50c_2^2c_4\lambda\mu^2-70c_2c_3c_4\lambda\mu^2-50c_3^2c_4\lambda\mu^2-30c_2c_3c_4\mu^3-30c_3^2c_4\mu^3+50c_2^2\lambda^3+$$
$$25c_2^2\lambda^2\mu+25c_2c_3\lambda^2\mu+75c_2c_4\lambda^2\mu-15c_2c_3\lambda\mu^2+30c_2c_4\lambda\mu^2+30c_3c_4\lambda\mu^2+18c_3c_4\mu^3-15c_2\lambda^2\mu)\hat{b}_7c_2=0,$$

where, as already described, $\hat{b}_7=0$ and $c_2=0$ do not give effective embedded pairs. Choosing to constrain c_4 (without loss of generality as already described) to satisfy the condition for a $5(4)_{6(7)}$ ERK pair gives the expression

$$c_4=\frac{-5(10c_2\lambda+5c_2\mu+5c_3\mu-3\mu)(c_3\mu+\lambda)c_2\lambda}{D},$$
$$D=(200c_2^2c_3\lambda^2+100c_2^2c_3\lambda\mu+100c_2c_3^2\lambda\mu+75c_2c_3^2\mu^2-100c_2^2\lambda^2-100c_2c_3\lambda^2-50c_2^2\lambda\mu-70c_2c_3\lambda\mu-$$
$$50c_3^2\lambda\mu-30c_2c_3\mu^2-30c_3^2\mu^2+75c_2\lambda^2+30c_2\lambda\mu+30c_3\lambda\mu+18c_3\mu^2)\mu.$$

Similar to other construction of other families of $5(4)_{6(7)}$ ERK pairs, the expression for $a_{6,3}$ is too complex to give here.

The seven-stage fourth-order component of the pair satisfies the fifth-order condition $[\tau^4]$, which can cause numerical issues, making this case difficult to recommend for practical usage.

Table 3.4: Cases and free parameters for the complete solution to six-stage fifth-order ERK embedded pairs. 6(6) refers to families of $5(4)_{6(6)}$ ERK pairs and refers to families of 6(7) refers to $5(4)_{6(7)}$ ERK pairs. N/S means that the associated tableaux constructed by the author took up too much space to include in this thesis.

Case	6(6) free params	Subcases	7(7) free params	Subcases	Examples
I (full)	N/A	N/A	$c_2, c_4, \frac{\mu}{\omega}, \hat{b}_7$	2	N/S
II (full)	N/A	N/A	$c_2, c_3, \frac{\lambda}{\omega}, \hat{b}_7$	2	N/S
III	N/A	N/A	$c_2, c_3, \frac{\lambda}{\mu}, \hat{b}_7,$	1	(A.19)
IV	N/A	N/A	$c_2, c_3, \frac{\lambda}{\mu}, \hat{b}_7$	2	N/S
V w/o C(2)	$c_3, c_5, \frac{\lambda}{\mu}, \hat{b}_6$	2	No	N/A	(A.15) (A.16)
VI w/o C(2)	$c_3, c_5, \frac{\lambda}{\mu}$	2	No	N/A	(A.17) (A.18)
V with C(2)	$c_2, c_3, c_5, \hat{b}_6$	1	$c_2, c_3, c_4, c_5, \hat{b}_7$	1	(A.5) (A.7)
VI with C(2)	$c_2, c_3, c_5, c_6, \hat{b}_6$	1	No	N/A	(A.6)

Chapter 4

The OCSage package for solving the order conditions and extensively studying the properties of IVP methods

"Since the local extrapolation mode is being adopted one way of choosing the three c's would be to make the principal truncation term 'small' for the 5th order formula. For general systems of equations this is of course a prohibitive task."
(J.R. Dormand and P.J. Prince 1980)

This chapter describes the `OCSage` (Order Conditions for Sage) package, which is comprised of a set of software tools developed using Sage, a relatively popular open-source CAS [57]. Development of the `OCSage` package was initiated in 2013 to support the study and construction of RK methods. In order to support the work in this thesis, `OCSage` has been under continuous development ever since. The version of `OCSage` presented along with this thesis (`OCSage` version `OCSage.ak2018.v1`) was developed exclusively by the author (Andrew Kroshko) over a period of 5 years. In addition, `OCSage` has been designed in a manner that can eventually be extended into a software package with wider applicability than the specific study in this thesis.

One of the principal reasons for developing `OCSage` is that previous publications that present new embedded RK pairs often gave little information on the tradeoffs examined, except in a qualitative sense (for example, Bogacki and Shampine use mostly qualitative reasoning to construct an embedded pair consisting of second- and third-order ERK methods, which is used in Matlab [15]) or by simply claiming that the authors found a "small" leading error coefficient, such as Dormand and Prince did for the DP5$(4)_{6(7)}$ pair (2.79) [47]. In addition, new RK methods that are presented in the literature were often not thoroughly compared against all of the popular existing RK methods that were in practical use at the time of publication, e.g., see the references [16, 30]. However, due to limitations in computing resources and publication space in the past, it is understandable why many publications used qualitative reasoning, did not perform thorough testing, and did not include substantial data supporting new RK method construction. It is seen in Chapter 5 that drawing clear conclusions about RK methods with similar qualitative properties requires a large amount of data to support well-defined conclusions. By using modern computing technology to develop tools such as `OCSage` (and `pythODE` in Chapter 5), it is possible to give a clear indication of the tradeoffs of different RK families and sets of free parameters.

Although the need for more sophisticated software to study numerical methods in addition to the need to simply conduct more detailed computational studies has been discussed in the literature [166], many of these discussions do not give specifics about the nature of the software that would have to be developed or the resources that would be required to develop it. A substantial contribution to the study of numerical methods that development of `OCSage` (as well as `pythODE` in Chapter 5) gives is demonstrating the specifics of the software required to conduct more detailed computational studies of numerical methods. It is only by constructing an integrated software system and demonstrating specific research problems where this new software is actually required, e.g., in this thesis thoroughly studying the construction of efficient higher-order ERK pairs without standard simplifying assumptions, that specific conclusions can be reached about how better computing technology can improve the study of numerical methods. In particular, in this chapter and Chapter 5, the software system developed is described in detail to give a good picture of the components required and their complexity. In addition, examples of execution times are included throughout to help give a clear idea of the computational resources required and where the bottlenecks might be, both in terms of computational performance and software development resources. This can greatly help other researchers who either want to use `OCSage` and `pythODE` directly or develop their own software for similar purposes.

In Section 4.4, the `OCSage` package applies the theory of rooted trees associated with RK methods to help generate PYTHON code that implements the scalar sums (2.39), after which the generated code is used to implement expressions for the order conditions (2.43) and error coefficients (2.47). In Section 4.7, the order conditions (2.43) can then be solved to further generate (as PYTHON-based source code) explicit expressions for the RK Butcher tableau coefficients (2.34) and RK error coefficients (2.47) in terms of the free parameters of the specific RK family, after which these expressions can also be used to determine more complex quantities, such as the other characteristic numbers (2.56). The code is generated in two different forms by `OCSage` to allow utilization from either the SAGE [57] or SCIPY [91] platforms (discussed below), therefore allowing complete integration of the capabilities of a CAS with high-performance numerical computing. As discussed in Section 4.9, the generated code is used for a high-performance numerical search of the parameter space of families of $5(4)_6$ ERK pairs, and the results are stored in a `PostgreSQL` database [92],[1] where it is used for plotting, data analysis within the `PostgreSQL` database itself, or by the `pythODE` package as described in Chapter 5.

4.1 SAGE, SCIPY, and other important software components

The SAGE platform [57] can narrowly be thought of as a CAS, or more broadly as an extensive platform to support scientific and mathematical computing, but with a strong computer algebra focus. SAGE uses the PYTHON programming language as the primary interface to both its own specific functionality and nearly 100 other software components that it incorporates, including many common PYTHON-based libraries such

[1] https://www.postgresql.org/

as SciPy [91] as well as non-Python-based libraries that are widely used for scientific computing, e.g., `ATLAS` [200] for numerical linear algebra and `Maxima`[2] for symbolic algebra. A principal reason to use a Python-based platform is because for many studies (including the one in this thesis) available computational resources are adequate and efficiency of programmer-hours is paramount. This is why Python-based platforms have become widely used for scientific computing and why they are used for both `OCSage` (and `pythODE` in Chapter 5). In addition, a majority of the tasks and development effort in many modern scientific software packages is for functionality that is non-mathematical and administrative [201], and Python excels at these tasks in comparison to many other platforms used for scientific computing.

Sage [57] has its own Python-based library of functionality for many common mathematical tasks, with a focus on computer algebra. Like many CASs, there is a large set of functionality for symbolic mathematics that can be pictured by examining the section of the Sage reference manual[3] called "Symbolic calculus"[4] that describes the functions and methods used for tasks such as the manipulation of algebraic expressions (for example, gathering terms, expanding expressions, sums and products of expressions), automatic simplification, factorization, and other standard mathematical operations (for example, square roots, trigonometric functions, series expansion, greatest common divisors). These are all obviously useful for the mathematics described in Section 2.7 and Chapter 3. Functionality to support linear algebra and matrices can be pictured by examining the section of the Sage reference manual called "Linear algebra",[5] which describes sophisticated classes that are used to implement objects representing matrices (as well as vectors) that have many features to support symbolic manipulation of these matrices and that are especially useful for studying the mathematics from Chapter 3. Other examples of functionality to support popular mathematical computing tasks, but not required for the study described in this thesis, can be pictured by examining sections of the Sage reference manual called "Geometry and Topology",[6] "Number Theory, Algebraic Geometry",[7] "Probability and Statistics",[8] etc.

There are some differences in the Python-based interpreter for Sage [57] in comparison to standard Python or most other scientific computing tools that also build on Python, such as SciPy [91]. This is because the standard Sage interpreter uses a *preparser* that converts non-standard Python-based syntax, which can be more convenient for computer algebra, into standard Python syntax. The two types of source files are generally differentiated because the standard Python interpreter uses source files with the `.py` extension and those using Sage interpreter use source files with the `.sage` extension. To illustrate the preparser, consider the Python expression `z = 1/2` that, in the absence of the Sage preparser, would assign the integer 0 to `z` in standard Python 2.x, or assign 0.5 to `z` when using standard Python 3.x or when using the statement `from __future__ import __division__` in Python 2.x. However, when

[2]http://maxima.sourceforge.net/
[3]http://doc.sagemath.org/html/en/reference/
[4]http://doc.sagemath.org/html/en/reference/calculus/index.html
[5]http://doc.sagemath.org/html/en/reference/#linear-algebra
[6]http://doc.sagemath.org/html/en/reference/#geometry-and-topology
[7]http://doc.sagemath.org/html/en/reference/#number-theory-algebraic-geometry
[8]http://doc.sagemath.org/html/en/reference/#probability-and-statistics

using the SAGE interpreter the value assigned to `z` when executing the expression `z = 1/2` is a SAGE object representing the rational number $\frac{1}{2}$. Rational expressions involving variables can be similarly defined when using the SAGE interpreter. If using the preparser is not desired, the standard PYTHON interpreter built into SAGE [57] can be run with the command

```
sage -python <<script name>>.py
```

after which normal PYTHON syntax can be used like the PYTHON interpreter with any other standard PYTHON installation, but with built-in access to all of the additional functionality provided with the SAGE package. The equivalent of the above expression, i.e., `z = 1/2` as evaluated when using the SAGE interpreter, in the syntax of the standard PYTHON interpreter, but not using the SAGE [57] preparser, are the statements

```
from sage.all import *
z = Rational((1,2))
```

However, this is not nearly as convenient for sophisticated symbolic computation with rational expressions (this code snippet also works in the SAGE interpreter). It should be obvious from the types of expressions used when constructing RK methods in Section 2.7 and Chapter 3 that the convenient syntax when using the preparser for working with rational expressions, including mixed symbolic/numeric expressions, is of great benefit to the study described in this thesis.

All the following examples of the SAGE interpreter syntax (using the preparser) have similarly equivalent forms as function calls for a standard PYTHON interpreter. The SAGE preparser allows both

```
a**b
a^b
```

to be used for exponentiation (the ˆ operator in the second expression is the bitwise *EXCLUSIVE OR* operation when used with the standard PYTHON interpreter). The SAGE preparser allows avoiding the multiplication operator, e.g., shown by the equivalent expressions

```
z = 3*exp(y)
z = 3exp(y)
```

where the second expression would be illegal syntax if used with the standard PYTHON interpreter. The SAGE preparser allows a compact syntax for symbolically defining mathematical functions, such as given by

```
f(x,y,z) = sin(x^3 - 4*y) + y^x
```

that would also be illegal syntax for the standard PYTHON interpreter because the function call syntax cannot be assigned to. This SAGE syntax is useful for symbolically defining a function to use for tasks such as working with standard calculus operations, e.g., integration and differentiation.

SAGE also offers a web-based notebook with interactive worksheets. This notebook has proven to be extremely useful for exploring the mathematics that led to finding the complete solution of the order conditions for $5(4)_6$ ERK pairs explicitly in terms of the free parameters that are described in Chapter 3. An example of a worksheet running in the Firefox web browser[9] that was used in exploring the mathematics presented in

[9]https://www.mozilla.org/en-US/firefox/products/

Chapter 3 is shown in Figure 4.1. The code provided with `OCSage` that allows for reproducing the worksheet shown in Figure 4.1 is further described in Section 4.6. Furthermore, functions provided by `OCSage` can use the SAGE notebook to generate a "report" on a particular ERK method using the generated code in Section 4.7 by specifying only the free parameters of the particular RK family. An example of this "report" for the DP5(4)$C_{6(7)}$ pair (2.81) [48] is given by Figure 4.2, where it can be seen that the Butcher tableau (2.34) is given, the order conditions (2.43) are checked, the characteristic numbers (2.56) are generated, and the linear stability regions discussed in Section 2.5.4 are plotted. In addition, although not shown because of space considerations a few other plots with information not used directly in this thesis, as well as bar graphs of PECs (2.47) similar to Figure 4.12 and others like it, are also included in the "report".

To install SAGE for this thesis work, the source code for SAGE is downloaded from the SAGE website[10] as a `tar.gz` file and compiled on the Linux machine where it is used. There are also binary packages of SAGE for many common Linux distributions and OS X, as well as a virtual appliance to use on Windows and a live Linux version that boots from a USB key. As an example, the tarball of the source for SAGE 8.0 (released July 21, 2017) is a 611.66 MB `tar.gz` file containing the source code for all the necessary libraries. Building SAGE is as simple as running the `make` command in the directory created by untarring. Using 8 threads took just under 2 hours to build SAGE 8.0 on the relatively generic desktop computer used for the study in this thesis.[11]

Software libraries used

The uniform PYTHON-based interface that SAGE [57] provides also integrates many standard scientific and mathematical computing libraries in order to make it much easier to pursue studies combining sophisticated symbolic computation, numeric computation, simulation, and sophisticated data analysis. Although there is neither space nor the necessity to describe all of the nearly 100 component libraries here, specific libraries included with SAGE [57] that are important to `OCSage` are described. Standard libraries in the PYTHON computing ecosystem included with SAGE that are used extensively by this study are:

- NUMPY:[12] A PYTHON-based library [125, 126, 185] that implements versatile numeric arrays, which are heavily inspired by and have similar syntax and functionality to those in the commercial MATLAB software platform. The NUMPY package also provides a great deal of the basic functionality (for example, fancy indexing, masking, matrix operations, iterators, type handling, basic mathematical functions such as exponential and trigonometric operations, as well as basic implementations of more complex mathematical operations such as Fourier transforms) required to used these numeric arrays for contemporary scientific computing.

[10]http://www.sagemath.org/

[11]A homebuilt quad-core 3.8GHZ AMD A10-5800K with 16GB of memory is used for all timings in this thesis. In addition, a homebuilt quad-core 3.0GHZ AMD Phenom II X4 945 with 4GB of RAM is used to assist in long computations. Both of these computers run Debian Linux Jessie, i.e. Debian Linux version 8.

[12]http://numpy.org/

- MATPLOTLIB:[13] A PYTHON-based library that can generate an enormous variety of plots from NUMPY arrays (see the sample gallery on the MATPLOTLIB website[14]) The MATPLOTLIB library is loosely based on the state-machine-like interface for plotting in MATLAB. However, it also includes an object-oriented API more suitable for use internally by software packages [87]. The MATPLOTLIB library is used for all the data plots in this thesis.

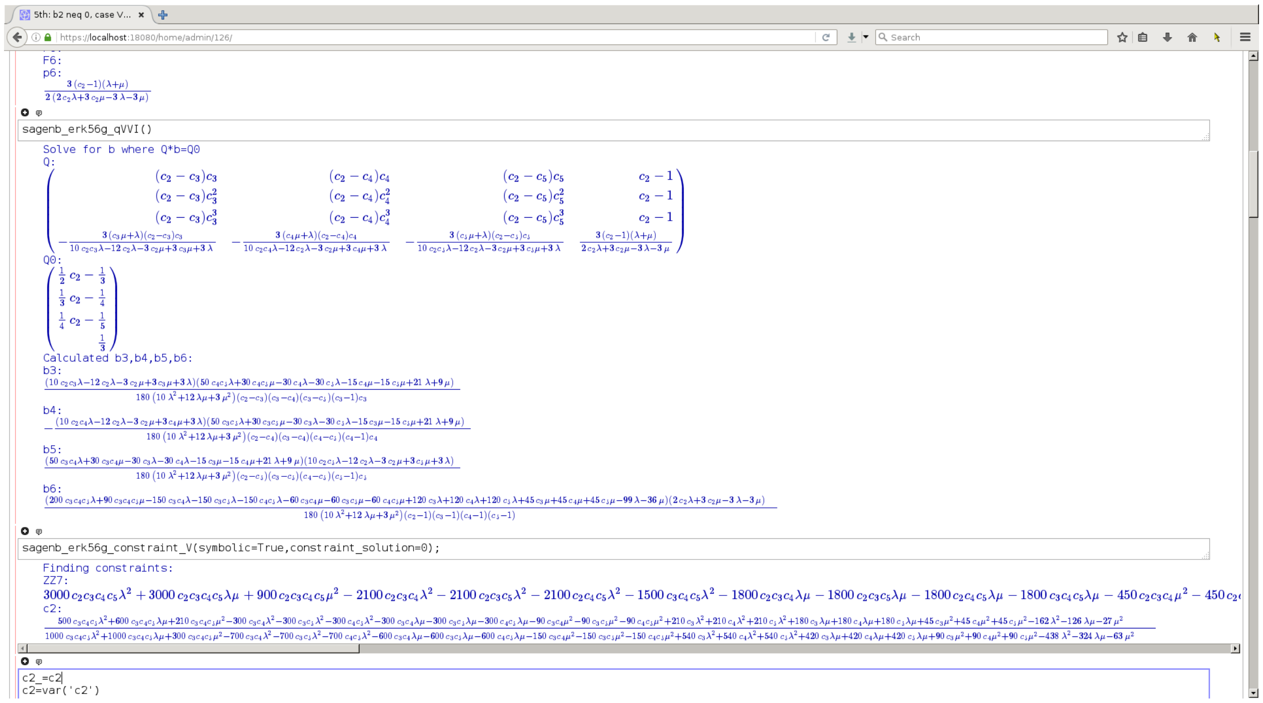
- SCIPY:[15] The SCIPY package [91] extends NUMPY with many optimized high-level functions for common tasks in scientific computing (for example, special functions, linear algebra, signal processing, optimization) that are also heavily inspired by, but not necessarily identical to, those in MATLAB. The SCIPY package [91] is included with SAGE [57] and used by `OCSage`, but it is more commonly used on its own (in a similar manner to the command-line interface of MATLAB) as the basis for scientific computing software.

Non-PYTHON-based libraries used directly by SAGE are:

- `Maxima`:[16] A CAS written in LISP that provides most of the core symbolic algebraic manipulations for SAGE [57].

- `Singular`:[17] A computer algebra library that is used by SAGE [57] for many more difficult tasks, such as factoring polynomials. It is noted in the documentation for SAGE 8.0 (see the relevant section of the reference manual[18]) that factoring polynomials with rational coefficients is done using `Singular` rather than `Maxima` because the former is "very fast" and the latter is "dreadfully slow". This difference substantially affects the speed of some operations in `OCSage` because the quadratic irrationality (square roots) in some cases of the complete solution of the six-stage fifth-order ERK order conditions means SAGE must resort to slower operations in `Maxima` rather than faster operations in `Singular` (see Sections 3.2 and 3.4 for more discussion of the quadratic irrationality that is introduced into some expressions when solving the six-stage fifth-order ERK order conditions). In practice, this means that operations that might require less than a minute with rational expressions for the calculated coefficients in terms of the free parameters (such as the families corresponding to Cases III, V, VI derived in Section 3.4.1), the equivalent operations with expressions containing quadratic irrationality (such as the families corresponding to Cases I, II, IV derived in Section 3.4.1) might take 20–30 minutes. Although notably slower and sometimes inconvenient, to complete the study in this thesis it was not necessary to improve this aspect of performance.

[13] http://matplotlib.org/
[14] http://matplotlib.org/gallery.html
[15] http://scipy.org/
[16] http://maxima.org
[17] https://www.singular.uni-kl.de/
[18] http://doc.sagemath.org/html/en/reference/calculus/sage/symbolic/expression.html#sage.symbolic.expression.Expression.factor



F6:
p6:

$$\frac{3\,(c_2-1)(\lambda+\mu)}{2\,(2\,c_2\lambda+3\,c_2\mu-3\,\lambda-3\,\mu)}$$

```
sagenb_erk56g_qVVI()
```

Solve for b where Q*b=Q0
Q:

$$\begin{pmatrix} (c_2-c_3)c_3 & (c_2-c_4)c_4 & (c_2-c_5)c_5 & c_2-1 \\ (c_2-c_3)c_3^2 & (c_2-c_4)c_4^2 & (c_2-c_5)c_5^2 & c_2-1 \\ (c_2-c_3)c_3^3 & (c_2-c_4)c_4^3 & (c_2-c_5)c_5^3 & c_2-1 \\ -\frac{3\,(c_3\mu+\lambda)(c_2-c_3)c_3}{10\,c_2c_3\lambda-12\,c_2\lambda-3\,c_2\mu+3\,c_3\mu+3\,\lambda} & -\frac{3\,(c_4\mu+\lambda)(c_2-c_4)c_4}{10\,c_2c_4\lambda-12\,c_2\lambda-3\,c_2\mu+3\,c_4\mu+3\,\lambda} & -\frac{3\,(c_5\mu+\lambda)(c_2-c_5)c_5}{10\,c_2c_5\lambda-12\,c_2\lambda-3\,c_2\mu+3\,c_5\mu+3\,\lambda} & \frac{3\,(c_2-1)(\lambda+\mu)}{2\,c_2\lambda+3\,c_2\mu-3\,\lambda-3\,\mu} \end{pmatrix}$$

Q0:

$$\begin{pmatrix} \frac{1}{2}\,c_2-\frac{1}{3} \\ \frac{1}{3}\,c_2-\frac{1}{4} \\ \frac{1}{4}\,c_2-\frac{1}{5} \\ \frac{1}{3} \end{pmatrix}$$

Calculated b3,b4,b5,b6:
b3:

$$\frac{(10\,c_2c_3\lambda-12\,c_2\lambda-3\,c_2\mu+3\,c_3\mu+3\,\lambda)(50\,c_4c_5\lambda+30\,c_4c_5\mu-30\,c_4\lambda-30\,c_5\lambda-15\,c_4\mu-15\,c_5\mu+21\,\lambda+9\,\mu)}{180\,(10\,\lambda^2+12\,\lambda\mu+3\,\mu^2)(c_2-c_3)(c_3-c_4)(c_3-c_5)(c_3-1)c_3}$$

b4:

$$-\frac{(10\,c_2c_4\lambda-12\,c_2\lambda-3\,c_2\mu+3\,c_4\mu+3\,\lambda)(50\,c_3c_5\lambda+30\,c_3c_5\mu-30\,c_3\lambda-30\,c_5\lambda-15\,c_3\mu-15\,c_5\mu+21\,\lambda+9\,\mu)}{180\,(10\,\lambda^2+12\,\lambda\mu+3\,\mu^2)(c_2-c_4)(c_3-c_4)(c_4-c_5)(c_4-1)c_4}$$

b5:

$$\frac{(50\,c_3c_4\lambda+30\,c_3c_4\mu-30\,c_3\lambda-30\,c_4\lambda-15\,c_3\mu-15\,c_4\mu+21\,\lambda+9\,\mu)(10\,c_2c_5\lambda-12\,c_2\lambda-3\,c_2\mu+3\,c_5\mu+3\,\lambda)}{180\,(10\,\lambda^2+12\,\lambda\mu+3\,\mu^2)(c_2-c_5)(c_3-c_5)(c_4-c_5)(c_5-1)c_5}$$

b6:

$$\frac{(200\,c_3c_4c_5\lambda+90\,c_3c_4c_5\mu-150\,c_3c_4\lambda-150\,c_3c_5\lambda-150\,c_4c_5\lambda-60\,c_3c_4\mu-60\,c_3c_5\mu-60\,c_4c_5\mu+120\,c_3\lambda+120\,c_4\lambda+120\,c_5\lambda+45\,c_3\mu+45\,c_4\mu+45\,c_5\mu-99\,\lambda-36\,\mu)(2\,c_2\lambda+3\,c_2\mu-3\,\lambda-3\,\mu)}{180\,(10\,\lambda^2+12\,\lambda\mu+3\,\mu^2)(c_2-1)(c_3-1)(c_4-1)(c_5-1)}$$

```
sagenb_erk56g_constraint_V(symbolic=True,constraint_solution=0);
```

Finding constraints:
ZZ7:

$$3000\,c_2c_3c_4c_5\lambda^2+3000\,c_2c_3c_4c_5\lambda\mu+900\,c_2c_3c_4c_5\mu^2-2100\,c_2c_3c_4\lambda^2-2100\,c_2c_3c_5\lambda^2-2100\,c_2c_4c_5\lambda^2-1500\,c_3c_4c_5\lambda^2-1800\,c_2c_3c_4\lambda\mu-1800\,c_2c_3c_5\lambda\mu-1800\,c_2c_4c_5\lambda\mu-1800\,c_3c_4c_5\lambda\mu-450\,c_2c_3c_4\mu^2-450\,c_2$$

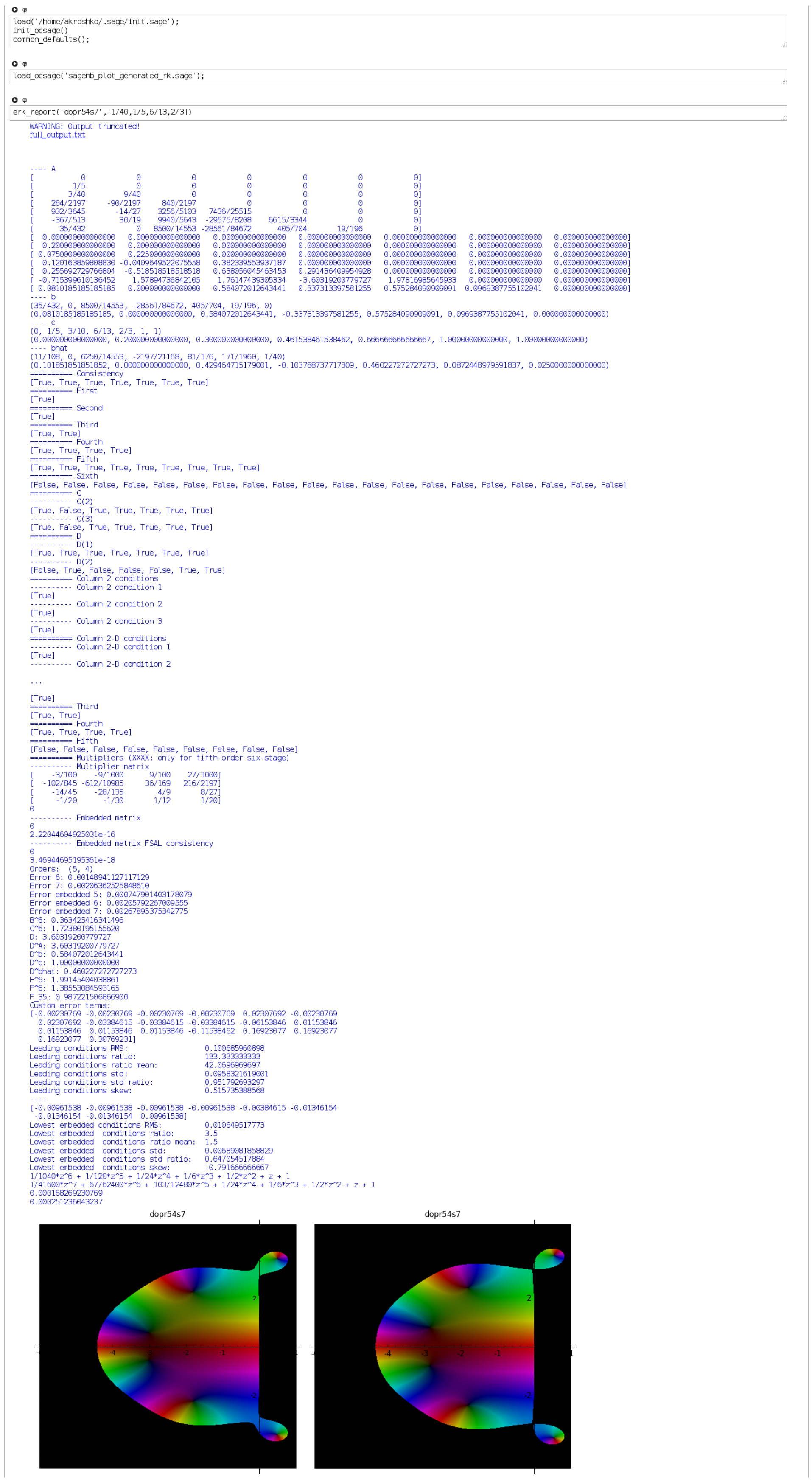
c2:

$$\frac{500\,c_3c_4c_5\lambda^2+600\,c_3c_4c_5\lambda\mu+210\,c_3c_4c_5\mu^2-300\,c_3c_4\lambda^2-300\,c_3c_5\lambda^2-300\,c_4c_5\lambda^2-300\,c_3c_4\lambda\mu-300\,c_3c_5\lambda\mu-300\,c_4c_5\lambda\mu-90\,c_3c_4\mu^2-90\,c_3c_5\mu^2-90\,c_4c_5\mu^2+210\,c_3\lambda^2+210\,c_4\lambda^2+210\,c_5\lambda^2+180\,c_3\lambda\mu+180\,c_4\lambda\mu+180\,c_5\lambda\mu+45\,c_3\mu^2+45\,c_4\mu^2+45\,c_5\mu^2-162\,\lambda^2-126\,\lambda\mu-27\,\mu^2}{1000\,c_3c_4c_5\lambda^2+1000\,c_3c_4c_5\lambda\mu+300\,c_3c_4c_5\mu^2-700\,c_3c_4\lambda^2-700\,c_3c_5\lambda^2-700\,c_4c_5\lambda^2-600\,c_3c_4\lambda\mu-600\,c_3c_5\lambda\mu-600\,c_4c_5\lambda\mu-150\,c_3c_4\mu^2-150\,c_3c_5\mu^2-150\,c_4c_5\mu^2+540\,c_3\lambda^2+540\,c_4\lambda^2+540\,c_5\lambda^2+420\,c_3\lambda\mu+420\,c_4\lambda\mu+420\,c_5\lambda\mu+90\,c_3\mu^2+90\,c_4\mu^2+90\,c_5\mu^2-438\,\lambda^2-324\,\lambda\mu-63\,\mu^2}$$

```
c2_=c2
c2=var('c2')
```

Figure 4.1: A sample of exploratory calculations for the family corresponding to Case V derived in Section 3.4.1 being done in a worksheet in the Sage notebook running in a Firefox web browser.

Many of the component libraries of Sage have a wide array of incompatible APIs in many different languages, e.g., `Maxima` uses Common Lisp and `Singular` uses C++. Therefore, one of the advantages of Sage [57] is that it provides a common interface with a consistent set of versions for a diverse array of libraries. Sometimes, the only interface to particular software packages is a command prompt or similar interface meant to be used by a human can occur for some older open-source software packages that provide particular functionality or when interfacing with closed-source commercial software such as Maple and Mathematica. In these cases, Sage uses the Python-based `Pexpect` library [57], which simulates sending textual commands to the command prompt just like a human would and reading the resulting output in order to allow software to interface with these packages. The `Singular` library used to have only a command-line-based interface requiring the use of `Pexpect` by Sage. However, the Sage project has now created a proper C++ API for `Singular`, and over time is also doing this for many other important libraries. However, `Pexpect` continues to be useful to the Sage project in order to provide interactive interfaces for the Sage notebook or interfacing with closed-source commercial software such as Maple and Mathematica through their command-line interfaces. Although Sage can interface to Maple and Mathematica, which have extremely powerful algebraic capabilities, the versions available to the author were often out-of-sync with the latest Sage version. Therefore, these commercial packages are not used for this study because any advantages they would confer are not strictly required.



```
load('/home/akroshko/.sage/init.sage');
init_ocsage()
common_defaults();
```

```
load_ocsage('sagenb_plot_generated_rk.sage');
```

```
erk_report('dopr54s7',[1/40,1/5,6/13,2/3])
```

```
WARNING: Output truncated!
full_output.txt



---- A
[           0            0            0            0            0            0            0]
[         1/5            0            0            0            0            0            0]
[        3/40         9/40            0            0            0            0            0]
[    264/2197     -90/2197     840/2197            0            0            0            0]
[    932/3645       -14/27    3256/5103  7436/25515            0            0            0]
[    -367/513        30/19    9940/5643 -29575/8208    6615/3344            0            0]
[      35/432            0  8500/14553 -28561/84672      405/704       19/196            0]
[  0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000]
[  0.200000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000]
[ 0.0750000000000000   0.225000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000]
[  0.120163859808830 -0.0409649522075558   0.382339553937187   0.000000000000000   0.000000000000000   0.000000000000000   0.000000000000000]
[  0.255692729766804  -0.518518518518518   0.638056045463453   0.291436409954928   0.000000000000000   0.000000000000000   0.000000000000000]
[ -0.715399610136452    1.57894736842105    1.76147439305334   -3.60319200779727    1.97816985645933   0.000000000000000   0.000000000000000]
[ 0.0810185185185185   0.000000000000000   0.584072012643441  -0.337313397581255   0.575284090909091  0.0969387755102041   0.000000000000000]
---- b
(35/432, 0, 8500/14553, -28561/84672, 405/704, 19/196, 0)
(0.0810185185185185, 0.000000000000000, 0.584072012643441, -0.337313397581255, 0.575284090909091, 0.0969387755102041, 0.000000000000000)
---- c
(0, 1/5, 3/10, 6/13, 2/3, 1, 1)
(0.000000000000000, 0.200000000000000, 0.300000000000000, 0.461538461538462, 0.666666666666667, 1.00000000000000, 1.00000000000000)
---- bhat
(11/108, 0, 6250/14553, -2197/21168, 81/176, 171/1960, 1/40)
(0.101851851851852, 0.000000000000000, 0.429464715179001, -0.103788737717309, 0.460227272727273, 0.0872448979591837, 0.0250000000000000)
========== Consistency
[True, True, True, True, True, True, True]
========== First
[True]
========== Second
[True]
========== Third
[True, True]
========== Fourth
[True, True, True, True]
========== Fifth
[True, True, True, True, True, True, True, True, True]
========== Sixth
[False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False, False]
========== C
---------- C(2)
[True, False, True, True, True, True, True]
---------- C(3)
[True, False, True, True, True, True, True]
========== D
---------- D(1)
[True, True, True, True, True, True, True]
---------- D(2)
[False, True, False, False, False, True, True]
========== Column 2 conditions
---------- Column 2 condition 1
[True]
---------- Column 2 condition 2
[True]
---------- Column 2 condition 3
[True]
========== Column 2-D conditions
---------- Column 2-D condition 1
[True]
---------- Column 2-D condition 2

...

[True]
========== Third
[True, True]
========== Fourth
[True, True, True, True]
========== Fifth
[False, False, False, False, False, False, False, False, False]
========== Multipliers (XXXX: only for fifth-order six-stage)
---------- Multiplier matrix
[    -3/100    -9/1000      9/100    27/1000]
[  -102/845 -612/10985     36/169   216/2197]
[    -14/45    -28/135        4/9       8/27]
[     -1/20      -1/30       1/12       1/20]
0
---------- Embedded matrix
0
2.22044604925031e-16
---------- Embedded matrix FSAL consistency
0
3.46944695195361e-18
Orders:  (5, 4)
Error 6: 0.00148941127117129
Error 7: 0.00206362525848610
Error embedded 5: 0.000747901403178079
Error embedded 6: 0.00205792267009555
Error embedded 7: 0.00267895375342775
B^6: 0.363425416341496
C^6: 1.72380195155620
D: 3.60319200779727
D^A: 3.60319200779727
D^b: 0.584072012643441
D^c: 1.00000000000000
D^bhat: 0.460227272727273
E^6: 1.99145404038861
F^6: 1.38553084593165
F_35: 0.987221506866900
Custom error terms:
[-0.00230769 -0.00230769 -0.00230769 -0.00230769  0.02307692 -0.00230769
  0.02307692 -0.03384615 -0.03384615 -0.03384615 -0.06153846  0.01153846
  0.01153846  0.01153846  0.01153846 -0.11538462  0.16923077  0.16923077
  0.16923077  0.30769231]
Leading conditions RMS:                 0.100685960898
Leading conditions ratio:               133.333333333
Leading conditions ratio mean:          42.0696969697
Leading conditions std:                 0.0958321619001
Leading conditions std ratio:           0.951792693297
Leading conditions skew:                0.515735388568
----
[-0.00961538 -0.00961538 -0.00961538 -0.00961538 -0.00384615 -0.01346154
 -0.01346154 -0.01346154  0.00961538]
Lowest embedded conditions RMS:         0.010649517773
Lowest embedded  conditions ratio:      3.5
Lowest embedded  conditions ratio mean:  1.5
Lowest embedded  conditions std:        0.00689081858829
Lowest embedded  conditions std ratio:  0.647054517884
Lowest embedded  conditions skew:       -0.791666666667
1/1040*z^6 + 1/120*z^5 + 1/24*z^4 + 1/6*z^3 + 1/2*z^2 + z + 1
1/41600*z^7 + 67/62400*z^6 + 103/12480*z^5 + 1/24*z^4 + 1/6*z^3 + 1/2*z^2 + z + 1
0.000168269230769
0.000251236043237
```

Figure 4.2: A "report" in the Sage notebook giving the properties of the DP5(4)C$_{6(7)}$ pair (2.81).

4.2 Review of existing software to analyze Runge–Kutta methods

Software for algebraically studying the RK order conditions (2.43) has already been described in the literature and gave substantial insights that assisted in developing `OCSage`. Therefore, despite most of these existing software packages being developed before 2005, it is important to review these published packages first. However, in comparison to the software that is reviewed here, the `OCSage` package is much larger, with approximately 11,000 lines of code rather than at most several hundred lines of code for the software packages reviewed here. `OCSage` also benefits from being developed in an era with readily available high-performance and multi-core computing, and where the Python scientific computing ecosystem is broadly integrated and widely used. These are all aspects that make developing a more complex software package much easier than in the past.

CASs date back to the 1960s [121], and commercial CASs such as Maple were in widespread use by the 1980s [93]. Some of the first dedicated software packages to build upon a CAS to systematically generate and solve the order conditions (2.43) are from Keiper in 1990 and Harrison in 1994 (neither of which were named or published in the literature, but were mentioned in the references and discussion by Famelis, Papakostas, and Tsitouras [58]). Two other early packages (also unnamed) that were published through peer-reviewed journals are from Sofroniou in 1994 [168] that was built upon the commercial CAS Mathematica, and from Bendtsen in 1996 [14] that was built upon the commercial CAS Maple. In 2001, Bornemann published two packages [18, 19] in order to give the option of either Mathematica or Maple, respectively, and these have been used and/or referenced by several authors in further research [51, 95, 196]. The (unnamed) Mathematica package by Famelis, Papakostas, and Tsitouras from 2004 [58] appears to have been extremely useful in both preceding and subsequent publications by the original authors [58, 181, 184]. An advantage to the algorithms used by Famelis, Papakostas, and Tsitouras is, compared to previous approaches, they were demonstrated to be up to many orders of magnitude faster for generating expressions above order twelve for scalar sums (2.39), order conditions (2.43), error coefficients (2.47) [58]. Another published tool is the unnamed software package by Cameron [28] that is built on the commercial software package Matlab, which is focused on numerical computing and hence is required to use Maple when CAS functionality is required. This package by Cameron [28] automatically generates explicit expressions for the PECs (2.47), and it has been used by at least one other author [142]. In comparison to `OCSage`, the preceding packages are relatively small. In many cases, the code even fits into the text of the original paper [58]. However, these packages act more as helper functions for studying RK methods with the commercial software package for which they were written, rather than as a more independent set of tools.

The studies by Tsitouras [182, 183] and Khashin [99] previously mentioned in Section 3.1 also use computer algebra to solve order conditions, but instead of generating explicit expressions that are amenable to high-performance evaluation, numeric values of at least some free parameters are substituted, and these simpler equations are then solved for each set of free parameters using relatively slow algebraic methods. The advantages of software packages such as Mathematica, which is similar to Sage in having both strong CAS and numerical computing capabilities, becomes extremely clear in these cases. However, it is seen in Section 4.15.1 that in the study by Tsitouras [182, 183] this led to ERK pairs that do not exactly satisfy the order conditions and do not have corresponding pairs in exact arithmetic.

4.3 Overview of the `OCSage` package

The `OCSage` package contains all needed components for studying the full process of solving order conditions and selecting coefficients to construct RK methods, in a more complete and integrated software package than those reviewed in the last section. Although it will be mentioned where the specific algorithms and functionality are inspired by the code in other published packages, the `OCSage` package is much larger and develops relevant functionality much further than any other known packages for a similar purpose. The additional complexity of `OCSage` comes from aspects such as:

- The functions and classes that represent the large number of details that must be addressed to support a thorough study of the mathematics as presented in Chapter 3.
- The functions and classes that support code generation and *unit testing* to ensure the mathematics remains correct throughout the development process, neither of which are provided by any other known package supporting the construction of RK methods.
- Code that supports searching the free parameters of RK families and storing the results in a `PostgreSQL` database, as well as plotting those results from that database.
- Many of the other large and small complexities that are commonly expected when moving from simple implementations of a task to software useful for serious research or other practical applications.

Despite being somewhat specialized to this study, `OCSage` could be extended to study other types of classically constructed RK methods with as little as a few hundred lines of code. Thoroughly implementing method construction using CMF (as shown in Chapter 3) for RK method families other than ERK or even ERK methods other than $5(4)_6$ ERK pairs would likely require several thousand additional lines of code specific to the RK method family of interest. Although not addressed currently, the issue that would arise of duplicated but specialized code for implementing CMF could be mitigated by a more general library of code based on the functions for, but not specialized to, $5(4)_6$ ERK pairs.

The `OCSage` package first finds the purely algebraic solution of the order conditions the families of $5(4)_6$ ERK pairs constructed in Section 2.7 and Chapter 3. This is in addition to replicating the algebraic solution of many published ERK families up to eighth order. The solutions of the order conditions are then used to generate Python code (for both the standard Python interpreter and Sage interpreter) that gives the method coefficients of the above-mentioned ERK families (as well as error coefficients, stability properties, and other characteristic numbers) as callable Sage or Python functions with the most general possible set of free parameters as arguments. These generated functions can then be used to narrow down candidate methods by searching for methods with promising characteristic numbers (2.56).

Due to the requirement for a purely symbolic solution of order conditions and numeric search in the same package, the `OCSage` package is a mix of code using the Sage interpreter with the preparser as described in Section 4.1, and code using the standard Python interpreter when making heavy use of SciPy, NumPy, and matplotlib. Although it would have been possible to make use of only one interpreter, it was found that it was most straightforward to use the interpreter best suited for either symbolic or numerical computing, as required by each source code file. The specific advantage in `OCSage` of using the Sage interpreter for manipulating symbolic expressions has already been discussed, and a specific advantage to using SciPy is that it allows some source files in `OCSage`, such as `erk_search.py` described in Section 4.9, to be used without a full Sage installation (the installation of Sage 8.0 used for this study takes up 7.7GB on disk). Using just SciPy could be an advantage for leveraging computing resources such as disk- or quota-limited computers that already have a standard SciPy installation.

Although most files for `OCSage` are split between a main package directory and subdirectories, in the following overview and the next sections, they are collected as functional units with respect to the main package directory.

- The source files for generating trees and the source files with the generated trees, as well as the source files making possible their utilization by providing code for the scalar sums (2.39), order conditions (2.43), and PECs (2.47) are:

  ```
  maketrees_cached.sage
  maketrees.sage
  generated_trees/*
  generated_trees/rk_trees01.py
  ...
  generated_trees/rk_trees12.py
  generated_trees/rk_trees.py
  orderconditions.py
  orderconditions_scipy.py
  ```

  that are covered in Section 4.4.

- The source file containing code specific to the mathematics of $5(4)_6$ ERK pairs in Chapter 3 is:

  ```
  erk5_6s_functions.sage
  ```

  that is just under 3000 lines long, covered in Section 4.5, and used extensively by the next two functional units of source code files that are about to be described, which themselves are covered fully in Section 4.6 and Section 4.7.

- The files specific to symbolically computing the solutions to the order conditions for contexts such as the SAGE notebook in Figure 4.1 and unit testing are:

  ```
  sagenb/*
  unittests_worksheet/*
  ```

  that are covered in Section 4.6.

- The source files used to actually solve the order conditions and generate expressions for Butcher tableau (2.34) coefficients and error coefficients in terms of the free parameters, generated as either SCIPY [91] or SAGE code, are:

  ```
  rkclasses.py
  erk_classic.sage
  erk_cmf.sage
  erk_fixed.sage
  generated_sage/*
  generated_scipy/*
  ```

  that are covered in Section 4.7.

- In order to verify that the order conditions are being solved correctly, several source files that implement unit tests are given by:

  ```
  unittests_generated_scipy.py
  unittests.py
  unittests_regenerate.py
  unittests_worksheet/unittests_worksheet.py
  ```

  where the files in `unittests_worksheet` have already been mentioned and are covered in Section 4.8.

- The source file that uses the generated code to numerically search the free parameters of the solved order conditions is:

  ```
  erk_search.py
  ```

  that is covered in Section 4.9. The output of this last source file is stored in a `PostgreSQL` database for either analysis within the database itself, use by plotting functions contained within the figures supporting Sections 4.12–4.14, or use by the `pythODE` package covered in Chapter 5.

4.4 Generating code to implement the scalar sums

For this study of $5(4)_6$ ERK pairs, code implementing the scalar sums (2.39) up to order ten is generated automatically and allows the order conditions (2.43) up to order five along with five additional orders of non-zero error coefficients to be examined (for the fourth-order method of a $5(4)_6$ ERK pair, this allows the order conditions (2.43) up to order four along with six additional orders of non-zero error coefficients to be examined). This puts the study in this thesis in line with other studies of fifth-order ERK methods that sometimes find numerical values for error coefficients up to order ten [16] for the small number of RK methods that they constructed or compared.

The program to generate the scalar sums (2.39) up to order twelve is the `maketrees.sage` source code file, and the files themselves are put into the directory `generated_trees/` as the files `rk_trees01.py` to `rk_trees12.py`. In order to avoid inefficiencies from having to evaluate unnecessarily high-order expressions for the scalar sums (2.39), the generated expressions for the scalar sums (2.39) are put in these separate files, which are called from the Python function `rk_classic_scalar_sums` in `rk_trees.py` and that must specify a maximum order for the scalar sums (2.39) to be returned. The `maketrees.sage` program generates the scalar sums (2.39) that can then be used to form the order conditions (2.43), PECs (2.47), and other characteristic numbers (2.56). An example of the generated code for specific scalar sums (2.39) in Python syntax can be seen in the last row of Figure 4.3 (with the corresponding tree in the first row), where the rest of the specific algorithm is described below in Section 4.4.1 after this overview of generating code implementing the scalar sums (2.39) is given. Also provided as part of the generated code in `rk_trees.py` are the calculated values of α, γ, σ corresponding to each scalar sum (2.39) generated. See the discussion and references in Section 2.5.1 for the definitions of α, γ, σ.

Using a single core of the desktop computer used for this study, `maketrees.sage` takes only about 2 seconds to generate code implementing the RK scalar sums (2.39) up to order six and takes only 17 seconds to generate code implementing the RK scalar sums (2.39) up to order twelve, after which point it starts to take much longer for each additional order. Although generating code implementing the scalar sums (2.39) as nested `sum` statements is relatively convenient, the statements themselves can be time consuming and resource intensive to evaluate, such as deeply nested summations or loops tend to be in most software. Although code for scalar sums (2.39) up to order twelve is generated because this only takes a few additional seconds more than order ten, the nested `sum` statements for orders above ten are currently too inefficient to actually utilize.

Actually running the nested `sum` statements returns the fully expanded expressions for the scalar sums (2.39) of general RK methods, with the total number of terms (the terms are the products of the Butcher tableau (2.34) coefficients inside the summations given by (2.39)) for all scalar sums (2.39) at each order for a particular number of stages given in column five of Table 4.1. However, by comparing column five of Table 4.1 to column six, which gives the total number of non-zero terms for ERK scalar sums (2.39), it can be seen that the effect of the zero coefficients, due to the strictly lower-triangular nature of the Butcher tableau (2.34) of

ERK methods, drastically reduces the number of terms in the expanded expressions for the scalar sums (2.39) of ERK methods in comparison to general RK methods, especially for scalar sums (2.39) of order of six or greater. This allows error terms and order conditions (2.43) associated with these higher-order ERK scalar sums (2.39) to be computed relatively easily from purely symbolic expressions. On the other hand, it can easily be seen that using fully expanded purely symbolic versions of the most general scalar sums (2.39) for high-order RK methods with large numbers of stages may in fact be intractable even at orders of ten or below.

Table 4.1: Some values of the number of non-unique trees generated and unique trees at each order, in addition to the number of terms in the expanded scalar sums for 6 and 7 stages. *Values specific to the algorithms and methodology used by `OCSage`.

order	stages	non-unique trees generated*	unique trees	terms in RK	terms in ERK
1	6	N/A (base case)	1	6	6
1	7			7	7
2	6	1	1	5	5
2	7			6	6
3	6	2	2	35	15
3	7			48	21
4	6	6	4	240	35
4	7			378	56
5	6	16	9	1710	90
5	7			3087	161
6	6	50	20	12255	231
6	7			25434	462
7	6	150	48	89520	630
7	7			213170	1399
8	6	490	115	659690	1654
8	7			1799820	4158
9	6	1576	286	4960310	4391
9	7			15466588	12623
10	6	5238	719	37678270	11461
10	7			134236420	37962

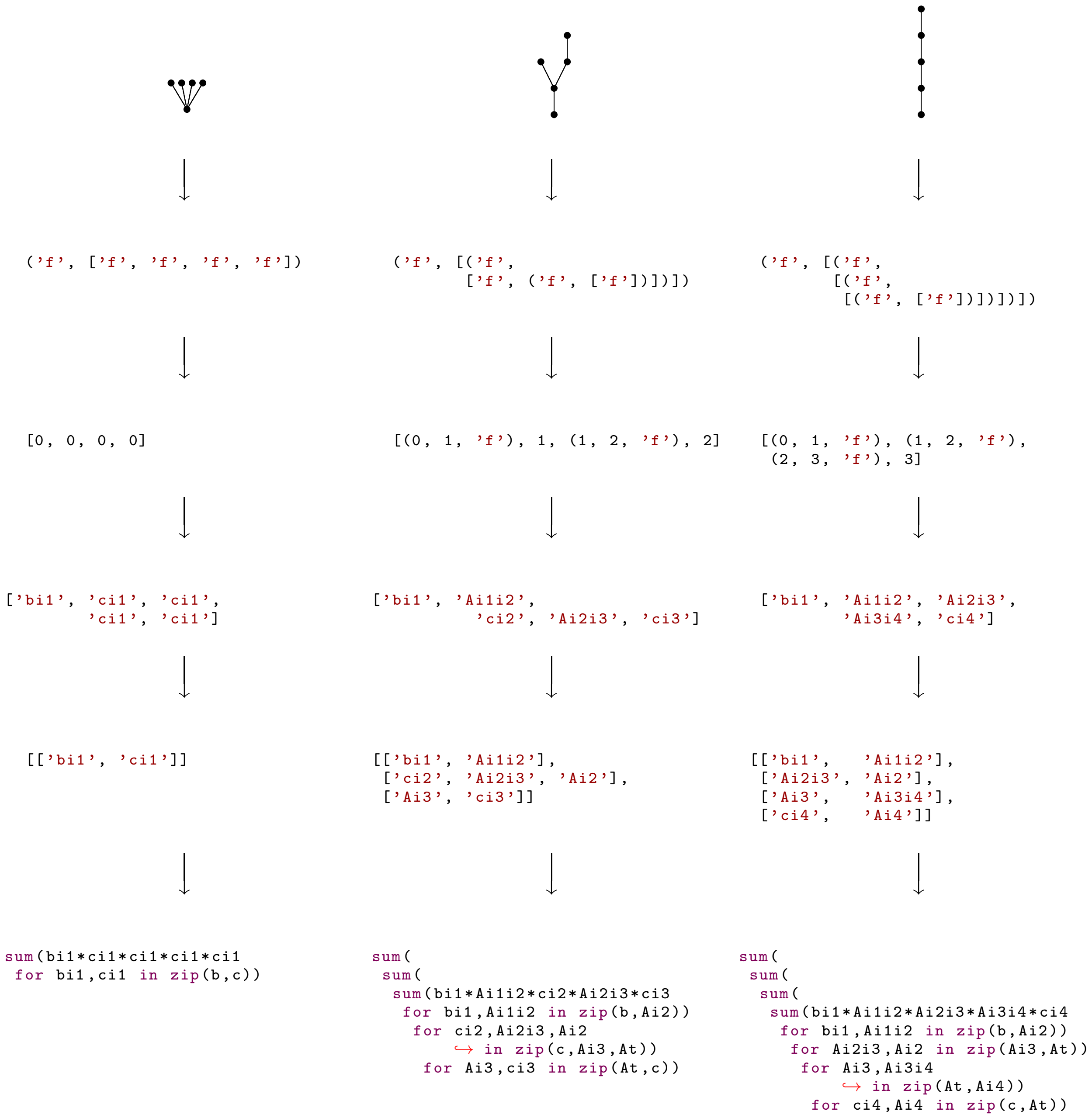
The reason that the expanded versions of the nested `sum` statements in the generated PYTHON code for scalar sums (2.39) can be used efficiently and within the resource limitations of a typical desktop computer from the nested `sum` statements, even up to order ten, is because it is easy to pre-calculate the relatively small number of non-zero terms of the fully expanded scalar sums (2.39) for ERK methods. In order to speed up the process of loading the expanded and simplified scalar sums (2.39) for individual scripts, the program `maketrees_cached.sage` uses the nested `sum` expressions for the trees generated (see example in the last row of Figure 4.3), expands and simplifies them for ERK methods, and stores these expressions in the quickly loadable binary format provided by SAGE, i.e., `.sobj` files. The pre-calculation of all ERK scalar sums (2.39) for up to order ten and seven stages takes 82 minutes the desktop computer used for this study, with the SAGE `.sobj` files taking up only 1.9MB and all `.sobj` files loading together taking about 5 seconds. With the `.sobj` files being so small on disk, these pre-calculated expressions do not consume a notable amount of memory when loaded into memory on a modern desktop computer. However, despite the expense of pre-calculating, except when being re-generated as part of the normal software development process, creating these `.sobj` files only has to be done once unless different orders and numbers of stages are required.

For convenience in some scripts or during some phases of software development, loading the scalar sums (2.39) for ERK methods using the nested `sum` statements directly (without pre-calculating) up to order seven with 7 stages adds only about 5 seconds to each execution of a script, although by order eight with 7 stages it adds about 45 seconds to each execution of a script.

Although with the simplifications that the lower triangular ERK Butcher tableau (2.34) allows, using scalar sums (2.39) for practical purposes above order ten would be possible, it would take much longer to pre-calculate the expanded expressions and the combinatorial computational complexity of the current implementation in `OCSage` would quickly make this intractable. To illustrate this, counting the number of terms in the RK scalar sums (2.39) for six-stage RK up to tenth order and seven-stage up to ninth order to fill in Table 4.1 took several hours on the desktop computer used for this study. Furthermore, as implemented currently, counting the number of terms in the RK scalar sums (2.39) for seven-stage tenth-order required a special script to count the terms of the scalar sums in small groups with 16GB of memory on the computer used. In fact, the process of expanding the appropriate nested `sum` expression and counting all 14,606,622 terms of just the tallest tenth-order general RK tree alone consumed 6881MB of memory and took nearly an hour on the desktop computer used for this study. Pre-computing the ERK scalar sums (2.39) up to order 12 would likely be possible in a couple of weeks on a computer with around 1TB of memory using the existing code, with the resulting precomputed scalar sums (2.39) still likely being small enough to be useful on a typical desktop computer. Beyond that, more sophisticated code generation algorithms other than just using the relatively simple nested `sum` statements might make pre-computing the ERK scalar sums for orders up to 12, or even greater, possible on a standard desktop computer. However, none of these possible enhancements are necessary for this study.

4.4.1 Algorithms used for `maketrees.sage`

The source file `maketrees.sage` generates the trees as nested PYTHON lists and tuples in a similar manner to that described by Bornemann [18, 19], which is also similar to the representation of the rooted trees as nested sets described by Butcher [27, pg.125]. All possible nested lists and tuples isomorphic to all appropriate rooted trees are generated using a *greedy* methodology, an example of these data structures can be seen in the second row of Figure 4.3 with the corresponding rooted tree in the first row. This is done by starting out with the data structure corresponding to the single vertex • and then adding a new branch and leaf to create the data structure isomorphic to the rooted tree of order 2, i.e, . At each iteration, new rooted trees are generated by adding a single branch and leaf to each vertex of all existing rooted trees of the currently highest order, in order to generate the complete (but non-unique) set of rooted trees with one additional order.



```
('f', ['f', 'f', 'f', 'f'])

[0, 0, 0, 0]

['bi1', 'ci1', 'ci1',
        'ci1', 'ci1']

[['bi1', 'ci1']]

sum(bi1*ci1*ci1*ci1*ci1
 for bi1,ci1 in zip(b,c))
```

```
('f', [('f',
       ['f', ('f', ['f'])])])

[(0, 1, 'f'), 1, (1, 2, 'f'), 2]

['bi1', 'Ai1i2',
          'ci2', 'Ai2i3', 'ci3']

[['bi1', 'Ai1i2'],
 ['ci2', 'Ai2i3', 'Ai2'],
 ['Ai3', 'ci3']]

sum(
 sum(
  sum(bi1*Ai1i2*ci2*Ai2i3*ci3
   for bi1,Ai1i2 in zip(b,Ai2))
    for ci2,Ai2i3,Ai2
        ↪ in zip(c,Ai3,At))
     for Ai3,ci3 in zip(At,c))
```

```
('f', [('f',
       [('f',
        [('f', ['f'])])])])

[(0, 1, 'f'), (1, 2, 'f'),
 (2, 3, 'f'), 3]

['bi1', 'Ai1i2', 'Ai2i3',
        'Ai3i4', 'ci4']

[['bi1',   'Ai1i2'],
 ['Ai2i3', 'Ai2'],
 ['Ai3',   'Ai3i4'],
 ['ci4',   'Ai4']]

sum(
 sum(
  sum(
   sum(bi1*Ai1i2*Ai2i3*Ai3i4*ci4
    for bi1,Ai1i2 in zip(b,Ai2))
     for Ai2i3,Ai2 in zip(Ai3,At))
      for Ai3,Ai3i4
          ↪ in zip(At,Ai4))
       for ci4,Ai4 in zip(c,At))
```

Figure 4.3: Examples of the sequence of data structures used to generate scalar sums corresponding to three representative fifth-order rooted trees.

During generation of the set of the next highest order of rooted trees, this set can contain duplicates of many of these rooted trees because each rooted tree can sometimes be constructed in several different ways from the current set of highest order trees using this branch addition procedure. However, these duplicates can be efficiently eliminated as each order is generated by use of PYTHON's built-in `sort` function on the nested data structures corresponding to the trees. This is similar to what was described by Bornemann [18, 19], where the MAPLE and MATHEMATICA `sort` functions he used behave identically to the PYTHON `sort` function for this application. Then the duplicates can easily be removed by checking for equality of the data structures between adjacent entries in the sorted lists using the PYTHON `==` operator. See columns three and four of Table 4.1 for the exact number of non-unique trees generated by `maketrees.sage` from the unique trees at each lower order and number of unique trees after deduplication for each order (that is also identical to the number of trees already given in Table 2.1). It was noted by Bornemann [19] that this greedy approach to generating trees is generally as efficient and much easier to code than earlier methods that required the use of efficient integer partition algorithms. Considering that scalar sums (2.39) only need to be generated infrequently and the greedy approach takes less than a minute on a typical modern desktop computer, this makes the greedy approach even more appropriate than in 2002.

Examples of the sequence of data structures used to get from the rooted tree to the nested `sum` expressions for the scalar sums (2.39) are given in Figure 4.3. The steps that correspond to each line in Figure 4.3, with the arrows showing the progression of the algorithm along the columns, are described as follows:

- On the first line of Figure 4.3 are three examples of the rooted trees that have code representing their scalar sums (2.39) shown as examples of code generation.

- On the second line of Figure 4.3 are examples of the PYTHON data structures consisting of nested lists and tuples that get generated greedily and correspond to rooted trees, and that are then sorted to help remove duplicates. This data structure is analogous to the square bracket notation shown in (2.40), except with a few more elements required for a PYTHON implementation. A vertex that branches is represented as a tuple, with `'f'` being the first element of the tuple representing the component of the elementary differential (2.36) corresponding to that vertex, and the second element of the tuple being a list of all the vertices that branch from the current vertex of interest. Any of these vertices in this list that branch further are represented by a similarly constructed tuple to what was just described in that list. Otherwise, vertices that terminate after branching are represented as just the string `'f'` in this list of vertices that branch from the same parent vertex.

  The particular data structure just described was originally conceived to support order conditions (2.43) for *partitioned Runge–Kutta methods* [11][72, pgs.302–308], which are not addressed in this study or currently implemented by the associated software, but could readily be added to the implementation by changing `'f'` to something else. For instance, the strings `'f1'` and `'f2'` have been used together in the past to represent the rooted trees and corresponding elementary differentials (2.36) of 2-partitioned RK methods known as *implicit-explicit Runge–Kutta* methods, see discussion by Ascher, Ruuth, and Spiteri [11], as well as others [94].

- On the third line of Figure 4.3, the data structures are given that begin to build the structure of the scalar sums (2.39) by giving the coupled pairs of indices in the scalar sums (2.39) as tuples, and using integers for terminating branches. The `'f'` is now redundant information for standard RK rooted trees (although required in the specific implementation in `OCSage`), but will be useful for future studies of partitioned methods as mentioned above.

- On the fourth line of Figure 4.3, the components of the rooted tree are now represented by the corresponding coefficients of the Butcher tableau (2.34) according to how the scalar sums (2.39) are constructed; see the discussion in Section 2.5.1. These list elements now correspond to the product of the indexed variables for each term of the specific scalar sum (2.39); observe how the groupings correspond to the multiplication in the sixth line of Figure 4.3 described below.

- On the fifth line of Figure 4.3 is the data structure that gives the set of variables that need to be iterated for each of the `sum` statements in the generated PYTHON code.

- The sixth line of Figure 4.3 is the final PYTHON expression that implements each of the scalar sum (2.39), by expanding to that scalar sum (2.39) itself when evaluated.

Finally, functions to actually use the scalar sums (2.39) for order conditions, error coefficients, and other expressions, such as (2.66), are defined in `orderconditions.py` (which is a PYTHON source file called from `.sage` scripts), along with `orderconditions_scipy.py` for a small number of functions that currently need to be called with PYTHON rather than SAGE.

4.5 Code specific to constructing $5(4)_6$ ERK pairs

The next two sections following this one extensively use code contained in the source file `erk5_6s_functions.sage`, which contains most of the functionality specific to constructing the $5(4)_6$ ERK pairs described in Chapter 3. The `erk5_6s_functions.sage` source file is just under 3000 lines long, which is about 25% of the `OCSage` package, indicating the complexity of fully implementing the mathematics described in Chapter 3 for study. As a comparison, the mathematics describing the classic $5(4)_6$ ERK pairs in Section 2.7 could probably be implemented with several hundred lines of code in commonly used CASs such as MAPLE, MATHEMATICA, and SAGE [57]. However, studying the method construction in Chapter 3 using the thorough and data-intensive methodology of this thesis would be extremely difficult to implement as a simple script using common CASs, and a specialized software tool similar to `OCSage` would still have to be built. There are two nearly identical applications of the functionality contained in `erk5_6s_functions.sage`:

1. Solving the order conditions (2.43) using the SAGE worksheets for exploring the mathematics, either using numerical values for the free parameters or finding expressions leading to the calculated Butcher tableau (2.34) coefficients. These scripts, which are also used for the unit testing described in Section 4.8, contain identical statements to those in the final versions of the worksheets for the SAGE notebook that was used for the mathematics in Section 2.7 and Chapter 3.

2. Solving the order conditions (2.43) in order to generate code giving the Butcher tableau (2.34) coefficients explicitly in terms of the free parameters. This generated code is primarily used to systematically

search the free parameters of ERK methods. The solution of the order conditions done in a way that supports code generation is described in Section 4.7, and using this generated code to search for specific RK formulae is then described in Section 4.9.

Although the mathematical expressions and relevant code are nearly the same, there are slight differences in applying the solution procedures as well as using the software interface for each of the above applications. Generally, each piece of functionality to implement the mathematics in Chapter 3 has three function definitions associated with it in `erk5_6s_functions.sage`:

- A SAGE function `sagenb_«name of functionality»(vardict,**kwds)` that is called for the first application above, i.e., worksheets for the SAGE notebook and unit testing.
- A SAGE function `«name of functionality»(vardict,**kwds)` that is used for the second application above, i.e., code generation.
- A SAGE function `_«name of functionality»(vardict,**kwds)` that is used internally for actually implementing the specific piece of mathematical functionality.

An example of these functions to calculate the $\{p_3, p_4, p_5\}$ vector from equation (3.51) for the family corresponding to Case III derived in Section 3.4.1 is given in Listing 4.1.

Listing 4.1: Functions to implement equation (3.51) used for the family corresponding to Case III derived in Section 3.4.1.

```
def sagenb_erk5g_6s_p_III(**kwds):
    vardict=get_globals_as_dict()
    global p3,p4,p5
    p3,p4,p5 = _erk5g_6s_p_III(vardict,**kwds)

def erk5g_6s_p_III(vardict,**kwds):
    return _erk5g_6s_p_III(vardict,**kwds)

def _erk5g_6s_p_III(vardict,symbolic=False):
    for v in vardict:
        exec("%s = vardict['%s']" % (v,v))
    p3 = -((c2 - c3)*(la + c3*mu)/om)/2
    p3 = factor_if_symbolic(p3)
    p4 = -((c2 - c4)*(la + c4*mu)/om)/2
    p4 = factor_if_symbolic(p4)
    p5 = -((c2 - c5)*(la + c5*mu)/om)/2
    p5 = factor_if_symbolic(p5)
    ocp("Calculated p3,p4,p5 from multipliers: ")
    ocp('p3: ');
    ocv(p3);
    ocp('p4: ');
    ocv(p4);
    ocp('p5: ');
    ocv(p5);
    return p3,p4,p5
```

The `vardict` argument to the functions in Listing 4.1 contains a dictionary of the SAGE variables or the already calculated values for these variables.

Some of the utility functions for output used in the above example are:

- The SAGE function `ocp(obj)` prints the string representation of a SAGE object as textual output, either for terminals or when used within a SAGE worksheet. See some of the description text within the example of a SAGE notebook in Figure 4.1.

- The SAGE function `ocv(obj)` prints SAGE objects as either their textual representation when called from a terminal or uses the pretty-print functionality built into SAGE worksheets. An example of pretty-printing of the algebraic expressions (that SAGE internally formats with LaTeX) is the example of a SAGE notebook in Figure 4.1.

Sometimes the algebraic procedures differ slightly between worksheets and code generation. Selecting the appropriate functionality is easily accomplished by adding a Boolean keyword argument `symbolic=«True/False»` to a small number of appropriate function calls. The keyword argument `constraint_solution=«0/1»` is also used to select one of two subcases for Cases I, II, IV, V, VI described in Section 3.4.1.

There are also global constants defined in `erk5_6s_functions.sage` that give a list of algebraic variables to help easily set up the SAGE notebook worksheets (and scripts derived from them) for the mathematics in Chapter 3. Specific examples of how the functions similar to the example given by Listing 4.1 are utilized in PYTHON and SAGE scripts are described in the appropriate sections below.

4.6 Exploring the mathematics using a SAGE notebook

To initially explore the mathematics described in Chapter 3, worksheets in the SAGE notebook were created, an example of part of one is already shown in Figure 4.1. The functions in the source file `erk5_6s_functions.sage` described in the previous section were developed over the course of an iterative exploration that eventually obtained the specific mathematical formulation of the complete solution of $5(4)_6$ ERK pairs presented in Chapter 3. The final versions of these worksheets use the functions from the source file `erk5_6s_functions.sage` exclusively. If any further exploration is needed, the functions from the source file `erk5_6s_functions.sage` can be used up to the point detailed exploration is desired, after which appropriate algebraic expressions can be used directly in the worksheet so the appropriate mathematics can be explicitly explored.

However, the configuration directories (by default stored in `~/.sage`) used by the SAGE notebooks are not easily portable between computers or easily archived. An appropriate textual format for export cannot be updated automatically as the notebooks are worked on (as opposed to the `iPython` notebook `.pynb` format commonly used for SCIPY-based `iPython` notebook worksheets and its successor JUPYTER[19]) but must be manually exported and imported each time a worksheet is stored in a portable format, transferred, or backed up. For future explorations of RK methods beyond the work in this thesis, using JUPYTER notebooks may be desirable because these support using SAGE in the notebook directly from a file that can easily be transferred between computers, stored in version control, or published.

[19] https://ipython.org/notebook.html

Therefore, to provide the code used in the notebooks for the purposes of this thesis, the statements from the notebooks are transferred to SAGE scripts that can be readily run from a terminal. A user can either run these scripts directly and view the output on a terminal, or the statements can be pasted one by one into a SAGE notebook or a JUPYTER notebook to recreate the notebooks used to explore the mathematics presented in Chapter 3.

An example is given by Listing 4.2, where the script `erk5gIIIs6_unittest.sage` solves for the explicit expressions of the calculated Butcher tableau coefficients in terms of the free parameters for Case III of the complete solution (as can be seen from the name, this script is also used for the unit tests described in Section 4.8) of the six-stage fifth-order ERK order conditions (see Section 3.4.1).

Listing 4.2: The `erk5gIIIs6_unittest.sage` file to demonstrate the solution of the family corresponding to Case III derived in Section 3.4.1 and use it in unit tests.

```
#!/usr/local/bin/sage
import sys

load(os.path.join(os.getenv("OCSAGE_ROOT"),'init.sage'))
load_ocsage('sagenb_erk5_6s_functions.sage');
# set some fixed parameters here:
c2=43/193;c3=257/3121;c4=133/143;c6=1;la=1;mu=3;a63=342/897
# load in some predefined values:
load_ocsage('sagenb_erk5_6s.sage');
om=-5/3*(2*c2-1)*la - (5/3*c2-1)*mu;nu=0;
sagenb_erk5g_6s_mult();
sagenb_erk5g_6s_b6_III();
sagenb_erk5g_6s_p_III();
sagenb_erk5g_6s_q();
sagenb_erk5g_6s_constraint_III();
sagenb_erk5g_6s_q_b1b2();
sagenb_erk5g_6s_higher();
sagenb_erk5g_6s_A1();
sagenb_erk5g_6s_A2();
sagenb_erk5g_6s_A3();
load_ocsage('sagenb_check_erk5_6s.sage');
load_ocsage('sagenb_checkerk.sage')
check_erk(A,b,c);

if len(sys.argv) > 1:
    fh=open(sys.argv[1],'w')
    fh.write(dumps((A,b,c)));
    fh.close()
```

In Listing 4.2, it can clearly be seen that the functions from the source file `erk5_6s_functions.sage` described in the previous section are the only substantial functionality. If a filename is included as a command-line argument to scripts such as the example in Listing 4.2, the SAGE function `dumps` is used to serialize the calculated Butcher tableau (2.34), and it is stored in the filename provided for later verification by the unit tests.

4.6.1 Code for setting up `OCSage`

The `OCSage` package provides the `init.sage` source file that is loaded in the second line of Listing 4.2, which contains functions such as `load_ocsage` that works like the standard the SAGE `load` function to load files in the `OCSage` directory based on the environment variable `$OCSAGE_ROOT`. The `sagenb/` directory in the `OCSage` package contains several source files with functions that help study RK methods in the interactive shell or notebooks, such as initializing the many variables seen in method construction in Chapters 2 and 3 or loading expressions for the order conditions.

4.7 Generating code from the solution of the order conditions

For the vast majority of studies of RK methods, the solution of the order conditions (2.43) is presented only for small sets of specific values (usually rational numbers) selected for the free parameters of the RK family of interest. As mentioned before, in this study, the expressions for the calculated Butcher tableau (2.34) coefficients in terms of the free parameters are purely symbolic. These symbolic solutions are used to generate the code for the search process described in Section 4.9, which allows the characteristic numbers (2.56) for millions of sets of free parameters to be found orders of magnitude faster than when solving the order conditions anew for each individual set of free parameters. If the order conditions (2.43) were algebraically solved anew for each set of free parameters tested, only thousands could be done in several hours, such as was indicated in the studies by Tsitouras [182, 183] (in addition to the issues mentioned in Chapter 3 with using numerical methods to directly solve the order conditions). Although many studies have found explicit symbolic expressions in terms of the free parameters for constructing the RK families of interest, the sheer quantity of code required to support the complete solution described in Chapter 3 in the way it is used in this study would make it extremely difficult to code manually.

Due to the large number of families describing $5(4)_6$ ERK pairs in Section 2.7 and Chapter 3 (as well as other RK methods that are were not derived in thesis), when generating the code that gives the calculated coefficients of a Butcher tableau (2.34) explicitly in terms of the free parameters, it is imperative to reuse the common functionality in solving the order conditions and code generation processes in order to keep the software complexity manageable. Ideally, solving the appropriate systems of equations (in this study this is only the order conditions (2.43), but this can easily include conditions similar to order conditions that are often imposed in other studies of RK methods [20, 21, 142]) in the appropriate sequence should be implemented so that it requires the minimum possible amount of code to specify the relevant information for a particular family. In conjunction with the functions in `erk5_6s_functions.sage`, the `ERKSolve` class and its subclasses that are described in this section largely accomplish this.

The Python classes defined by the `OCSage` source file `rkclasses.py` are:

- The class `ERKSolve` that is used for solving ERK methods using procedures such as those in Section 2.7.
- The subclass `ERKCassitySolve` of `ERKSolve` for six-stage fifth-order methods and $5(4)_{6(6)}$ ERK pairs using CMF defined in Chapter 3.
- The subclass `ERKCassitySolveFSAL` of `ERKSolve` for $5(4)_{6(7)}$ ERK pairs using the CMF defined in Chapter 3.

The `rkclasses.py` source file includes support for parallelism using the `multiprocessing` library. All solutions and code generation for all ERK families studied in this thesis took several weeks of computation time using 4 cores on the desktop computer used for this study. However, development of OCSage required many iterations, and the sometimes large execution time for solving the order conditions during development became onerous during testing. This was especially the case when the procedures were not yet in a more optimized form or when the code still had bugs that could substantially add to the execution time before they became apparent. In these cases, using 4 cores instead of 1 core was found to be even more beneficial than it was for the final data itself.

Solving the order conditions for specific RK families is implemented either by overriding or adding certain methods to one of ERKSolve, ERKCassitySolve, or ERKCassitySolveFSAL. The actual solution of the order conditions occurs by executing the method solve(self) for these classes in order to generate the source files with the generated code, an example of which is given in Listing 4.3, in the directories generated_sage/ and generated_scipy/.

4.7.1 Solving for classic ERK families with the ERKSolve class

An example of using the ERKSolve class to solve for the classic ERK pair family containing the RKF4(5)$_{6(6)}$ pair, described in Section 2.7.2, is given by Listing 4.3.

Listing 4.3: The Fehl45_6 subclass of ERKSolve to set up code generation for the ERK family containing the RKF4(5)$_{6(6)}$ pair.

```
class Fehl45_6(ERKSolve):
    label  = 'fehl45s6'
    n=6
    methodorders=(5,4)
    def banner(self):
        self.wo('##############################################################################')
        self.wo('6 stage ERK Fehlberg 4(5)')
        self.wo('##############################################################################')

    def init(self):
        self.substitutions = {self.b[1]:0}
        self.symbolic_substitutions = {self.c[2]:3/2*self.c[1],
                                       self.c[3]:3*self.c[1] / ((4 - 24*self.c[1]) + 45*self.c[1]*
                                           ↪ self.c[1])}
        self.conditions = [self.step1,
                           self.step2,
                           self.step3,
                           self.step4,
                           self.step5]

    def step1(self,A,b,c):
        conditions = [self.classic_order_conditions[1][0],
                      self.classic_order_conditions[2][0],
                      self.classic_order_conditions[3][0],
                      self.classic_order_conditions[4][0],
                      self.classic_order_conditions[5][0]]
        conditions = self.substitute(conditions)
        variables = [self.b[0],self.b[2],self.b[3],self.b[4],self.b[5]]
        output = solve(conditions,*variables,solution_dict=True)
        return output[0]

    def step2(self,A,b,c):
        conditions = butcher_simplifying_C(self.A,self.b,self.c,2,start=3,end=4) + \
                     butcher_simplifying_C(self.A,self.b,self.c,3,start=3,end=4)
        conditions = self.substitute(conditions)
        variables = [self.A[2][1],
                     self.A[3][1],
                     self.A[3][2]]
        output = solve(conditions,*variables,solution_dict=True)
        return output[0]

    def step3(self,A,b,c):
        conditions = butcher_simplifying_C(self.A,self.b,self.c,2,start=5,end=5) + \
                     butcher_simplifying_C(self.A,self.b,self.c,3,start=5,end=5) + \
                     [erk_column_conditions1(self.A,self.b,2),
                      erk_column_conditions2(self.A,self.b,self.c,2)]
        variables = [self.A[4][1],
                     self.A[4][2],
                     self.A[4][3],
                     self.A[5][1]]
        conditions = self.substitute(conditions)
        output = solve(conditions,*variables,solution_dict=True)
        return output[0]

    def step4(self,A,b,c):
        conditions = [sum(
                        sum(bi*Aij*cj**3
                          for bi,Aij in zip(self.b,Aj))
                            for Aj,cj in zip(self.A.T,self.c)) == 1/20] + \
                     butcher_simplifying_C(self.A,self.b,self.c,2,start=6,end=6) + \
                     butcher_simplifying_C(self.A,self.b,self.c,3,start=6,end=6)
        conditions = self.substitute(conditions)
        variables = [self.A[5][2],self.A[5][3],self.A[5][4]]
        self.wo("----------")
        self.pp(conditions)
        self.wo(variables)
        output = solve(conditions,*variables,solution_dict=True)
        return output[0]

    def step5(self,A,b,c):
        conditions = self.classic_order_conditions[0]
        conditions = self.substitute(conditions)
        variables = [self.A[1][0],
                     self.A[2][0],
                     self.A[3][0],
                     self.A[4][0],
                     self.A[5][0]]
        output = solve(conditions,*variables,solution_dict=True)
        return output[0]

    def init_emb(self):
        self.substitutions_emb = {self.bhat[1]:0,
                                  self.bhat[5]:0}
        self.conditions_emb = [self.step1_emb]

    def step1_emb(self,A,b,c,bhat):
        conditions = [self.classic_order_conditions_emb[1][0],
                      self.classic_order_conditions_emb[2][0],
                      self.classic_order_conditions_emb[3][0],
                      self.classic_order_conditions_emb[4][0]]
        conditions = self.substitute(conditions)
        variables = [bhat[0],bhat[2],bhat[3],bhat[4]]
        output = solve(conditions,*variables,solution_dict=True)
        return output[0]

<<...>>

if __name__ == '__main__':
    STARTTIME = time.time()
    # XXXX: these must be defined before the pool to have effect!
    rkclasses.SUPPRESS_EMB=False
    rkclasses.SUPPRESS_STABILITY=False
    rkclasses.SUPPRESS_ERRORTERMS=False
    rkclasses.REDIRECT_STDOUT=True
    rkclasses.SOLVE_PARALLEL=True
    if rkclasses.SOLVE_PARALLEL:
        from multiprocessing import Pool
        rkclasses.POOL = Pool(processes=4)
    ERROR_TERMS=9
    <<...>>
    rkclasses.rksolvemaster(Fehl45_6,  do_emb=True, do_stability=True,do_errorterms=ERROR_TERMS)
    <<...>>
    if rkclasses.SOLVE_PARALLEL:
        rkclasses.POOL.close()
        rkclasses.POOL.join()
    print("--- %s seconds ---" % (time.time() - STARTTIME))
    print "Preparsing ./generated_sage"
    p = subprocess.Popen(['bash','-c','sage -preparse *.sage'], cwd=os.path.join(os.getcwd(),'
        ↪ generated_sage'))
    p.wait()
```

In Listing 4.3, steps similar to what is described in Section 2.7.2 are defined using the methods `step1(self)` through `step5(self)`. In each of the `step«number»` methods, the relevant order conditions (including any simplifying assumptions or reworking of the order conditions) are defined and solved using the built-in SAGE `solve` function. Analogous to the `step«number»` methods when an embedded pair is being constructed are the `step«number»_emb` methods that are called after the sequence of `step«number»` methods. In this example, it can be noted that the reduced system for the family containing the RKF4(5)$_{6(6)}$ pair (2.78) could in fact be solved simply by using a single call of the built-in SAGE `solve` function. However, experience has shown that except for extremely simple reduced systems, solving with the SAGE `solve` function, as opposed to setting up a sequence of systems, does not necessarily give a good indication whether all solutions are being returned or whether these returned solutions are complete solution of the appropriate reduced system. In fact, once the complexity of the reduced system for the family containing the DP5(4)$_{6(7)}$ pair (2.79) is reached, the SAGE `solve` function (at least in SAGE version 8.0) is no longer sufficient to find an appropriate solution without solving the reduced system as a sequence of solutions.

The attributes provided by the `ERKSolve` class for use while solving are:

- The `classic_order_conditions` attribute contains expressions for the order conditions for the higher-order component of an RK pair.
- The `classic_order_conditions_emb` attribute contains expressions for the order conditions for the higher-order method of an RK pair, but only if the option for solving for an embedded pair is enabled.
- The `A` attribute contains a SAGE matrix of the SAGE variables corresponding to the $\mathbf{A}$ matrix of the Butcher tableau (2.34).
- The `b` attribute contains a SAGE vector of the SAGE variables corresponding to the $\mathbf{b}$ vector of the Butcher tableau (2.34).
- The `c` attribute contains a SAGE vector of the SAGE variables corresponding to the $\mathbf{c}$ vector of the Butcher tableau (2.34).
- The `bhat` attribute contains a SAGE vector of the SAGE variables corresponding to the $\hat{\mathbf{b}}$ vector of the Butcher tableau (2.34), but only if the option for solving for an embedded pair is enabled.

The user-configurable attributes of `ERKSolve` that are demonstrated by Listing 4.3 are:

- The `label` attribute is set as a unique and descriptive string corresponding to each RK family, which is used for tasks such as naming the filenames of the generated code or identifying the ERK method family in the `PostgreSQL` database.
- The `n` attribute is set to the number of stages, the natural name for this attribute would be `s`, but `s` is used elsewhere (for example, see (3.1) and (3.27c)). Because the number of stages is used so often in the code, a single letter attribute is desirable.

- The `methodorders` attribute is set as a Python tuple of the higher and lower method orders, respectively, of an ERK pair. If only a single method is being solved for, rather than a pair, then the second element of the tuple can be `None`. In the case of the RKF4$(5)_{6(6)}$ pair (2.78), although the classic implementation uses the fifth-order method as error estimator, hence the 4(5) in RKF4$(5)_{6(6)}$ pair (2.78); the implementation to construct ERK pairs in `OCSage` requires the higher-order method to be first with `self.methodorders=(5,4)`. The actual implementation of the resulting pair in a numerical solver can choose to use local extrapolation independent of how the order conditions are solved.

- The `substitutions` attribute is set as a dictionary of coefficients or other mathematical variables in the solution that have a numerical value (usually a rational number) they are always equal to.

- The `symbolic_substitutions` attribute is set as a dictionary of coefficients or other mathematical variables in the solution that have algebraic expressions they always equal. The `symbolic_substitutions` attribute must currently be separate from the `substitutions` attribute because internally the values in the `substitutions` attribute need to be applied first.

- The `conditions` attribute is set as a Python list containing the sequence of methods to be run without arguments, generally the user-defined `step«number»` methods (described below), in order to solve the order conditions.

- The `conditions_emb` attribute is set as a list of methods to run without arguments, generally the user-defined `step«number»_emb` methods (described below), in order to solve the order conditions of the lower-order component of an ERK pair. The `conditions_emb` attribute is optional and generally set in the `init_emb` method. The methods listed in the `conditions_emb` attribute are used to solve the order conditions for the lower-order method of an embedded pair after the order conditions for the higher-order method have been solved with the methods listed by the `conditions` attribute.

The methods of the `ERKSolve` class as demonstrated by the example above are:

- The `banner(self)` method is overridden to provide a textual description that is printed to the terminal before solving the order conditions and generating the code.

- The `init(self)` method sets up the class prior to solving the order conditions, it is here that the attributes described above are generally set by the user.

- The `init_emb(self)` method is optional and sets up the class prior to solving the order conditions for the lower-order method of an embedded pair in an analogous manner to `init(self)`.

- The `solve(self)` method is what is called by the user to actually solve the order conditions and is typically called right after instantiating the user-defined subclass of `ERKSolve`.

- The `substitute(self,expressions)` method (only shown in Listing 4.5) can be used like the Sage `substitute` function to substitute already calculated values into a list of expressions given by the `expressions` argument. These used to be done automatically after each function given by the `conditions` attribute, but as already mentioned it was found that this led to excessively large generated expressions.

- The `wo(self,obj)` method and `pp(self,obj)` method are convenience methods for printing and pretty-printing objects to the terminal or SAGE notebook, respectively. The output includes a newline and also assures the textual output can be redirected to a file if the proper options are configured.

- The user-defined `step«number»` methods contain the code for each step of solving the order conditions (for a single ERK method or for the higher-order component of an ERK pair). The `step«number»` methods are also listed in the `conditions` attribute. These methods must return a dictionary giving the solved-for variables and the SAGE expressions giving their solutions.

- The user-defined `step«number»_emb` methods contain the code for each step of solving the order conditions (for the lower-order component of an ERK pair). The `step«number»_emb` methods are also listed in the `conditions` attribute. These methods must return a dictionary giving the solved-for variables and the SAGE expressions giving their solutions.

The generated code

Listing 4.4 demonstrates the generated code through the `generated_scipy/fehl45s6.py` `OCSage` file, that is generated by running the `solve(self)` method from the `Fehl45_6` class in Listing 4.3.

Listing 4.4: A representative sample of the generated SCIPY code for the ERK family containing RKF4(5)$_{6(6)}$ pair (2.78).

```
from __future__ import division
from scipy import array
from scipy import roots
from scipy import linalg
from scipy import sqrt
import scipy as sp
import numpy as np

def fehl45s6_coefficients(x):
    c2 = x[0]
    c5 = x[1]
    c6 = x[2]
    b2 = 0
    c4 = 3*c2/(45*c2**2 - 24*c2 + 4)
    c3 = 3/2*c2
    b6=-1/20*(675*c2**3 - 780*c2**2 - 20*(45*c2**3 - 51*c2**2 + 18*c2 - 2)*c5 + 282*c2 - 32)/(2*(45*
      ↪ c2**2 - 24*c2 + 4)*c6**4 - 9*c2**2*c5*c6 - (135*c2**3 - 72*c2**2 + 2*(45*c2**2 - 24*c2 +
      ↪ 4)*c5 + 18*c2)*c6**3 + 9*(c2**2 + (15*c2**3 - 8*c2**2 + 2*c2)*c5)*c6**2)
    b5=-1/20*(675*c2**3 - 780*c2**2 - 20*(45*c2**3 - 51*c2**2 + 18*c2 - 2)*c6 + 282*c2 - 32)/(2*(45*
      ↪ c2**2 - 24*c2 + 4)*c5**4 + 9*c2**2*c5**2 - 9*(15*c2**3 - 8*c2**2 + 2*c2)*c5**3 - (2*(45*c2
      ↪ **2 - 24*c2 + 4)*c5**3 + 9*c2**2*c5 - 9*(15*c2**3 - 8*c2**2 + 2*c2)*c5**2)*c6)
    b4=1/540*(184528125*c2**9 - 492075000*c2**8 + 590490000*c2**7 - 419904000*c2**6 + 195372000*c2**5
      ↪  - 61772544*c2**4 + 13291776*c2**3 - 1880064*c2**2 - 30*(8201250*c2**9 - 21596625*c2**8 +
      ↪ 25660800*c2**7 - 18098640*c2**6 + 8361792*c2**5 - 2627424*c2**4 + 562176*c2**3 - 79104*c2
      ↪ **2 + 6656*c2 - 256)*c5 - 10*(24603750*c2**9 - 64789875*c2**8 + 76982400*c2**7 - 54295920*
      ↪ c2**6 + 25085376*c2**5 - 7882272*c2**4 + 1686528*c2**3 - 237312*c2**2 - (36905625*c2**9 -
      ↪ 95134500*c2**8 + 111099600*c2**7 - 77215680*c2**6 + 35217504*c2**5 - 10938240*c2**4 +
      ↪ 2315520*c2**3 - 322560*c2**2 + 26880*c2 - 1024)*c5 + 19968*c2 - 768)*c6 + 158976*c2 -
      ↪ 6144)/(405*c2**6 - 216*c2**5 + 18*c2**4 - 3*(2025*c2**7 - 2160*c2**6 + 846*c2**5 - 144*c2
      ↪ **4 + 8*c2**3)*c5 - (6075*c2**7 - 6480*c2**6 + 2538*c2**5 - 432*c2**4 + 24*c2**3 - (91125*
      ↪ c2**8 - 145800*c2**7 + 98010*c2**6 - 35424*c2**5 + 7200*c2**4 - 768*c2**3 + 32*c2**2)*c5)*
      ↪ c6)
```

```
b3=4/135*(540*c2**2 - 15*(45*c2**2 - 28*c2 + 4)*c5 - 5*(135*c2**2 - 2*(90*c2**2 - 57*c2 + 8)*c5 -
    ↪  84*c2 + 12)*c6 - 333*c2 + 48)/(405*c2**6 - 216*c2**5 + 18*c2**4 - 6*(45*c2**5 - 24*c2**4
    ↪ + 2*c2**3)*c5 - 2*(135*c2**5 - 72*c2**4 + 6*c2**3 - 2*(45*c2**4 - 24*c2**3 + 2*c2**2)*c5)*
    ↪ c6)
b1=-1/540*(2025*c2**3 - 2340*c2**2 - 60*(45*c2**3 - 51*c2**2 + 18*c2 - 2)*c5 - 10*(270*c2**3 -
    ↪ 306*c2**2 - (405*c2**3 - 450*c2**2 + 150*c2 - 16)*c5 + 108*c2 - 12)*c6 + 846*c2 - 96)/(c2
    ↪ **2*c5*c6)
a43=-6*(45*c2**3 - 24*c2**2 + 2*c2)/(91125*c2**6 - 145800*c2**5 + 102060*c2**4 - 39744*c2**3 +
    ↪ 9072*c2**2 - 1152*c2 + 64)
a42=9/2*(135*c2**3 - 72*c2**2 + 8*c2)/(91125*c2**6 - 145800*c2**5 + 102060*c2**4 - 39744*c2**3 +
    ↪ 9072*c2**2 - 1152*c2 + 64)
a32=9/8*c2
a62=-1/2*(135*a32*b3*c2**3 - 72*a32*b3*c2**2 + 6*(2*a32*b3 + a42*b4)*c2 - 2*(45*(a32*b3 + a42*b4)
    ↪ *c2**2 + 4*a32*b3 + 4*a42*b4 - 24*(a32*b3 + a42*b4)*c2)*c5)/(45*b6*c2**2*c6 - 24*b6*c2*c6
    ↪ - (45*b6*c2**2 - 24*b6*c2 + 4*b6)*c5 + 4*b6*c6)
a54=-1/54*(18225*a32*b3*c2**7 - 2430*(a32*b3*(5*c6 + 8) + 5*a42*b4*c6)*c2**6 + 162*(4*a32*b3*(20*
    ↪ c6 + 13) + 5*a42*b4*(16*c6 + 1))*c2**5 - 432*(a32*b3*(13*c6 + 4) + a42*b4*(13*c6 + 1))*c2
    ↪ **4 - 4*(2025*b5*c2**4 - 2160*b5*c2**3 + 936*b5*c2**2 - 192*b5*c2 + 16*b5)*c5**4 + 72*(a42
    ↪ *b4*(16*c6 + 1) + 2*a32*b3*(8*c6 + 1))*c2**3 + (18225*b5*c2**5 + 1620*b5*c2**4*(5*c6 - 12)
    ↪  - 216*b5*c2**3*(40*c6 - 39) + 288*b5*c2**2*(13*c6 - 6) - 48*b5*c2*(16*c6 - 3) + 64*b5*c6)
    ↪ *c5**3 - 96*(a32*b3*c6 + a42*b4*c6)*c2**2 - 9*(2025*b5*c2**5*c6 - 2160*b5*c2**4*c6 + 936*
    ↪ b5*c2**3*c6 - 192*b5*c2**2*c6 + 16*b5*c2*c6)*c5**2)/(45*b5*c2**4*c6 - 24*b5*c2**3*c6 + 2*
    ↪ b5*c2**2*c6 - (45*b5*c2**4 - 24*b5*c2**3 + 2*b5*c2**2)*c5)
a53=-2/27*(18225*a32*b3*c2**7 - 2430*(a32*b3*(5*c6 + 8) + 5*a42*b4*c6)*c2**6 + 81*(a32*b3*(160*c6
    ↪  + 89) + 10*a42*b4*(16*c6 + 1))*c2**5 - 54*(a32*b3*(89*c6 + 20) + a42*b4*(89*c6 + 8))*c2
    ↪ **4 + 2*(2025*b5*c2**4 - 2160*b5*c2**3 + 936*b5*c2**2 - 192*b5*c2 + 16*b5)*c5**4 + 18*(a42
    ↪ *b4*(40*c6 + 1) + 2*a32*b3*(20*c6 + 1))*c2**3 - (4050*b5*c2**4*c6 - 135*b5*c2**3*(32*c6 -
    ↪ 3) + 72*b5*c2**2*(26*c6 - 3) - 12*b5*c2*(32*c6 - 3) + 32*b5*c6)*c5**3 - 24*(a32*b3*c6 +
    ↪ a42*b4*c6)*c2**2 + 9*(45*b5*c2**3*c6 - 24*b5*c2**2*c6 + 4*b5*c2*c6)*c5**2)/(2025*b5*c2**6*
    ↪ c6 - 2160*b5*c2**5*c6 + 846*b5*c2**4*c6 - 144*b5*c2**3*c6 + 8*b5*c2**2*c6 - (2025*b5*c2**6
    ↪  - 2160*b5*c2**5 + 846*b5*c2**4 - 144*b5*c2**3 + 8*b5*c2**2)*c5)
a52=1/2*(135*a32*b3*c2**3 - 18*(a32*b3*(5*c6 + 4) + 5*a42*b4*c6)*c2**2 - 8*a32*b3*c6 - 8*a42*b4*
    ↪ c6 + 6*(a42*b4*(8*c6 + 1) + 2*a32*b3*(4*c6 + 1))*c2)/(45*b5*c2**2*c6 - 24*b5*c2*c6 - (45*
    ↪ b5*c2**2 - 24*b5*c2 + 4*b5)*c5 + 4*b5*c6)
a65=-1/20*(455625*(8*a32*b3 + (8*a42 + 27*a43)*b4 + (8*a52 + 27*a53)*b5 - 4*a62*b6)*c2**9 -
    ↪ 729000*(8*a32*b3 + (8*a42 + 27*a43)*b4 + (8*a52 + 27*a53)*b5 - 4*a62*b6)*c2**8 +
    ↪ 24300*(168*a32*b3 + 21*(8*a42 + 27*a43)*b4 + 21*(8*a52 + 27*a53)*b5 + (75*c6**3 - 79*a62)*
    ↪ b6)*c2**7 - 270*(5888*a32*b3 + 736*(8*a42 + 27*a43)*b4 + 736*(8*a52 + 27*a53)*b5 + (10800*
    ↪ c6**3 + 675*c6**2 - 2464*a62)*b6 + 675)*c2**6 + 1080*(336*a32*b3 + 42*(8*a42 + 27*a43)*b4
    ↪ + 42*(8*a52 + 27*a53)*b5 + (1965*c6**3 + 180*c6**2 - 116*a62)*b6 + 270)*c2**5 - 360*(128*
    ↪ a32*b3 + 16*(8*a42 + 27*a43)*b4 + 16*(8*a52 + 27*a53)*b5 + 2*(1224*c6**3 + 117*c6**2 - 16*
    ↪ a62)*b6 + 567)*c2**4 + 8*(320*a32*b3 + 40*(8*a42 + 27*a43)*b4 + 5*(64*a52 + 216*a53 + 27*
    ↪ a54)*b5 + 40*(684*c6**3 + 54*c6**2 - a62)*b6 + 9936)*c2**3 - 96*(5*(64*c6**3 + 3*c6**2)*b6
    ↪  + 189)*c2**2 + 384*(5*b6*c6**3 + 6)*c2 - 128)/(2*(91125*b6*c2**6 - 145800*b6*c2**5 +
    ↪ 102060*b6*c2**4 - 39744*b6*c2**3 + 9072*b6*c2**2 - 1152*b6*c2 + 64*b6)*c5**3 - 9*(30375*b6
    ↪ *c2**7 - 48600*b6*c2**6 + 35370*b6*c2**5 - 14688*b6*c2**4 + 3648*b6*c2**3 - 512*b6*c2**2 +
    ↪  32*b6*c2)*c5**2 + 9*(2025*b6*c2**6 - 2160*b6*c2**5 + 936*b6*c2**4 - 192*b6*c2**3 + 16*b6*
    ↪ c2**2)*c5)
a64=1/540*(1366875*(8*a32*b3 + (8*a42 + 27*a43)*b4 + (8*a52 + 27*a53)*b5 - 4*a62*b6)*c2**9 -
    ↪ 2187000*(8*a32*b3 + (8*a42 + 27*a43)*b4 + (8*a52 + 27*a53)*b5 - 4*a62*b6)*c2**8 +
    ↪ 218700*(56*a32*b3 + 7*(8*a42 + 27*a43)*b4 + 7*(8*a52 + 27*a53)*b5 + (25*c6**3 - 28*a62)*b6
    ↪ )*c2**7 - 810*(5888*a32*b3 + 736*(8*a42 + 27*a43)*b4 + 736*(8*a52 + 27*a53)*b5 + 16*(675*
    ↪ c6**3 - 184*a62)*b6 + 675)*c2**6 + 19440*(56*a32*b3 + 7*(8*a42 + 27*a43)*b4 + 7*(8*a52 +
    ↪ 27*a53)*b5 + 7*(45*c6**3 - 4*a62)*b6 + 45)*c2**5 - 1080*(128*a32*b3 + 16*(8*a42 + 27*a43)*
    ↪ b4 + 16*(8*a52 + 27*a53)*b5 + 32*(69*c6**3 - 2*a62)*b6 + 567)*c2**4 + 24*(320*a32*b3 +
    ↪ 40*(8*a42 + 27*a43)*b4 + 5*(64*a52 + 216*a53 + 27*a54)*b5 + 40*(567*c6**3 - 4*a62)*b6 +
    ↪ 9936)*c2**3 - 864*(80*b6*c6**3 + 63)*c2**2 + 768*(5*b6*c6**3 + 9)*c2 + 10*(546750*a62*b6*
    ↪ c2**8 - 54675*(15*c6**2 + 16*a62)*b6*c2**7 + 14580*(25*c6**3 + 90*c6**2 + 42*a62)*b6*c2**6
    ↪  - 324*(1800*c6**3 + 2835*c6**2 + 736*a62)*b6*c2**5 + 3888*(105*c6**3 + 92*c6**2 + 14*a62)
    ↪ *b6*c2**4 - 432*(368*c6**3 + 189*c6**2 + 16*a62)*b6*c2**3 + 192*(189*c6**3 + 54*c6**2 + 2*
    ↪ a62)*b6*c2**2 + 256*b6*c6**3 - 576*(8*c6**3 + c6**2)*b6*c2)*c5 - 384)/(135*b6*c2**5 - 72*
    ↪ b6*c2**4 + 6*b6*c2**3 - (2025*b6*c2**6 - 2160*b6*c2**5 + 846*b6*c2**4 - 144*b6*c2**3 + 8*
    ↪ b6*c2**2)*c5)
a63=-1/135*(1366875*(8*a32*b3 + (8*a42 + 27*a43)*b4 + (8*a52 + 27*a53)*b5 + 8*a62*b6)*c2**9 -
    ↪ 2187000*(8*a32*b3 + (8*a42 + 27*a43)*b4 + (8*a52 + 27*a53)*b5 + 8*a62*b6)*c2**8 +
    ↪ 72900*(168*a32*b3 + 21*(8*a42 + 27*a43)*b4 + 21*(8*a52 + 27*a53)*b5 + 158*a62*b6)*c2**7 -
    ↪ 810*(5888*a32*b3 + 736*(8*a42 + 27*a43)*b4 + 736*(8*a52 + 27*a53)*b5 + 4928*a62*b6 + 675)*
    ↪ c2**6 + 3240*(336*a32*b3 + 42*(8*a42 + 27*a43)*b4 + 42*(8*a52 + 27*a53)*b5 + (75*c6**3 +
    ↪ 232*a62)*b6 + 270)*c2**5 - 1080*(128*a32*b3 + 16*(8*a42 + 27*a43)*b4 + 16*(8*a52 + 27*a53)
    ↪ *b5 + 16*(15*c6**3 + 4*a62)*b6 + 567)*c2**4 + 24*(320*a32*b3 + 40*(8*a42 + 27*a43)*b4 +
    ↪ 5*(64*a52 + 216*a53 + 27*a54)*b5 + 40*(117*c6**3 + 2*a62)*b6 + 9936)*c2**3 - 288*(80*b6*c6
    ↪ **3 + 189)*c2**2 + 384*(5*b6*c6**3 + 18)*c2 - 20*(546750*a62*b6*c2**8 - 874800*a62*b6*c2
    ↪ **7 - 7290*(25*c6**3 - 79*a62)*b6*c2**6 + 81*(3600*c6**3 + 225*c6**2 - 2464*a62)*b6*c2**5
    ↪ - 648*(315*c6**3 + 30*c6**2 - 58*a62)*b6*c2**4 + 216*(368*c6**3 + 39*c6**2 - 16*a62)*b6*c2
    ↪ **3 - 96*(189*c6**3 + 18*c6**2 - a62)*b6*c2**2 - 128*b6*c6**3 + 144*(16*c6**3 + c6**2)*b6*
    ↪ c2)*c5 - 384)/(273375*b6*c2**9 - 437400*b6*c2**8 + 294030*b6*c2**7 - 106272*b6*c2**6 +
    ↪ 21600*b6*c2**5 - 2304*b6*c2**4 + 96*b6*c2**3 - 2*(91125*b6*c2**8 - 145800*b6*c2**7 +
    ↪ 98010*b6*c2**6 - 35424*b6*c2**5 + 7200*b6*c2**4 - 768*b6*c2**3 + 32*b6*c2**2)*c5)
a51=-a52 - a53 - a54 + c5
a41=-(45*(a42 + a43)*c2**2 - 3*(8*a42 + 8*a43 + 1)*c2 + 4*a42 + 4*a43)/(45*c2**2 - 24*c2 + 4)
a61=-a62 - a63 - a64 - a65 + c6
```

```
    a31=-a32 + 3/2*c2
    a21=c2
    bhat4=1/54*(546750*c2**7 - 1148175*c2**6 + 1049760*c2**5 - 544644*c2**4 + 173664*c2**3 - 34128*c2
        ↪ **2 - (820125*c2**7 - 1676700*c2**6 + 1501740*c2**5 - 765936*c2**4 + 240624*c2**3 - 46656*
        ↪ c2**2 + 5184*c2 - 256)*c5 + 3840*c2 - 192)/(135*c2**5 - 72*c2**4 + 6*c2**3 - (2025*c2**6 -
        ↪  2160*c2**5 + 846*c2**4 - 144*c2**3 + 8*c2**2)*c5)
    bhat3=2/27*(135*c2**2 - 2*(90*c2**2 - 57*c2 + 8)*c5 - 84*c2 + 12)/(135*c2**5 - 72*c2**4 + 6*c2**3
        ↪  - 2*(45*c2**4 - 24*c2**3 + 2*c2**2)*c5)
    bhat1=1/54*(270*c2**3 - 306*c2**2 - (405*c2**3 - 450*c2**2 + 150*c2 - 16)*c5 + 108*c2 - 12)/(c2
        ↪ **2*c5)
    bhat5=-(45*c2**3 - 51*c2**2 + 18*c2 - 2)/(2*(45*c2**2 - 24*c2 + 4)*c5**3 + 9*c2**2*c5 - 9*(15*c2
        ↪ **3 - 8*c2**2 + 2*c2)*c5**2)
    return c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,
        ↪ a21,bhat4,bhat3,bhat1,bhat5

def fehl45s6_tableau(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return {'A':array([[0,0,0,0,0,0],
                       [a21,0,0,0,0,0],
                       [a31,a32,0,0,0,0],
                       [a41,a42,a43,0,0,0],
                       [a51,a52,a53,a54,0,0],
                       [a61,a62,a63,a64,a65,0]]),
            'b':array([b1,0,b3,b4,b5,b6]),
    'c':array([0,c2,3/2*c2,3*c2/(45*c2**2 - 24*c2 + 4),c5,c6]),
    'bhat':array([bhat1,0,bhat3,bhat4,bhat5,0])}

def fehl45s6_numpy_mask(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    param_mask = sp.zeros(x[0].shape,dtype=bool)
    param_mask = param_mask | sp.logical_not(sp.isreal(b6))
    param_mask = param_mask | sp.logical_not(sp.isreal(b5))
    param_mask = param_mask | sp.logical_not(sp.isreal(b4))
    param_mask = param_mask | sp.logical_not(sp.isreal(b3))
    param_mask = param_mask | sp.logical_not(sp.isreal(b1))
    param_mask = param_mask | sp.logical_not(sp.isreal(a43))
    param_mask = param_mask | sp.logical_not(sp.isreal(a42))
    param_mask = param_mask | sp.logical_not(sp.isreal(a32))
    param_mask = param_mask | sp.logical_not(sp.isreal(a62))
    param_mask = param_mask | sp.logical_not(sp.isreal(a54))
    param_mask = param_mask | sp.logical_not(sp.isreal(a53))
    param_mask = param_mask | sp.logical_not(sp.isreal(a52))
    param_mask = param_mask | sp.logical_not(sp.isreal(a65))
    param_mask = param_mask | sp.logical_not(sp.isreal(a64))
    param_mask = param_mask | sp.logical_not(sp.isreal(a63))
    param_mask = param_mask | sp.logical_not(sp.isreal(a51))
    param_mask = param_mask | sp.logical_not(sp.isreal(a41))
    param_mask = param_mask | sp.logical_not(sp.isreal(a61))
    param_mask = param_mask | sp.logical_not(sp.isreal(a31))
    param_mask = param_mask | sp.logical_not(sp.isreal(a21))
    param_mask = param_mask | sp.logical_not(sp.isreal(bhat4))
    param_mask = param_mask | sp.logical_not(sp.isreal(bhat3))
    param_mask = param_mask | sp.logical_not(sp.isreal(bhat1))
    param_mask = param_mask | sp.logical_not(sp.isreal(bhat5))
    return param_mask

def fehl45s6_max_coefficient_magnitude(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    coefficient_list = [b1,0,b3,b4,b5,b6] + [bhat1,0,bhat3,bhat4,bhat5,0] + [0,0,0,0,0,0,a21
        ↪ ,0,0,0,0,0,a31,a32,0,0,0,0,a41,a42,a43,0,0,0,a51,a52,a53,a54,0,0,a61,a62,a63,a64,a65,0] +
        ↪ [0,c2,3/2*c2,3*c2/(45*c2**2 - 24*c2 + 4),c5,c6]
    coefficient_max = sp.zeros_like(coefficient_list[0])
    for c in coefficient_list:
        coefficient_max = sp.maximum(coefficient_max,sp.absolute(c))
    return coefficient_max
```

```
def fehl45s6_min_c_distance(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    c_list = [0,c2,3/2*c2,3*c2/(45*c2**2 - 24*c2 + 4),c5,c6]
    c_list[0] = np.zeros_like(c2)
    if len([0,c2,3/2*c2,3*c2/(45*c2**2 - 24*c2 + 4),c5,c6]) == 7:
        c_list =  c_list[:-1]
        c_list[-1] = np.ones_like(c2)
    else:
        if not hasattr(c_list[-1],'shape'):
            c_list[-1] = np.ones_like(c2)
    c_array = np.stack(c_list,axis=0)
    min_c_distance = np.min(np.diff(np.sort(c_array,axis=0),axis=0),axis=0)
    return min_c_distance

def fehl45s6_b_diff(x,cached_coefficients=None):
    # XXXX: workaround for fehl45s6
    bhat6=None
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    if bhat6 is None:
        return sp.nan*sp.ones_like(c2)
    else:
        return abs(b6-bhat6)

def fehl45s6_orders(x,cached_coefficients=None):
    return (5, 4)

def fehl45s6_stability(x,z,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return 1/2*(2*(45*a32*a43*a54*a65*b6*c2**3 - 24*a32*a43*a54*a65*b6*c2**2 + 4*a32*a43*a54*a65*b6*
        ↪ c2)*z**6 + (45*(2*a32*a43*a54*b5 + (2*a32*a43*a64 + (2*a32*a53 + (2*a42 + 3*a43)*a54)*a65)
        ↪ *b6)*c2**3 - 24*(2*a32*a43*a54*b5 + (2*a32*a43*a64 + (2*a32*a53 + (2*a42 + 3*a43)*a54)*a65
        ↪ )*b6)*c2**2 + 4*(2*a32*a43*a54*b5 + (2*a32*a43*a64 + (2*a32*a53 + (2*a42 + 3*a43)*a54)*a65
        ↪ )*b6)*c2)*z**5 + (45*(2*a32*a43*b4 + (2*a32*a53 + (2*a42 + 3*a43)*a54)*b5 + (2*a32*a63 +
        ↪ (2*a42 + 3*a43)*a64 + (2*a52 + 3*a53)*a65)*b6)*c2**3 - 24*(2*a32*a43*b4 + (2*a32*a53 + (2*
        ↪ a42 + 3*a43)*a54)*b5 + (2*a32*a63 + (2*a42 + 3*a43)*a64 + (2*a52 + 3*a53)*a65)*b6)*c2**2 +
        ↪  2*(4*a32*a43*b4 + 2*(2*a32*a53 + (2*a42 + 3*a43)*a54)*b5 + (4*a32*a63 + 2*(2*a42 + 3*a43)
        ↪ *a64 + (4*a52 + 6*a53 + 3*a54)*a65)*b6)*c2)*z**4 + (45*(2*a32*b3 + (2*a42 + 3*a43)*b4 +
        ↪ (2*a52 + 3*a53)*b5 + (2*a62 + 3*a63)*b6)*c2**3 - 24*(2*a32*b3 + (2*a42 + 3*a43)*b4 + (2*
        ↪ a52 + 3*a53)*b5 + (2*a62 + 3*a63)*b6)*c2**2 + 2*(4*a32*b3 + 2*(2*a42 + 3*a43)*b4 + (4*a52
        ↪ + 6*a53 + 3*a54)*b5 + (4*a62 + 6*a63 + 3*a64)*b6)*c2 + 2*(45*a65*b6*c2**2 - 24*a65*b6*c2 +
        ↪  4*a65*b6)*c5)*z**3 + (135*b3*c2**3 - 72*b3*c2**2 + 6*(2*b3 + b4)*c2 + 2*(45*b5*c2**2 -
        ↪ 24*b5*c2 + 4*b5)*c5 + 2*(45*b6*c2**2 - 24*b6*c2 + 4*b6)*c6)*z**2 + 90*c2**2 + 2*(45*(b1 +
        ↪ b3 + b4 + b5 + b6)*c2**2 - 24*(b1 + b3 + b4 + b5 + b6)*c2 + 4*b1 + 4*b3 + 4*b4 + 4*b5 + 4*
        ↪ b6)*z - 48*c2 + 8)/(45*c2**2 - 24*c2 + 4)

def fehl45s6_stability_length(x,cached_coefficients=None):
    eps=1e-12
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    if type(x[0]) == sp.ndarray:
        stability_length_array = sp.empty_like(x[0])
        poly = sp.array([<<...>>])
    else:
       stability_length_array = sp.array([0.])
       poly = sp.array([<<...>>])
    poly_size_one = poly[0].shape
    new_poly = []
    for e in poly:
        if not type(e) is sp.ndarray:
            new_poly.append(sp.ones(poly_size_one)*e)
        else:
            new_poly.append(e)
    poly = sp.array(new_poly)
    it = sp.nditer(poly[0], flags=['multi_index','refs_ok'])
    while not it.finished:
        # if sp.isfinite(it):
        if True:
            poly_individual = sp.array([t[it.multi_index] for t in poly])
            poly_p1 = sp.array(poly_individual,copy=True)
            poly_n1 = sp.array(poly_individual,copy=True)
            poly_p1[-1] = poly_p1[-1] - 1.
            poly_n1[-1] = poly_n1[-1] + 1.
            try:
                roots_p = sp.roots(poly_p1)
                roots_n = sp.roots(poly_n1)
            except linalg.LinAlgError:
                roots_p = [0.]
                roots_n = [0.]
            real_p = -sp.inf
            real_n = -sp.inf
            for root in roots_p:
                if abs(root.imag) < eps and root.real < -eps:
                    real_p = max(real_p,root.real)
            for root in roots_n:
                if abs(root.imag) < eps and root.real < -eps:
                    real_n = max(real_n,root.real)
            stability_length_array[it.multi_index] = -max(real_n,real_p)
        it.iternext()
    return stability_length_array

def fehl45s6_stability_emb(x,z,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return 1/2*(2*(45*a32*a43*a54*bhat5*c2**3 - 24*a32*a43*a54*bhat5*c2**2 + 4*a32*a43*a54*bhat5*c2)*
        ↪ z**5 + (45*(2*a32*a43*bhat4 + (2*a32*a53 + (2*a42 + 3*a43)*a54)*bhat5)*c2**3 - 24*(2*a32*
        ↪ a43*bhat4 + (2*a32*a53 + (2*a42 + 3*a43)*a54)*bhat5)*c2**2 + 4*(2*a32*a43*bhat4 + (2*a32*
        ↪ a53 + (2*a42 + 3*a43)*a54)*bhat5)*c2)*z**4 + (45*(2*a32*bhat3 + (2*a42 + 3*a43)*bhat4 +
        ↪ (2*a52 + 3*a53)*bhat5)*c2**3 - 24*(2*a32*bhat3 + (2*a42 + 3*a43)*bhat4 + (2*a52 + 3*a53)*
        ↪ bhat5)*c2**2 + 2*(4*a32*bhat3 + 2*(2*a42 + 3*a43)*bhat4 + (4*a52 + 6*a53 + 3*a54)*bhat5)*
        ↪ c2)*z**3 + (135*bhat3*c2**3 - 72*bhat3*c2**2 + 6*(2*bhat3 + bhat4)*c2 + 2*(45*bhat5*c2**2
        ↪ - 24*bhat5*c2 + 4*bhat5)*c5)*z**2 + 90*c2**2 + 2*(45*(bhat1 + bhat3 + bhat4 + bhat5)*c2**2
        ↪  - 24*(bhat1 + bhat3 + bhat4 + bhat5)*c2 + 4*bhat1 + 4*bhat3 + 4*bhat4 + 4*bhat5)*z - 48*
        ↪ c2 + 8)/(45*c2**2 - 24*c2 + 4)

def fehl45s6_stability_emb_length(x,cached_coefficients=None):
    eps=1e-12
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    if type(x[0]) == sp.ndarray:
        stability_length_array = sp.empty_like(x[0])
        poly = sp.array([<<...>>])
    else:
        stability_length_array = sp.array([0.])
        poly = sp.array([<<...>>])
    poly_size_one = poly[0].shape
    new_poly = []
    for e in poly:
        if not type(e) is sp.ndarray:
            new_poly.append(sp.ones(poly_size_one)*e)
        else:
            new_poly.append(e)
    poly = sp.array(new_poly)
    it = sp.nditer(poly[0], flags=['multi_index','refs_ok'])
    while not it.finished:
        # if sp.isfinite(it):
        if True:
            poly_individual = sp.array([t[it.multi_index] for t in poly])
            poly_p1 = sp.array(poly_individual,copy=True)
            poly_n1 = sp.array(poly_individual,copy=True)
            poly_p1[-1] = poly_p1[-1] - 1.
            poly_n1[-1] = poly_n1[-1] + 1.
            try:
                roots_p = sp.roots(poly_p1)
                roots_n = sp.roots(poly_n1)
            except linalg.LinAlgError:
                roots_p = [0.]
                roots_n = [0.]
            real_p = -sp.inf
            real_n = -sp.inf
            for root in roots_p:
                if abs(root.imag) < eps and root.real < -eps:
                    real_p = max(real_p,root.real)
            for root in roots_n:
                if abs(root.imag) < eps and root.real < -eps:
                    real_n = max(real_n,root.real)
            stability_length_array[it.multi_index] = -max(real_n,real_p)
        it.iternext()
    return stability_length_array

def fehl45s6_error_6(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return sqrt(<<....>>)

def fehl45s6_error_trees_6(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return array([<<....>>])

def fehl45s6_error_conditions_6(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return sqrt(<<....>>)

def fehl45s6_error_conditions_trees_6(x,cached_coefficients=None):
    if cached_coefficients:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = cached_coefficients
    else:
        c2,c5,c6,b2,c4,c3,b6,b5,b4,b3,b1,a43,a42,a32,a62,a54,a53,a52,a65,a64,a63,a51,a41,a61,a31,a21,
            ↪ bhat4,bhat3,bhat1,bhat5 = fehl45s6_coefficients(x)
    return array([<<....>>])

<<....>>
```

The initial `import` statements ensure the proper libraries are loaded. The `«...»` placeholder shows where large expressions have been omitted; they can be found in full in the `OCSage` code. The last placeholder in Listing 4.4 indicates the last four functions (described below) are duplicated to whatever order of error coefficients are necessary. Similar sets of four functions are also generated for the lower-order method of an embedded pair.

Generating code describing mathematical expressions for PYTHON-family interpreters from a PYTHON or SAGE script turns out to be remarkably easy. The string representations (that are obtained by the `__repr__` method built into every PYTHON object) of the objects containing SAGE expressions are executable SAGE code. Only a small number of changes are necessary to make this code executable in a standard PYTHON interpreter using SCIPY, e.g., changing the `^` operator to `**`, ensuring appropriate library functions are imported for operations such as square roots, etc.

In order to search the space of free parameters (described below in Section 4.9) as efficiently as possible, most of the generated SCIPY functions [91] other than `«label»_tableau` can be used *vectorized*, i.e, the argument `x` to the functions in Listing 4.4 can accept SAGE or SCIPY vectors and matrices for the free parameters, and give back the desired characteristic numbers (2.56) as a vector or matrix, respectively. Note that although the generated function for the Butcher tableau `fehl45s6_tableau(x)` in Listing 4.4 cannot be vectorized because it returns a dictionary, this does not substantially affect any studies because the actual Butcher tableau coefficients only need to be generated when performance testing them and the computational work to test them on sets of IVPs greatly exceeds the time spent calculating the corresponding tableaux. This vectorization reduces the penalty of using a high-level language like PYTHON to the minimum by taking advantage of the high-performance array operations provided by SCIPY [91].

The functions generated as shown in Listing 4.4 are:

- The `fehl45s6_tableau` function returns a dictionary containing the calculated values of the Butcher tableau (2.34) based on the free parameters given by the argument `x`.
- The `fehl45s6_numpy_mask` function is generated code giving a NUMPY array mask representing where invalid values (often `nan` or complex numbers) occur for any calculated coefficients. Use of this function is not strictly necessary to use the generated code in Listing 4.4 for the classic case of RKF4(5)$_{6(6)}$ pair (2.78). However, masks are essential to using vectorization for certain ERK families, such as Cases I, II, IV derived in Section 3.4.1 and many other potential RK families, where the expressions for the coefficients can evaluate to complex numbers or otherwise invalid values. Checking for these invalid values and returning a mask is of great assistance in using the results of these generated functions, including tasks such as plotting with MATPLOTLIB. The white patches within the coloured areas in the figures in Sections 4.12 and 4.13 are invalid values, i.e., `nan`, `inf`, or complex values, that are handled seamlessly by NUMPY and MATPLOTLIB once masked off.
- The `fehl45s6_max_coefficient_magnitude` function returns the maximum absolute value of the Butcher tableau coefficients (2.34).
- The `fehl45s6_orders` function returns the tuple of the orders for the ERK family containing the RKF4(5)$_{6(6)}$ pair (2.78).
- The `fehl45s6_stability` function returns the stability polynomial (a rational expression in generated SAGE code or numerical value in generated SCIPY code)) for the fifth-order method of the ERK family containing the RKF4(5)$_{6(6)}$ pair (2.78).
- The `fehl45s6_stability_length` function returns the length of the stability region for the fifth-order method of the ERK family containing the RKF4(5)$_{6(6)}$ pair (2.78).
- The `fehl45s6_stability_emb` function returns the stability polynomial (that is generally an exact numerical value or symbolic expression when using generated SAGE code, or a floating-point value when using generated SCIPY code) for the fourth-order method of the ERK family containing the RKF4(5)$_{6(6)}$ pair (2.78).

- The `fehl45s6_stability_emb_length` function returns the length of the stability region for the fourth-order method of the ERK family containing the RKF4(5)$_{6(6)}$ pair (2.78).
- The `fehl45s6_error_6` returns the sixth-order error coefficient of the fifth-order method in the ERK family containing the RKF4(5)$_{6(6)}$ pair (2.78).
- The `fehl45s6_error_trees_6` returns the individual values of the PECs (2.47) rather than their norm.
- The `fehl45s6_error_conditions_6` function returns the norm of the $\overline{\text{PEC}}$s (2.48).
- The `fehl45s6_error_conditions_trees_6` function returns the individual values of the $\overline{\text{PEC}}$s (2.48) rather than their norm.

4.7.2 Solving for ERK families with CMF with the `CassitySolve` class

The `ERKCassitySolve` subclass of `ERKSolve` supports generating code for the specific $5(4)_6$ ERK pairs that use CMF and are described in Chapter 3. The main additional functionality added by the `ERKCassitySolve` subclass is that the calculated variables used extensively in Chapter 3 are added as class members, e.g., $F_6, G_3, \mathbf{p}, \mathbf{s}, \mathbf{r}-\mathbf{s}, r_3^*, s_3^*, r_4^*, r_4^\dagger, r_5^*, \lambda, \mu, \nu, \omega$, as well as specialized methods for substituting these variables into expressions properly. An example of using the `ERKCassitySolve` class that sets up the solution of the order conditions and code generation for the family corresponding to Case III derived in Section 3.4.1 is given in Listing 4.5.

Listing 4.5: The `ERKCondensed_erk5gso1s6_mixin` mixin and `ERKCondensed_erk5gIIIs6` subclass of `ERKSolve` setting up code generation for the family corresponding to Case III derived in Section 3.4.1.

```
class ERKCondensed_erk5gso1s6_mixin(object):
    def step1(self,A,b,c):
        vardict=deepcopy(self.vardict)

        G3 = erk5g_6s_mult(vardict,symbolic=True)
        G3 = self.substitute(G3)
        return {self.G3:G3}

    def step2(self,A,b,c):
        # calculate b6, etc.
        raise NotImplementedError

    def step3(self,A,b,c):
        # find constraints
        raise NotImplementedError

    def step4(self,A,b,c):
        vardict=deepcopy(self.vardict)
        b3,b4,b5 = erk5g_6s_q(vardict,symbolic=True)
        return {self.b[2]:b3,
                self.b[3]:b4,
                self.b[4]:b5}

    def step5(self,A,b,c):
        vardict=deepcopy(self.vardict)
        # find Q using real values
        b1,b2 = erk5g_6s_q_b1b2(vardict,symbolic=True)
        return {self.b[0]:b1,
                self.b[1]:b2}

    def step6(self,A,b,c):
        # find constraints
        raise NotImplementedError
    def step8(self,A,b,c):
        vardict=deepcopy(self.vardict)
        a21,a32,a54,a65 = erk5g_6s_A1(vardict,symbolic=True)
        return {self.A[1][0]:a21,
                self.A[2][1]:a32,
                self.A[4][3]:a54,
                self.A[5][4]:a65}

    def step9(self,A,b,c):
        vardict=deepcopy(self.vardict)
        a42,a43,a52,a53,a62,a64 = erk5g_6s_A2(vardict,symbolic=True)
        return {self.A[3][1]:a42,
                self.A[3][2]:a43,
                self.A[4][1]:a52,
                self.A[4][2]:a53,
                self.A[5][1]:a62,
                self.A[5][3]:a64}

    def step10(self,A,b,c):
        vardict=deepcopy(self.vardict)
        _,_,_,a31,a41,a51,a61 = erk5g_6s_A3(vardict,symbolic=True)
        return {self.A[2][0]:a31,
                self.A[3][0]:a41,
                self.A[4][0]:a51,
                self.A[5][0]:a61}

class ERKCondensed_erk5gIIIs6_mixin(ERKCondensed_erk5gso1s6_mixin,ERKCassitySolve):
    ""
    def init(self):
        self.substitutions = {self.nu:0,
                              self.c[5]:1}
        self.symbolic_substitutions = {self.om:-5/3*(2*self.c[1]-1)*self.la - (5/3*self.c[1]-1)*self.
            ↪ mu}
        self.conditions = [self.step1,
                           self.step2,
                           self.step3,
                           self.step4,
                           self.step5,
                           self.step6,
                           self.step7,
                           self.step8,
                           self.step9,
                           self.step10]

    def step2(self,A,b,c):
        vardict=deepcopy(self.vardict)
        # calculate b6 symbolically
        G3 = erk5g_6s_mult(vardict,symbolic=True)
        G3 = self.substitute(G3)
        vardict['G3']=G3
        # p6,b6 = erk5g_6s_b6_III(3/5,self.b[5],self.p[5],self.F6,G3,0,self.mu,-3/5*self.om,self.om)
        p6,b6 = erk5g_6s_b6_III(vardict,symbolic=True)
        return {self.b[5]:b6,
                self.p[5]:p6}

    def step3(self,A,b,c):
        vardict=deepcopy(self.vardict)
        # TODO: update to case 2
        # calculate Q symbolically
        b3,b4,b5 = erk5g_6s_q(vardict,symbolic=True)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        # calculate p
        p3,p4,p5 = erk5g_6s_p_III(vardict,symbolic=True)
        vardict['p3']=p3
        vardict['p4']=p4
        vardict['p5']=p5
        b3,b4,b5,b6,c5,p5 = erk5g_6s_constraint_III(vardict,symbolic=True)
        # return restriction to give formula for c5
        return {self.c[4]:c5}

    def step6(self,A,b,c):
        vardict=deepcopy(self.vardict)
        # find p using real values
        # p3,p4,p5 = erk5g_6s_p_III(self.c[1],self.c[2],self.c[3],self.c[4],self.p[2],self.p[3],self.
            ↪ p[4],self.la,self.mu,self.nu,self.om)
        p3,p4,p5 = erk5g_6s_p_III(vardict,symbolic=True)
        return {self.p[2]:p3,
                self.p[3]:p4,
                self.p[4]:p5}

    def step7(self,A,b,c):
        vardict=deepcopy(self.vardict)
        p3,p4,p5=erk5g_6s_p_III(vardict,symbolic=True)
        vardict['p3']=p3
        vardict['p4']=p4
        vardict['p5']=p5
        r3,r4,r5,rs3,rs4,s3,s4 = erk5g_6s_higher(vardict,symbolic=True)
        return {self.r[2]:r3,
                self.r[3]:r4,
                self.r[4]:r5,
                self.rs[2]:rs3,
                self.rs[3]:rs4,
                self.s[2]:s3,
                self.s[3]:s4}

class ERKCondensed_erk5gIIIs6(ERKCondensed_erk5gIIIs6_mixin,ERKCassitySolve):
    ""
    label = 'erk5gIIIs6'
    n=6
    methodorders=(5,None)
    def banner(self):
        print '################################################################################'
        print '6 Stage ERK 5 with stage order 1 and b_2 \neq 0, case III, solved using condensed
            ↪ matrix notation'
        print '################################################################################'

class ERKCondensed_erk5gIIIs6FSAL7(ERKCondensed_erk5gIIIs6,ERKCassitySolve):
    ""
    label = 'erk5gIIIs6FSAL7'
    n=7
    methodorders=(5,4)
    def banner(self):
        print '################################################################################'
        print '6 Stage ERK 5 with stage order 1 and b_2 \neq 0, case III FSAL, solved using condensed
            ↪  matrix notation'
        print '################################################################################'

    def init(self):
        ERKCondensed_erk5gIIIs6_mixin.init(self)
        self.substitutions.update({self.c[6]:1,
                                   self.b[6]:0})
        self.conditions = [self.step0a,
                           self.step0b,
                           self.step1,
                           self.step2,
                           self.step3,
                           self.step4,
                           self.step5,
                           self.step6,
                           self.step7,
                           self.step8,
                           self.step9,
                           self.step10,
                           self.step11]

    def init_emb(self):
        self.symbolic_substitutions_emb = {}
        self.substitutions_emb = {}
        self.conditions_emb = [self.step1_emb,
                               self.step2_emb]

    def step0a(self,A,b,c):
        vardict=deepcopy(self.vardict)
        vardict['bhat7']=var('bhat7')
        vardict['c7']=1
        G3 = erk5g_6s_mult(vardict,symbolic=True)
        G3 = self.substitute(G3)
        vardict['G3']=G3
        p6,b6 = erk5g_6s_b6_III(vardict)
        vardict['b6']=b6
        vardict['p6']=p6
        b3,b4,b5 = erk5g_6s_q(vardict,symbolic=True)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        p3,p4,p5 = erk5g_6s_p_III(vardict,symbolic=True)
        vardict['p3']=p3
        vardict['p4']=p4
        vardict['p5']=p5
        b3,b4,b5,b6,c5,p5 = erk5g_6s_constraint_III(vardict)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        vardict['b6']=b6
        vardict['c5']=c5
        vardict['p5']=p5
        r3,r4,r5,rs3,rs4,s3,s4 = erk5g_6s_higher(vardict,symbolic=True)
        vardict['r3']=r3
        vardict['r4']=r4
        vardict['r5']=r5
        vardict['rs3']=rs3
        vardict['rs4']=rs4
        vardict['s3']=s3
        vardict['s4']=s4
        a21,a32,a54,a65 = erk5g_6s_A1(vardict,symbolic=True)
        vardict['a21'] = a21
        vardict['a32'] = a32
        vardict['a54'] = a54
        vardict['a65'] = a65
        a42,a43,a52,a53,a62,a64 = erk5g_6s_A2(vardict,symbolic=True)
        vardict['a42'] = a42
        vardict['a43'] = a43
        vardict['a52'] = a52
        vardict['a53'] = a53
        vardict['a62'] = a62
        vardict['a64'] = a64
        bhat3,bhat4,bhat5,bhat6 = erk5g_6s_q_emb_FSAL(vardict)
        vardict['bhat3'] = bhat3
        vardict['bhat4'] = bhat4
        vardict['bhat5'] = bhat5
        vardict['bhat6'] = bhat6
        a63,c3,c4 = erk5g_6s_higher_FSAL(vardict,case=3);
        return {self.c[3]:c4}

    def step0b(self,A,b,c):
        vardict=deepcopy(self.vardict)
        vardict['bhat7']=var('bhat7')
        vardict['c7']=1
        G3 = erk5g_6s_mult(vardict,symbolic=True)
        G3 = self.substitute(G3)
        vardict['G3']=G3
        p6,b6 = erk5g_6s_b6_III(vardict)
        vardict['b6']=b6
        vardict['p6']=p6
        b3,b4,b5 = erk5g_6s_q(vardict,symbolic=True)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        p3,p4,p5 = erk5g_6s_p_III(vardict,symbolic=True)
        vardict['p3']=p3
        vardict['p4']=p4
        vardict['p5']=p5
        b3,b4,b5,b6,c5,p5 = erk5g_6s_constraint_III(vardict)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        vardict['b6']=b6
        vardict['c5']=c5
        vardict['p5']=p5
        r3,r4,r5,rs3,rs4,s3,s4 = erk5g_6s_higher(vardict,symbolic=True)
        vardict['r3']=r3
        vardict['r4']=r4
        vardict['r5']=r5
        vardict['rs3']=rs3
        vardict['rs4']=rs4
        vardict['s3']=s3
        vardict['s4']=s4
        a21,a32,a54,a65 = erk5g_6s_A1(vardict,symbolic=True)
        vardict['a21'] = a21
        vardict['a32'] = a32
        vardict['a54'] = a54
        vardict['a65'] = a65
        a42,a43,a52,a53,a62,a64 = erk5g_6s_A2(vardict,symbolic=True)
        vardict['a42'] = a42
        vardict['a43'] = a43
        vardict['a52'] = a52
        vardict['a53'] = a53
        vardict['a62'] = a62
        vardict['a64'] = a64
        bhat3,bhat4,bhat5,bhat6 = erk5g_6s_q_emb_FSAL(vardict)
        vardict['bhat3'] = bhat3
        vardict['bhat4'] = bhat4
        vardict['bhat5'] = bhat5
        vardict['bhat6'] = bhat6
        a63,c3,c4 = erk5g_6s_higher_FSAL(vardict,case=3);
        return {self.A[5][2]:a63}

    def step11(self,A,b,c):
        return {A[6][0]:b[0],
                A[6][1]:b[1],
                A[6][2]:b[2],
                A[6][3]:b[3],
                A[6][4]:b[4],
                A[6][5]:b[5]}

    def step1_emb(self,A,b,c,bhat):
        vardict=deepcopy(self.vardict)
        G3 = erk5g_6s_mult(vardict,symbolic=True)
        G3 = self.substitute(G3)
        vardict['G3']=G3
        p3,p4,p5 = erk5g_6s_p_III(vardict,symbolic=True)
        vardict['p3']=p3
        vardict['p4']=p4
        vardict['p5']=p5
        p6,b6 = erk5g_6s_b6_III(vardict,symbolic=True)
        vardict['b6']=b6
        vardict['p6']=p6
        b3,b4,b5 = erk5g_6s_q(vardict)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        b3,b4,b5,b6,c5,p5 = erk5g_6s_constraint_III(vardict,symbolic=True)
        vardict['b3']=b3
        vardict['b4']=b4
        vardict['b5']=b5
        vardict['b6']=b6
        vardict['p5']=p5
        vardict['p6']=p6
        vardict['b6']=var('b6')
        b6 = erk5g_6s_b6_II_full(vardict)
        bhat3,bhat4,bhat5,bhat6 = erk5g_6s_q_emb_FSAL(vardict)
        vardict['bhat3'] = bhat3
        vardict['bhat4'] = bhat4
        vardict['bhat5'] = bhat5
        vardict['bhat6'] = bhat6
        return {self.bhat[2]:bhat3,
                self.bhat[3]:bhat4,
                self.bhat[4]:bhat5,
                self.bhat[5]:bhat6}

    def step2_emb(self,A,b,c,bhat):
        vardict=deepcopy(self.vardict)
        bhat1,bhat2=erk5g_6s_q_bhat1bhat2_FSAL(vardict);
        return {self.bhat[0]:bhat1,
                self.bhat[1]:bhat2}
```

In Listing 4.5, it can clearly be seen that the general-purpose functions for the mathematics from Chapter 3 that are contained in `erk5_6s_functions.sage`, already described in Section 4.5, are used extensively. It can also be noted that in some methods in Listing 4.5 that certain mathematical operations are repeated, such as the `step3` method of the `ERKCondensed_erk5gIIIs6` class where the $\mathbf{p}$ vector is solved for again. This repetition was found to be the best way to deal with issues where certain Sage expressions needed to be substituted into other expressions, without either requiring all expressions substituted in or requiring additional functionality to cache computations in the `ERKSolve` class. Because solving the order conditions and generating the code for all families studied in this thesis takes only a few hours, adding software complexity to remove this repetition of code and calculations is not desirable at this time.

Also notice the use of *mixins*, which allow adding code to a class without being a parent class. Mixins allow many types of code reuse without leading to the well-known issues that multiple inheritance can cause, e.g., the “diamond problem” [140, pgs.347–369]. Mixins are used to ensure that there is minimal repetition of code for supporting all of the cases, subcases, and types of embedded pairs described in Chapter 3.

Although setting up the generation of code for a corresponding embedded pair is not shown in Listing 4.5 due to space considerations, solving the order conditions and generating code for the families of embedded pairs derived in Chapter 3 follows a similar pattern to that used for the family containing the RKF4(5)$_{6(6)}$ method (2.78) in Listing 4.3 and can be found in the `erk_cmf.sage` source file contained within `OCSage`.

The generated code

The generated code for the Butcher tableau (2.34) of ERK methods in the family corresponding to Case III is given by Listing 4.6.

Listing 4.6: A representative sample of the generated code for the family corresponding to Case III of the complete solution of six-stage fifth-order.

```
def erk5gIIIs6FSAL7_coefficients(x):
    bhat7 = x[0]
    c2 = x[1]
    c3 = x[2]
    la = x[3]
    mu = x[4]
    b7 = 0
    c7 = 1
    c6 = 1
    nu = 0
    om = -5/3*(2*c2 - 1)*la - 1/3*(5*c2 - 3)*mu
    c4=3*(10*c2*c3*la + 5*c2*c3*mu - 5*c3*la + 3*la)*(c3*mu + la)*c2*la/(200*c2**2*c3**2*la**3 + 200*
        ↪ c2**2*c3**2*la**2*mu + 20*c2**2*c3**2*la*mu**2 - 15*c2**2*c3**2*mu**3 - 200*c2*c3**2*la**3
        ↪  - 120*c2**2*c3*la**2*mu - 160*c2*c3**2*la**2*mu - 60*c2**2*c3*la*mu**2 - 15*c2*c3**2*la*
        ↪ mu**2 - 90*c2**2*la**3 + 60*c2*c3*la**3 + 50*c3**2*la**3 - 45*c2**2*la**2*mu + 90*c2*c3*la
        ↪ **2*mu + 30*c3**2*la**2*mu + 18*c2*c3*la*mu**2 + 45*c2*la**3 - 30*c3*la**3 + 18*c2*la**2*
        ↪ mu - 18*c3*la**2*mu)
    a63=1/18*(2560000*c2**5*c3**4*c4**3*la**5*mu + 640000*c2**5*c3**4*c4**3*la**4*mu**2 - 224000*c2
        ↪ **5*c3**4*c4**3*la**3*mu**3 - 40000*c2**5*c3**4*c4**3*la**2*mu**4 + 4400*c2**5*c3**4*c4
        ↪ **3*la*mu**5 - 100*c2**5*c3**4*c4**3*mu**6 + 160000*c2**5*c3**4*c4**2*la**6 + 2400000*c2
        ↪ **5*c3**3*c4**3*la**6 - 4600000*c2**5*c3**4*c4**2*la**5*mu - 4090000*c2**5*c3**3*c4**3*la
        ↪ **5*mu - 5440000*c2**4*c3**4*c4**3*la**5*mu - 1698000*c2**5*c3**4*c4**2*la**4*mu**2 -
        ↪ 1895000*c2**5*c3**3*c4**3*la**4*mu**2 - 1376000*c2**4*c3**4*c4**3*la**4*mu**2 + 188000*c2
        ↪ **5*c3**4*c4**2*la**3*mu**3 + 154500*c2**5*c3**3*c4**3*la**3*mu**3 + 264000*c2**4*c3**4*c4
        ↪ **3*la**3*mu**3 + 72700*c2**5*c3**4*c4**2*la**2*mu**4 + 79450*c2**5*c3**3*c4**3*la**2*mu
        ↪ **4 + 50200*c2**4*c3**4*c4**3*la**2*mu**4 - 3300*c2**5*c3**4*c4**2*la*mu**5 - 2350*c2**5*
        ↪ c3**3*c4**3*la*mu**5 - 2500*c2**4*c3**4*c4**3*la*mu**5 - 90000*c2**5*c3**4*c4*la**6 -
        ↪ 4890000*c2**5*c3**3*c4**2*la**6 - 290000*c2**4*c3**4*c4**2*la**6 - 4350000*c2**5*c3**2*c4
        ↪ **3*la**6 - 5150000*c2**4*c3**3*c4**3*la**6 + 2901000*c2**5*c3**4*c4*la**5*mu + 6159000*c2
        ↪ **5*c3**3*c4**2*la**5*mu + 9699000*c2**4*c3**4*c4**2*la**5*mu + 1365000*c2**5*c3**2*c4**3*
        ↪ la**5*mu + 8638000*c2**4*c3**3*c4**3*la**5*mu + 4570000*c2**3*c3**4*c4**3*la**5*mu +
        ↪ 1431000*c2**5*c3**4*c4*la**4*mu**2 + 3645000*c2**5*c3**3*c4**2*la**4*mu**2 + 3310500*c2
        ↪ **4*c3**4*c4**2*la**4*mu**2 + 1573500*c2**5*c3**2*c4**3*la**4*mu**2 + 3573000*c2**4*c3**3*
        ↪ c4**3*la**4*mu**2 + 1063500*c2**3*c3**4*c4**3*la**4*mu**2 + 63450*c2**5*c3**4*c4*la**3*mu
        ↪ **3 + 34050*c2**5*c3**3*c4**2*la**3*mu**3 - 179350*c2**4*c3**4*c4**2*la**3*mu**3 + 129450*
        ↪ c2**5*c3**2*c4**3*la**3*mu**3 - 132700*c2**4*c3**3*c4**3*la**3*mu**3 - 90600*c2**3*c3**4*
        ↪ c4**3*la**3*mu**3 - 30675*c2**5*c3**4*c4*la**2*mu**4 - 107625*c2**5*c3**3*c4**2*la**2*mu
        ↪ **4 - 82800*c2**4*c3**4*c4**2*la**2*mu**4 - 33450*c2**5*c3**2*c4**3*la**2*mu**4 - 86725*c2
        ↪ **4*c3**3*c4**3*la**2*mu**4 - 15325*c2**3*c3**4*c4**3*la**2*mu**4 + 1350*c2**5*c3**3*c4
        ↪ **2*la*mu**5 + 1800*c2**4*c3**4*c4**2*la*mu**5 - 750*c2**5*c3**2*c4**3*la*mu**5 + 1170*c2
        ↪ **4*c3**3*c4**3*la*mu**5 + 3114000*c2**5*c3**3*c4*la**6 + 165000*c2**4*c3**4*c4*la**6 +
        ↪ 8004000*c2**5*c3**2*c4**2*la**6 + 10233000*c2**4*c3**3*c4**2*la**6 + 195000*c2**3*c3**4*c4
        ↪ **2*la**6 + 2676000*c2**5*c3*c4**3*la**6 + 9315000*c2**4*c3**2*c4**3*la**6 + 4375000*c2
        ↪ **3*c3**3*c4**3*la**6 - 634500*c2**5*c3**4*la**5*mu - 3204000*c2**5*c3**3*c4*la**5*mu -
        ↪ 6021000*c2**4*c3**4*c4*la**5*mu - 1066500*c2**5*c3**2*c4**2*la**5*mu - 13486500*c2**4*c3
        ↪ **3*c4**2*la**5*mu - 8116500*c2**3*c3**4*c4**2*la**5*mu + 651000*c2**5*c3*c4**3*la**5*mu -
        ↪  3058500*c2**4*c3**2*c4**3*la**5*mu - 7308000*c2**3*c3**3*c4**3*la**5*mu - 1895000*c2**2*
        ↪ c3**4*c4**3*la**5*mu - 386100*c2**5*c3**4*la**4*mu**2 - 2398500*c2**5*c3**3*c4*la**4*mu**2
        ↪  - 2623050*c2**4*c3**4*c4*la**4*mu**2 - 2354400*c2**5*c3**2*c4**2*la**4*mu**2 - 6821550*c2
        ↪ **4*c3**3*c4**2*la**4*mu**2 - 2375850*c2**3*c3**4*c4**2*la**4*mu**2 - 318600*c2**5*c3*c4
        ↪ **3*la**4*mu**2 - 2771550*c2**4*c3**2*c4**3*la**4*mu**2 - 2493300*c2**3*c3**3*c4**3*la**4*
        ↪ mu**2 - 351500*c2**2*c3**4*c4**3*la**4*mu**2 - 58725*c2**5*c3**4*la**3*mu**3 - 229950*c2
        ↪ **5*c3**3*c4*la**3*mu**3 - 130350*c2**4*c3**4*c4*la**3*mu**3 - 272325*c2**5*c3**2*c4**2*la
        ↪ **3*mu**3 - 155025*c2**4*c3**3*c4**2*la**3*mu**3 + 34425*c2**3*c3**4*c4**2*la**3*mu**3 -
        ↪ 88500*c2**5*c3*c4**3*la**3*mu**3 - 205875*c2**4*c3**2*c4**3*la**3*mu**3 + 16175*c2**3*c3
        ↪ **3*c4**3*la**3*mu**3 + 7300*c2**2*c3**4*c4**3*la**3*mu**3 + 35325*c2**5*c3**3*c4*la**2*mu
        ↪ **4 + 32850*c2**4*c3**4*c4*la**2*mu**4 + 38475*c2**5*c3**2*c4**2*la**2*mu**4 + 119925*c2
        ↪ **4*c3**3*c4**2*la**2*mu**4 + 23400*c2**3*c3**4*c4**2*la**2*mu**4 - 1875*c2**5*c3*c4**3*la
        ↪ **2*mu**4 + 34185*c2**4*c3**2*c4**3*la**2*mu**4 + 23535*c2**3*c3**3*c4**3*la**2*mu**4 +
        ↪ 450*c2**5*c3**2*c4**2*la*mu**5 - 810*c2**4*c3**3*c4**2*la*mu**5 + 450*c2**4*c3**2*c4**3*la
        ↪ *mu**5 - 634500*c2**5*c3**3*la**6 - 4837500*c2**5*c3**2*c4*la**6 - 6417000*c2**4*c3**3*c4*
        ↪ la**6 - 112500*c2**3*c3**4*c4*la**6 - 4491000*c2**5*c3*c4**2*la**6 - 17068500*c2**4*c3**2*
        ↪ c4**2*la**6 - 8478000*c2**3*c3**3*c4**2*la**6 - 57500*c2**2*c3**4*c4**2*la**6 - 558000*c2
        ↪ **5*c4**3*la**6 - 5700000*c2**4*c3*c4**3*la**6 - 7897500*c2**3*c3**2*c4**3*la**6 -
        ↪ 1837500*c2**2*c3**3*c4**3*la**6 + 576900*c2**5*c3**3*la**5*mu + 1291500*c2**4*c3**4*la**5*
        ↪ mu - 26100*c2**5*c3**2*c4*la**5*mu + 7277400*c2**4*c3**3*c4*la**5*mu + 4970250*c2**3*c3
        ↪ **4*c4*la**5*mu - 1413000*c2**5*c3*c4**2*la**5*mu + 3168000*c2**4*c3**2*c4**2*la**5*mu +
        ↪ 11853450*c2**3*c3**3*c4**2*la**5*mu + 3367250*c2**2*c3**4*c4**2*la**5*mu - 333000*c2**5*c4
        ↪ **3*la**5*mu - 1094400*c2**4*c3*c4**3*la**5*mu + 2855250*c2**3*c3**2*c4**3*la**5*mu +
        ↪ 3099500*c2**2*c3**3*c4**3*la**5*mu + 387500*c2*c3**4*c4**3*la**5*mu + 530325*c2**5*c3**3*
        ↪ la**4*mu**2 + 679950*c2**4*c3**4*la**4*mu**2 + 1188675*c2**5*c3**2*c4*la**4*mu**2 +
        ↪ 4481100*c2**4*c3**3*c4*la**4*mu**2 + 1780875*c2**3*c3**4*c4*la**4*mu**2 + 344700*c2**5*c3*
        ↪ c4**2*la**4*mu**2 + 4356225*c2**4*c3**2*c4**2*la**4*mu**2 + 4752450*c2**3*c3**3*c4**2*la
        ↪ **4*mu**2 + 743550*c2**2*c3**4*c4**2*la**4*mu**2 - 53550*c2**5*c4**3*la**4*mu**2 + 552150*
        ↪ c2**4*c3*c4**3*la**4*mu**2 + 1825200*c2**3*c3**2*c4**3*la**4*mu**2 + 763525*c2**2*c3**3*c4
        ↪ **3*la**4*mu**2 + 41875*c2*c3**4*c4**3*la**4*mu**2 + 90225*c2**5*c3**3*la**3*mu**3 +
        ↪ 87300*c2**4*c3**4*la**3*mu**3 + 192825*c2**5*c3**2*c4*la**3*mu**3 + 378990*c2**4*c3**3*c4*
        ↪ la**3*mu**3 + 81900*c2**3*c3**4*c4*la**3*mu**3 + 110025*c2**5*c3*c4**2*la**3*mu**3 +
        ↪ 424980*c2**4*c3**2*c4**2*la**3*mu**3 + 132930*c2**3*c3**3*c4**2*la**3*mu**3 + 4050*c2**2*
        ↪ c3**4*c4**2*la**3*mu**3 - 1125*c2**5*c4**3*la**3*mu**3 + 118110*c2**4*c3*c4**3*la**3*mu**3
        ↪  + 105840*c2**3*c3**2*c4**3*la**3*mu**3 + 5670*c2**2*c3**3*c4**3*la**3*mu**3 - 11025*c2
        ↪ **5*c3**2*c4*la**2*mu**4 - 40770*c2**4*c3**3*c4*la**2*mu**4 - 8775*c2**3*c3**4*c4*la**2*mu
        ↪ **4 - 43695*c2**4*c3**2*c4**2*la**2*mu**4 - 33480*c2**3*c3**3*c4**2*la**2*mu**4 + 1350*c2
        ↪ **4*c3*c4**3*la**2*mu**4 - 8775*c2**3*c3**2*c4**3*la**2*mu**4 - 270*c2**4*c3**2*c4**2*la*
        ↪ mu**5 + 963000*c2**5*c3**2*la**6 + 1291500*c2**4*c3**3*la**6 + 2532600*c2**5*c3*c4*la**6 +
        ↪  10278900*c2**4*c3**2*c4*la**6 + 5242500*c2**3*c3**3*c4*la**6 + 33750*c2**2*c3**4*c4*la**6
        ↪  + 846000*c2**5*c4**2*la**6 + 9747000*c2**4*c3*c4**2*la**6 + 14409000*c2**3*c3**2*c4**2*la
        ↪ **6 + 3473250*c2**2*c3**3*c4**2*la**6 + 6250*c2*c3**4*c4**2*la**6 + 1179000*c2**4*c4**3*la
        ↪ **6 + 4806000*c2**3*c3*c4**3*la**6 + 3311250*c2**2*c3**2*c4**3*la**6 + 381250*c2*c3**3*c4
        ↪ **3*la**6 + 109800*c2**5*c3**2*la**5*mu - 1360350*c2**4*c3**3*la**5*mu - 1045125*c2**3*c3
        ↪ **4*la**5*mu + 972450*c2**5*c3*c4*la**5*mu - 776250*c2**4*c3**2*c4*la**5*mu - 6614100*c2
        ↪ **3*c3**3*c4*la**5*mu - 2040000*c2**2*c3**4*c4*la**5*mu + 493200*c2**5*c4**2*la**5*mu +
        ↪ 2448900*c2**4*c3*c4**2*la**5*mu - 3685275*c2**3*c3**2*c4**2*la**5*mu - 5221725*c2**2*c3
        ↪ **3*c4**2*la**5*mu - 691875*c2*c3**4*c4**2*la**5*mu + 600300*c2**4*c4**3*la**5*mu +
        ↪ 598050*c2**3*c3*c4**3*la**5*mu - 1378875*c2**2*c3**2*c4**3*la**5*mu - 659375*c2*c3**3*c4
        ↪ **3*la**5*mu - 31250*c3**4*c4**3*la**5*mu - 203400*c2**5*c3**2*la**4*mu**2 - 997515*c2**4*
        ↪ c3**3*la**4*mu**2 - 443250*c2**3*c3**4*la**4*mu**2 - 97425*c2**5*c3*c4*la**4*mu**2 -
        ↪ 2333475*c2**4*c3**2*c4*la**4*mu**2 - 3118455*c2**3*c3**3*c4*la**4*mu**2 - 531000*c2**2*c3
        ↪ **4*c4*la**4*mu**2 + 70650*c2**5*c4**2*la**4*mu**2 - 678555*c2**4*c3*c4**2*la**4*mu**2 -
        ↪ 3021210*c2**3*c3**2*c4**2*la**4*mu**2 - 1461330*c2**2*c3**3*c4**2*la**4*mu**2 - 85500*c2*
        ↪ c3**4*c4**2*la**4*mu**2 + 78345*c2**4*c4**3*la**4*mu**2 - 366795*c2**3*c3*c4**3*la**4*mu
        ↪ **2 - 535650*c2**2*c3**2*c4**3*la**4*mu**2 - 86625*c2*c3**3*c4**3*la**4*mu**2 - 45225*c2
        ↪ **5*c3**2*la**3*mu**3 - 141210*c2**4*c3**3*la**3*mu**3 - 42525*c2**3*c3**4*la**3*mu**3 -
        ↪ 46800*c2**5*c3*c4*la**3*mu**3 - 307260*c2**4*c3**2*c4*la**3*mu**3 - 206820*c2**3*c3**3*c4*
        ↪ la**3*mu**3 - 16200*c2**2*c3**4*c4*la**3*mu**3 - 450*c2**5*c4**2*la**3*mu**3 - 161820*c2
        ↪ **4*c3*c4**2*la**3*mu**3 - 218430*c2**3*c3**2*c4**2*la**3*mu**3 - 31860*c2**2*c3**3*c4**2*
        ↪ la**3*mu**3 + 900*c2**4*c4**3*la**3*mu**3 - 51030*c2**3*c3*c4**3*la**3*mu**3 - 17550*c2
        ↪ **2*c3**2*c4**3*la**3*mu**3 + 675*c2**5*c3*c4*la**2*mu**4 + 13770*c2**4*c3**2*c4*la**2*mu
        ↪ **4 + 11745*c2**3*c3**3*c4*la**2*mu**4 + 540*c2**4*c3*c4**2*la**2*mu**4 + 12420*c2**3*c3
        ↪ **2*c4**2*la**2*mu**4 - 135*c2**3*c3*c4**3*la**2*mu**4 - 479250*c2**5*c3*la**6 - 2040300*
        ↪ c2**4*c3**2*la**6 - 1045125*c2**3*c3**3*la**6 - 438750*c2**5*c4*la**6 - 5567850*c2**4*c3*
        ↪ c4*la**6 - 8654175*c2**3*c3**2*c4*la**6 - 2121750*c2**2*c3**3*c4*la**6 - 3750*c2*c3**4*c4*
        ↪ la**6 - 1872900*c2**4*c4**2*la**6 - 8354250*c2**3*c3*c4**2*la**6 - 6014625*c2**2*c3**2*c4
        ↪ **2*la**6 - 703125*c2*c3**3*c4**2*la**6 - 985500*c2**3*c4**3*la**6 - 2004000*c2**2*c3*c4
        ↪ **3*la**6 - 686250*c2*c3**2*c4**3*la**6 - 31250*c3**3*c4**3*la**6 - 215325*c2**5*c3*la**5*
        ↪ mu - 22770*c2**4*c3**2*la**5*mu + 1271700*c2**3*c3**3*la**5*mu + 420750*c2**2*c3**4*la**5*
        ↪ mu - 257175*c2**5*c4*la**5*mu - 1718370*c2**4*c3*c4*la**5*mu + 1513620*c2**3*c3**2*c4*la
        ↪ **5*mu + 2999250*c2**2*c3**3*c4*la**5*mu + 416250*c2*c3**4*c4*la**5*mu - 949410*c2**4*c4
        ↪ **2*la**5*mu - 1427130*c2**3*c3*c4**2*la**5*mu + 2085300*c2**2*c3**2*c4**2*la**5*mu +
        ↪ 1150875*c2*c3**3*c4**2*la**5*mu + 56250*c3**4*c4**2*la**5*mu - 401850*c2**3*c4**3*la**5*mu
        ↪  - 75750*c2**2*c3*c4**3*la**5*mu + 340125*c2*c3**2*c4**3*la**5*mu + 56250*c3**3*c4**3*la
        ↪ **5*mu + 2700*c2**5*c3*la**4*mu**2 + 426735*c2**4*c3**2*la**4*mu**2 + 696060*c2**3*c3**3*
        ↪ la**4*mu**2 + 126900*c2**2*c3**4*la**4*mu**2 - 35775*c2**5*c4*la**4*mu**2 + 246645*c2**4*
        ↪ c3*c4*la**4*mu**2 + 1715040*c2**3*c3**2*c4*la**4*mu**2 + 958365*c2**2*c3**3*c4*la**4*mu**2
        ↪  + 58725*c2*c3**4*c4*la**4*mu**2 - 114885*c2**4*c4**2*la**4*mu**2 + 511785*c2**3*c3*c4**2*
        ↪ la**4*mu**2 + 935100*c2**2*c3**2*c4**2*la**4*mu**2 + 167400*c2*c3**3*c4**2*la**4*mu**2 -
        ↪ 38205*c2**3*c4**3*la**4*mu**2 + 112455*c2**2*c3*c4**3*la**4*mu**2 + 59625*c2*c3**2*c4**3*
        ↪ la**4*mu**2 + 7425*c2**5*c3*la**3*mu**3 + 74790*c2**4*c3**2*la**3*mu**3 + 72495*c2**3*c3
        ↪ **3*la**3*mu**3 + 6750*c2**2*c3**4*la**3*mu**3 + 675*c2**5*c4*la**3*mu**3 + 76410*c2**4*c3
        ↪ *c4*la**3*mu**3 + 161487*c2**3*c3**2*c4*la**3*mu**3 + 37260*c2**2*c3**3*c4*la**3*mu**3 +
        ↪ 810*c2**4*c4**2*la**3*mu**3 + 76761*c2**3*c3*c4**2*la**3*mu**3 + 36720*c2**2*c3**2*c4**2*
        ↪ la**3*mu**3 - 135*c2**3*c4**3*la**3*mu**3 + 7020*c2**2*c3*c4**3*la**3*mu**3 - 810*c2**4*c3
        ↪ *c4*la**2*mu**4 - 4293*c2**3*c3**2*c4*la**2*mu**4 - 324*c2**3*c3*c4**2*la**2*mu**4 +
        ↪ 78300*c2**5*la**6 + 1062045*c2**4*c3*la**6 + 1714950*c2**3*c3**2*la**6 + 420750*c2**2*c3
```

```
    ↪ **3*la**6 + 1003995*c2**4*c4*la**6 + 4834935*c2**3*c3*c4*la**6 + 3606750*c2**2*c3**2*c4*la
    ↪ **6 + 425250*c2*c3**3*c4*la**6 + 1629900*c2**3*c4**2*la**6 + 3532500*c2**2*c3*c4**2*la**6
    ↪ + 1240875*c2*c3**2*c4**2*la**6 + 56250*c3**3*c4**2*la**6 + 407250*c2**2*c4**3*la**6 +
    ↪ 413250*c2*c3*c4**3*la**6 + 56250*c3**2*c4**3*la**6 + 47925*c2**5*la**5*mu + 385965*c2**4*
    ↪ c3*la**5*mu - 193185*c2**3*c3**2*la**5*mu - 587925*c2**2*c3**3*la**5*mu - 84375*c2*c3**4*
    ↪ la**5*mu + 517185*c2**4*c4*la**5*mu + 1038015*c2**3*c3*c4*la**5*mu - 1061955*c2**2*c3**2*
    ↪ c4*la**5*mu - 677025*c2*c3**3*c4*la**5*mu - 33750*c3**4*c4*la**5*mu + 673245*c2**3*c4**2*
    ↪ la**5*mu + 247995*c2**2*c3*c4**2*la**5*mu - 572175*c2*c3**2*c4**2*la**5*mu - 101250*c3**3*
    ↪ c4**2*la**5*mu + 118125*c2**2*c4**3*la**5*mu - 29025*c2*c3*c4**3*la**5*mu - 33750*c3**2*c4
    ↪ **3*la**5*mu + 7425*c2**5*la**4*mu**2 - 22680*c2**4*c3*la**4*mu**2 - 333693*c2**3*c3**2*la
    ↪ **4*mu**2 - 213570*c2**2*c3**3*la**4*mu**2 - 13500*c2*c3**4*la**4*mu**2 + 62640*c2**4*c4*
    ↪ la**4*mu**2 - 230121*c2**3*c3*c4*la**4*mu**2 - 560142*c2**2*c3**2*c4*la**4*mu**2 - 109755*
    ↪ c2*c3**3*c4*la**4*mu**2 + 61911*c2**3*c4**2*la**4*mu**2 - 177336*c2**2*c3*c4**2*la**4*mu
    ↪ **2 - 109620*c2*c3**2*c4**2*la**4*mu**2 + 6210*c2**2*c4**3*la**4*mu**2 - 13635*c2*c3*c4
    ↪ **3*la**4*mu**2 - 12960*c2**4*c3*la**3*mu**3 - 40743*c2**3*c3**2*la**3*mu**3 - 12150*c2
    ↪ **2*c3**3*la**3*mu**3 - 810*c2**4*c4*la**3*mu**3 - 40338*c2**3*c3*c4*la**3*mu**3 - 27864*
    ↪ c2**2*c3**2*c4*la**3*mu**3 - 324*c2**3*c4**2*la**3*mu**3 - 11502*c2**2*c3*c4**2*la**3*mu
    ↪ **3 + 243*c2**3*c3*c4*la**2*mu**4 - 181980*c2**4*la**6 - 931770*c2**3*c3*la**6 - 714825*c2
    ↪ **2*c3**2*la**6 - 84375*c2*c3**3*la**6 - 902610*c2**3*c4*la**6 - 2070540*c2**2*c3*c4*la**6
    ↪  - 743850*c2*c3**2*c4*la**6 - 33750*c3**3*c4*la**6 - 696825*c2**2*c4**2*la**6 - 736875*c2*
    ↪ c3*c4**2*la**6 - 101250*c3**2*c4**2*la**6 - 83250*c2*c4**3*la**6 - 33750*c3*c4**3*la**6 -
    ↪ 97470*c2**4*la**5*mu - 238707*c2**3*c3*la**5*mu + 183330*c2**2*c3**2*la**5*mu + 134325*c2*
    ↪ c3**3*la**5*mu + 6750*c3**4*la**5*mu - 382293*c2**3*c4*la**5*mu - 206658*c2**2*c3*c4*la
    ↪ **5*mu + 324135*c2*c3**2*c4*la**5*mu + 60750*c3**3*c4*la**5*mu - 207900*c2**2*c4**2*la**5*
    ↪ mu + 39285*c2*c3*c4**2*la**5*mu + 60750*c3**2*c4**2*la**5*mu - 12825*c2*c4**3*la**5*mu +
    ↪ 6750*c3*c4**3*la**5*mu - 12960*c2**4*la**4*mu**2 + 32886*c2**3*c3*la**4*mu**2 + 114858*c2
    ↪ **2*c3**2*la**4*mu**2 + 24300*c2*c3**3*la**4*mu**2 - 36045*c2**3*c4*la**4*mu**2 + 94365*c2
    ↪ **2*c3*c4*la**4*mu**2 + 68607*c2*c3**2*c4*la**4*mu**2 - 11016*c2**2*c4**2*la**4*mu**2 +
    ↪ 23976*c2*c3*c4**2*la**4*mu**2 + 7533*c2**3*c3*la**3*mu**3 + 7290*c2**2*c3**2*la**3*mu**3 +
    ↪  243*c2**3*c4*la**3*mu**3 + 6804*c2**2*c3*c4*la**3*mu**3 + 167481*c2**3*la**6 + 403650*c2
    ↪ **2*c3*la**6 + 147825*c2*c3**2*la**6 + 6750*c3**3*la**6 + 397710*c2**2*c4*la**6 + 437130*
    ↪ c2*c3*c4*la**6 + 60750*c3**2*c4*la**6 + 146475*c2*c4**2*la**6 + 60750*c3*c4**2*la**6 +
    ↪ 6750*c4**3*la**6 + 73629*c2**3*la**5*mu + 51273*c2**2*c3*la**5*mu - 61965*c2*c3**2*la**5*
    ↪ mu - 12150*c3**3*la**5*mu + 122715*c2**2*c4*la**5*mu - 16767*c2*c3*c4*la**5*mu - 36450*c3
    ↪ **2*c4*la**5*mu + 23490*c2*c4**2*la**5*mu - 12150*c3*c4**2*la**5*mu + 7533*c2**3*la**4*mu
    ↪ **2 - 17010*c2**2*c3*la**4*mu**2 - 14580*c2*c3**2*la**4*mu**2 + 6804*c2**2*c4*la**4*mu**2
    ↪ - 14337*c2*c3*c4*la**4*mu**2 - 1458*c2**2*c3*la**3*mu**3 - 75735*c2**2*la**6 - 86265*c2*c3
    ↪ *la**6 - 12150*c3**2*la**6 - 85860*c2*c4*la**6 - 36450*c3*c4*la**6 - 12150*c4**2*la**6 -
    ↪ 24300*c2**2*la**5*mu + 2187*c2*c3*la**5*mu + 7290*c3**2*la**5*mu - 14337*c2*c4*la**5*mu +
    ↪ 7290*c3*c4*la**5*mu - 1458*c2**2*la**4*mu**2 + 2916*c2*c3*la**4*mu**2 + 16767*c2*la**6 +
    ↪ 7290*c3*la**6 + 7290*c4*la**6 + 2916*c2*la**5*mu - 1458*c3*la**5*mu - 1458*la**6)/((800*c2
    ↪ **2*c3**2*c4*la**2 + 200*c2**2*c3**2*c4*la*mu - 10*c2**2*c3**2*c4*mu**2 - 450*c2**2*c3**2*
    ↪ la**2 - 900*c2**2*c3*c4*la**2 - 650*c2*c3**2*c4*la**2 - 135*c2**2*c3**2*la*mu - 270*c2**2*
    ↪ c3*c4*la*mu - 115*c2*c3**2*c4*la*mu + 480*c2**2*c3*la**2 + 375*c2*c3**2*la**2 + 240*c2**2*
    ↪ c4*la**2 + 750*c2*c3*c4*la**2 + 125*c3**2*c4*la**2 + 150*c2**2*c3*la*mu + 75*c2*c3**2*la*
    ↪ mu + 75*c2**2*c4*la*mu + 150*c2*c3*c4*la*mu - 135*c2**2*la**2 - 420*c2*c3*la**2 - 75*c3
    ↪ **2*la**2 - 210*c2*c4*la**2 - 150*c3*c4*la**2 - 45*c2**2*la*mu - 90*c2*c3*la*mu - 45*c2*c4
    ↪ *la*mu + 117*c2*la**2 + 90*c3*la**2 + 45*c4*la**2 + 27*c2*la*mu - 27*la**2)*(50*c2*c3*c4*
    ↪ la - 5*c2*c3*c4*mu - 30*c2*c3*la - 30*c2*c4*la - 25*c3*c4*la + 15*c2*la + 15*c3*la + 15*c4
    ↪ *la - 9*la)*(4*c2*la + c2*mu - la)*(c3*mu + la)*(c2 - c3)*(c3 - c4)*la)
G3=1/6*(3*c2 - 2)*la - 5/12*(2*c2 - 1)*la - 1/12*(5*c2 - 3)*mu + 1/12*(4*c2 - 3)*mu
b6=1/4*(4*c2*la + c2*mu - la)/(7*c2*la + 2*c2*mu - 2*la)
p6=1/2
c5=3/5*(60*b6*c2*c3*c4 - 60*b6*c2*c3 - 60*b6*c2*c4 - 60*b6*c3*c4 - 30*c2*c3*c4 + 60*b6*c2 + 60*b6
    ↪ *c3 + 20*c2*c3 + 60*b6*c4 + 20*c2*c4 + 20*c3*c4 - 60*b6 - 15*c2 - 15*c3 - 15*c4 + 12)*la
    ↪ /(240*b6*c2*c3*c4*la*p6 + 120*b6*c2*c3*c4*mu*p6 - 36*b6*c2*c3*c4*mu - 120*b6*c3*c4*la*p6 -
    ↪  72*b6*c3*c4*mu*p6 - 36*b6*c2*c3*la - 36*b6*c2*c4*la - 40*c2*c3*c4*la + 36*b6*c3*c4*mu -
    ↪ 2*c2*c3*c4*mu + 36*b6*c2*la + 36*b6*c3*la + 18*c2*c3*la + 36*b6*c4*la + 18*c2*c4*la + 20*
    ↪ c3*c4*la - 36*b6*la - 12*c2*la - 12*c3*la - 12*c4*la + 9*la)
b5=-1/60*(60*b6*c2*c3*c4 - 60*b6*c2*c3 - 60*b6*c2*c4 - 60*b6*c3*c4 - 30*c2*c3*c4 + 60*b6*c2 + 60*
    ↪ b6*c3 + 20*c2*c3 + 60*b6*c4 + 20*c2*c4 + 20*c3*c4 - 60*b6 - 15*c2 - 15*c3 - 15*c4 + 12)/((
    ↪ c2 - c5)*(c3 - c5)*(c4 - c5)*c5)
b4=1/60*(60*b6*c2*c3*c5 - 60*b6*c2*c3 - 60*b6*c2*c5 - 60*b6*c3*c5 - 30*c2*c3*c5 + 60*b6*c2 + 60*
    ↪ b6*c3 + 20*c2*c3 + 60*b6*c5 + 20*c2*c5 + 20*c3*c5 - 60*b6 - 15*c2 - 15*c3 - 15*c5 + 12)/((
    ↪ c2 - c4)*(c3 - c4)*(c4 - c5)*c4)
b3=-1/60*(60*b6*c2*c4*c5 - 60*b6*c2*c4 - 60*b6*c2*c5 - 60*b6*c4*c5 - 30*c2*c4*c5 + 60*b6*c2 + 60*
    ↪ b6*c4 + 20*c2*c4 + 60*b6*c5 + 20*c2*c5 + 20*c4*c5 - 60*b6 - 15*c2 - 15*c4 - 15*c5 + 12)/((
    ↪ c2 - c3)*(c3 - c4)*(c3 - c5)*c3)
b2=-1/2*(2*b3*c3 + 2*b4*c4 + 2*b5*c5 + 2*b6 - 1)/c2
b1=-1/2*(2*(b3 + b4 + b5 + b6 - 1)*c2 - 2*b3*c3 - 2*b4*c4 - 2*b5*c5 - 2*b6 + 1)/c2
p3=3/2*(c3*mu + la)*(c2 - c3)/(10*c2*la + 5*c2*mu - 5*la - 3*mu)
p5=3/2*(c5*mu + la)*(c2 - c5)/(10*c2*la + 5*c2*mu - 5*la - 3*mu)
p4=3/2*(c4*mu + la)*(c2 - c4)/(10*c2*la + 5*c2*mu - 5*la - 3*mu)
s4=-1/360*(20*c2*c3*c4*la + 80*c2*c3*c5*la - 5*c2*c3*c4*mu - 5*c2*c3*c5*mu - 60*c2*c3*la - 15*c2*
    ↪ c4*la - 10*c3*c4*la - 45*c2*c5*la - 40*c3*c5*la + 30*c2*la + 30*c3*la + 6*c4*la + 24*c5*la
    ↪  - 18*la)/((c2 - c4)*(c3 - c4)*(c4 - c5)*la)
r4=-1/180*(50*c2*c3*c5*la - 5*c2*c3*c5*mu - 30*c2*c3*la - 30*c2*c5*la - 25*c3*c5*la + 15*c2*la +
    ↪ 15*c3*la + 15*c5*la - 9*la)/((c2 - c4)*(c3 - c4)*(c4 - c5)*la)
rs4=1/360*(20*c2*c3*la - 5*c2*c3*mu - 15*c2*la - 10*c3*la + 6*la)/((c2*c3 - c2*c4 - c3*c4 + c4
    ↪ **2)*la)
    s3=1/360*(20*c2*c3*c4*la + 80*c2*c4*c5*la - 5*c2*c3*c4*mu - 5*c2*c4*c5*mu - 15*c2*c3*la - 60*c2*
        ↪ c4*la - 10*c3*c4*la - 45*c2*c5*la - 40*c4*c5*la + 30*c2*la + 6*c3*la + 30*c4*la + 24*c5*la
        ↪  - 18*la)/((c2 - c3)*(c3 - c4)*(c3 - c5)*la)
    r3=1/180*(50*c2*c4*c5*la - 5*c2*c4*c5*mu - 30*c2*c4*la - 30*c2*c5*la - 25*c4*c5*la + 15*c2*la +
        ↪ 15*c4*la + 15*c5*la - 9*la)/((c2 - c3)*(c3 - c4)*(c3 - c5)*la)
    r5=1/180*(50*c2*c3*c4*la - 5*c2*c3*c4*mu - 30*c2*c3*la - 30*c2*c4*la - 25*c3*c4*la + 15*c2*la +
        ↪ 15*c3*la + 15*c4*la - 9*la)/((c2 - c5)*(c3 - c5)*(c4 - c5)*la)
    rs3=-1/360*(20*c2*c4*la - 5*c2*c4*mu - 15*c2*la - 10*c4*la + 6*la)/((c2 - c3)*(c3 - c4)*la)
    a65=r5/b6
    a32=p3/c2
    a54=-rs4/(b5*(c5 - 1))
    a21=c2
    a43=(a63*b6*c5 - a63*b6 - c5*r3 + s3)/(b4*c4 - b4*c5)
    a52=(a63*b6*c3*c4 - a54*b5*c4**2 + a54*b5*c4*c5 - a63*b6*c3 - c3*c4*r3 + (b5*c4 - b5*c5)*p5 + c3*
        ↪ s3)/(b5*c2*c4 - b5*c2*c5)
    a64=-(a54*b5 - r4)/b6
    a62=-(a63*b6*c3 - a54*b5*c4 + a65*b6*c5 - b6*p6 + c4*r4)/(b6*c2)
    a42=-(a63*b6*c3*c5 - a63*b6*c3 - c3*c5*r3 - (b4*c4 - b4*c5)*p4 + c3*s3)/(b4*c2*c4 - b4*c2*c5)
    a53=-(a63*b6*c4 - a63*b6 - c4*r3 + s3)/(b5*c4 - b5*c5)
    a51=-a52 - a53 - a54 + c5
    a41=-a42 - a43 + c4
    a61=-a62 - a63 - a64 - a65 + 1
    a31=-a32 + c3
    a74=b4
    a72=b2
    a76=b6
    a73=b3
    a71=b1
    a75=b5
    bhat4=-1/36*(160*c2**2*c3*c5*la**2 + 40*c2**2*c3*c5*la*mu - 2*c2**2*c3*c5*mu**2 - 90*c2**2*c3*la
        ↪ **2 - 90*c2**2*c5*la**2 - 130*c2*c3*c5*la**2 - 27*c2**2*c3*la*mu - 27*c2**2*c5*la*mu - 23*
        ↪ c2*c3*c5*la*mu + 48*c2**2*la**2 + 75*c2*c3*la**2 + 75*c2*c5*la**2 + 25*c3*c5*la**2 + 15*c2
        ↪ **2*la*mu + 15*c2*c3*la*mu + 15*c2*c5*la*mu - 42*c2*la**2 - 15*c3*la**2 - 15*c5*la**2 - 9*
        ↪ c2*la*mu + 9*la**2)/((7*c2*la + 2*c2*mu - 2*la)*(c2 - c4)*(c3 - c4)*(c4 - c5)*la)
    bhat3=1/36*(160*c2**2*c4*c5*la**2 + 40*c2**2*c4*c5*la*mu - 2*c2**2*c4*c5*mu**2 - 90*c2**2*c4*la
        ↪ **2 - 90*c2**2*c5*la**2 - 130*c2*c4*c5*la**2 - 27*c2**2*c4*la*mu - 27*c2**2*c5*la*mu - 23*
        ↪ c2*c4*c5*la*mu + 48*c2**2*la**2 + 75*c2*c4*la**2 + 75*c2*c5*la**2 + 25*c4*c5*la**2 + 15*c2
        ↪ **2*la*mu + 15*c2*c4*la*mu + 15*c2*c5*la*mu - 42*c2*la**2 - 15*c4*la**2 - 15*c5*la**2 - 9*
        ↪ c2*la*mu + 9*la**2)/((7*c2*la + 2*c2*mu - 2*la)*(c2 - c3)*(c3 - c4)*(c3 - c5)*la)
    bhat6=-1/4*(28*bhat7*c2*la + 8*bhat7*c2*mu - 8*bhat7*la - 4*c2*la - c2*mu + la)/(7*c2*la + 2*c2*
        ↪ mu - 2*la)
    bhat5=1/36*(160*c2**2*c3*c4*la**2 + 40*c2**2*c3*c4*la*mu - 2*c2**2*c3*c4*mu**2 - 90*c2**2*c3*la
        ↪ **2 - 90*c2**2*c4*la**2 - 130*c2*c3*c4*la**2 - 27*c2**2*c3*la*mu - 27*c2**2*c4*la*mu - 23*
        ↪ c2*c3*c4*la*mu + 48*c2**2*la**2 + 75*c2*c3*la**2 + 75*c2*c4*la**2 + 25*c3*c4*la**2 + 15*c2
        ↪ **2*la*mu + 15*c2*c3*la*mu + 15*c2*c4*la*mu - 42*c2*la**2 - 15*c3*la**2 - 15*c4*la**2 - 9*
        ↪ c2*la*mu + 9*la**2)/((7*c2*la + 2*c2*mu - 2*la)*(c2 - c5)*(c3 - c5)*(c4 - c5)*la)
    bhat2=-1/2*(2*bhat3*c3 + 2*bhat4*c4 + 2*bhat5*c5 + 2*bhat6 + 2*bhat7 - 1)/c2
    bhat1=-1/2*(2*(bhat3 + bhat4 + bhat5 + bhat6 + bhat7 - 1)*c2 - 2*bhat3*c3 - 2*bhat4*c4 - 2*bhat5*
        ↪ c5 - 2*bhat6 - 2*bhat7 + 1)/c2
    return bhat7,c2,c3,la,mu,b7,c7,c6,nu,om,c4,a63,G3,b6,p6,c5,b5,b4,b3,b2,b1,p3,p5,p4,s4,r4,rs4,s3,
        ↪ r3,r5,rs3,a65,a32,a54,a21,a43,a52,a64,a62,a42,a53,a51,a41,a61,a31,a74,a72,a76,a73,a71,a75,
        ↪ bhat4,bhat3,bhat6,bhat5,bhat2,bhat1

def erk5gIIIs6FSAL7_tableau(x,cached_coefficients=None):
    if cached_coefficients:
        bhat7,c2,c3,la,mu,b7,c7,c6,nu,om,c4,a63,G3,b6,p6,c5,b5,b4,b3,b2,b1,p3,p5,p4,s4,r4,rs4,s3,r3,
            ↪ r5,rs3,a65,a32,a54,a21,a43,a52,a64,a62,a42,a53,a51,a41,a61,a31,a74,a72,a76,a73,a71,a75
            ↪ ,bhat4,bhat3,bhat6,bhat5,bhat2,bhat1 = cached_coefficients
    else:
        bhat7,c2,c3,la,mu,b7,c7,c6,nu,om,c4,a63,G3,b6,p6,c5,b5,b4,b3,b2,b1,p3,p5,p4,s4,r4,rs4,s3,r3,
            ↪ r5,rs3,a65,a32,a54,a21,a43,a52,a64,a62,a42,a53,a51,a41,a61,a31,a74,a72,a76,a73,a71,a75
            ↪ ,bhat4,bhat3,bhat6,bhat5,bhat2,bhat1 = erk5gIIIs6FSAL7_coefficients(x)
    return {'A':Matrix([[0,0,0,0,0,0,0],
                        [a21,0,0,0,0,0,0],
                        [a31,a32,0,0,0,0,0],
                        [a41,a42,a43,0,0,0,0],
                        [a51,a52,a53,a54,0,0,0],
                        [a61,a62,a63,a64,a65,0,0],
                        [a71,a72,a73,a74,a75,a76,0]]),
            'b':vector([b1,b2,b3,b4,b5,b6,0]),
            'c':vector([0,c2,c3,c4,c5,1,1]),
            'bhat':vector([bhat1,bhat2,bhat3,bhat4,bhat5,bhat6,bhat7])}
```

Where the other functions analogous to those given in Listing 4.4 are also provided by the generated code (not shown due to space limitations). The size of the code in Listing 4.6 compared to Listing 4.4 gives a good indication of the additional complexity that solving six-stage fifth-order order conditions without standard simplifying assumptions entails. The total size of generated code for all methods examined in this study is

225MB, indicating the difficulty that would exist in conducting a similar study to the one in this thesis by hand coding rather than with generative code.

4.7.3 Generating code for existing methods without solving order conditions

Some published methods are only used in this thesis for comparative purposes, such as pairs by Tsitouras [182, 183], the CK4(5)$_{6(6)}$ pair (2.83) that is part of a family with a singularity at $b_5 = 0$, or ERK methods higher than fifth order. For these methods, it is not necessary or desirable to work out the symbolic expressions for the Butcher tableau coefficients in terms of the free parameters. However, it is still desirable to incorporate these methods into the framework of `OCSage`. In these cases, where the published coefficients are already known, it is possible to have the `step«number»` methods (and the analogous `step«number»_emb` methods) to simply return the already-known (published) values of coefficients in the dictionary. The code is generated that provides all the same functions as seen in Listings 4.4 and 4.6, except the `x` argument is ignored.

4.8 Unit testing `OCSage`

Experience has shown that it can be extremely difficult to verify that code implementing sophisticated mathematics, especially code implementing numerical methods, is working to specification because subtle errors can give plausible results that can only be detected by careful study and/or hand verification. Additionally, the large number of ERK families examined in this thesis would be time consuming to manually test after each change in the code. Because the interface for using `OCSage` to construct RK methods involves coding, it is necessary to test and not just verify manually that the code for a particular RK family is implemented correctly. Therefore, because `OCSage` has been under continuous development throughout this study and will likely remain so if it is used for more studies of RK methods in the future, having unit tests can quickly give a high level of confidence that any new changes in the code have not broken any existing functionality or that an installation on a different platform is functioning correctly. The unit tests also fail if PYTHON exceptions occur, ensuring that all code they cover at least runs without producing runtime errors. As mentioned in Section 4.6, the SAGE source code files derived from the SAGE worksheets are used as unit tests, and they are just one of several sets of unit tests conducted.

The first set of unit tests, in the `OCSage` source file `unittests_worksheet/unittests_worksheet.py` that are already mentioned in Section 4.6, verify the `sagenb_«...»` and `_«..»` functions in `erk5_6s_functions.sage` that are discussed in Section 4.5. By solving each of the cases described in Sections 3.3 and 3.4.1 with several sets of rational values for the free parameters, then ensuring the order conditions are satisfied by the resulting Butcher tableau (2.34), this verifies that RK methods of the desired order are in fact constructed. This set of unit tests takes just under 10 minutes using one processor core on the desktop computer used for this study.

The second set of unit tests are in the `OCSage` source file `unittests_generated_scipy.py`, which tests the generated code (described in Section 4.7) in the `OCSage` directory `generated_scipy/`. This is done by substituting in two sets (although it could be extended to more) of numerical (floating-point) values as the free parameters of the solution for each ERK family, in order to find two sets of Butcher tableau coefficients, and then verifying that these Butcher tableau coefficients satisfy the order conditions. Because floating-point values are used, the residual after substituting the calculated Butcher tableau coefficients back into the order conditions will generally not be zero, but it should be extremely small. Generally, when order conditions are expected to be satisfied, ensuring the residual is $< 10^{-12}$ is adequate in double precision. However, cases with quadratic irrationality, such as Cases I, II, IV derived in Section 3.4.1, can lead to large and complicated expressions for some Butcher tableau coefficients in terms of the free parameters that introduce significant roundoff error due to the large number of terms. In general, for order conditions that are not expected to be satisfied, e.g., the sixth-order conditions for a fifth-order method, the residuals are typically around 10^{-3}–10^{-7}. This test runs extremely fast, in 1.2 seconds, using one processor core on the desktop computer used for this study because code, such as that in Listings 4.4 and 4.6, runs extremely fast compared to finding the algebraic solution anew as does the previous unit test does. Due to this short execution time, this set of unit tests caught many bugs and other issues during the development of `OCSage`. Additionally, the nearly identical generated code in the `OCSage` directory `generated_sage/` can also be verified, specifically by using the Sage arbitrary precision data type with 512 bits of precision. Using 512 bits of precision, the residuals from satisfying the order conditions are typically about 10^{-150}, giving a high-degree of confidence in the accuracy of the generated code. The drawback is that these units tests using this higher-precision arithmetic take about 11 minutes; this is still short enough to be run regularly but it is not as convenient as the tests run on the generated code in `generated_scipy/`. When these tests are run frequently during the normal write-test-debug cycle, this test suite instills a high degree of confidence that the `OCSage` package is continuing to solve the order conditions correctly.

The third set of tests in the source file `unittests_regenerate.py` incorporates the second set of unit tests just described, but reruns the solution of the order conditions and generation of code described in Section 4.7, i.e., the `OCSage` `erk_classic.sage`, `erk_fixed.sage`, and `erk_cmf.sage` programs. This set of tests takes much longer, about 65 minutes total using up to four processor cores for the code generation on the desktop computer used for this study, but it ensures the code generation and generated code continue to work error free.

All of the above unit tests can be run together using the source file `unittests.py`, which takes a total of 75 minutes on the desktop computer used for this study. This is a reasonable time for the developer to have confidence in full functionality by running everything overnight, during a break, or when otherwise occupied. In fact, the unit test script is often used to regenerate the generated code described in Section 4.7 because it runs all appropriate scripts and automatically catches many programming errors that may have been inadvertently introduced.

4.9 The `ERKSearch` class for systematically searching coefficients

The `erk_search.py` source file is used to search the free parameters of the ERK families described in this thesis. This includes support for parallelism and visualization. Although data showing detailed results of searching through free parameters of RK method families does not appear to have been published before, several authors have mentioned doing searches or using optimization to select RK method coefficients. Dormand and Prince [47, 136] appear to have used unspecified numerical optimization methods to find promising regions in the space of free parameters for the family containing the DP$5(4)_{6(7)}$ pair (2.79) and they specifically mention using numerical optimization for their ERK embedded pair [136] with seventh- and eighth-order methods that was constructed from an ERK family with 10 free parameters. Sharp describes an "interactive" grid search as the methodology he used to find a minimal leading error coefficient for ERK methods higher than fifth-order [156]. Tsitouras has extensively used numerical optimization methods to assist the studies already mentioned on six-stage fifth-order methods [182, 183] and some of his other publications [180]. However, like other authors, Tistouras presents few details or data showing how the searches were conducted or why the specific published methods were chosen. It is likely that many authors studying RK methods have used some kind of optimization technique to narrow down the candidate sets of free parameters and present the best set of coefficients for their particular study, without specifically mentioning that this was the case. Due to the lack of specifics in existing publications, the study in this thesis is the first to present and analyze the results of searching the space of free parameters for RK methods in an explicit, detailed, and thorough manner.

In order to fully utilize the generated tableaux, error coefficients, and characteristic numbers for constructing better RK methods, it is necessary to examine the characteristic numbers (2.56) of candidate sets of RK method coefficients. Brute-force search is possible for $5(4)_6$ ERK pairs using the generated code from `OCSage` because valid and practical values for RK free parameters do not extend over the entire real number line. RK method construction is typically done so that most free parameters are the components of the $\mathbf{c}$ vector, where only values in the interval $[0, 1]$ are traditionally used for ERK methods (values of the $\mathbf{c}$ components less than 0 and greater than 1 are excluded for stability reasons). Free parameters other than components of the $\mathbf{c}$ vector can also be reasonably constrained. For instance, it is commonly accepted [16, 47, 190] that free parameters involving the $\mathbf{A}$ matrix of the Butcher tableau can be constrained to a reasonable interval, such as $[-20, 20]$, meaning the whole real number line does not have to be examined in the case of free parameters from the $\mathbf{A}$ matrix. Ratios of the multipliers $\lambda, \mu, \nu, \omega$ defined in Chapter 3 and given by Table 3.4 are also a free parameter that could in principle extend the whole real number line, but the experience of conducting searches in the course of this study shows that the space of free parameters is smooth enough that one multiplier can be fixed at 1.0 and the other varied in the interval $[-1.0, 1.0]$, and then vice versa.

It is widely known that the most practical and efficient numerical formulae are generally selected to have reasonably small leading error coefficients in comparison to the family they are constructed from [16, 47, 146, 183]. However, given that Dormand and Prince demonstrated in 1986 [48] that ERK pairs with smaller leading-error coefficients do not necessarily outperform methods with somewhat larger (but still reasonably small) leading-error coefficients on some test sets of IVPs, it is more appropriate to search for six-stage fifth-order ERK methods with a leading error coefficient below a chosen upper bound rather than simply minimizing the leading error coefficient. In fact, the families of embedded pairs containing the DP5$(4)_{6(7)}$ pair (2.79) can have arbitrarily small (but not zero) leading error coefficients, i.e., where $A^6 \to 0$ and $D \to \infty$ as $c_4 \to 1$ and $c_5 \to 1$ at a particular value of c_3, a phenomenon noted in ERK methods by Fehlberg as far back as 1966 [60]. However, in Chapter 5 it is demonstrated that aggressive minimization of the leading error coefficient, even with values of the Butcher tableau coefficients that are of "reasonable" magnitude, can sometimes lead to ERK pairs that perform unexpectedly poorly solving some IVPs. For $5(4)_6$ ERK pairs, rough guidelines for small-enough leading error coefficients can be found by taking the ERK embedded pair derived by Dormand and Prince with an extended stability region [47] as an ERK method with a "good" leading error coefficient of 0.00181 and the widely used DP5$(4)_{6(7)}$ pair (2.79) as an ERK method with an "excellent" leading error coefficient of 0.000399.

The scalar sums (2.39), order conditions, and expressions for particular Butcher tableau coefficients are typically rational functions (except for Cases I, II, IV from Section 3.4.1 in the case of six-stage fifth-order ERK families that must contain square roots except for certain reduced systems), which can actually be amenable to study by analytic techniques. Verner analyzes the error estimate for the quadrature problem (2.2) only, i.e., using the bushy trees only [187]. However, the performance data presented in Chapter 5 clearly indicates current analytic techniques may not be sufficient to choose the most robust and efficient ERK pairs. This is because, as shown in Chapter 5, the individual values of the more difficult to analyze PECs are often the most important, and that tradeoffs between properties, rather than extreme values (for example, not necessarily minimizing leading error coefficients to the smallest magnitude possible), lead to the best performing ERK pairs for practical IVPs. For example, consider that the minimum possible leading error coefficient for the much simpler family of popular four-stage fourth-order ERK methods had to be found using a combination of analytic techniques and numerical optimization [139]. Furthermore, once a family has been explicitly constructed, numerical search or optimization are generally the main techniques for finding the best numerical formulae. Therefore, it is seen in Chapter 5 that performance testing in conjunction with studying the mathematical properties is required find the pairs with the best performance.

A brute-force search of just over 1 million candidate sets of free parameters for the three-parameter family containing the fifth-order component of the DP5$(4)_{6(7)}$ pair (2.79) (with c_2, c_4, c_5 as free parameters, and 101 points in $[0, 1]$ tested for each) gives only 15030 sets of free parameters corresponding to leading error coefficients smaller than 0.0020, and only 202 sets of free parameters corresponding to leading error coefficients smaller than the 0.000399 of the DP5$(4)_{6(7)}$ pair (2.79). From these numbers, it can clearly be seen that using rough but well-established guidelines to restrict leading error coefficients (2.47) can drastically reduce the number of candidate RK formulae to be evaluated. With appropriate insight, other properties can be used, in order to reduce the number of candidate formulae even further. The `erk_search.py` script searches 2-D *slices* of parameter space, i.e., it does a grid search of two free parameters while keeping others constant. These two variables that define the 2-D slice can be chosen in a way that gets the maximum possible benefit from vectorization. In particular, when searching with the `ERKSearch` class, if the leading error coefficient for a particular slice of the space of free parameters is always greater than a certain threshold (for example, greater than 0.0064, for giving the best overall picture of space of free parameters, or greater than 0.0020, for fine-grained searching for efficient methods) the other properties are not calculated for that slice, allowing for a huge savings in computational cost and database storage for some ERK families studied. It is also seen in the figures in the following sections that the space of free parameters tends to be smooth. This makes using brute-force search a reasonable methodology to give a relatively thorough and systematic search of the sets of free parameters. For $5(4)_6$ ERK pairs, there are also free parameters $\hat{b}_6$ or $\hat{b}_7$ for the fourth-order method, depending on the family of ERK pairs, that can also be reasonably examined by checking a range of values for regions where the specific fifth-order methods show promise.

The free parameters for each case of the complete solution for six-stage fifth-order ERK methods are given in Table 3.3, where it can be seen that there are never more than five free parameters. The free parameters for the six-stage fifth-order component for each family of $5(4)_6$ ERK pairs are given in Table 3.4, where it can be seen that there are never more than four free parameters. Even with restrictions on search space due to insight into reasonable method properties, searches of four- and five-dimensional space of free parameters are probably close to the limit of what brute-force search can effectively do. However, some modifications based on the insights gained by the brute-force searches done for this study may allow more sophisticated searching to effectively find better RK methods from families with more than five free parameters. In addition to the relatively low-dimensional space of free parameters, brute-force search as the sole method is really only possible because the smoothness of the space of free parameters allows refinement from broad and coarse-grained searches to narrow and fine-grained searches with a high degree of confidence that nothing has been missed. Examples of the relative smoothness of the space of free parameters can be seen in figures in Sections 4.12–4.15. Therefore, for this study, only brute-force search needs to be used because this provides the most insight into the process of choosing families and selecting coefficients, and it is unlikely to have missed anything substantial. Despite clever restrictions to avoid needing to go over the whole search space, the largest searches done for this study sometimes require `PostgreSQL` tables that take up to 300GB, although more sophisticated search strategies would drastically reduce this. For future studies into RK methods with a greater number of free parameters, it may not be tractable to search as thoroughly as done for this study. However, the data and resulting insights gained by using brute-force search in this study can guide other numerical optimization techniques that may be able to find better RK methods for families where brute-force search is not suitable.

The internals of the `ERKSearch` class are not given in detail because the usage consists only of setting attributes in the `__init__` method when the `ERKSearch` class is subclassed for a particular ERK family. The attribute `search_functions` that demonstrates the choices for sets of functions for calculating characteristic numbers is given in Listing 4.7.

Listing 4.7: The `search_functions` attribute of the `ERKSearch` class showing the choices of characteristic numbers (2.56) possible for different searches.

```
class ERKSearch()
    <<...>>
    self.search_functions={
      'simple':lambda: [[self.get_error_6_function(), 'error_6', (0.0, 0.0020)]],
      <<...>>
      'simple-wide-embedded':lambda: [
        [self.get_error_6_function(),                        'error_6',                (0.0,
            ↪ 0.0020)],
        [self.construct_error_ratio_function(),              'error_ratio',            (0.0, 10.0)
            ↪ ],
        [self.construct_characteristic_B_function(),         'char_B',                 (0.0, 5.0)],
        [self.construct_characteristic_C_function(),         'char_C',                 (0.0, 5.0)],
        [self.construct_characteristic_E_function(),         'char_E',                 (0.0, 5.0)],
        [self.get_min_c_distance_function(),                 'min_c_distance',         (0.0, 0.2)],
        [self.get_max_coefficient_magnitude_function(),      'max_coeff',              (0.0, 100.0)
            ↪ ],
        [self.get_b_diff_function(),                         'b6_diff',                (0.0, 2.0)],
        [self.construct_error_6_max_condition_function(),    'conditions_6_max',       (0.0, 5.0)],
        [self.construct_error_7_max_condition_function(),    'conditions_7_max',       (0.0, 5.0)],
        [self.construct_error_8_max_condition_function(),    'conditions_8_max',       (0.0, 5.0)],
        [self.construct_error_9_max_condition_function(),    'conditions_9_max',       (0.0, 5.0)],
        [self.construct_error_5_max_condition_emb_function(),   'conditions_5_emb_max',   (0.0, 5.0)],
        [self.construct_error_6_max_condition_emb_function(),   'conditions_6_emb_max',   (0.0, 5.0)],
        [self.construct_error_7_max_condition_emb_function(),   'conditions_7_emb_max',   (0.0, 5.0)],
        [self.construct_error_8_max_condition_emb_function(),   'conditions_8_emb_max',   (0.0, 5.0)],
        [self.construct_error_9_max_condition_emb_function(),   'conditions_9_emb_max',   (0.0, 5.0)
            ↪ ]],
      <<...>>
      'default-embedded':lambda :[
        [self.get_error_6_function(),                        'error_6',                (0.0,
            ↪ 0.0013)],
        [self.get_error_7_function(),                        'error_7',                (0.0,
            ↪ 0.0064)],
        [self.get_error_8_function(),                        'error_8',                (0.0,
            ↪ 0.0064)],
        [self.get_error_9_function(),                        'error_9',                (0.0,
            ↪ 0.0064)],
        [self.construct_error_ratio_function(),              'error_ratio',            (0.0, 32.0
            ↪   )],
        [self.construct_error_ratio_79_function(),           'error_ratio_79',         (1.0, 4.0)],
        [self.get_error_emb_5_function(),                    'error_emb_5',            (0.0, 0.0064
            ↪  )],
        [self.get_error_emb_6_function(),                    'error_emb_6',            (0.0, 0.0064
            ↪  )],
        [self.get_error_emb_7_function(),                    'error_emb_7',            (0.0, 0.0064
            ↪  )],
        [self.get_error_emb_8_function(),                    'error_emb_8',            (0.0, 0.0064
            ↪  )],
        [self.get_error_emb_9_function(),                    'error_emb_9',            (0.0, 0.0064
            ↪  )],
        [self.construct_stability_last_term_function(),      'stab_term',              (0.0,
            ↪ 1./256.)],
        [self.get_stability_length_function(),               'stab_length',            (2.0, 7.0)],
        [self.get_stability_length_emb_function(),           'stab_length_emb',        (2.0, 7.0)],
        [self.construct_stability_length_difference_function(), 'stab_length_difference',(-2.0,2.0)],
        [self.get_max_coefficient_magnitude_function(),      'max_coeff',              (0.0, 50.0)
            ↪ ],
        [self.get_min_c_distance_function(),                 'min_c_distance',         (0.0, 0.5)],
        [self.get_b_diff_function(),                         'b6_diff',                (0.0, 2.0)],
        [self.construct_characteristic_B_function(),         'char_B',                 (0.0, 5.0)],
        [self.construct_characteristic_C_function(),         'char_C',                 (0.0, 5.0)],
        [self.construct_characteristic_E_function(),         'char_E',                 (0.0, 5.0)],
        [self.construct_error_6_max_condition_function(),    'conditions_6_max',       (0.0, 10.0)
            ↪ ],
        [self.construct_error_7_max_condition_function(),    'conditions_7_max',       (0.0, 10.0)
            ↪ ],
        [self.construct_error_8_max_condition_function(),    'conditions_8_max',       (0.0, 10.0)
            ↪ ],
        [self.construct_error_9_max_condition_function(),    'conditions_9_max',       (0.0, 10.0)
            ↪ ],
        [self.construct_error_5_max_condition_emb_function(),   'conditions_5_emb_max',   (0.0, 10.0)
        ↪ ],
    [self.construct_error_6_max_condition_emb_function(),   'conditions_6_emb_max',  (0.0, 10.0)
        ↪ ],
    [self.construct_error_7_max_condition_emb_function(),   'conditions_7_emb_max',  (0.0, 10.0)
        ↪ ],
    [self.construct_error_8_max_condition_emb_function(),   'conditions_8_emb_max',  (0.0, 10.0)
        ↪ ],
    [self.construct_error_9_max_condition_emb_function(),   'conditions_9_emb_max',  (0.0, 10.0)
        ↪ ]]}
```

In Listing 4.7, observe the use of the PYTHON keyword `lambda` to create anonymous functions in the data structures; this reduces loading time and ensures that unnecessary code that might cause errors for some families is never evaluated. Smaller sets of functions, such as those represented by keys such as `'simple'` and `'simple-wide-embedded'`, are much faster than computing all characteristic numbers that might be of interest, which is done by the more numerous and slower set of functions stored under `'default-embedded'`. Because the selection of functions to be computed can be selected by appropriate command-line arguments to `erk_search.py`, this allows quick and preliminary searches to be done before more time-consuming final searches.

An example of defining a subclass of `ERKSearch` for the family containing the DP5(4)$_{6(7)}$ pair (2.79), which shows a set of many different searches conducted in the course of this study, is given in Listing 4.8.

Listing 4.8: An outline of the `erk_search.py` showing some module variables and the `ERKSearch_dopr54s7` subclass of the `ERKSearch` class.

```
<<...>>
PLOT_FUNCTIONS_STANDARD          = ('error_6','error_7','error_ratio','max_coeff','stab_length')
PLOT_FUNCTIONS_FULL              = ('error_6','error_7','error_8','error_9','error_ratio','
    ↪ error_ratio_79','max_coeff','stab_length','conditions_6_ratio','conditions_7_ratio')
PLOT_FUNCTIONS_WIDE_EMBEDDED     = ('error_6','char_B','min_c_distance','max_coeff','b6_diff','
    ↪ conditions_6_ratio','conditions_7_ratio','conditions_5_ratio_emb','conditions_6_max','
    ↪ conditions_7_max','conditions_8_max','conditions_9_max','conditions_5_emb_max','
    ↪ conditions_6_emb_max','conditions_7_emb_max','conditions_8_emb_max','conditions_9_emb_max',)
PLOT_FUNCTIONS_FULL_EMBEDDED     = ('error_6','error_7','error_8','error_9','error_emb_5','
    ↪ error_ratio_79','error_emb_6','error_emb_7','error_ratio','max_coeff','stab_length','
    ↪ stab_length_emb','stab_length_difference','char_B','char_C','char_E','conditions_6_ratio','
    ↪ conditions_5_ratio_emb',)
<<...>>
class ERKSearch_dopr54s7(ERKSearch):
    label='dopr54s7'
    plots={'simple':(('c4','c5'),                              ('error_6',)),
           'default':(('c4','c5'),                             PLOT_FUNCTIONS_STANDARD),
           'full':(('c4','c5'),                                PLOT_FUNCTIONS_FULL),
           'full-limited':(('c4','c5'),                        PLOT_FUNCTIONS_FULL,{'c2':(20,30,1)}),
           'full-embedded-parameter':(('c5','bhat7'),          PLOT_FUNCTIONS_FULL_EMBEDDED),
           'full-embedded-parameter-limited':(('c5','bhat7'),  PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(20,30,1)}),
           'progression':(('c4','c5'),('error_6','c2',13,48,1)),
           # classic dopr m,s,c
           'full-embedded-interesting-m':(('c5','bhat7'),       PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(2,3,1),'c4':(48,49,1)}),
           'full-embedded-interesting-s':(('c5','bhat7'),       PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(5,6,1),'c4':(15,16,1)}),
           'full-embedded-interesting-c':(('c5','bhat7'),       PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(2,3,1),'c4':(3,4,1)}),
           'full-embedded-interesting-spread-1':(('c5','bhat7'),  PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(1,2,1),'c4':(31,32,1)}),
           'full-embedded-interesting-spread-2':(('c5','bhat7'),  PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(1,2,1),'c4':(33,34,1)}),
           # 0.99 dopr
           'full-embedded-interesting-99':(('c5','bhat7'),      PLOT_FUNCTIONS_FULL_EMBEDDED,{'c2'
               ↪ :(4,5,1),'c4':(61,62,1)})}
    def __init__(self):
        ERKSearch.__init__(self)
        self.search_variables = {'simple':{'bhat7':[(0.0, 0.0,  1),   False],
                                           'c2':   [(0.0, 1.0,  51),  False],
                                           'c4':   [(0.0, 1.0,  51),  True],
          'c5':   [(0.0, 1.0,  51),  True]},
# 136  (135+1) hits the published formulae that use demoninators of
    ↪ 3,5,9,etc. used for classic dopr methods
'default':{'bhat7':[(0.0, 0.0,  1),   False],
           'c2':   [(0.0, 1.0,  136), False],
           'c4':   [(0.0, 1.0,  136), True],
           'c5':   [(0.0, 1.0,  136), True]},
'interesting':{'bhat7':[(0.0,  0.0,  1),  False],
               'c2':   [(0.17, 0.23, 25), False],
               'c4':   [(0.5,  1.0,  271),True],
               'c5':   [(0.5,  1.0,  271),True]},
'full-embedded':{'bhat7':[(-0.2, 0.2,  41),    True],
                 'c2':   [( 0.0, 1.0,  21),    False],
                 'c4':   [( 0.0, 1.0,  21),    False],
                 'c5':   [( 0.0, 1.0,  21),    True]},
'complete-embedded':{'bhat7':[(-0.2, 0.2,  41),True],
                     'c2':   [( 0.0, 1.0,  21),False],
                     'c4':   [( 0.0, 1.0,  21),False],
                     'c5':   [( 0.0, 1.0,  21),True]},
# makes sure it captures current embedded stuff, not much choice but
    ↪  200
# TODO: this hits everything used currently, may need higher
    ↪ resolution
'interesting-embedded':{'bhat7':[(-0.2,     0.2,     81), True],
                        'c2':   [( 25./135.,32./135.,8),  False],
                        'c4':   [( 60./135.,1.0,     76), False],
                        'c5':   [( 60./135.,1.0,     76), True]}}
```

The attributes that must be defined in subclasses of `ERKSearch` are:

- The `label` attribute must correspond to the `label` attribute used for the particular subclass of the `ERKSolve` class, i.e., the RK family, that is used for generating the code used in the search. This `label` attribute is used within the `PostgreSQL` database for both keys and table names, as well as for the filenames of MATPLOTLIB plots.

- The `plots` attribute describes, for the standard plotting functionality (to be described below), the free parameters used for the x-axis and y-axis respectively along with a list of characteristic numbers (2.56) to display in the plot (these lists are typically defined as global constants such as those shown at the top of Listing 4.7, and the functions specified must have been part of the search).

  – The `PLOT_FUNCTIONS_STANDARD`, `PLOT_FUNCTIONS_FULL`, and `PLOT_FUNCTIONS_FULL_EMBEDDED` global constants define the list of functions to plot and are used to avoid repetitive code in the `plots` attribute of subclasses of `ERKSearch`.

- The `search_variables` attribute contains a list of dictionaries that represent possible searches (to be selected on the command line), where each key-value pair in each dictionary gives the information required for specific free parameters for that particular search, and the dictionary key is a string that corresponds to the variable name used in the generated code. This information is in a list that gives in order:

  – A tuple giving the arguments to the SciPy `linspace` function to specify the grid points for searching with this particular variable.

  – A Boolean variable, which can only be `True` for two of the variables, specifying the variables that constitute a slice of the search. Due to the benefits of vectorization in searching, it can often be best to incorporate those variables that lead to slices with the most points. Another strategy is, because all pairs that are identical except the value of $\hat{b}_6$ or $\hat{b}_7$ will have the same A^6, incorporating $\hat{b}_6$ or $\hat{b}_7$ into a slice means that slices with too large an A^6 will often not need to compute other characteristic numbers.

Arithmetic operations, e.g., `z = 2+2`, in PYTHON can be orders of magnitude slower than low-level languages such as C, C++, or FORTRAN, largely because of the overhead of the PYTHON interpreter and object creation. However, leveraging vectorization in combination with libraries such as SCIPY for heavy numerical computation can often bring that difference down to a factor of two to three [201] or even less.[20] In particular, the overall process of brute-force search of the free parameters of RK method families is extremely amenable to vectorization, hence the performance disadvantage of the PYTHON interpreter is much smaller than would be expected by benchmarking simple arithmetic operations.

To take advantage of vectorization, a 2D slice of a *hypercube* of the space of free parameters for the appropriate RK method family have all of their characteristic numbers calculated together. Although NUMPY supports arrays with more than two dimensions, it was found that 2D slices were generally large enough to fully exploit vectorization while being small enough to ensure that all searches necessary for this study could be done within the memory that was available. This is accomplished because the generated code (except the functions that return the actual Butcher tableau) can take multi-dimensional arrays as arguments, as already discussed in Section 4.7. The following examples give an idea of how much vectorization speeds up the searches of the free parameters. When testing using a selection of the functions from Listing 4.7 for the family containing the DP5(4)$_{6(7)}$ pair (2.79), going from 1×1 input arrays to 21×21 input arrays takes 2.6 times as long, with the latter having 441 points and clearly illustrating the benefits of vectorization. When moving from 21×21 input arrays to 101×101 input arrays, it takes about 12 times longer, with about 23 times as many points, indicating a continued but increasingly marginal benefit to vectorization. On the desktop computer used for this study, the scaling starts to become worse, with input arrays of 401×401 taking 22 times longer but with only about 16 times more points than 101×101 input arrays, likely due to bottlenecks in processes such as memory management. These observations show that for this study, using 2D slices of the parameter space rather than much larger 3D cubes is best for the number of points in a typical search. However, although these observations may give guidelines for other users, they are dependent on many factors, such as the specifications of the particular machine and architectures used, the particular version used for PYTHON and libraries, etc.

An example of how to run the `erk_search.py` source file is:

```
python erk_search.py --dopr54s7 complete-embedded default-embedded
```

that given the class `ERKSearch_dopr54s7` in Listing 4.8, the command line argument `--dopr54s7` selects the class defined in the `erk_search.py` source file with the `label` attribute equal to `'dopr54s7'` and `complete-embedded` as the second argument indexes the key `'complete-embedded'` in the attribute

[20]https://www.tcm.phy.cam.ac.uk/~mjr/linpack/

search_variables of the ERKSearch class for the search to perform. The additional 'default-embedded' argument indexes the appropriate set of functions in the search_functions attribute described in Listing 4.7. The textual output of running the erk_search.py source file ensures that the user can monitor the slices as they are being searched, an advantage due to the length of time (many hours) that large searches may require.

4.9.1 Storage in a PostgreSQL database

It is well-known that using a database management system such as PostgreSQL has many advantages in comparison to simple file storage. An enormous advantage that a database management system (DBMS), such as PostgreSQL, gives lies in supporting concurrent storage and retrieval of data by many processes across the different machines over a network. Experience has shown that declarative languages such as SQL (as opposed to imperative languages, such as Python) confer many advantages for storage, retrieval, and transformation of data that can drastically reduce software development costs and eliminate many sources of error. Databases generally have preexisting functionality to handle many implementation details required for managing large and complex data sets efficiently, e.g., error handling, managing storage space, indexing, scalability, etc. This frees the user/developer from having to consider many implementation details, and allows access to functionality for which they would not otherwise have the development resources. The largest size of the datasets generated by using erk_search.py for this thesis are 300GB, which for complex data sets is large enough to be difficult to manage without a tool like a proper database.

For OCSage searches, a partial example of the PostgreSQL table storing data from a search is given in Listing 4.9. Each search is stored within a table with the name ocsage_data_«datestamp»_«label». The first sets of columns are pairs of columns named after the strings used as variable names throughout OCSage; with the variable searched as the first column name in a pair, e.g., c2 for c_2; and an integer index within the hypercube searched as the second column name in a pair, with the column name being the variable name with the suffix _index, e.g., c2_index refers to the c_2 variable. The «variable»_index column is included because integers are far more efficient and less error prone than floating-point values for grouping and comparison operations within the PostgreSQL database. Subsequent columns use the string name from Listing 4.7 that refers to the property from the search_functions attribute and contains its floating-point value at the set of free parameters for that particular row.

In addition, the searches that have been conducted are contained within the table ocsage_searches, which contains information such as the timestamp, label, number of datapoints stored, the hostname used, the command line arguments used, the total execution time, whether the search is successful, and several more pieces of information. This makes OCSage more convenient to use for a large number of families, such as this study does, because a lot of essential information is stored automatically and therefore does not need to be manually recorded and organized. An example of several rows of a PostgreSQL table from a search, along with the column headings, is given in Listing 4.9. An example of an SQL query for finding ERK pairs with certain restrictions on Butcher tableau coefficients and properties is:

```
SELECT * FROM ocsage_data_20180421T042053_erk5sVVIs6FSAL7 WHERE error_6 > 0.0003
    AND error_6 < 0.0005 AND char_e < 10 AND c4 < 0.7 AND c5 < 0.93 ORDER BY
    error_6;
```

Additionally, `PostgreSQL` allows easily creating new tables from existing tables by either combining searches or narrowing down a large search into a smaller set of data for examination. The `psycopg` library already mentioned in Section 4.1 easily allows querying the database from PYTHON and using `pythODE` with the `«label»_tableau` functions from the generated SCIPY code to quickly reproduce the Butcher tableau from an appropriate table row when needed.

4.9.2 Plotting the results of a search

The data from the searches that stored in the `PostgreSQL` database can be used to generate plots and figures. The default methodology in the `erk_search.py` source file for plotting simply plots the slices from searches for inspection, in some instances this can be many thousands of slices if a general overview of the space of free parameters is wanted. Although data in the database could be manipulated for further plotting, for this study just plotting individual slices is found to be sufficient. An example of these slices can be seen in Figure 4.8, which is described in detail below along with others like it.

The best MATPLOTLIB function to plot figures such as in Sections 4.12 and 4.13 is the `imshow` function with the interpolation option turned off. An example of plotting these slices is Figure 4.10, which are only two of the many plots of slices generated for a search. The command to plot can be given as

```
python erk_search.py --plot <<datestamp>> <<plot selection>>
```

that uses `«datestamp»` to infer the SQL table and the label to plot and where `«plot selection»` refers to the key in the `plots` attribute shown in Listing 4.8. Alternately

```
python erk_search.py --plotrecent <<label>> <<plot selection>>
```

finds the most recent search with the `label` attribute `«label»` and plots this. In addition to the default `.png` format of the plots, a `--pdf` argument can be added to generate the plots as `pdf` format.

If too many slices are used or plotting is not carefully restricted when doing fine-grained searches, there can be issues with performance and resource utilization. For instance, if too many points or slices are used for families where there can be five parameters, it is easy to generate over 10000 slices, which would need to be plotted to picture the whole space of free parameters. In one case during initial investigations for this study, it took only 8 hours to actually do the search but over 24 hours to generate all the plots on the particular desktop computer used in this study. In addition, many file managers and image viewers have difficulty efficiently thumbnailing or otherwise allowing efficient browsing of that many images. In these cases, it is better to narrow down the region of interest in the space of free parameters using just textual database searches before doing any plotting. For future studies, more sophisticated data visualization tools may assist in constructing new ERK pairs.

4.10 Comparison of `OCSage` output with published values in the literature

In order to verify the correctness of `OCSage`, using known free parameters and manually checking the resulting Butcher tableaux against ones published in the literature is relatively straightforward, especially when using rational numbers to give calculated coefficients that can be compared exactly. However, ensuring the calculated characteristic numbers (2.56) are all correct is a bit more difficult because they are not uniformly presented in every publication. For instance, they are often presented as floating-point values with few decimal places, and this is unhelpful for finding the subtle errors that can sometimes creep into the implementations. Therefore, to ensure `OCSage` is correct, it is necessary to make comparisons with as many published characteristic numbers as possible. Table 4.2 shows many of the known values of characteristic numbers (2.56) that have been published for classic ERK methods along with the values calculated by `OCSage`. It can clearly be seen that all of these values match those calculated with `OCSage` when taking into account the precision of the original publication. In some cases, characteristic numbers were discussed, but numerical values were not given, or they may have only been shown graphically. In these cases, the values calculated by `OCSage` have been deemed consistent with the original publication. Table 4.2 should also give the reader a good idea of typical characteristic numbers for classic ERK methods.

The code generated to implement the RK scalar sums (2.39) in Section 4.4 can be verified to tenth order by testing the Butcher tableau coefficients of published higher-order RK methods (up to tenth order) [69][72, pgs.179–185][136], which is done by verifying that the expressions for the order conditions vanish. This helps verify the characteristic numbers that incorporate higher-order scalar sums (2.39). However, care is still needed to ensure that the algorithms and resulting calculated values for α and σ are correct. The values of α and σ were verified up to eighth order by manually checking with a publication by Butcher [24], which was tedious because the ordering of the values presented by Butcher is not the same as that resulting from the sort order produced by generating the code described in Section 4.4.

Table 4.2: Comparing characteristic numbers of ERK pairs calculated by `OCSage` with published values.
-=not examined, nc=not calculated but discussed, g=only graphics provided but consistent with `OCSage`.

Formula	A^6	A^7	A^8	A^9	$\hat{A}^5$	$\hat{A}^6$	$\hat{A}^7$	stability term	stability length	fourth-order stability length
RKF4(5)$_{6(6)}$ [16]	0.0034	0.0068	0.0081	0.008	-	-	-	-	-	-
RKF4(5)$_{6(6)}$ [30]	0.0034	0.0068	-	-	0.0018	0.0058	-	-	-	-
RKF4(5)$_{6(6)}$ [47]	0.00336	-	-	-	-	-	-	-	-	-
RKF4(5)$_{6(6)}$ `OCSage`	0.003357	0.006765	0.008069	0.008039	0.001839	0.005805	0.009445	0.00048077	3.677706	-
DP5(4)$_{6(7)}$ [16]	0.00040	0.0040	0.0043	0.0042	-	-	-	-	-	-
DP5(4)$_{6(7)}$ [47]	0.000399				-	-	-	$\frac{1}{600}$ (0.001666...)	3.3 [48]	g
DP5(4)$_{6(7)}$ `OCSage`	0.000399	0.003956	0.004260	0.00421653	0.001183	0.0018238	0.004141	0.0016666	3.306568	-
DP5(4)S$_{6(7)}$ [47]	0.00181	-	-	-	-	-	-	$\frac{1}{1280}$ (0.00078125)	5.7 [48]	g
DP5(4)S$_{6(7)}$ `OCSage`	0.001813	0.002510	0.002494	0.002246	0.000421	0.002092	0.002824	0.00077160	5.704636	-
DP5(4)C$_{6(7)}$ [48]	0.00149					-	-	-	4.4	-
DP5(4)C$_{6(7)}$ `OCSage`	0.001489	0.002064	0.002176	0.002037	0.000748	0.0020579	0.002679	0.00096154	4.436779	-
DP5(4)M$_{6(6)}$ [47]	0.00123	-	-	-	-	-	-	nc	g	-
DP5(4)M$_{6(6)}$ `OCSage`	0.001227	0.001642	0.001619	0.001485	0.002227	0.002852	0.003236	0.00125000	3.734360	-
Bogacki–Shampine 5(4) [16]	0.000022	0.00021	0.00035	0.00042	-	-	-	-	-	-
CK4(5)$_{6(6)}$ [30]	0.0009	0.0013	-	-	0.0005	0.0011	-	-	-	-
CK4(5)$_{6(6)}$ `OCSage`	0.000948	0.001368	0.001452	0.001351	0.000539	0.001153	0.001358	0.00125000	3.734360	-

Listing 4.9: Part of the `PostgreSQL` table resulting from a search of the ERK family containing DP5(4)$_{6(7)}$.

```
 c2  | c2_index | bhat7 | bhat7_index |  c5  | c5_index |  c4  | c4_index |      error_6       |      error_7       |      error_8       |      error_9       |  error_ratio   |  stab_length
-----+----------+-------+-------------+------+----------+------+----------+--------------------+--------------------+--------------------+--------------------+----------------+---------------
0.11 |       11 |     0 |           0 | 0.68 |       68 | 0.41 |       41 | 0.00159442275658 | 0.00225720855214 | 0.00247981806725 | 0.00227977509487 | 1.41569012536 | 3.10877267581
0.11 |       11 |     0 |           0 | 0.69 |       69 | 0.41 |       41 |  0.0015889918708 | 0.00225214596756 | 0.00247291083356 | 0.00227805141468 | 1.41734266169 | 3.10877267581
0.11 |       11 |     0 |           0 |  0.7 |       70 | 0.41 |       41 | 0.00158608255374 | 0.00225141948593 | 0.00247154818703 | 0.00228243535035 |  1.4194844276 | 3.10877267581
0.11 |       11 |     0 |           0 | 0.71 |       71 | 0.41 |       41 | 0.00158570868451 | 0.00225519764721 | 0.00247628897157 | 0.00229376985036 | 1.42220173808 | 3.10877267581
0.11 |       11 |     0 |           0 | 0.72 |       72 | 0.41 |       41 | 0.00158787205405 | 0.00226366289544 | 0.00248780777601 | 0.00231309066195 | 1.42559527367 | 3.10877267581
0.11 |       11 |     0 |           0 | 0.73 |       73 | 0.41 |       41 | 0.00159256232248 | 0.00227702926094 | 0.00250695532523 | 0.00234171412468 | 1.42978973494 | 3.10877267581
<<...>>
```

4.11 Examination of lower-order ERK methods for comparison

It is instructive to briefly examine some properties of popular lower-order ERK methods in order to give a frame of reference for the data obtained from searching families of $5(4)_6$ ERK pairs. In order to construct the best ERK pairs in a well-defined way, in this study it is shown that the individual PECs must be examined. Recall from Chapter 2 that the standard definition of PECs (2.47) can be normalized to the definition of $\overline{\text{PECs}}$ (2.48) so that a value of 1.0 represents the error if the Taylor series of the local solution (2.38) were simply truncated. The values of the individual PECs (2.47) for the FE method (2.25) are shown by a bar graph in Figure 4.4 (left side), where the ordering of the bars is the sort ordering for the data structures used by `OCSage` to represent the rooted trees. This sort ordering can be seen for scalar sums (2.39) up to sixth order by their ordering in Tables B.1 and B.2 or up to twelfth order by examining the code in the `OCSage` directory `generated_trees/`. The analogous bar graph for the $\overline{\text{PECs}}$ (2.48) is shown in Figure 4.4 (right side), where all bars have a value of 1.0 because the FE method (2.25) can be seen as a simple truncation of the Taylor series. The simple truncation of the Taylor series of the local solution (2.38) done by the FE method (2.25) is obvious in Figure 4.4 (right side), but not in Figure 4.4 (left side) because of the additional scaling by $\frac{\alpha(\mathbf{F})}{q!}$ of the PECs (2.47). However, the PECs (2.47) continue to be useful for constructing quantities such as A^q because each bar is proportional to the contribution to the error from each elementary differential $\mathbf{F} \in \mathfrak{F}^q(\mathbf{f}(\mathbf{y}(t)))$ (2.36) would make if all $\mathbf{F}$ were equal, which is probably still the best assumption that can be made if no particular IVP or class of IVPs is specified [16, 146]. However, in Chapter 5 it is shown that this assumption does not necessarily apply to at least some specific classes of IVPs from practical applications.

To illustrate a non-trivial example of the values of the individual PECs (2.47), an analogous figure to Figure 4.4 for the RK4 method (2.26) is given by Figure 4.5 (left side), with $\overline{\text{PECs}}$ (2.48) for the RK4 method (2.26) given by Figure 4.5 (right side). It can be seen in Figure 4.5 (right side) that with increasingly higher order, an increasing fraction of $\overline{\text{PECs}}$ (2.48) are 1.0 because an increasing number of scalar sums (2.39) evaluate to 0.0. This can easily be seen to be because the scalar sums (2.39) of an ERK method evaluate to 0.0 if the height of the rooted tree corresponding to a scalar sum (2.39) is larger than the number of stages. The idea of avoiding vanishing PECs has been discussed in the literature [16, 190] but is not seen to specifically be an issue in the data examined for this thesis. That individual PECs can assume excessive magnitude does not appear to have been discussed in the literature either but appears to explain the poor performance of some $5(4)_6$ ERK pairs at coarse accuracies.

4.12 Search results for classic $5(4)_6$ ERK pairs

Although the classic six-stage fifth-order ERK methods discussed in Section 2.7 have been under study for several decades, the specific tradeoffs between the possible families and in the coefficient selection process have not been presented in any kind of detail in the literature. Only general discussions of the families used and a

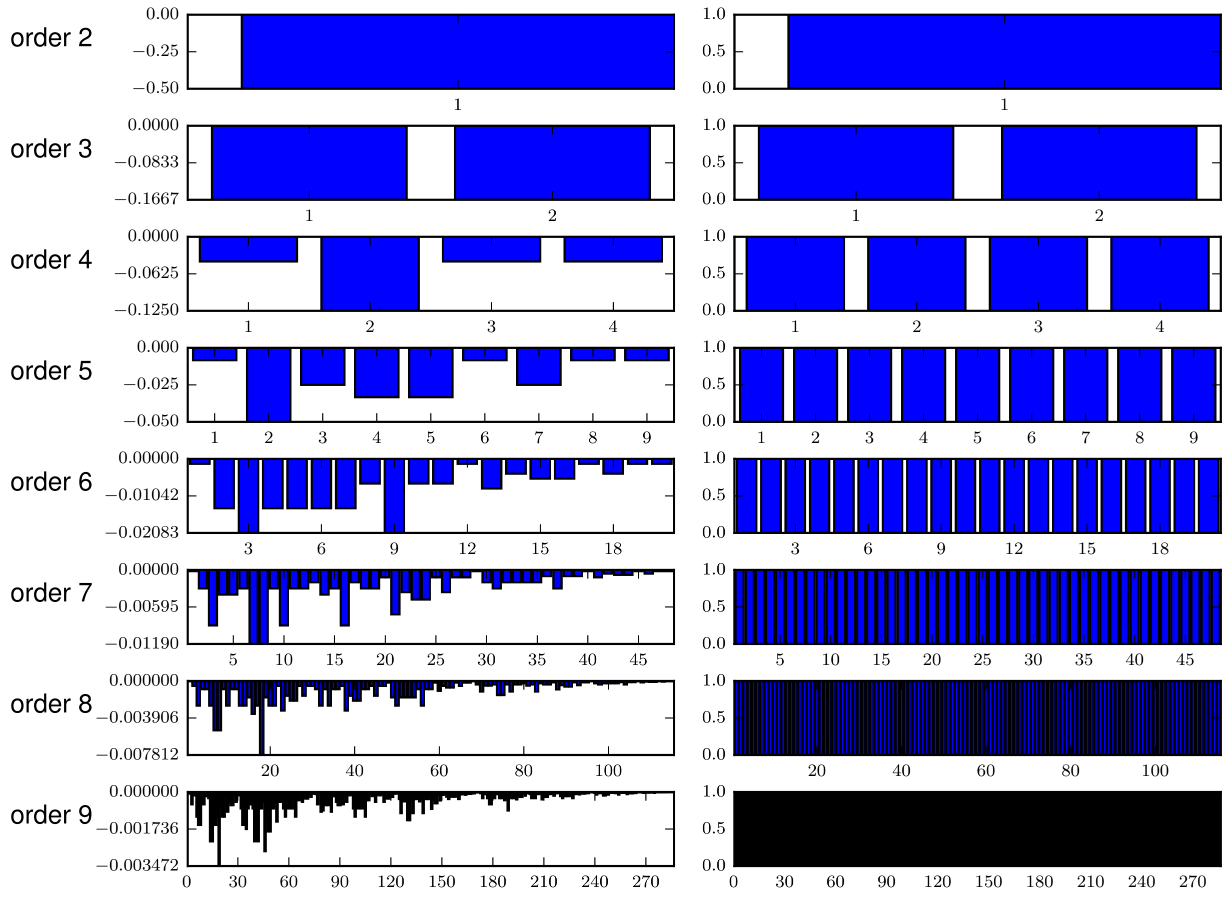


Figure 4.4: The individual PECs (2.47) for the FE method (2.25). (left) The individual $\overline{\text{PEC}}$s (2.48). (right)

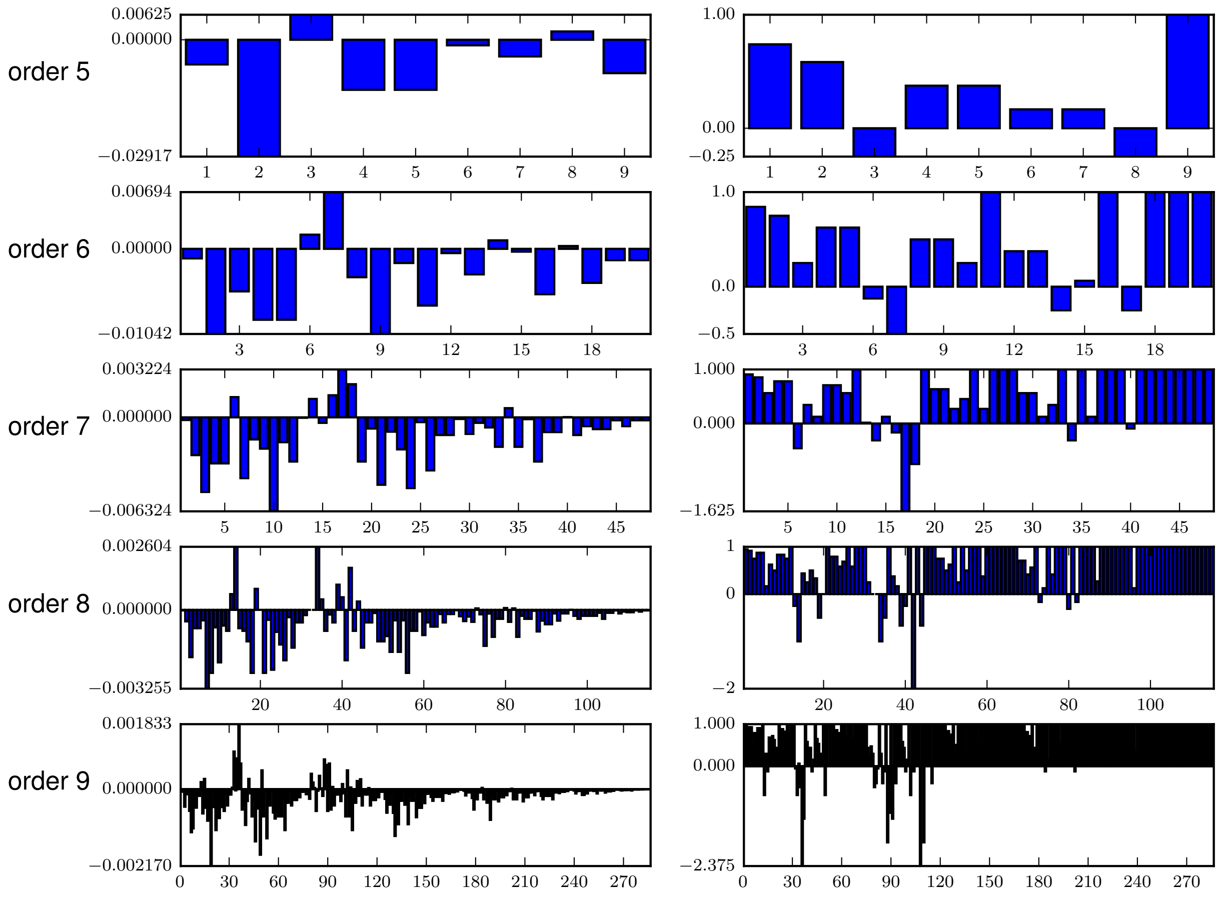


Figure 4.5: The individual PECs (2.47) for the RK4 method (left). The individual $\overline{\text{PEC}}$s (2.48) (right).

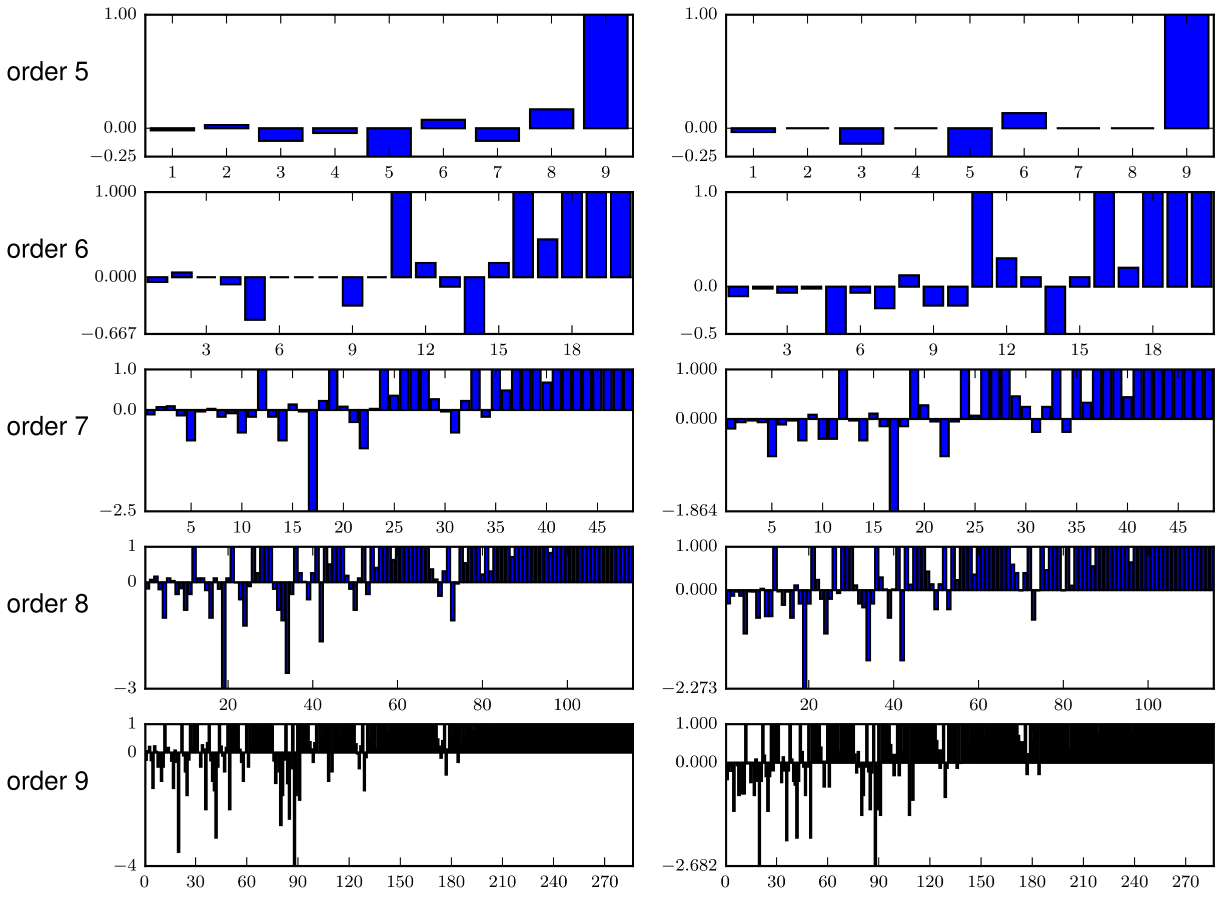


Figure 4.6: The individual $\overline{\text{PEC}}$s (2.48) for the "three-eighths" four-stage fourth-order method (left) and Ralston's four-stage fourth-order method (right).

small number of specific methods themselves are usually presented in the literature. Due to this, the relative merits of different ERK methods are still the subject of at least informal discussion. In this section, `OCSage` is demonstrated by presenting data that helps picture the space of free parameters and specific tradeoffs for the families that were originally used to construct the RKF4(5)$_{6(6)}$ (2.78) and DP5(4)$_{6(7)}$ (2.79) pairs. The remainder of this chapter should give a clear idea of what the capabilities and limitations of actually using `OCSage` to analyze RK families might be.

For illustrative purposes, the upper bound for the leading error coefficient A^6 in many searches conducted with `OCSage` is often much larger than what is typically used for efficient pairs. Although it is seen in Chapter 5 that performance is not as simplistically predicted by the magnitude of the leading error coefficient as sometimes been implied in the literature [16, 146], having error coefficients as small as possible is desirable and ERK methods with overly large error coefficients do not need to be considered. Although the new efficient $5(4)_6$ ERK pairs derived for this thesis will have an A^6 smaller than about 0.0012, displaying A^6 up to 0.0064 gives a better picture of the space of free parameters for the purposes of study. In general, comparing error coefficients between methods of different orders should only be done to get a rough comparative picture because in some cases comparing the magnitude of error coefficients between methods of different order can be misleading. Due to this, even though error coefficients of different orders are actually incorporated into the formula of classic characteristic numbers like B (2.56a), C (2.56b), and E (2.56d), the precise values of these classic characteristic numbers (2.56) do not have a precise meaning.

4.12.1 The family derived by Fehlberg containing the RKF4(5)$_{6(6)}$ pair

In this subsection, the original family derived by Fehlberg [61] containing the RKF4(5)$_{6(6)}$ pair (2.78) is used for illustrative purposes because there are only three free parameters. However, to properly study the tradeoffs of many classic ERK pairs composed of six-stage fourth- and fifth-order methods, it is much better to examine the more general family, i.e., the family of $5(4)_{6(6)}$ ERK pairs from Section 3.3.4, that is studied in Section 4.13.

Despite using a more restrictive family, many observations of the fifth-order component of the $5(4)_6$ ERK pairs in this subsection generally hold for $5(4)_{6(6)}$ ERK pairs other than the RKF4(5)$_{6(6)}$ pair (2.78), e.g., the DP5(4)M$_{6(6)}$ pair (2.82) published by Dormand and Prince [47] in the same paper as the DP5(4)$_{6(7)}$ pair (2.79), the CK4(5)$_{6(6)}$ pair (2.83), and the multitude of $5(4)_6$ ERK pairs that have been constructed by Verner [187]. The CK4(5)$_{6(6)}$ pair (2.83) has $b_5 = 0$, which is not handled by the family of $5(4)_{6(6)}$ ERK pairs from Section 3.3.4 and therefore the code currently generated by `OCSage`. However, perturbing the free parameters slightly (c_2 and c_5 by $\frac{1}{300}$ as well as c_3 by $\frac{1}{200}$ to preserve the C(3) simplifying assumptions (2.44b)) gives a method with nearly identical characteristic numbers (2.56) (all within 0.1%) to the CK4(5)$_{6(6)}$ pair (2.83) that also performs nearly identically in the performance testing described in Chapter 5. Therefore, for the purposes of the discussion in this chapter, conclusions can be made about the CK4(5)$_{6(6)}$ pair (2.83) using the `OCSage` generated code for the $5(4)_{6(6)\mathrm{C}(2)}$ family from Section 3.3.4 by using this perturbed method. A modified family that actually contains the CK4(5)$_{6(6)}$ pair (2.83) [30] would be possible to incorporate. However, to find the families that properly handle non-trivial but unimportant singularities arising from values such as $b_5 = 0$ would require substantially expanding on Chapter 3 and not currently yield any additional insights on ERK method construction.

As a representative example of searching the space of free parameters with `OCSage`, the family that Fehlberg [61] used to derive the RKF4(5)$_{6(6)}$ pair (2.78) is searched by a uniform sampling of each of the free parameters c_2, c_5, c_6 at 101 points in the interval $[0, 1]$. Given these 1,030,301 potential points, a small number will be invalid for the families presented in this thesis, for example, either $\mathbf{c}$ components that are equal or, similarly to the CK4(5)$_{6(6)}$ pair (2.83), the value of the specific coefficients leads to a singularity.

The smallest sixth-order leading error coefficient A^6 of the six-stage fifth-order method found is 0.00087955035750 from using the free parameters $c_2 = \frac{19}{100}$, $c_5 = \frac{82}{100}$, and $c_6 = \frac{88}{100}$. Searches at much higher resolution give only a slightly smaller A^6. However, in the search just described there are also 1458 other points that have an $A^6 \leq 0.0010$, many of which could often expected to have at least relatively similar or even better performance because of the importance of the individual PECs for some classes of IVPs, which is shown in Chapter 5. It can be noted that for a fifth-order method, $\left(\frac{0.001}{0.00088}\right)^{\frac{1}{5}} = 1.0259$ implies a potential of 2.6% better performance based on the simplistic assumption that global error is directly related to the magnitude of the leading error coefficient. This simplistic assumption that global error is directly related to the magnitude of the leading error coefficient is called "relative efficiency" by Shampine [146] and is mentioned again in Chapter 5 as part of a more detailed discussion of the relative performance of ERK pairs.

The smoothness of the space of free parameters can be seen in Figure 4.7, which shows the value of the leading error coefficient A^6 of six-stage fifth-order component of the pair, with c_5 on the x-axis, c_6 on the y-axis, and each sub-figure representing a value of c_2 progressing incrementally from 0.02 to 0.36. Outside of this range of c_2 the value of A^6 is always extremely large, i.e., close to 0.0064 or even much greater. Observe that despite the space of free parameters being relatively smooth, there is a rapid increase in A^6 from $c_2 = 0.21$ to $c_2 = 0.24$, which indicates that at least some precision may be required when selecting coefficients. Figure 4.7 also reflects what the possible choices in leading error coefficient for the six-stage fifth-order component of the other similar classic pairs mentioned above generally looks like [30, 47, 187]. Observe that in constructing the CK4(5)$_{6(6)}$ pair (2.83), Cash and Karp found the region with a nearly minimal A^6, and that the results of performance experiments in Chapter 5 can easily be used to argue that they found one of the best overall classic 5(4)$_6$ ERK pairs. Therefore, for the RKF4(5)$_{6(6)}$ pair (and 5(4)$_{6(6)\mathrm{C}(2)}$ family from Section 3.3.4, as well as the similar families corresponding to Cases V and VI derived in Section 3.4.1), looking at other properties (such as the other characteristic numbers (2.56) or individual values of the $\overline{\mathrm{PEC}}$s (2.48)) beyond the leading error coefficient is necessary to improve on the current state-of-the-art pairs.

Although space does not permit showing the plots for all characteristic numbers for all 35 slices, i.e., $c_2 = 0.02$ to $c_2 = 0.36$ in 0.01 increments, that correspond to the slices given by Figure 4.7, two interesting ones corresponding to $c_2 = 0.2$ and $c_2 = 0.25$ are given by Figure 4.8. Observe in Figure 4.8a (corresponding

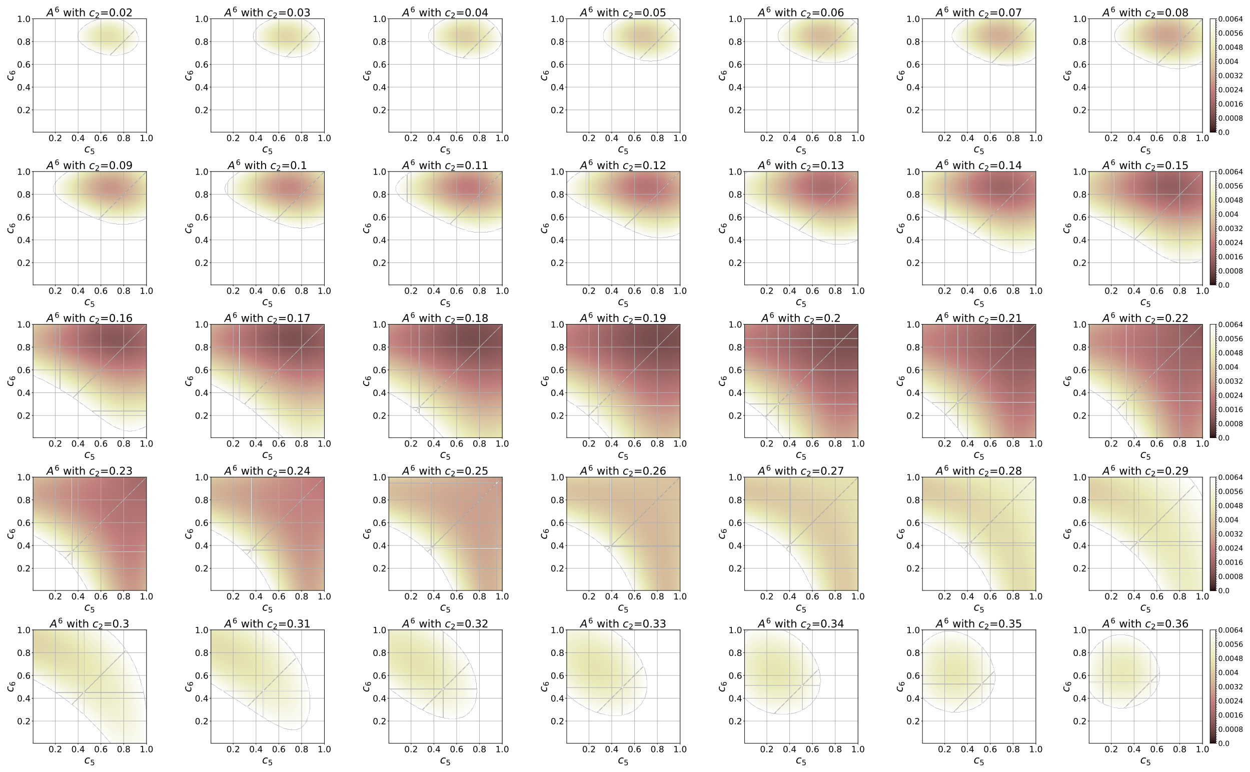


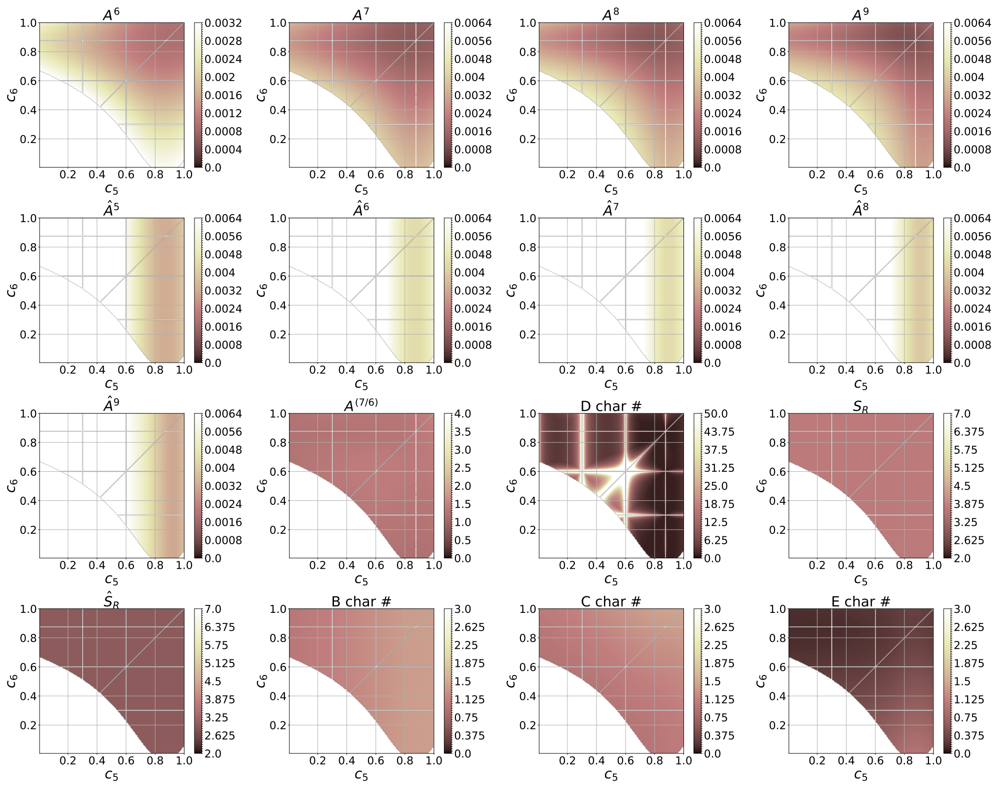
Figure 4.7: The leading error coefficient A^6 for the family originally used to derive the RKF4(5)$_{6(6)}$ pair (2.78) with c_2 in the interval $[0.02, 0.36]$.

to $c_2 = 0.2$) that despite the existence of small leading error coefficients A^6 for the fifth-order method, these occur in conjunction with a much larger leading error coefficient $\hat{A}^5$ for the fourth-order method. Having a much larger leading error coefficient in the error estimator than the method used to advance the solution, i.e., usually when $|\hat{b}_6 - b_6|$ is large, is seen in Section 5.12 to sometimes be advantageous and sometimes disadvantageous. The free parameter $c_2 = 0.2$ used by Cash and Karp [30] for the CK4(5)$_{6(6)}$ pair (2.83) allows finding the region of smallest A^6 when constructing that pair. In Figure 4.8b corresponding to $c_2 = 0.25$, $\hat{A}^5$ can be relatively small but only occurs with A^6 that is relatively large. The free parameter $c_2 = 0.25$ was chosen by Fehlberg [61] for the RKF4(5)$_{6(6)}$ pair (2.78) and although not ideal, it is the probably the best compromise for this family if the fourth-order component is used to advance the solution, such as Fehlberg originally intended. The performance experiments done in support of this thesis show that too few free parameters, such as with the family originally used [61] to construct the RKF4(5)$_{6(6)}$ pair (2.78) in comparison to the more general 5(4)$_{6(6)\mathrm{C}(2)}$ family, can severely impact the potential for good performance.

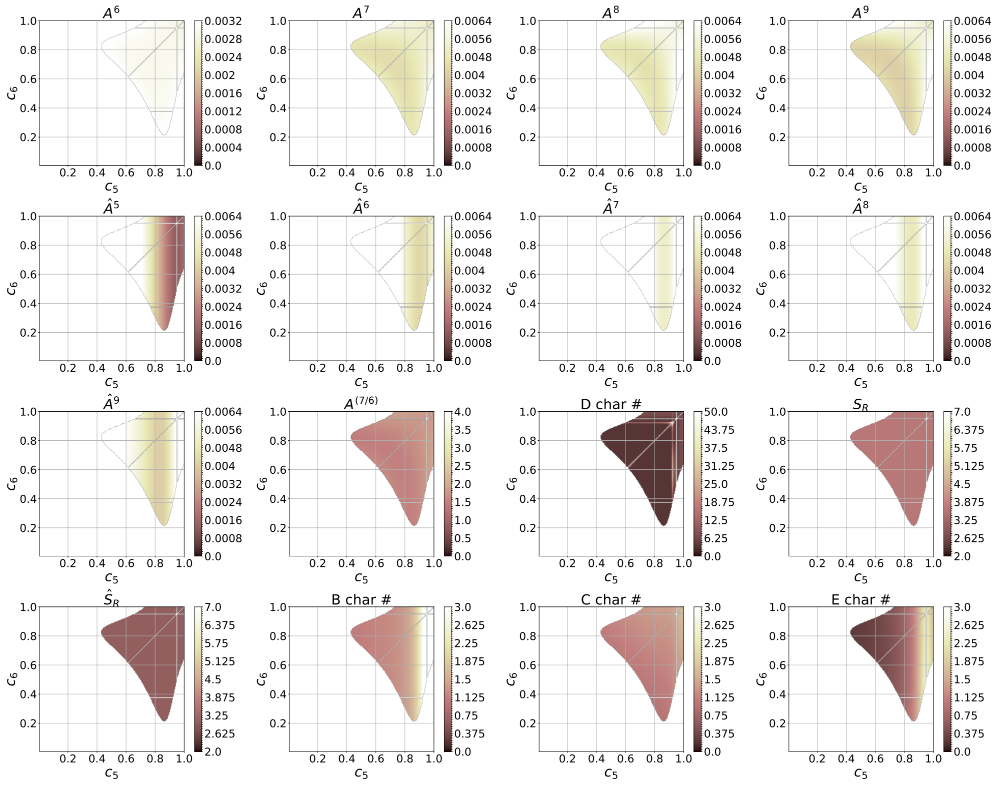
4.12.2 The family derived by Dormand and Prince containing the DP5(4)$_{6(7)}$ pair

Figure 4.9 is a similar figure to Figure 4.7 for demonstrating that the leading error coefficient changes smoothly, but instead shows the fifth-order method of the family containing the DP5(4)$_{6(7)}$ pair (2.79) that was originally constructed by Dormand and Prince [47]. It can clearly be seen that between $c_2 = 0.18$ and $c_2 = 0.23$ there are regions of extremely small leading error coefficients (the darkest shading is less than or equal to the leading error coefficient of the DP5(4)$_{6(7)}$ pair (2.79)). A simplistic relation between the leading error coefficient and performance indicates that the smallest leading error coefficients found, which are five times smaller than the A^6 of the DP5(4)$_{6(7)}$ pair (2.79), could lead to a potential 41% performance gain over the DP5(4)$_{6(7)}$ pair (2.79), i.e., $(4)^{0.2} = 1.41$. In fact, increasingly finer grained searches reveal extremely small leading error coefficients and experimentation shows that this can be taken to an arbitrary degree, although eventually the Butcher tableau coefficients get extremely large, i.e., $D \gg 100$.

However, even if the Butcher tableau coefficients are of reasonable magnitude, e.g., < 40, there are further practical considerations discussed further in Chapter 5 meaning that even when high accuracies are desired, the performance gains from aggressively minimized leading error coefficients are often lower than what would be expected by simplistic considerations. For instance, there are pairs in the family containing the DP5(4)$_{6(7)}$ pair (2.79) that have a much smaller leading error coefficient than DP5(4)$_{6(7)}$ pair (2.79), but with $c_4 > 0.94$ and $c_5 > 0.98$. These values of c_4 and c_5 are close enough to $c_6 = 1$ that they may hurt performance [47]. In addition, the searches conducted with `OCSage` show that whenever the leading error coefficient is around the size of that of the DP5(4)$_{6(7)}$ pair (2.79) or much smaller, certain $\overline{\text{PEC}}$s (2.48) always assume a magnitude greater than 1.0 and this seems to negatively impact performance at coarse accuracies. An example of a published pair with an extremely small leading error coefficient is the PP5(4)$_{6(7)}$ pair (2.84) [128] with $A^6 \approx 0.0000654$, which was originally constructed from a family with one less free



(a) The characteristic numbers (2.56) when $c_2 = 0.2$.



(b) The characteristic numbers (2.56) when $c_2 = 0.25$.

Figure 4.8: Characteristic numbers (2.56) for the family containing the RKF4(5)$_{6(6)}$ pair (2.78).

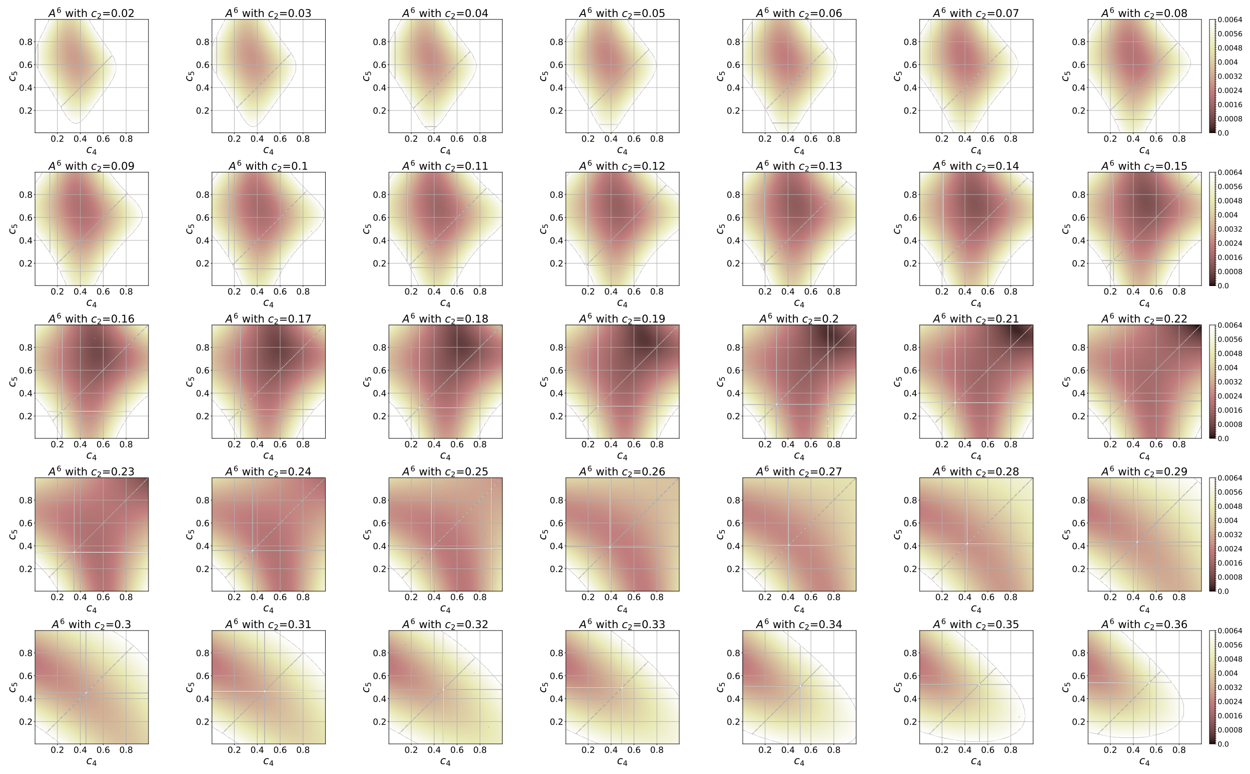


Figure 4.9: A^6 for the family containing the DP5(4)$_{6(7)}$ pair (2.79) in the interval $c_2 = [0.02, 0.37]$.

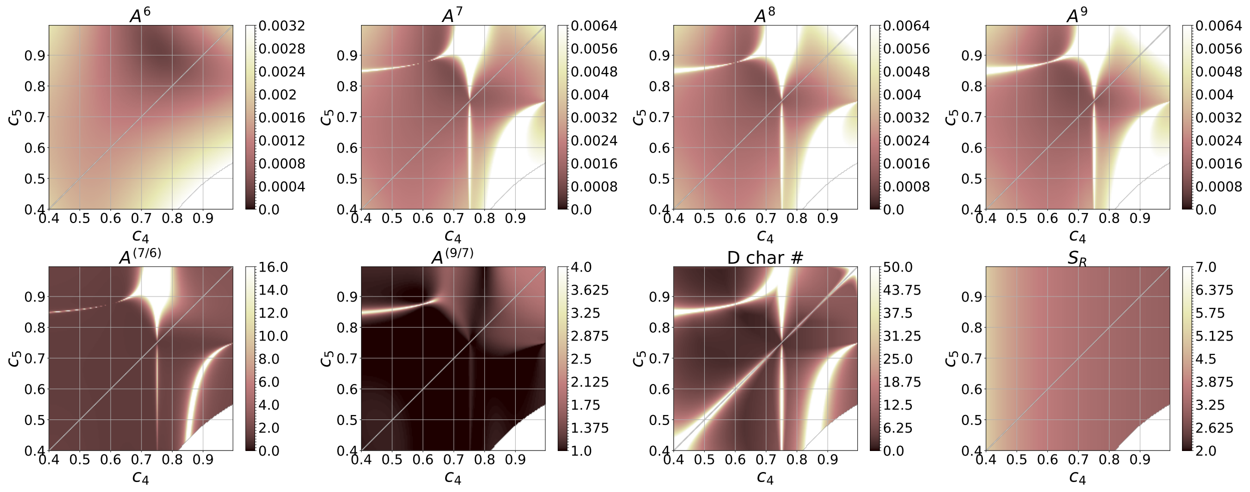


Figure 4.10: Characteristic numbers (2.56) for the family containing the DP5(4)$_{6(7)}$ pair (2.79) with $c_2 = 0.2$.

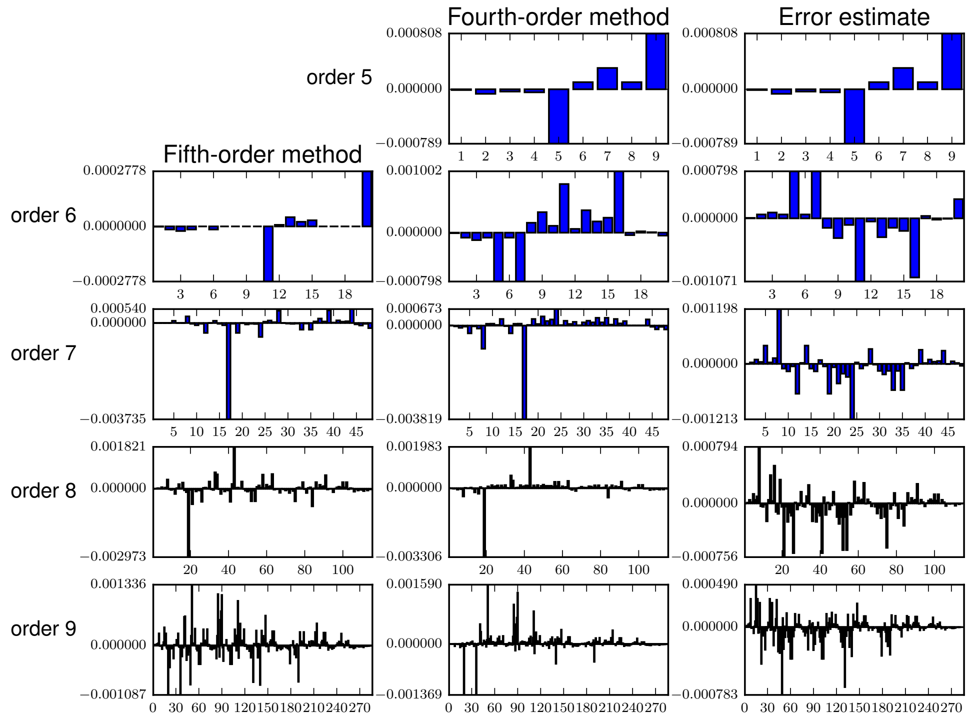
parameter than the $5(4)_{6(7)\mathrm{C}(2)}$ family from Section 3.3.5. The performance data in Chapter 5 shows that the PP5$(4)_{6(7)}$ pair (2.84) [128] often has a small (approximately 10–15%) performance advantage over pairs such as the CK4$(5)_{6(6)}$ pair (2.83) for high accuracy calculations, despite simplistic considerations of the leading error coefficient indicating there should be a much greater performance advantage. However, when solving some IVPs, pairs with small leading error coefficients, such the DP5$(4)_{6(7)}$ (2.79) and PP5$(4)_{6(7)}$ (2.84) pairs, can unexpectedly be 30–40% less efficient than pairs such as CK4$(5)_{6(6)}$ (2.83) even at high accuracies, which is investigated further in Chapter 5.

Figure 4.10 shows the space of free parameters for $c_2 = 0.2$ that includes both the DP5$(4)_{6(7)}$ pair (2.79) [47] and the DP5(4)C$_{6(7)}$ pair [48]. Although Figure 4.10 only reflects the characteristics of the fifth-order method and not the pair as a whole, an obvious property is that the DP5(4)C$_{6(7)}$ pair (2.81) has much smaller values of the higher-order error coefficients A^q, $q \in \{7, 8, 9\}$ than the corresponding error coefficients for the DP5$(4)_{6(7)}$ pair (2.79)). This seems to improve performance at coarse tolerances and helps explain why for some sets of IVPs Dormand and Prince observed the DP5(4)C$_{6(7)}$ pair (2.81) outperforming the DP5$(4)_{6(7)}$ pair (2.79) [48]. Dormand and Prince gave an explanation for this observation based on large matching stability regions. However, no observations made in the course of the study in this thesis indicate that this is the likely reason.

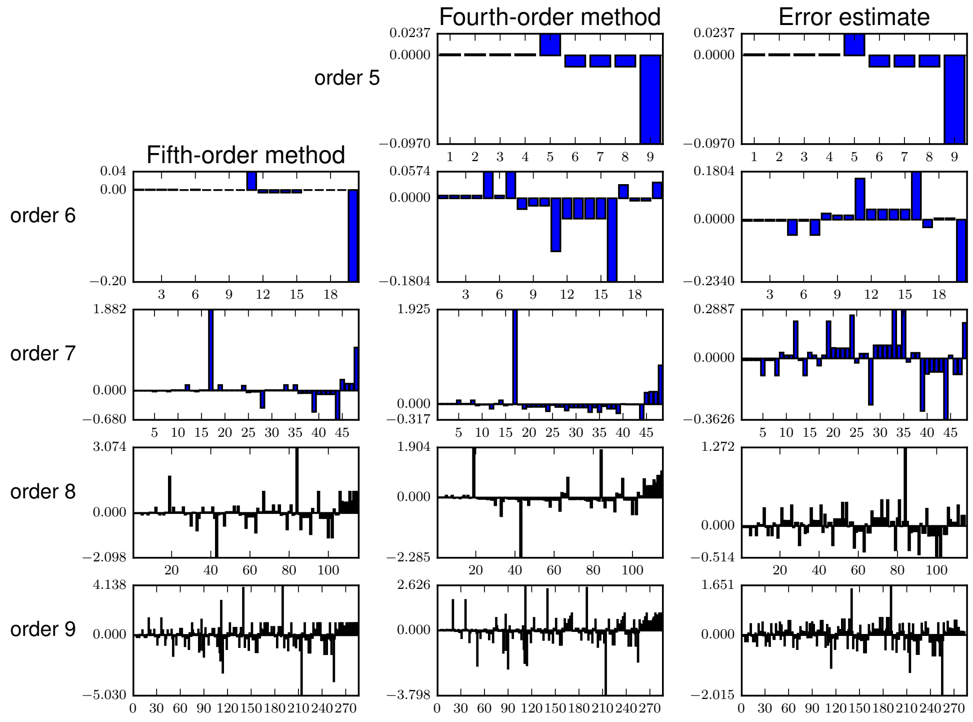
For a closer look at how choosing the free parameters might affect the error coefficients and resulting global behaviour, bar graphs of the PECs (2.47) of the DP5$(4)_{6(7)}$ pair (2.79) are given in Figure 4.11a and the $\overline{\text{PEC}}$s (2.48) are given in Figure 4.11b. In both subfigures of Figure 4.11 the first row to last row give the fifth-order to eighth-order PECs (2.47) and $\overline{\text{PEC}}$s (2.48) respectively. The first column of both subfigures of Figure 4.11 corresponds to the fifth-order method, the second column corresponds to the fourth-order method used for error estimation, and the third column corresponds to their difference that is the actual error estimate. The other figures with three columns of bar graphs in this chapter also follow this scheme.

A similar figure to Figure 4.11b for the DP5(4)C$_{6(7)}$ pair (2.81) using the $\overline{\text{PEC}}$s (2.48) is given by Figure 4.12. Comparing Figure 4.11b with Figure 4.12, observe that for the DP5$(4)_{6(7)}$ pair (2.81) many PECs nearly vanish for both the error estimate and the fifth-order component. Also observe that for the higher-order error terms, some $\overline{\text{PEC}}$s (2.48) can have a magnitude greater than one, which for these components leads to errors greater than a simple truncation of the Taylor series of the local solution (2.38) would. This is undesirable for ERK pairs because it can cause a loss in efficiency, especially at coarse tolerances. In Figure 4.12 the $\overline{\text{PEC}}$s (2.48)) for the DP5(4)C$_{6(7)}$ pair (2.81) are much closer to the same size and avoid values greater than 1.0 for the individual PECs.

The characteristic numbers around the DP5$(4)_{6(7)}$ (2.79), DP5(4)S$_{6(7)}$ (2.80), and DP5(4)C$_{6(7)}$ (2.81) pairs can respectively be seen in Figures 4.13, 4.14, and 4.15. Although these figures are limited to 2D slices because seeing the full region of the space of free parameters around a pair would require visualizing a 4D space of free parameters, they give a good idea of what the choice of the $\hat{b}_7$ free parameter affects, along with helping visualize other tradeoffs in coefficient selection. It can also be seen that if the difference in



(a) The individual PECs (2.47).



(b) The individual $\overline{\text{PEC}}$s (2.48).

Figure 4.11: The individual PECs for the DP5(4)$_{6(7)}$ pair (2.79).

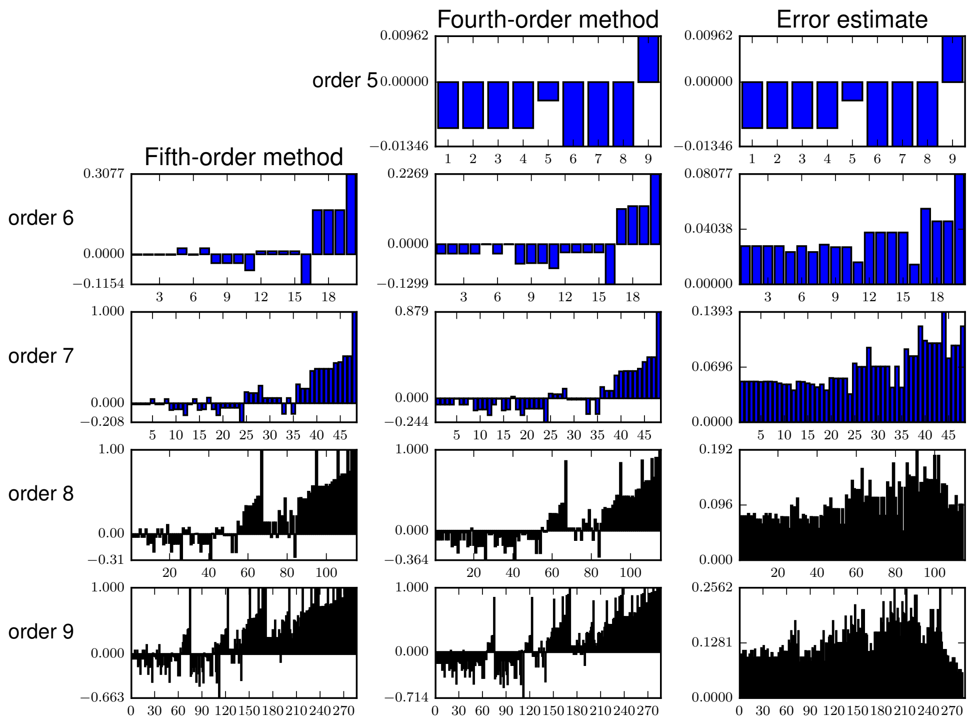


Figure 4.12: The individual $\overline{\text{PEC}}$s (2.47), for the DP5(4)C$_{6(7)}$ pair (2.81).

the length along the negative real axis of the stability regions is important, it is possible to get variety of values including precisely matched stability regions with $\hat{b}_7 \neq 0$. As well, there is only a small range where the stability region of the fourth-order method can be larger along the negative real axis than that of the fifth-order method. This is in contrast to Figure 4.15 that contains the DP5(4)C$_{6(7)}$ pair (2.81) where it can be seen that the stability regions are generally the same size with reasonable values of $\hat{b}_7$ and there is little control otherwise. However, for the nonstiff IVPs used for performance testing in this study, the sizes of stability regions do not seem to matter and they are not studied further.

With the benefit of visualizations such as those presented in this section as well as the use of a database such as `PostgreSQL` described in Section 4.9 to handle large datasets, it is now possible to make well-defined improvements on some classic 5(4)$_6$ ERK pairs. For instance, it can be seen in Figure 4.10, that there are other sets of free parameters with essentially the same leading error coefficient as the DP5(4)$_{6(7)}$ pair (2.79), but with much smaller higher-order error coefficients, which is further discussed in Section 5.9. In this study, the more general 5(4)$_{6(7)\text{C}(2)}$ family from Section 3.3.5 is used for 5(4)$_{6(7)}$ ERK pairs with the C(2) simplifying assumptions (2.44b) because the C(3) simplifying assumptions (2.44b) are not necessarily required.

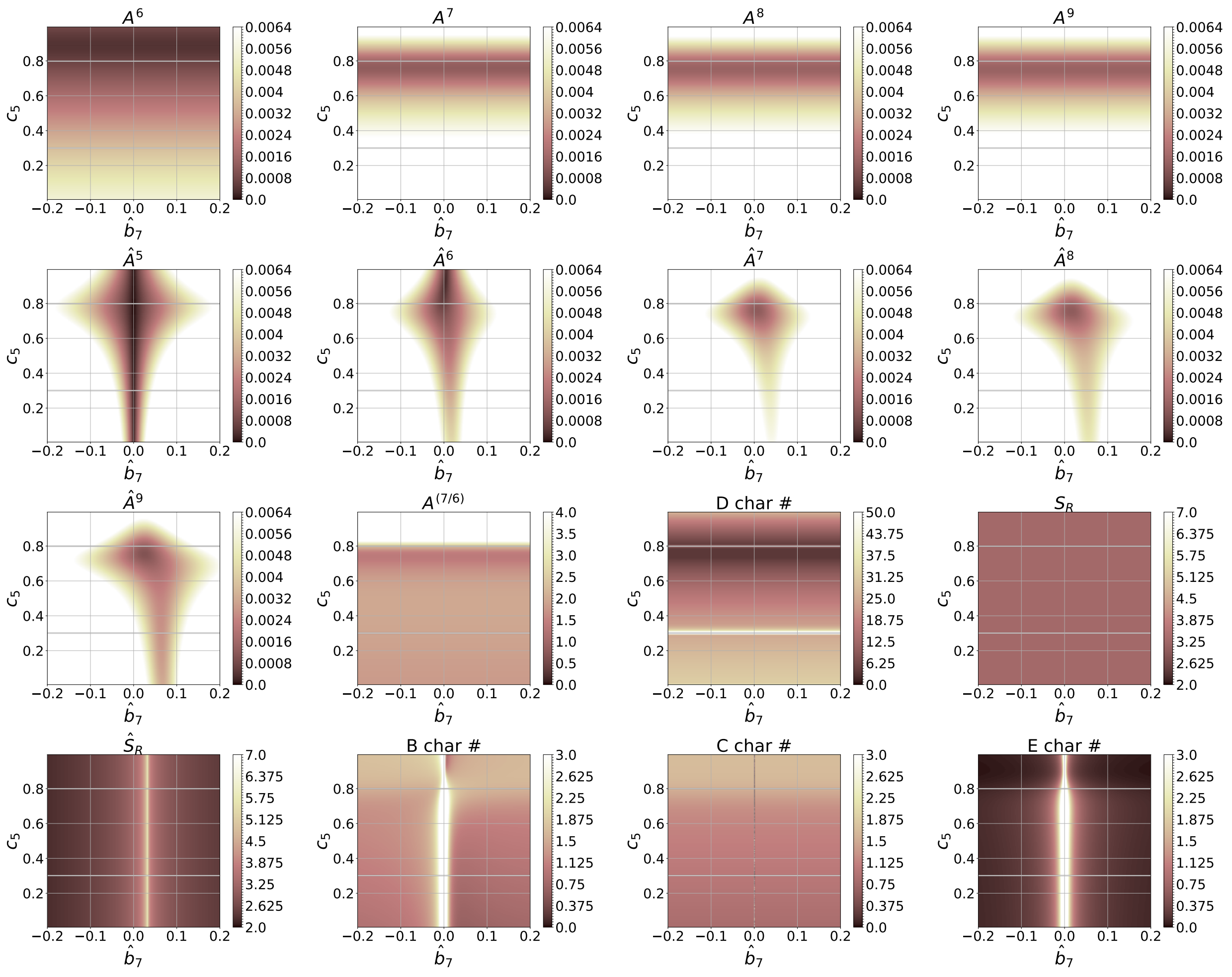


Figure 4.13: Characteristic numbers (2.56) in a search slice with $c_2 = 0.2$ and $c_4 = 0.8$ containing the DP5(4)$_{6(7)}$ pair (2.79).

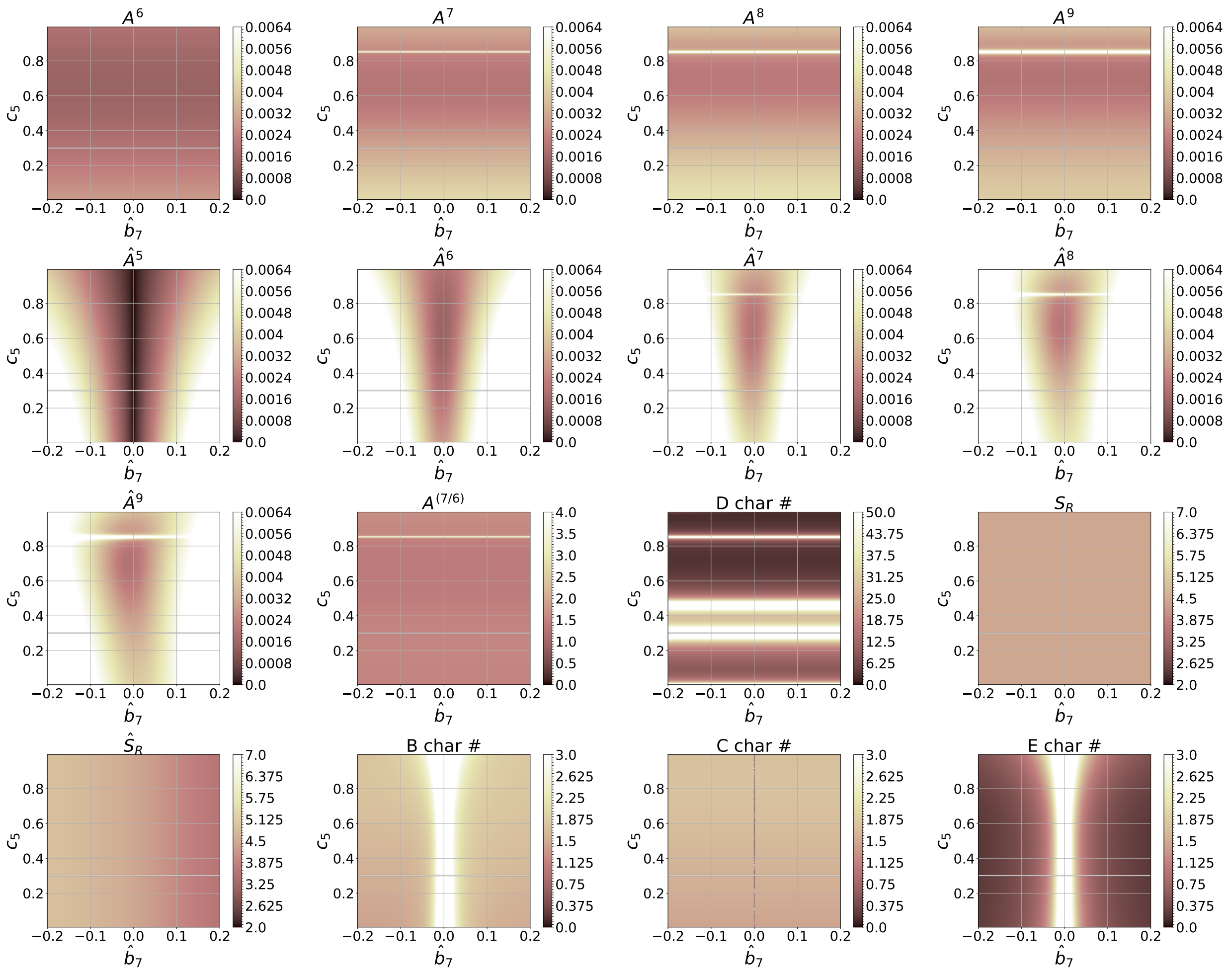


Figure 4.14: Characteristic numbers (2.56) in a search slice with $c_2 = 0.2222$ and $c_4 = 0.5556$ containing the DP5(4)S$_{6(7)}$ pair (2.80).

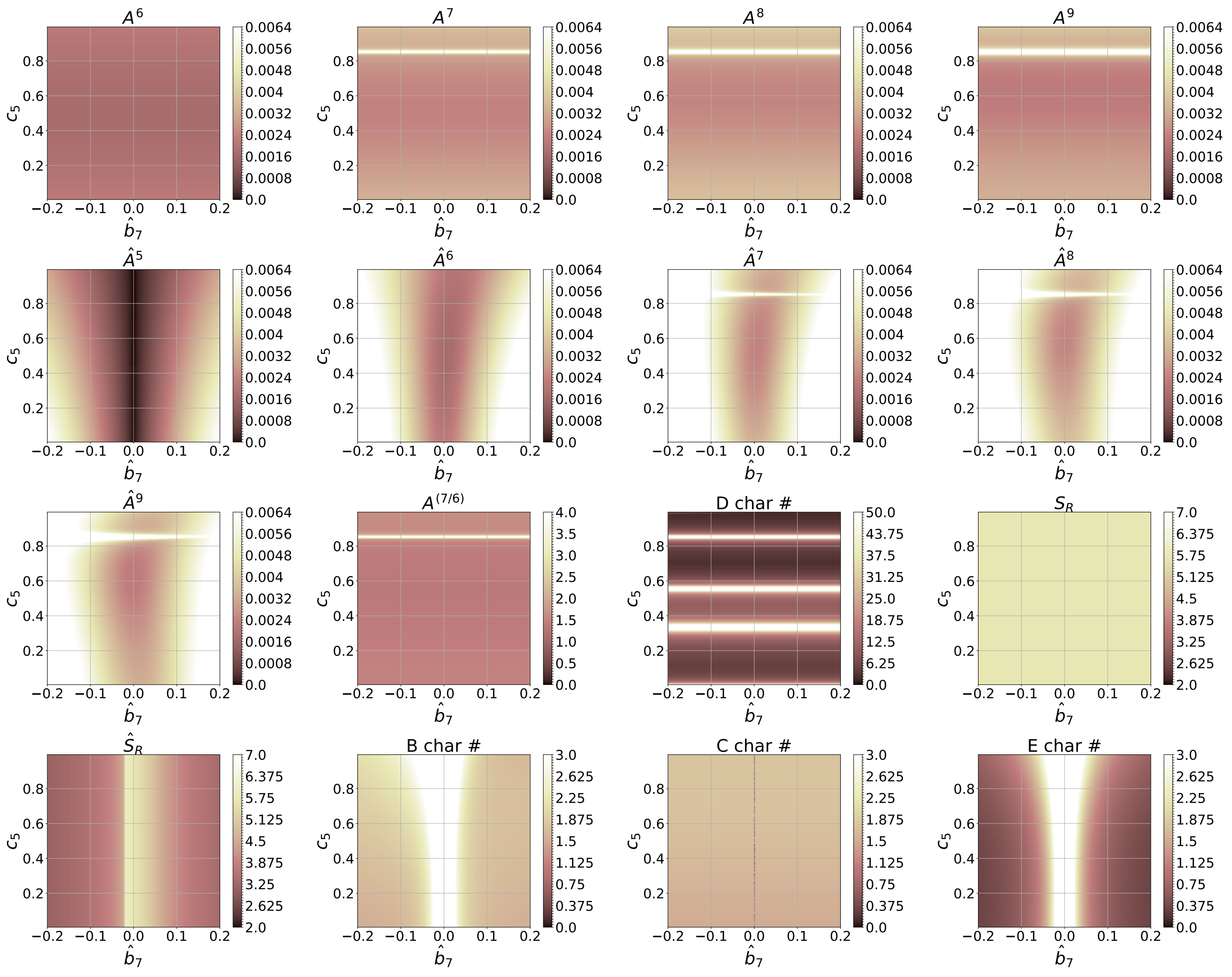


Figure 4.15: Characteristic numbers (2.56) in a search slice with $c_2 = 0.2222$ and $c_4 = 0.4667$ containing the DP5(4)C$_{6(7)}$ pair (2.81).

4.13 Search results for the $5(4)_{6(6)\mathrm{C}(2)}$ and $5(4)_{6(7)\mathrm{C}(2)}$ families

The families of $5(4)_6$ ERK pairs that use the C(2) simplifying assumptions (2.44b), i.e., the $5(4)_{6(6)\mathrm{C}(2)}$ and $5(4)_{6(7)\mathrm{C}(2)}$ families, are derived in Sections 3.3.4 and 3.3.5, respectively. Both of these families are more general than, but also include nearly all of the families used to derive the $5(4)_6$ ERK pairs that have been published in the literature. As already discussed in Section 3.3, the families of $5(4)_6$ ERK pairs derived in Section 3.3 do not require the C(3) simplifying assumptions (2.44b), and this allows the c_3 coefficient to be chosen as a free parameter rather than being dependent on the c_2 free parameter. Performance testing (not shown) indicates the differences in performance from foregoing the C(3) simplifying assumptions (2.44b) and allowing c_2 to be independent of c_3 turns out to be relatively small for most IVPs, although the best performance when solving at least some IVPs comes from when the C(3) simplifying assumptions (2.44b) are not satisfied.

4.13.1 Search results for the $5(4)_{6(6)\mathrm{C}(2)}$ family

The $5(4)_{6(6)\mathrm{C}(2)}$ family has $\hat{b}_6$ as a free parameter, which allows greater control over the error estimate than the family originally used to derive the RKF$4(5)_{6(6)}$ pair (2.78) does. Both Dormand and Prince [47], as well as Verner [187], derive families of $5(4)_6$ ERK pairs that contain an additional free parameter for the fourth-order method and both of these families are contained in the $5(4)_{6(6)\mathrm{C}(2)}$ family. Figure 4.17 shows the change in leading error coefficient A^6 with c_3 fixed at two different values and c_2 varying between 0.05 and 0.39 in increments of 0.01, which demonstrates the limited additional control over minimizing A^6 and further reflects the minor amount of control the additional free parameter gives over most other properties.

Generally, when $\hat{b}_6$ is too far away from b_6 the magnitude of the error coefficients for the fourth-order method always becomes excessive, i.e., > 0.0064. For most published ERK methods the magnitude of the components of $\mathbf{b}$ are limited to the interval $[-2, 2]$ allowing this to be a reasonable search interval in `OCSage` for $\hat{b}_6$. This can clearly be seen by inspecting Figures 4.18 and 4.19. It can also be seen in Figure 4.19 that sometimes only an extremely narrow interval for $\hat{b}_6$ gives acceptable characteristic numbers.

The procedure given here gives a general idea of the procedures used for the remainder of the families of $5(4)_6$ ERK pairs discussed in this study, although significant differences are noted where they occur. To find suitable candidate pairs, a broad search is first conducted that just examines the leading error coefficient A^6, although other properties were tried, restricting them to reasonable values does not appreciably reduce the search space. Each free parameter for the fifth-order method of a pair is sampled throughout the feasible range at enough points to resolve the features of the space of free parameters (this is checked graphically). The $5(4)_{6(6)\mathrm{C}(2)}$ family has c_2, c_3, c_5, c_6 as free parameters that are each sampled at 101 points. Because only A^6 needs to be calculated, this only takes about an hour on the computer used for this study and gives ranges of free parameters with $A^6 \leq 0.0015$ of $c_2 \in [0.0,\ 1.0]$, $c_3 \approx [0.22,\ 0.34]$, $c_5 \approx [0.32,\ 1.0]$, and $c_6 \approx [0.54,\ 1.0]$. Only these ranges are used for more time consuming searches involving more properties than just A^6 and this means only 3.75% of the points in the feasible space of free parameters for the fifth-order method of the $5(4)_6$ ERK pairs need to be further evaluated for the best properties. Preliminary searches such as this were often done in order to narrow down the search space before conducting expensive searches where many more properties were calculated.

The minimum leading error coefficient of the $5(4)_{6(6)\mathrm{C}(2)}$ family, which forgoes the C(3) simplifying assumptions (2.44b), when searched with a grid spacing of 0.01 for all free parameters is 0.00087088315076 at $c_2 = 0.23, c_3 = 0.28, c_5 = 0.8, c_6 = 0.88$, which is only slightly smaller than the family containing $\mathrm{RKF4(5)}_{6(6)}$ pair (2.78). A simplistic relation between the leading error coefficients and performance would indicate about 17% worse performance for even the best pair from the family of $5(4)_{6(6)}$ ERK pairs from Section 3.3.4, i.e., $\left(\frac{0.000871}{0.000399}\right)^{0.2} \approx 1.17$, in comparison to the $\mathrm{DP5(4)}_{6(7)}$ pair (2.79). However, it will be seen in the remainder of the thesis that due to the effect of other properties, methods with somewhat larger error coefficients can often have equivalent or even better performance solving many IVPs than methods with somewhat smaller error coefficients.

To further illustrate the choices made in coefficient selection for classic $5(4)_{6(6)}$ ERK pairs, some of the tradeoffs of different sets of the free parameters chosen for the $\mathrm{DP5(4)M}_{6(6)}$ pair (2.82) can be seen in Figure 4.19. Figures throughout this chapter, e.g., Figure 4.19, show there is not necessarily a best ERK pair and there are wide ranges of pairs with similar properties, but there do exist specific tradeoffs depending on what the most important properties of an ERK pair are considered to be. Many pairs with favourable combinations of properties were often observed to have components of the $\mathbf{c}$ vector that are closely spaced. However, closely spaced components of the $\mathbf{c}$ vector may lead to performance loss due to issues such as roundoff errors [47]. Although performance tests in Chapter 5 do not specifically show performance loss due to closely spaced components of the $\mathbf{c}$ vector, for real-world usage it is often good practice to be conservative.

4.13.2 Search results for the $5(4)_{6(7)\mathrm{C}(2)}$ family

The $5(4)_{6(7)\mathrm{C}(2)}$ family is more general than but contains the family originally used to construct the $\mathrm{DP5(4)}_{6(7)}$ pair (2.79), which is discussed in Section 4.12.2, and all of the same observations also apply. Searching the space of free parameters and performance testing done during the course of this study (not shown) indicate that $5(4)_{6(7)}$ ERK pairs with $A^6 \geq 0.0010$ do not provide any advantage over $5(4)_{6(6)}$ ERK pairs, which is at least in part because $5(4)_{6(7)}$ ERK pairs incur an extra stage evaluation when a step is rejected. Therefore, the most significant advantages to $5(4)_{6(7)}$ ERK pairs are either the smaller error coefficients than possible with $5(4)_{6(6)}$ ERK pairs or large stability regions, such as the $\mathrm{DP5(4)S}_{6(7)}$ pair (2.80) has.

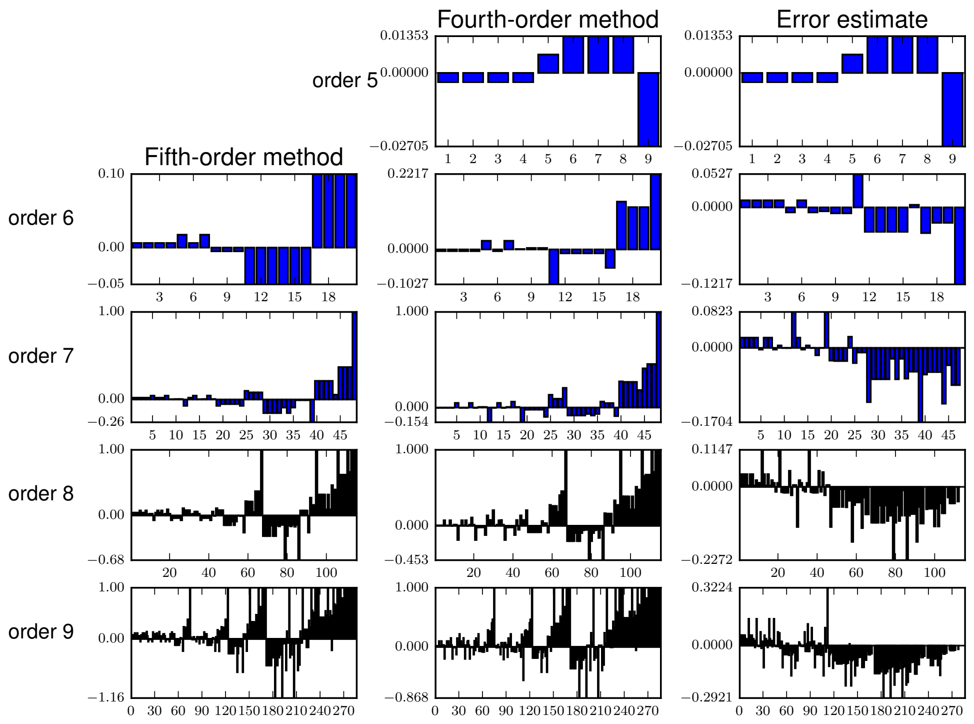


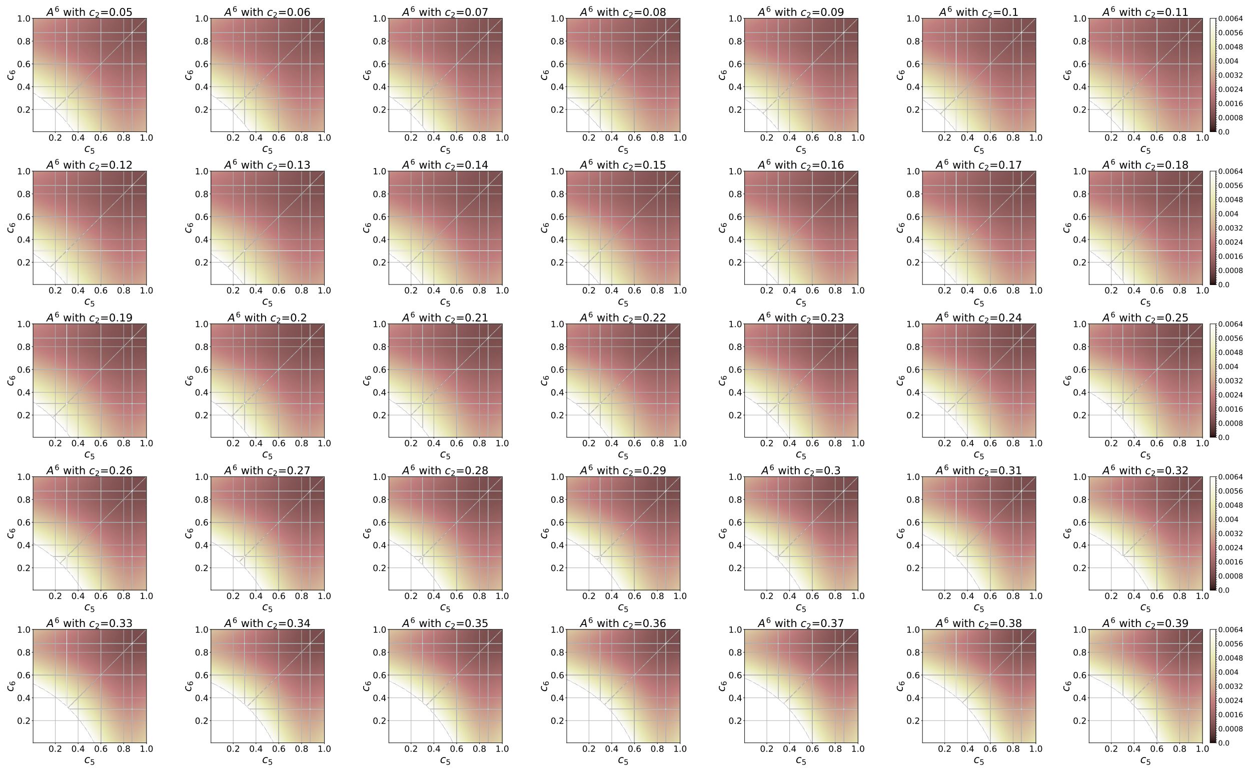
Figure 4.16: The individual $\overline{\text{PEC}}$s (2.47) for the CK4(5)$_{6(6)}$ pair (2.83).

4.14 Observations on families of $5(4)_{6(6)}$ ERK pairs without the C(2) simplifying assumptions

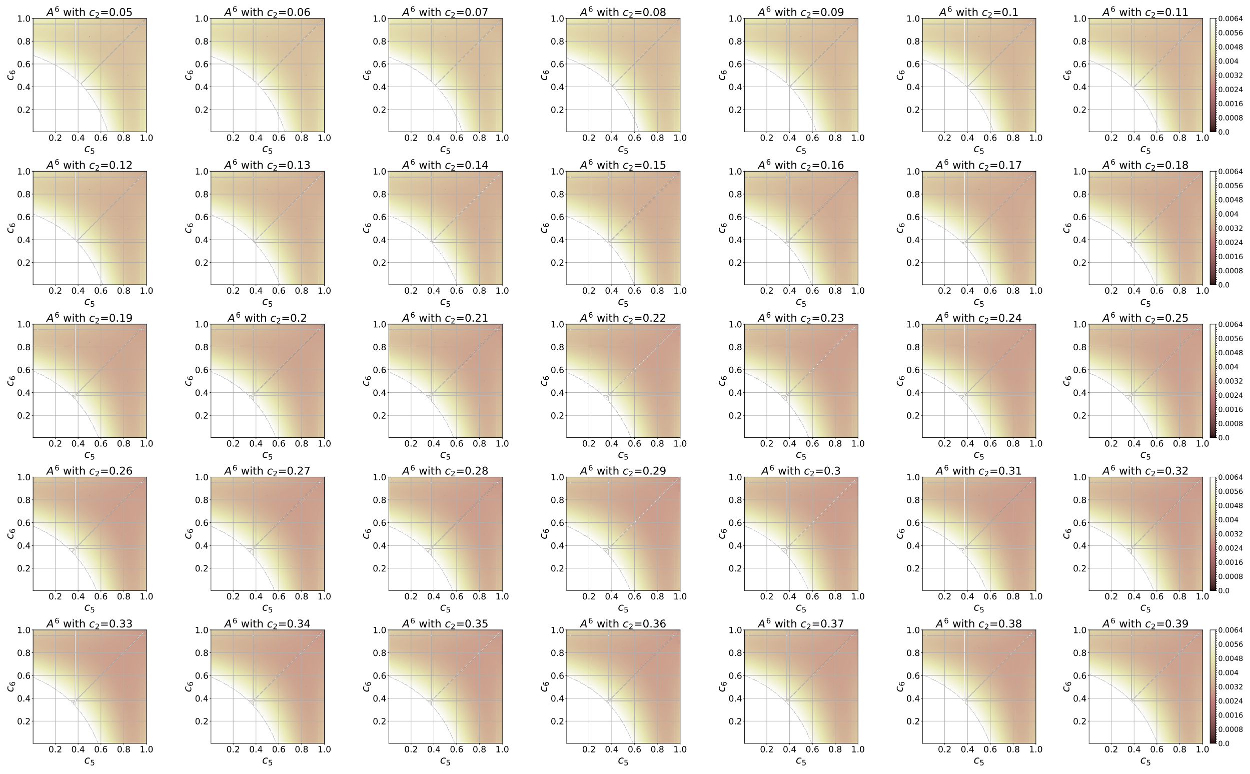
In this section, some representative search results are shown for the new families of $5(4)_6$ ERK pairs without the C(2) simplifying assumptions (2.44b), which are described in Section 3.5. However, the space of free parameters for $5(4)_{6(6)}$ ERK pairs without the C(2) simplifying assumptions (2.44b) is often not as smooth and can be more fragmented in comparison to families with the C(2) simplifying assumptions (2.44b). Therefore, a tool such as `OCSage` becomes essential to narrowing down the regions of the space of free parameters that might give the best candidates for efficient pairs.

Given that the ratio $\frac{\lambda}{\mu}$ itself is a free parameter for Cases V and VI, rather than individual values of λ and μ, the best strategy found for searching the families of $5(4)_{6(6)}$ ERK pair without the C(2) simplifying assumptions (2.44b) is to search λ in the interval $[-1.0, 1.0]$ with $\mu = 1$ and then μ in the interval $[-1.0, 1.0]$ with $\lambda = 1$. In this section, searching $\mu \in [-1.0, 1.0]$ with $\lambda = 1$ is shown, although searching $\lambda \in [-1.0, 1.0]$ with $\mu = 1$ is similar.

Because of the fewer free parameters than other families of $5(4)_6$ ERK pairs, pairs from Case V are generally not as efficient as pairs from Case VI or the $5(4)_{6(6)\text{C}(2)}$ and $5(4)_{6(7)\text{C}(2)}$ families, this limitation is

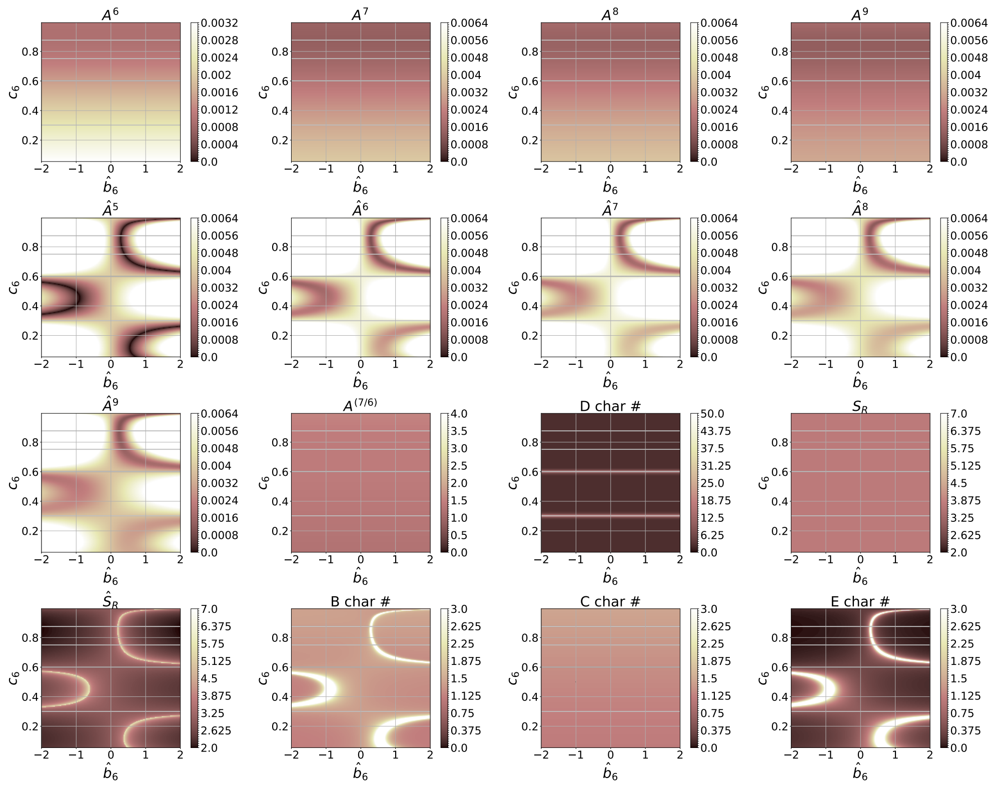


(a) Near the free parameters of the CK4(5)$_{6(6)}$ pair (2.83) with $c_3 = \frac{3}{10}$.



(b) Near the free parameters of the RKF4(5)$_{6(6)}$ pair (2.78) with $c_3 = \frac{3}{8}$.

Figure 4.17: A^6 for the $5(4)_{6(6)\mathrm{C}(2)}$ family with c_2 varying between 0.05 and 0.39.


(a) Near the free parameters for the CK4(5)$_{6(6)}$ pair (2.83) with $c_2 = 0.2$, $c_3 = 0.3$, and $c_5 = 1.0$.

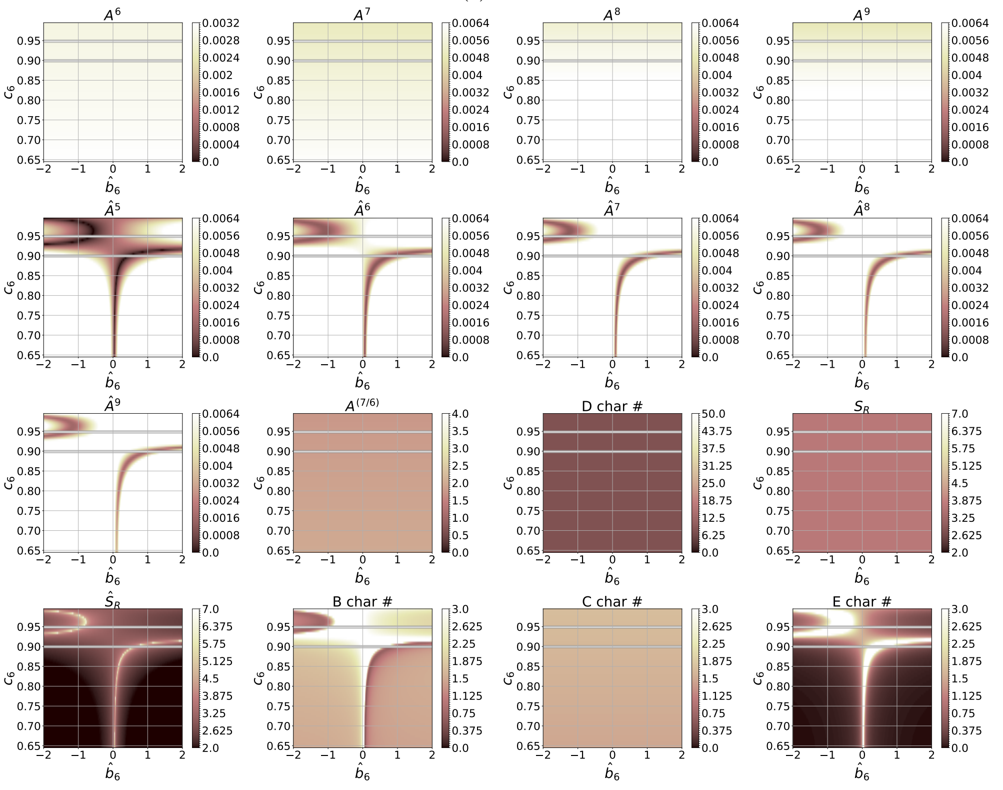

(b) Near the free parameters for the RKF4(5)$_{6(6)}$ pair (2.78) with $c_2 = 0.25$, $c_3 = 0.375$, and $c_5 = 1.0$.

Figure 4.18: Characteristic numbers (2.56) in search slices for the 5(4)$_{6(6)\mathrm{C}(2)}$ family. The free parameter $\hat{b}_6$ is on the x-axis and c_6 is on the y-axis.

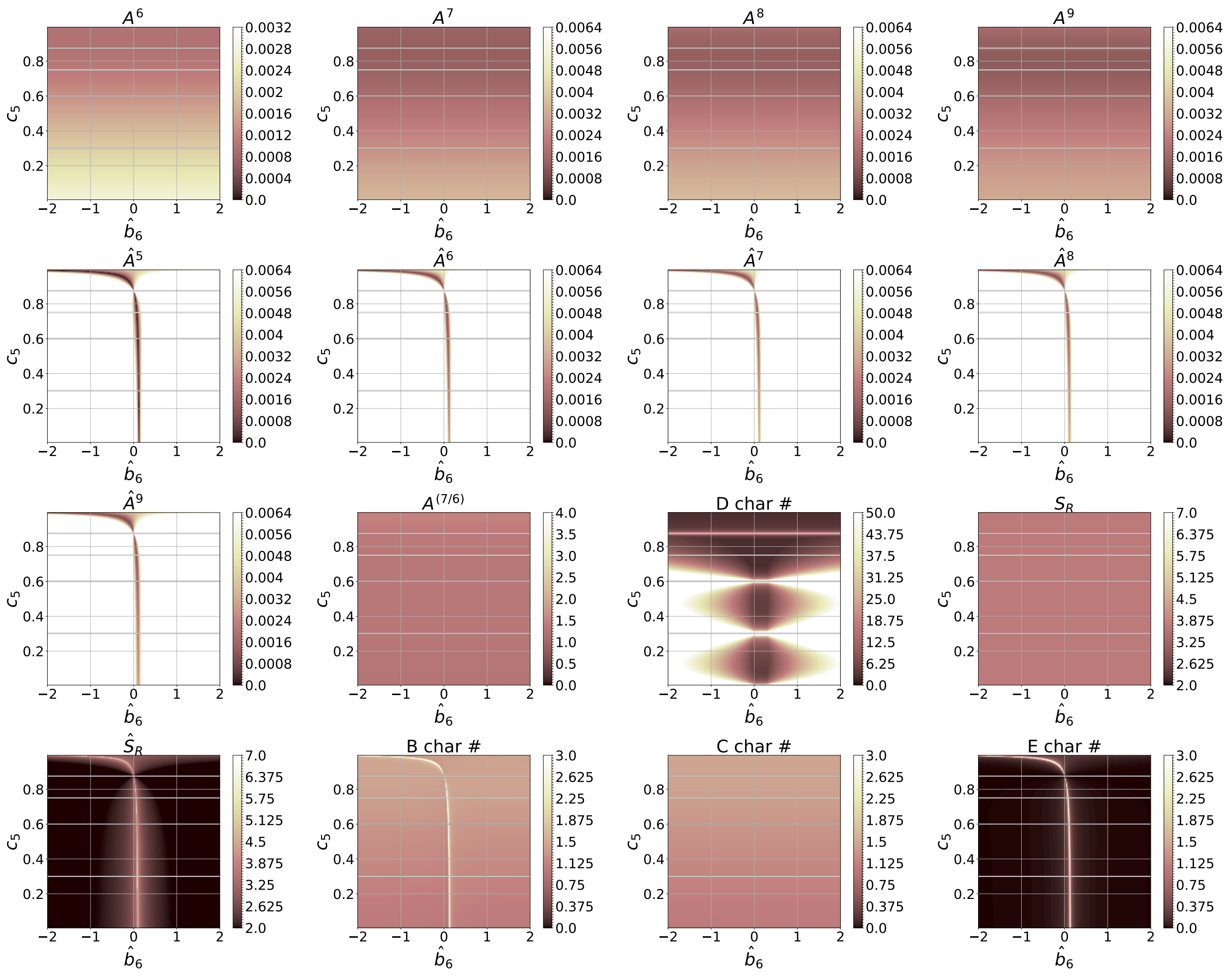
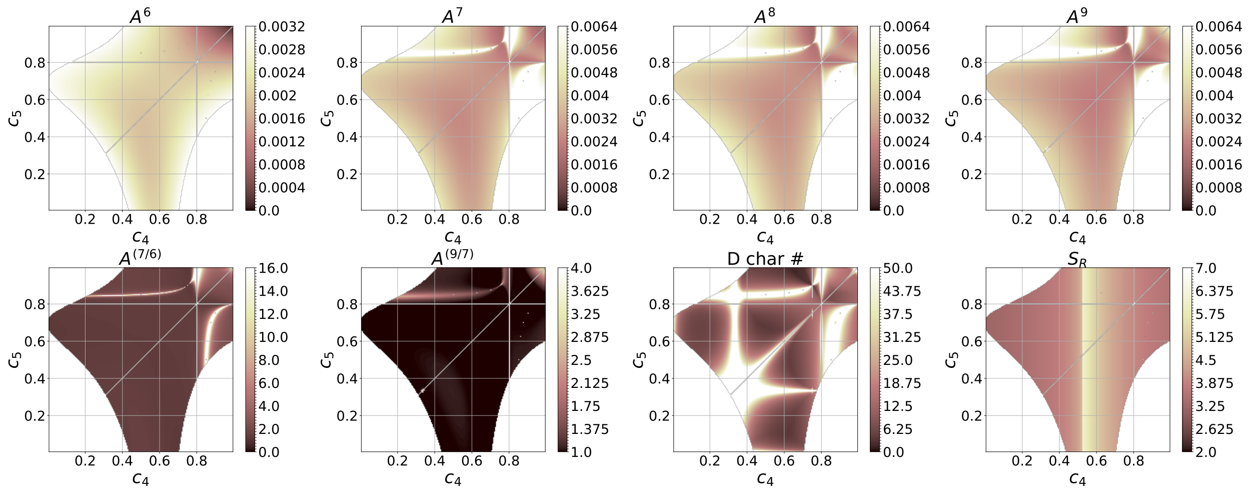
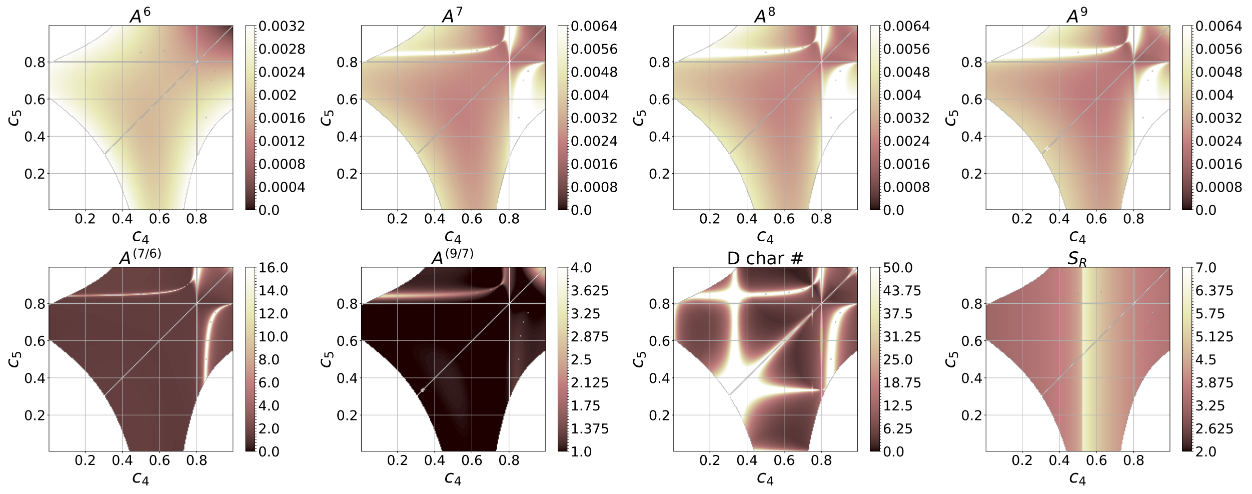


Figure 4.19: Characteristic numbers (2.56) in a search slice for the $5(4)_{6(6)\mathrm{C}(2)}$ family near the DP5(4)$\mathrm{M}_{6(6)}$ pair (2.82) with $c_2 = 0.2$, $c_3 = 0.3$, and $c_6 = 1.0$. The free parameter $\hat{b}_6$ is on the x-axis and c_6 is on the y-axis.



(a) With $c_2 = 0.1111$ and not satisfying the C(3) simplifying assumptions (2.44b).



(b) With $c_2 = 0.2222$ and satisfying the C(3) simplifying assumptions (2.44b).

Figure 4.20: Slices showing characteristic numbers of the fifth-order method of the $5(4)_{6(7)\mathrm{C}(2)}$ family of $5(4)_{6(7)}$ ERK pairs from Section 3.3.5 at $c_3 = 0.3333$.

discussed further in Chapter 5. This lack of competitiveness is despite that both subcases of Case V have minimum leading error coefficients A^6 of only slightly greater than 0.0010. In addition, many pairs found in preliminary searches of Case V with a relatively small leading error coefficients, i.e., $A^6 < 0.0011$, have issues such as extremely large Butcher tableau coefficients (> 1000) or values of the $\overline{\text{PEC}}$s greater than 1.0, both of which are likely cause performance loss.

Case VI with $c_6 \neq 1$

As seen in Section 3.5.1, all $5(4)_{6(6)}$ ERK pairs in Case VI require $\hat{b}_6 = 0$. This means that there is no parameter to "tune" the fourth-order component and hence the error estimate of an embedded pair. Based on the performance data in Section 5.12, the biggest disadvantage to Case VI is that some IVPs do benefit from a large value $|\hat{b}_6 - b_6|$. However, searches into the space of free parameters of Case VI show the family has trouble achieving this in combination with other desirable properties. Other than the issue just described finding large values of $|\hat{b}_6 - b_6|$, many pairs from Case VI are competitive with the best $5(4)_{6(6)}$ ERK pairs with the C(2) simplifying assumptions, including CK4$(5)_{6(6)}$ (2.83). The space of free parameters in comparison to the leading error coefficient A^6 for Cases VIa and VIb is shown in Figures 4.21 and 4.22, respectively. Figure 4.21 shows the more complicated nature of the space of free parameters for Case VIa that comes from more complicated expression such as (3.63). The free parameters chosen for examination in Figures 4.21 and 4.22, respectively, include the pairs presented in Section 5.14.

4.15 Properties of the $5(4)_{6(7)}$ ERK pairs from the complete solution without the C(2) simplifying assumptions

In this section, the families of the $5(4)_{6(7)}$ ERK pair from Cases I–VI, which are summarized in Table 3.4, are discussed. The problem with Cases II, III, IV having the fourth-order component satisfying at least one fifth-order condition is already mentioned in Chapter 3 and in the extensive experimentation done for the study in this thesis (not shown) pairs from these families perform unexpectedly poorly given that some pairs can have leading error coefficients as small as $A^6 \approx 0.0007$. In addition, searches done with `OCSage` indicate that although Case I does not have this problem with the fourth-order component satisfying at least one fifth-order condition, Case I always has a leading error coefficient $A^6 > 0.0022$. This is a minimal A^6 almost triple that of any other family of $5(4)_6$ ERK pairs from Chapter 3. Therefore, given the excellent performance of $5(4)_6$ ERK pairs from other families, this means Case I cannot currently be recommended either, unless specific drawbacks of other families are found that Case I does not have.

4.15.1 Tsitouras' $5(4)_{6(7)}$ ERK pairs without the C(2) simplifying assumptions

Even though the pairs published by Tsitouras [182, 183], that are already mentioned in Section 3.1, were claimed to be $5(4)_{6(7)}$ ERK pairs, the coefficients are published as 64-bit floating-point numbers rather than

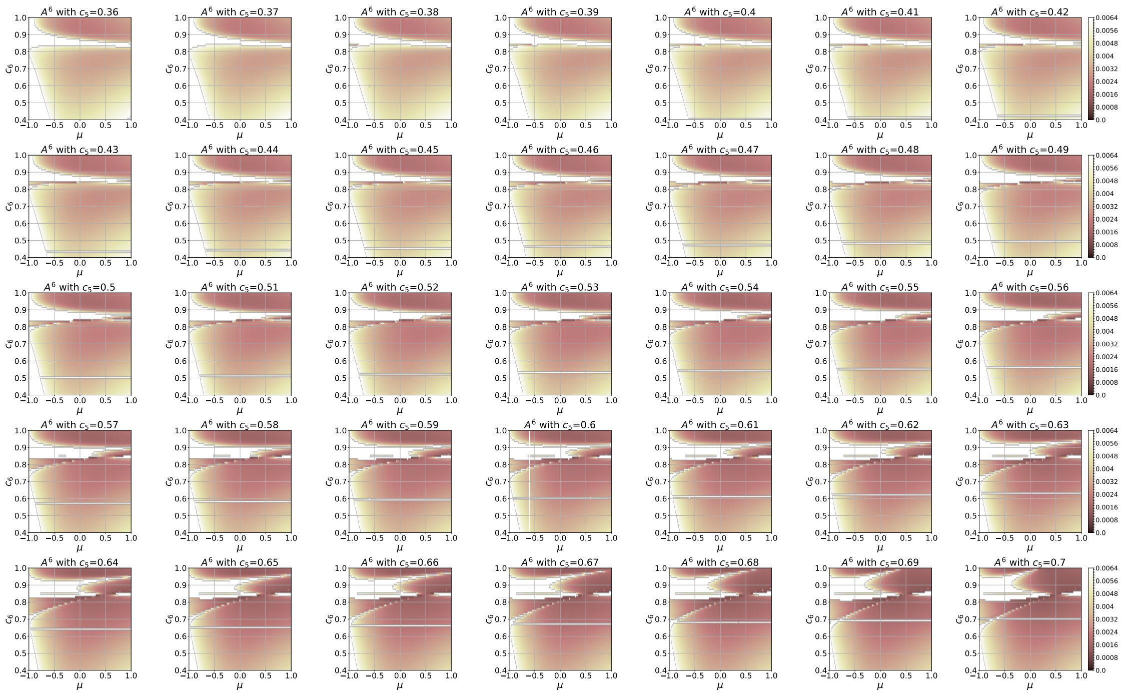


Figure 4.21: Values of A^6 of the family corresponding to Case VIa from Section 3.5.1 with $c_3 = 0.05$ and $\lambda = 1$.

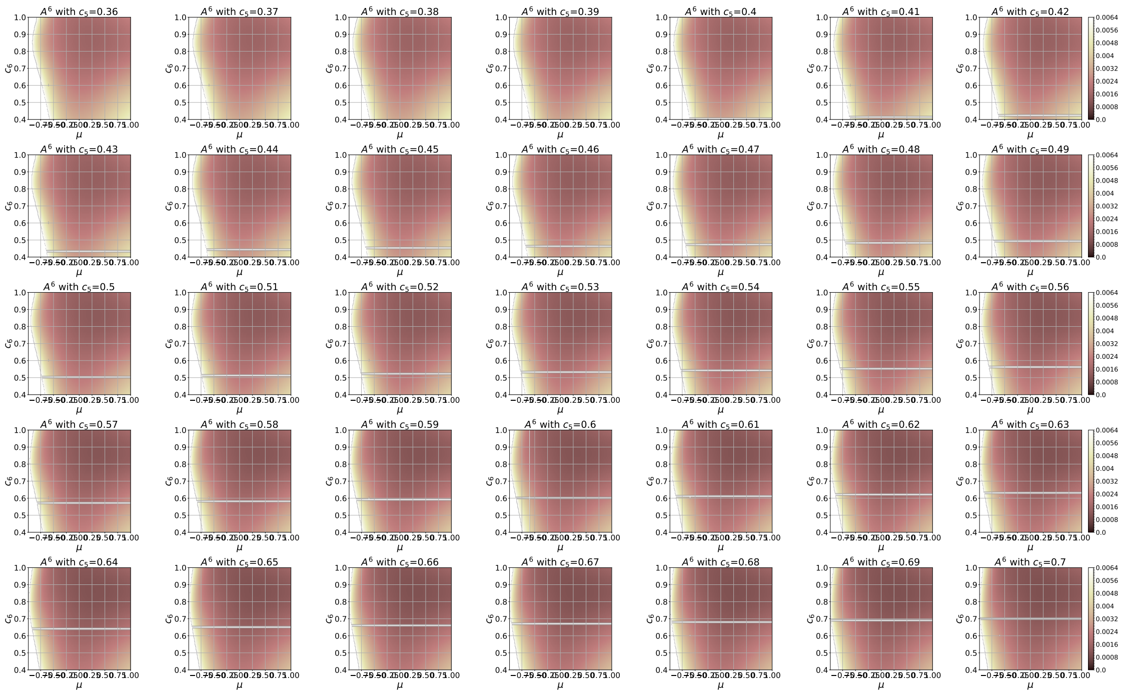


Figure 4.22: Values of A^6 of the family corresponding to Case VIb from Section 3.5.1 with $c_3 = 0.12$ and $\lambda = 1$.

exact rational numbers. This is because they were found using numerical optimization to solve the order conditions, rather than an exact symbolic solution of the order conditions, such as that demonstrated in Section 2.7 and Chapter 3. However, it has already been mentioned in Section 3.1 that Tsitouras' published pairs [182, 183] have coefficients that do not satisfy the appropriate order conditions properly at all, likely due to a transcription error. Peter Stone[21] has published (on his personal website) coefficients for Tsitouras' pair from 2009 [182][22] and corrected coefficients for Tsitouras' pair from 2011 [183][23] using higher-precision arithmetic in MAPLE to calculate the coefficients to 85 digits (about 283 bits of precision).

Within SAGE, both standard 64-bit floating-point numbers and the arbitrary precision real number data type, i.e., `sage.rings.real_mpfr.RealNumber`,[24] allow exact conversion to rational numbers through the `as_integer_ratio` and `exact_rational` methods, respectively. Converting floating point numbers to an exact representation in terms of rational numbers allows determining information like the degree they satisfy order conditions (this calculation is better done in exact or arbitrary-precision arithmetic because it is never actually carried out while using an RK method) by using either exact arithmetic, or arbitrary precision arithmetic if exact arithmetic turns out to be unfeasible. When using 64-bit floating-point arithmetic, the residuals from substitution of Tsitouras' Butcher tableau coefficients into the order conditions are no more than approximately an order of magnitude greater than 64-bit floating point unit roundoff, i.e., $\approx 1.1 \times 10^{-16}$, and the magnitudes of these residuals do not change substantially when 512-bit precision arithmetic is used instead. This can be seen in the summary of the residuals for the "fifth-order" components of Tsitouras' pairs in Table 4.3, along with two classic ERK pairs for comparison. Notice that Stone's corrected and higher-precision coefficients give extremely small residuals for Tsitouras' pair from 2009 [182] when computing them with 512-bit precision arithmetic, but not for Tsitouras' pair from 2011 [183]. Results from using both standard 64-bit floating-point and 512-bit precision arithmetic are shown in Table 4.3 to demonstrate the utility of using arbitrary-precision arithmetic to study numerical methods, even if the numerical methods themselves are only ever implemented with 64-bit floating-point numbers.

Analogous residuals are given for the "fourth-order" components of Tsitouras' pairs in Table 4.4. Notice that the "fourth-order" component of Tsitouras' pair published in 2011 [183] has residuals from the substitution of the coefficients into the appropriate order conditions required for fourth-order that are much larger than any other published pairs, i.e., as large as -0.97, and cannot be considered to even approximately satisfy the necessary order conditions. Because the residuals are close to 64-bit floating-point unit roundoff for Tsitouras' pair published in 2009 [182] and Peter Stone has given corrected coefficients for Tsitouras' pair published in 2011 that give residuals similar in magnitude to 64-bit floating-point unit roundoff, the issue with extremely large residuals for the fourth-order method from Tsitouras' pair in his 2011 paper [183] is obviously just a

[21]Peter Stone was a mathematics professor at Vancouver Island University who has since retired. He has not published his calculations related to Tsitouras' methods (and many others) beyond his personal website. The author replicated similar calculations to both correct and find Tsitouras' pairs to higher precision, but Peter Stone's calculations were done give much lower residuals.

[22]`http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK5/RKcoeff5m_1.pdf`

[23]`http://www.peterstone.name/Maplepgs/Maple/nmthds/RKcoeff/Runge_Kutta_schemes/RK5/RKcoeff5n_1.pdf`

[24]`http://doc.sagemath.org/html/en/reference/rings_numerical/sage/rings/real_mpfr.html`

simple error transcribing the coefficients for publication.

It is also possible to find the particular solution case of the complete solution from Section 3.4 that Tsitouras' pairs are closest to because they also have distinct components of the $\mathbf{c}$ vector, which is an assumption of this study. By substituting the coefficients of Tsitouras' pair published in 2009 [182] into the coefficient matrix of (3.28) and using SVD (singular value decomposition) to find the nullspace, this gives multipliers of approximately $\lambda : \mu : \nu : \omega = -0.7298 : 0.0553 : 0.0368 : -0.6804$. A corresponding operation for Tsitouras' pair published in 2011 [182] gives multipliers of approximately $\lambda : \mu : \nu : \omega = -0.0392 : -0.6880 : 0.1019 : -0.7175$. Therefore, because Cases I–IV from Chapter 3 require one of the multipliers to be zero, the "fifth-order" method of Tsitouras' pairs approximate Case V of the complete solution of six-stage fifth-order ERK order conditions. In addition, for Tsitouras' pairs the condition that must vanish for $5(4)_{6(7)}$ ERK pairs, i.e., (3.72), evaluates to much larger than 64-bit floating-point unit roundoff (in the range of 10^{-6} to 10^{-4}, rather than vanishing). Therefore, despite the small residuals from substituting the Butcher tableau coefficients into the order conditions, this shows that Tsitouras' pairs do not correspond to $5(4)_6$ ERK pairs in exact arithmetic. In Chapter 5 it is shown that Tsitouras' pairs (using Stone's corrected coefficients) perform similarly to many classic $5(4)_6$ ERK pairs but sometimes have some unexpected behaviour when solving some IVPs to high accuracy. Therefore, the main disadvantages of using numerical optimization to solve the order conditions is not only that exploration of the space of parameters is not as easy or efficient, but also that the RK methods found may not be part of families that are able to satisfy the order condition in exact arithmetic.

Many published numerical methods, including many based on Runge–Kutta methods, often use numerical optimization to approximately solve the order conditions or only use 64-bit floating point arithmetic to compute and publish Butcher tableau coefficients. Several examples are found in the references [34, 94]. However, the drawback of using numerical optimization to solve order conditions is that methods may be constructed that appear to have compelling properties, but that are actually impossible in exact arithmetic. Tsitouras' pairs [182, 183] are an excellent published example of this. Although solving the order conditions numerically using 64-bit arithmetic may be suitable for many cases, the results of this thesis indicate that a CAS should be used to find symbolic solutions to explore the families and find the best regions in space of free parameters. If symbolic solution of the relevant conditions is not possible, then arithmetic more precise than 64-bit floating point should be used and the residuals from the conditions used to construct the numerical method should be checked carefully.

Table 4.3: The residuals when the coefficients of the fifth-order component of $5(4)_6$ ERK pairs are substituted into the order conditions. The rows indicating 53-bits use standard 64-bit floating point arithmetic with 53 bits of precision. The rows indicating 512-bits use SAGE arbitrary precision arithmetic with 512 bits of precision.

Order condition	max(order 1)	max(order 2)	max(order 3)	max(order 4)	max(order 5)	max(order 6)
53-bit res $CK4(5)_{6(6)}$	0.000000	0.000000	2.775558×10^{-17}	1.387779×10^{-17}	1.387779×10^{-17}	4.166667×10^{-4}
512-bit res $CK4(5)_{6(6)}$	0.000000	0.000000	$1.864585 \times 10^{-155}$	$4.661463 \times 10^{-156}$	$4.661463 \times 10^{-156}$	4.166667×10^{-4}
53-bit res $DP5(4)_{6(7)}$	2.220446×10^{-16}	1.110223×10^{-16}	1.387779×10^{-16}	4.857226×10^{-17}	2.636780×10^{-16}	2.777778×10^{-4}
512-bit res $DP5(4)_{6(7)}$	0.000000	0.000000	$1.118751 \times 10^{-154}$	$3.729170 \times 10^{-155}$	$2.260810 \times 10^{-154}$	2.777778×10^{-4}
53-bit res Tsit(2009)	1.110223×10^{-15}	4.440892×10^{-16}	2.220446×10^{-16}	1.942890×10^{-16}	8.326673×10^{-17}	2.106951×10^{-4}
512-bit res Tsit(2009)	9.853229×10^{-16}	4.406602×10^{-16}	2.263352×10^{-16}	1.502776×10^{-16}	9.518337×10^{-17}	2.106951×10^{-4}
53-bit res Tsit(2009) (Stone)	0.000000	0.000000	2.775558×10^{-17}	2.081668×10^{-17}	6.938894×10^{-17}	2.106974×10^{-4}
512-bit res Tsit(2009) (Stone)	2.412918×10^{-86}	4.881948×10^{-86}	1.588092×10^{-86}	3.904451×10^{-87}	4.085032×10^{-87}	2.106974×10^{-4}
53-bit res Tsit(2011)	4.440892×10^{-16}	6.661338×10^{-16}	2.026157×10^{-15}	1.443290×10^{-15}	2.237793×10^{-15}	7.484703×10^{-5}
512-bit res Tsit(2011)	1.474515×10^{-16}	2.381915×10^{-16}	1.169977×10^{-15}	1.118861×10^{-15}	5.892587×10^{-16}	7.484703×10^{-5}
53-bit res Tsit(2011) (Stone)	2.220446×10^{-16}	0.000000	6.383782×10^{-16}	4.302114×10^{-16}	2.345346×10^{-15}	7.484703×10^{-5}
512-bit res Tsit(2011) (Stone)	2.081668×10^{-19}	3.687615×10^{-17}	2.958858×10^{-17}	1.836509×10^{-17}	1.360600×10^{-17}	7.484703×10^{-5}

Table 4.4: The residuals when the coefficients of the fourth-order component of $5(4)_6$ ERK pairs are substituted into the order conditions. The rows indicating 53-bits use standard 64-bit floating point arithmetic with 53 bits of precision. The rows indicating 512-bits use SAGE arbitrary precision arithmetic with 512 bits of precision.

Order condition	max(order 1 emb)	max(order 2 emb)	max(order 3 emb)	max(order 4 emb)	max(order 5 emb)
53-bit res $CK4(5)_{6(6)}$	0.000000	5.551115×10^{-17}	2.775558×10^{-17}	0.000000	3.381348×10^{-4}
512-bit res $CK4(5)_{6(6)}$	0.000000	0.000000	0.000000	$4.661463 \times 10^{-156}$	3.381348×10^{-4}
53-bit res $DP5(4)_{6(7)}$	1.110223×10^{-16}	0.000000	5.551115×10^{-17}	6.938894×10^{-17}	8.083333×10^{-4}
512-bit res $DP5(4)_{6(7)}$	$7.458341 \times 10^{-155}$	0.000000	$9.322926 \times 10^{-155}$	$3.729170 \times 10^{-155}$	8.083333×10^{-4}
53-bit res Tsit(2009)	0.000000	1.110223×10^{-16}	5.551115×10^{-17}	1.110223×10^{-16}	6.558704×10^{-4}
512-bit res Tsit(2009)	3.469447×10^{-17}	7.480331×10^{-17}	5.563652×10^{-17}	6.709570×10^{-17}	6.558704×10^{-4}
53-bit res Tsit(2009) (Stone)	0.000000	0.000000	2.775558×10^{-17}	2.775558×10^{-17}	6.558709×10^{-4}
512-bit res Tsit(2009) (Stone)	1.387779×10^{-18}	1.387779×10^{-18}	6.938894×10^{-19}	6.938894×10^{-19}	6.558709×10^{-4}
53-bit res Tsit(2011)	9.696970×10^{-1}	4.696970×10^{-1}	1.515152×10^{-1}	1.098485×10^{-1}	4.228466×10^{-2}
512-bit res Tsit(2011)	9.696970×10^{-1}	4.696970×10^{-1}	1.515152×10^{-1}	1.098485×10^{-1}	4.228466×10^{-2}
53-bit res Tsit(2011) (Stone)	2.220446×10^{-16}	2.220446×10^{-16}	4.996004×10^{-16}	9.714451×10^{-16}	8.665277×10^{-4}
512-bit res Tsit(2011) (Stone)	8.636234×10^{-85}	3.372745×10^{-17}	2.672661×10^{-17}	1.701186×10^{-17}	8.665277×10^{-4}

Chapter 5

The `pythODE` package for studying large amounts of performance data from IVP methods

"From the epistemological point of view this is well-known: a single counterexample falsifes a theory, and a single example can adequately demonstrate the proper interpretation of a theory; in contrast, a large number of carefully planned and precisely controlled experiments are needed in order to corroborate even a modest empirical claim."
(G. Söderlind and L. Wang 2006)

This chapter describes the `pythODE` (Python ODE solver) software package that uses RK methods (2.33) to solve IVPs.[1] Performance data from `pythODE` is used in combination with `OCSage` to systematically construct efficient pairs for IVPs from celestial mechanics. IVPs from celestial mechanics are chosen because there are a sufficient number and variety in published test sets, and it is demonstrated in this chapter that there can be well-defined optimal pairs for this class of IVPs.

Although not strictly necessary for extremely basic usage, the `pythODE` package is generally used with a `PostgreSQL` database [92][2] that assists setting up numerical experiments, stores the numerical solutions to the IVPs as well as any other data associated with the numerical solution, manages network transparent and concurrent storage for the parallel solution of IVPs across many computers and CPU cores, and readily allows further processing of the data from the numerical experiments for study and visualization. The `pythODE` package can also directly utilize the data stored in the `PostgreSQL` database from the searches of the space of free parameters conducted by `OCSage`, such as those described in Section 4.9, by adding performance data for a specific IVP to each row of a database table corresponding to an ERK method. Overall, this study would have been much more difficult without the robustness and functionality typical of a proper database system. In particular, using a proper database helped immensely with allowing reasonably quick and versatile exploration and transformation of performance data in ways not anticipated when the performance data was generated. This includes using the robust concurrency of a *relational database management system* (RDBMS) to easily query preliminary results of experiments that are not finished running without ever having to specifically build this feature into `pythODE` (this would be much harder to incorporate into code managing basic data files), easily check-pointing long-running experiments to protect against loss in case of interruption, and the

[1]The `pythODE` package is designed so that other types of DEs and numerical methods for DEs can be incorporated in the future.
[2]`https://www.postgresql.org/`

use of the declarative language SQL (that can plan and optimize queries in a way that would be onerous to do for each individual query in an imperative language) often makes querying and processing data faster and more robust than direct use of imperative languages such as PYTHON. These tasks are all non-trivial enough that they would have been difficult without some type of proper database, and it cannot be overemphasized that using a proper database is one of the key factors making this study possible as a single-thesis study by one person. Although using a proper database, such as `PostgreSQL`, rather than ad-hoc approaches for scientific studies and simulations is becoming increasingly common [64, 75, 76, 89, 100],[3] databases seem to be rarely used for studies into numerical methods themselves. However, based on this study, it is strongly recommended that future studies incorporating numerical simulations use a proper, robust, and well-tested database such as `PostgreSQL`. One of the important contributions of this study is a demonstration of how to use modern software technology to enhance studies of numerical methods because it is not always obvious how to leverage even widely used software technology, such as dynamic languages like PYTHON and RDBMSs like `PostgreSQL`, for specific scientific and research purposes. This was clearly shown by many of the functional components of `OCSage` described in Section 4.3, where they are based on published work that does not appear to have been utilized further after initial publication and probably required integration into a software system for practical utility. How to do this, and its utility for research, was not at all obvious based on the common usage patterns and other guidelines that have been published for PYTHON and `PostgreSQL`.

Development of the `pythODE` package started in 2010 to support the Master's thesis work [103] of the author. Since that time, the `pythODE` package has been used by the author to support work that resulted in two peer-reviewed publications [104, 105]. However, the `pythODE` package has been under continuous development since its conception, and the version used for this thesis (`pythODE` version `pythODE.ak2018.v1`) is different than, although a direct development of, the versions of the `pythODE` package used in these other publications. Once properly set up, the use of a `PostgreSQL` database makes the current version of `pythODE` substantially more capable and easier to work with than earlier versions that tried to implement directly in PYTHON functionality that is more appropriately implemented with a relational database. Like `OCSage`, it can be expected that with use in other studies subsequent to this study, the `pythODE` package will continue to evolve substantially.

Unlike `OCSage`, the `pythODE` package also saw work by another author, Adam Preuss in 2011 and 2012, who later developed a similar package in C++ for his Master's work [135]. Most of the few thousand lines of code that Adam wrote, to manage networking (for cluster computing) as well as designing some important data structures, were largely superseded when `pythODE` started using a `PostgreSQL` database in 2016. However, his valuable contributions substantially shaped the development leading to the version of `pythODE` presented in this thesis.

The `pythODE` package is also a direct descendant of the `odeToJava` package that is a subject of the peer-reviewed paper by the author and the advisor (Dr. Raymond J. Spiteri) of this thesis [106]. The `odeToJava`

[3] http://www.datacarpentry.org/

package did not originate with the author but was actually started in the early 2000s by Murray Patterson at Acadia University [131], with the version published being based on further development by Jesse Rusak at Dalhousie University. However, the author of this thesis was solely responsible for the initial preparation of the manuscript [106], and the `odeToJava` package required substantial additional software development work in order to be a suitable subject for a peer-reviewed publication. This process of releasing and publishing on the `odeToJava` package contributed greatly to the design of `pythODE` as a software package.

5.1 Tradeoffs in choosing a suitable platform for numerical computing

One of the important issues encountered when testing numerical algorithms on a broad scale is managing the many different variations of code components, algorithms, coefficients, heuristics, and other parameters for the individual algorithms and the problems being solved [17, 149]. Moreover, to effectively evaluate the test results, there needs to be a flexible and efficient approach for data analysis and presentation. In comparison to many modern programming languages, although the traditional numerical computing language FORTRAN continues to be highly effective for implementing specific numerical formulae as well as standard array and linear algebra operations, FORTRAN is not an especially effective platform for the more complex software engineering required to integrate many diverse software components and libraries together. This is because without robust and contemporary programming language features, e.g., those that support object-oriented programming, software to manage the combinations of program input, algorithm components, and output data can quickly become unnecessarily complex and brittle [8]. Modern features supporting techniques such as object-oriented programming are now incorporated into recent versions of traditional languages for numerical computing, such as in more recent versions of FORTRAN and MATLAB [38, 141]. However, for more complex software development projects, these modern features in FORTRAN and MATLAB are not as well-integrated, and the software ecosystem is not as mature because it has not seen widespread use beyond scientific computing. By contrast, popular object-oriented languages used for scientific computing such as C++ or PYTHON have modern features that are much better integrated, which is probably because they have seen extensive use and refinement for general-purpose computing. The language features and built-in data types available to languages such as C++ or PYTHON, which are also well-integrated with available libraries and software components, are extremely good for building software with diverse and complex specifications. See the article [87] by John D. Hunter, the late original creator of MATPLOTLIB,[4] on the problematic experience of trying to integrate FORTRAN, C++, and MATLAB; or subsequently using MATLAB and JAVA together; that were experiences that ultimately led to his creation of MATPLOTLIB. Like `OCSage`, `pythODE` generated many gigabytes of data for many of the experiments (including the less rigorous initial exploration of the

[4] http://blog.fperez.org/2013/07/in-memoriam-john-d-hunter-iii-1968-2012.html

problem space) conducted in support of this thesis. Managing the diverse array of software requirements to effectively use this amount of data is best done with modern software ecosystems that have already been proven for this type of data management and accompanying software functionality.

Not all languages and software ecosystems are widely used for serious scientific or numerical computing. The consequences of being the largest and most ambitious user of a particular tool for a particular task include: requiring a great deal of extra labour to accomplish tasks that others are readily doing using standard tools, not easily being able to "Google" issues and bugs with the platform and associated libraries when they inevitably happen, needing to fix these issues and bugs with limited resources, and likely having to "reinvent the wheel" on the non-standard platform of choice. When using a non-standard platform for a particular task, it can be extremely difficult to know where the platform will "fail" in actual usage until many person-years are invested. In many cases, it has been a widespread community effort to develop certain platforms as mature tools for the tasks for which they are now commonly used [87, 132, 176, 185]. Therefore, any attempt to use non-traditional tools for scientific or numerical computing requires careful consideration.

5.1.1 Numerical computing in Java and `odeToJava`

All of the just-mentioned issues would have affected any continued development or use of `odeToJava` because although when first conceived in the early 2000s, many researchers were trying Java as a platform for numerical computing (see the discussion in the references [9, 101, 175]), continued development of `odeToJava` and scaling it up for a wide range of DEs and associated applications (as it is hoped `pythODE` will be) would mean taking on an extremely ambitious numerical computing project for Java. This would probably include considerable development effort for solving issues that might arise with tasks that are otherwise routine in platforms typically used for numerical computing such as Fortran, Matlab, and Python/SciPy.

In contrast, numerical computing platforms such as Matlab and Maple use Java extensively for components such as their GUI [3]. In fact, it has been widely observed that modern scientific and numerical computing applications increasingly involve a great deal of non-numerical and non-mathematical functionality in order to be effective research tools [149, 166]. Sophisticated commercial software packages for scientific computing such as Matlab and Maple utilize Java because it is a widely used and extremely suitable for general-purpose application development, but they also provide a domain specific language for the user-facing numerical computing component. However, this type of approach is better suited to development by extremely well-funded organizations that can continuously support a full-time development team, such as commercial software companies or government-funded laboratories. The need for a domain-specific language to make Java more suitable for complex numerical computing tasks is not just based on the author's experience. In a recent book on numerical computing in Java, the second chapter [35, pg.27] is on using Jython, a Python interpreter built on top of the Java virtual machine. However, jython does not have a direct interface to NumPy but instead allows a NumPy-like interface to the similar library `Nd4j` written in Java [35, pg.185]. However, the author does not feel that the available Java libraries are currently as mature, capable, or scalable as NumPy. In addition, NumPy itself is now extremely well-tested through being used by too many numerical computing projects to easily enumerate. A Java-based package, similar to `pythODE` (such as `odeToJava`), moving beyond the relatively small and simple DEs used for this study and using an array and linear algebra library not based on BLAS and LAPACK would require a substantial amount of "reinventing the wheel". In addition, although there are basic equivalents of the many of the mathematical libraries described in Section 4.1 written in Java, the author does not feel that many of these libraries are currently as capable or robust as the industry standard numerical libraries already interfaced from SciPy and Sage. In general, although it also definitely possible to call C/C++/Fortran based libraries from the JVM (Java virtual machine), if a well-tested and robust interface does not already exist, this will generally greatly increase the complexity of a project. John D. Hunter also described hitting a wall with using a combination of Matlab and Java (to handle a database and networking) for analyzing 3D medical image data as one of the reasons to rewrite his project in Python, which included the initial development of matplotlib [87]. For Python, it is still better that a well-tested and robust interface to low-level libraries already exists, such as SciPy or the additional ones that Sage interfaces to that are described in Section 4.1. However, the barriers to interfacing lower-level languages can be much lower and a quick search online shows that many research groups have written their own interfaces to any low-level code they needed to interface.

One of the most significant issues with using Java for serious numerical computing is the lack of support for operator overloading. When developing `odeToJava`, it became apparent that the basic code implementing numerical algorithms would not be able to be readily reused for operations on datatypes other than double-precision floating point, such as complex numbers, quadruple-precision floating-point numbers, arrays, matrices, etc., whereas with both `OCSage` and `pythODE`, using more advanced data types in place of double-precision floating point would be possible with relatively minor modifications and testing, albeit possibly with a significant performance penalty for some datatypes. The undesirability of user-defined operator overloading is understandable for the large codebases of commercial projects that Java is designed for because it is difficult to reason about and collaborate on code that has too many layers of abstraction across a large codebase. Conversely, without operator overloading for numeric data types, including complex numbers and matrices, it is much more difficult to reason about complex mathematical formulas, especially when working between written mathematics and source code. Therefore, some of the disadvantages to Java for numerical computing should be seen more of as tradeoffs with regards to specific requirements of different problem domains and language design choices.

Another issue with Java (and many low-level languages such as C/C++/Fortran), in comparison to Python, is that a lot of boilerplate code can be required for relatively simple tasks. This is probably why Jython is introduced so early in the book mentioned above on numerical computing in Java [35, pg.27]. Static typing and compiled code is of course ideal for Java's primary purpose of being used for the large codebases of many commercial projects. Depending on the specific research program and task, languages requiring a lot of boilerplate code can substantially increase the time required for many experimental and research computing tasks in comparison to MATLAB or PYTHON. In particular, the author feels that PYTHON strikes the right balance for both quick scripts and more involved projects for experimental computing because it has strong but dynamic typing, rather than the weak typing of MATLAB or the static typing of JAVA that often requires an enormous amount of boilerplate code. With `pythODE` (specifically using the `pymath_common.py` file described in Section 5.4), it is fairly easy to create an environment for PYTHON that requires minimal preparation or boilerplate code, similar to how MATLAB is often used, for experimentation within packages such as `pythODE`.

Therefore, despite these many available numerical libraries for JAVA, the author feels that JAVA is only suitable for numerical computing projects if there are project-specific reasons to use JAVA (prior experience, an existing setup, community adoption of JAVA for a particular problem domain, available human resources, non-numerical computing considerations such as database or GUI support, etc.). However, FORTRAN/C/C++ and now PYTHON are more suitable if doing numerical computing that requires robust and well-tested numerical libraries for projects that do not specifically need to incorporate JAVA in any way. Despite some modern features that have been added to languages like FORTRAN, a modern application such as `OCSage`/`pythODE` involving a lot of non-numerical functionality, such as incorporating a diverse set of software libraries, networking, data management, complex data structures, and engineering the software for modularity and flexibility with respect to the problem solved, would probably be one of the more ambitious research computing projects of the type attempted in pure FORTRAN. Ultimately, any programming language, software component, or platform choice by an individual or research group should take into consideration community support, the availability, and the integration of libraries and other software components, the maturity and robustness of the platform, monetary resources available for software development, and issues regarding either recruiting people with the appropriate skills or training them. Although for the non-numerical parts of numerical computing projects, JAVA does have the advantage that it is mature, robust, there is an enormous community using it, and it easy to find developers with the appropriate skills, for different numerical computing projects and different research groups, the best choice may be FORTRAN, C, C++, PYTHON, R, or some combination. It would be difficult to recommend JAVA for a project focused on breaking new ground in numerical computing because the widespread use of PYTHON means that the PYTHON ecosystem now shares many of these advantages of the JAVA as a platform for serious software engineering. In addition, PYTHON is now an extremely mature development platform because of its widespread use for general-purpose computing and application development, but that is also widely used for numerical computing. In summary, there are a great number of organizational considerations in choosing the most suitable and sustainable software platform for any project involving numerical computing.

5.1.2 An examination of the choices made in developing `pythODE`

Despite the sometimes lower performance in comparison to languages such as FORTRAN, C, and C++ for an individual researcher or small research group, as indicated by the its explosion in popularity for numerical computing, Python can be the ideal choice. Although better performance can always be appreciated, on modern computing hardware it is often the case for many studies that the amount of development effort required to just implement a study is a much greater bottleneck than the extra computational time a language like Python might require. For studies such as the one conducted in this thesis, that are well within the range of powerful contemporary desktop computers but that have difficult software development challenges involving a great deal of iteration and experimentation, the Python scientific computing ecosystem built around SciPy and Sage can eliminate an enormous amount of software development work by allowing all desired functionality to occur within one language and platform. The author feels that a study such as the one presented in this thesis has been possible as the work of mostly one person only since approximately 2010 because the scientific computing ecosystem around Python only became mature enough after that point. Although a software system similar to `OCSage` and `pythODE` probably would have been possible for a team to develop pre-2010 in a language such as C or C++ using available libraries, the scientific Python ecosystem was not as extensive and robust at that time as it now is at the time of the publication of this thesis. Although C or C++ are often better performing, the generally more complex codebase that results from manual memory management and static typing, the greater difficulty initially experimenting with quick and simple scripts that were essential to the long incremental development process for `OCSage` and `pythODE`, and the system administration tasks and continual maintenance to manage particular combinations of libraries that would have been required to support the functionality would probably have been prohibitive for one person. This is supported by the fact that large scientific software projects that do incorporate diverse and disparate functionality are invariably developed by well-funded teams. For example, the C++ based `Chaste` project [118], partially because of the complexity of the specific problem domain, lists approximately 40 people as having been part of the development process.[5] Even the `SUNDIALS` project [80], which is a robust and production-ready C++ based IVP solver described in more detail the next section, lists current and past contributions of 15 people and is developed in the context a government-funded laboratory.[6] Software packages such as PETSc for solving PDEs in a high-performance computing environment often incorporate RK methods, but PETSc is integrated into a large software stack that is developed by a large team and community.[7]

The list of topical software on the SciPy website[8] shows an incredible diversity of SciPy-based tools that have proliferated, including some that are highly capable tools developed by individuals and ones that have seen development by a much larger number of individuals. Examination of the content of the annual SciPy conference[9] indicates that there are many projects being developed as specialized research tools with a diverse array of functionality, many similar to or greater in size and complexity to `OCSage` and `pythODE`, being developed by individuals or small teams by leveraging the computing ecosystem built around SciPy.

[5] http://www.cs.ox.ac.uk/chaste/theteam.html
[6] https://computation.llnl.gov/projects/sundials/team
[7] https://www.mcs.anl.gov/petsc/
[8] https://www.scipy.org/topical-software.html
[9] http://conference.scipy.org/

The design choices for the `OCSage` and `pythODE` packages after 2015 were strongly influenced by examining the `mystic` project[10] (used for uncertainty quantification and highly constrained non-convex optimization) because their optimization process can involve thousands of numerical simulations that can include DEs [116]. However, due to `pythODE` and `OCSage` being mostly one-person projects targeted to a different purpose, the mathematical size of the DEs solved by `pythODE` is much smaller than the types of problems the `mystic` project is designed for [127].

Importantly, even for the numerous software projects surrounding `mystic`,[11] which were originally developed using funding from a 15 million dollar NSF grant,[12] it is still emphasized that they used pure PYTHON to reduce any accessibility barriers to the `mystic` platform through, for instance, a reduction in the number of dependencies [116]. Because a stated goal was also to give the `mystic` platform and supporting packages widespread availability to scientists and engineers, the pure PYTHON approach also furthers that goal. Even SCIPY is listed as an optional dependency, probably because not all projects would have needed it, and historically it has sometimes been quite difficult to install/compile the correct version of SCIPY on some platforms, such as Windows, for which pure PYTHON is readily available.

In addition, depending on the individual constraints of a research program, availability of computing resources, and ability to leverage available high-performance libraries, pure PYTHON may not give enough of a performance drawback compared to C/C++ to make the extra development time, barriers to entry for potential users and developers, and fewer number of computers required worthwhile. However, at the time of writing this thesis in 2018, the author of this thesis feels that what are now standard, widely used, and robust PYTHON libraries such as SCIPY and CYTHON are not onerous dependencies and would not contribute excessively to development and usage barriers. Despite that, especially in a research setting where the intellectual focus is not the software itself, it should be assumed that every extra language, platform, and other software dependency a software project relies upon will increase the barrier to setup, usage, and development as well as the costs of continuing use. Even the mixture of PYTHON and SQL in `OCSage` and `pythODE` contributes to the complexity of the software system, although this is an excellent tradeoff to get the previously mentioned benefits of an RDBMS.

5.2 Review of software to evaluate the performance of IVP methods

Like `OCSage`, the `pythODE` package has a significantly different and potentially more complex structure than many other packages for testing IVPs. This helps `pythODE` give meaningful insights that are not apparent with a basic IVP solver because it fully utilizes the capabilities of the PYTHON/SCIPY ecosystem, takes advantage of available parallelism in a way that supports this study, utilizes modern programming language features, allows a sophisticated experimental setup, and integrates with a `PostgreSQL` database. However,

[10]http://trac.mystic.cacr.caltech.edu/project/mystic/wiki.html
[11]https://github.com/uqfoundation
[12]http://public.enthought.com/~mmckerns/cit/my/Home.html

the `pythODE` package incorporates ideas from software packages that have been published by other authors. Therefore, it is important to review existing techniques and software packages for evaluating the performance of IVP methods.

Much of the performance data in published and peer-reviewed studies of numerical methods consists of a limited number of IVPs and IVP methods that have been implemented in an ad-hoc way, sometimes in lower-level languages such as FORTRAN or C, but often utilizing a higher-level language or a platform such as MATLAB. The amount of performance data and context from ad-hoc scripts tends to be extremely limited and does not give a complete picture of the relative performance from choices made in constructing numerical methods. The limitations of and possibly misleading conclusions that might be drawn from only small-scale performance testing are clearly shown below in Section 5.8 when performance testing thousands of ERK pairs. Small-scale and ad-hoc performance testing methodologies generally only provide enough information for showing adequate performance, indicating that the mathematics is done correctly and that a performance advantage may exist over previously known numerical methods [166]. For instance, the only existing IVP methods that new IVP methods are sometimes compared against are FE (2.25) or RK4 (2.26). Because ad-hoc scripts are rarely released or discussed in detail, the performance results obtained from them cannot be peer-reviewed in any detail. This lack of transparency has been discussed as an issue in reproducibility of published results [98, 111].

There have also been software packages that have specifically been developed for testing numerical algorithms for IVPs. However, it is impossible to describe all software developed to test IVP methods because individual pieces of published software can have varying levels of maturity, as well as it being relatively common that unpublished software is used internally by a research group on a continuing basis for peer-reviewed research. Software is sometimes informally published on the internet or formally published but not peer-reviewed (such as in association with a technical report [115] or book [72, 73, 134]). Sometimes software has been released in a limited manner that may be difficult to access because of issues such as internet links that are broken or have vanished, e.g., see the references [133]. Only a small minority of software packages developed and used to test IVP methods actually end up being the subject of peer-reviewed publications. Many software packages tie together existing standalone IVP solvers in order to test the algorithms involved but they do not have the granularity to easily modify all of the distinct algorithms and the many other components that constitute a modern IVP solver. Demonstrating how specifically to implement the component algorithms of an IVP solver with relatively fine granularity and modularity is one of the major contributions of the `odeToJava` package and resulting paper [106].

Before reviewing software packages focused on testing IVP methods in a research setting, there are also commonly used packages that are "black-box" IVP solvers for users not directly interested in the numerical algorithms or implementations used. These often have the advantage of a long period of usage and development that makes them robust and stable. However, due to the flexibility that software generally allows, there is a great deal of overlap in any categorization, including which available IVP solver might be considered "black box". Some representative examples of "black-box" IVP solvers are:

- The FORTRAN-based ODEPACK [79][13] is a collection of various FORTRAN and C based IVP solvers that are developed by U.S. government national laboratories (primarily Lawrence Livermore national laboratory). Two of five component packages of ODEPACK are ideal for solving non-stiff IVPs, that includes the LSODE package [137] (used by SCIPY as discussed below) and the VODE package [23] that instead implements IVP methods with coefficients that can vary. These solvers generally give extremely good performance and are considered extremely robust because they are developed for the extensive array of extremely difficult problems that the work of these labs and defence-related industries can require solving. The LSODE [137] and VODE packages [23] do not use RK methods but instead rely exclusively on the multistep methods that are briefly mentioned in Section 2.4.

- The `SUNDIALS` (SUite of Nonlinear and DIfferential/ALgebraic equation Solvers) package [80] is a C++ package that is derived from ODEPACK (among several other packages) and is developed by the same national laboratories as ODEPACK. A component package of `SUNDIALS` for solving IVPs is CVODE, which is a C/C++ version of VODE that was originally (and still can be found as) an independent package [39]. `SUNDIALS` is also widely used by practitioners who require a high performance and robust IVP solver. Like VODE and CVODE, `SUNDIALS` exclusively uses the multi-step methods that are briefly mentioned in Section 2.4.

- The MATLAB IVP solver `ode45` uses the DP5(4)$_{6(7)}$ pair (2.79) (`ode45` originally used the RKF4(5)$_{6(6)}$ pair (2.78); see a blog entry[14] by the original developer of MATLAB, Cleve Moler, for more information). The MATLAB platform also provides several alternatives for solving non-stiff IVPs, such as third- and second-order ERK pairs [15] as well as variable-order multi-step methods [153]. However, although the MATLAB code for these IVP solvers is available, it is generally too monolithic and specialized to the specific integration methods to be used for conducting the type of fine-grained experimentation for which `odeToJava` and `pythODE` are designed.

- Most popular mathematical and statistical software platforms now have robust and well-tested for DEs even if this is not their primary focus [151]. This includes MAPLE, MATHEMATICA, and R [167]. Similar to MATLAB, they have a selection of well-tested but monolithic IVP solvers, giving the practitioner a choice of methods. The MAPLE and R platforms both have the CK4(5)$_{6(6)}$ pair (2.83) as one of the available IVP methods.

- The `FATODE` package by Zhang and Sandu [203] incorporates discrete FORTRAN IVP solvers, which were heavily inspired by existing publicly available FORTRAN solvers [202, pg.11], into a single package that also include ERK methods and IRK methods. One of the major contributions of `FATODE` is that

[13] https://computation.llnl.gov/casc/odepack/
[14] https://blogs.mathworks.com/cleve/2014/05/26/ordinary-differential-equation-solvers-ode23-and-ode45/

it implements methods for *sensitivity analysis* of the initial conditions for IVPs, an analysis feature that gives additional insights into many application areas involving numerical computing, in a package described by its authors as being of comparable efficiency and utility to software such as `SUNDIALS` [202, pg.10]. Although a reasonably effective effort has been made to engineer the methods and solvers being experimented with into a common platform, `FATODE` still ties together distinct FORTRAN solvers that each implement distinct IVP methods but with some sharing of subroutines and being general enough to allow specifying arbitrary Butcher tableaux (2.34).[15] This includes several embedded ERK pairs that have a higher-order component ranging from third- to eighth-order, including the classic DP5(4)$_{6(7)}$ pair (2.79). Also included in `FATODE` is an advanced eighth-, fifth-, and third-order ERK method that was already mentioned in Section 2.6.

- The SCIPY platform has IVP solvers implemented in the package `scipy.integrate`.[16] These provide many of the same features, such as interpolation, event detection, and numerous output options, that give general-purpose full-featured IVP solvers with a similar interface and functionality to the popular MATLAB ones. There are eight different IVP solvers in `scipy.integrate`, and the specific ones relevant to the discussion in this thesis are:
  - The `scipy.integrate.RK23` solver uses the same three-stage third- and second-order Bogacki–Shampine ERK pair [15] as the MATLAB `ode23` solver. This solver plays a similar role to `ode23` in that lower-order methods can often perform better at coarse tolerances (shown in Section 5.5) or as an alternative to higher-order methods if undesirable behaviour is seen.
  - The `scipy.integrate.RK45` solver uses the classic DP5(4)$_{6(7)}$ pair (2.79) and plays a similar role to the `ode45` solver of MATLAB as a general-purpose non-stiff IVP solver.
  - The `scipy.integrate.LSODA` solver wraps the FORTRAN solvers `LSODE` described above and includes automatic detection of stiffness with switching between the appropriate stiff/non-stiff IVP method.
- The `SciKit` (supplementary add-on to SCIPY) `odes` package[17] wraps the `SUNDIALS` package [80] in addition to including another implementation of the DP5(4)$_{6(7)}$ pair (2.79) and the same eighth-, fifth-, and third-order ERK method from Section 2.6 mentioned above for `FATODE`.

Some representative examples of published software packages focused on testing IVP methods are:

- The "non-stiff DE" test set [83] was first published in 1972 as a collection of FORTRAN implementations of IVPs and IVP methods, which could be further expanded to support research. Various versions of this test set have been used by many authors over the years [40, 48, 56, 84, 145, 155], not always

[15] http://people.cs.vt.edu/asandu/Software/FATODE/downloads.html
[16] https://docs.scipy.org/doc/scipy/reference/integrate.html
[17] https://github.com/bmcage/odes

using the original FORTRAN software [16, 128, 181, 182, 183], nor has the code that these other authors developed and used when testing with the "non-stiff DE" test set always been released. An overview of the specific IVPs in the "non-stiff DE" test set are described in the next section, with some examples of IVPs already given, i.e., (2.11), (2.13), and (2.14).

- The "stiff DE" test set [55] was first published in 1975 as a similar software package to the "non-stiff DE" test set. Although not used in this study because most of the IVPs are too stiff to practically solve with ERK methods, it is mentioned because of the connection to the "non-stiff DE" test set [83] and because the IVPs have been implemented in `pythODE`. The "B1" IVP (2.12) from this test set is used as a demonstration in Chapter 2.

- The IVPs included in the books by Hairer et al. [72, 73], which have been heavily cited throughout this thesis [72, 73], are published as code.[18] The code is available on Hairer's personal website.[19] Although this code is not the subject of either specific technical reports or peer-reviewed publications, it is well-known, and influential nonetheless.

- The package by Nowak and Gebauer [124] is what the authors call a "test frame" consisting of an ODE solver with a web interface, which they describe in a technical report. Although not widely cited, it is notable because it is one of the first packages to use more modern software technology beyond the traditional monolithic FORTRAN solvers to attempt to make testing IVP methods easier.

- The "Bari" test set [115] implements stiff IVPs and DAEs (differential-algebraic equations, which are beyond the scope of this study) and solves them with implicit methods, including IRK methods. However, despite being a non-peer-reviewed technical report released along with code that has been used in support of many publications by the research group, it is widely cited and is one of the most recent packages specifically for testing IVP methods. It contains two IVPs that are used in this study, discussed below, as well as valuable discussion on testing IVP solvers. One of these IVPs, the "Pleiades" IVP (2.16), is also in the collection of IVPs by Hairer et al. already mentioned [72].

- Ketcheson has released a package called `NodePy`[20] that at first glance seems to have similar goals to the combination of `OCSage` and `pythODE`. However, `NodePy` and the two packages presented in this thesis are currently targeted to quite different purposes. `NodePy` seems to be focused on providing an alternative to by-hand construction of RK methods and ad-hoc programs for basic performance evaluation. It allows the implementation of a broad range of numerical methods beyond just RK methods. By contrast, the current versions of `OCSage` and `pythODE` have only been used for this study and the limited resources for software development have largely focused on the requirements of this study. Specifically, the software packages described in this thesis are currently focused exclusively on ERK methods and handling of the

[18] http://www.unige.ch/~hairer/software.html
[19] http://www.unige.ch/~hairer/
[20] http://nodepy.readthedocs.io/en/latest/index.html#

large amounts of data required to study the coefficient selection process for six-stage fifth-order ERK pairs, that then allows finding distinguishing features between the large number of possible families (including the new families introduced in Chapter 3).

- The `odeToJava` package [106] by the author and supervisor of this thesis has already been mentioned. It applies modern software engineering techniques to demonstrate fine-grained experimentation with the different algorithms that make up a complete IVP solver.

What should be apparent from the lists of software in this section is that there is a broad range of software to solve IVPs that has been released and published, all to suit different purposes.

5.3 The IVPs from common non-stiff IVP test sets

As indicated in the review of software in Section 5.2, IVPs are often collected into test sets for performance testing of IVP methods. Sometimes these test sets are presented as a named collection of IVPs and other times as ad-hoc collections of IVPs that generally appear together in a publication. In order to use them with `pythODE`, all tested IVPs have been re-implemented into PYTHON as described in Section 5.4.1. Although space does not permit giving the mathematical formulation of all IVPs implemented in `pythODE`, a check of the `pythODE` source code shows over 100 distinct IVPs have been used to study the families of ERK pairs described in this thesis. It is extremely important to test new IVP methods with many different IVPs because some performance or accuracy issues only become apparent with certain IVPs. For instance, during the initial development and testing of `pythODE` the author experienced at least one bug that only occurred when solving for non-autonomous IVPs. Because the IVPs commonly used to test IVP methods for some classes of problems, such as celestial mechanics problems, are usually autonomous, these types of bugs will not always become apparent with limited testing. The examples given in Section 2.2 are representative of the properties of most IVPs in the following test sets.

5.3.1 The non-stiff DE test set

One of the earliest published collections of IVPs for testing IVP methods is the "non-stiff DE" test set [83], where the original collection of problems and accompanying code, which is mentioned in the previous section, was published in 1972. There were five original "classes" of IVPs, with a sixth "class" of IVPs that have discontinuous RHSs proposed in 1987 by Enright and Pryce [56]. The author feels that these "classes" are mostly no longer a useful and informative classification system for helping researchers and other practitioners to test and choose IVP methods. For historical purposes, these "classes" are given by:

- The "non-stiff" IVPs of "Class A" are IVPs that are derived from single ODEs (2.1a), i.e., $m = 1$, with all but the first IVP in the class being nonlinear. The simplicity of these ODEs is such that one could expect to find them incorporated into an exercise in an elementary calculus textbook. One of the two

IVPs used by Dormand and Prince [47] to first demonstrate the DP5(4)$_{6(7)}$ method (2.79) is the "A3" IVP (2.11) of this "class".

- The "non-stiff" IVPs of "Class B" are IVPs that are derived from ODEs (2.1a) that are systems of size $m = 3$, except the first that is size $m = 2$. The first two equations have a linear RHS and the last three have a non-linear RHS. The author has found that these IVPs rarely have the same performance characteristics and therefore "Class B" is not a particularly informative grouping. However, the individual IVPs each have different numerical behaviour and several of these IVPs are individually discussed below.

- The "non-stiff" IVPs of "Class C" are called "moderate systems" in the original publication [83]. The first two IVPs (the "non-stiff C1" and "non-stiff C2" IVPs) are linear IVPs of size 10 derived from radioactive decay chains. The "non-stiff C3" and "non-stiff C4" IVPs are linear IVPs that are similar to the heat transfer IVP described in Section 2.2 with sizes $m = 10$ and $m = 51$, respectively. The fifth IVP (the "non-stiff C5" IVP is of size $m = 20$ that simulates of the five outer planets (including Pluto because it was still classified as a planet at the 1972 publication date) of the solar system. During performance testing in this study, it was generally observed that the performance of pairs on linear and non-linear IVPs are not well correlated. Although there are performance issues that strongly affect linear IVPs (such as the "non-stiff C1"–"non-stiff C4" IVPs) and it is important to include some examples, they should not be a dominant feature of IVP test sets for numerical methods focused on non-linear IVPs.

- The "non-stiff" IVPs of "Class D" are called "orbit equations" in the original publication [83]. These are all IVPs based on the basic Kepler IVP (2.13) that is described in Section 2.2 and using a range of eccentricities from 0.1 to 0.9. The "D3" IVP (2.13) of this "class" is the second of two IVPs used by Dormand and Prince [47] to demonstrate the DP5(4)$_{6(7)}$ method (2.79).

- The "non-stiff" IVPs of "Class E" are "higher-order" equations of size $m = 2$ that are derived from single second order ODEs using the procedure described in Section 2.1.2. The "non-stiff E2" IVP is the Van der Pol equation (2.14) with $\epsilon = 1$, although this parameter is not explicit in the implementation in the "non-stiff DE" test set [83]. However, for examining the effect of ϵ, the specific formulation (2.14) used by the "Bari" test set (described below) is used in Section 2.2 and the implementation in `pythODE`.

- The "non-stiff" IVPs of "Class F" are "problems with discontinuities", which can provide a severe test to certain IVP solvers. Crude solutions of discontinuous IVPs [72, pg.197] are possible with the solvers in `pythODE` using only the mathematical theory described in Chapter 2. However, these IVPs generally require dense output (interpolation) and event detection for reliable and accurate solution. Dense output and event detection are beyond the scope of this study and not currently implemented in `pythODE`. Discontinuous ODEs are well supported by the IVP solvers built into software such as MATLAB, OCTAVE, SCIPY, and many others used in production settings, with details that can be found in the references [72, pgs.189–202][152, 153].

The “stiff DE” test set is also implemented in `pythODE` and divided up into similar “classes”. However, like the “non-stiff DE” test set, it is not apparent whether the division into these “classes” are actually useful to either the researcher or practitioner. In order to make implementation in languages such as FORTRAN 77 easier, IVPs in the “DE test sets” are all scaled to have $t \in [0, 20]$. However, this type of normalization is analogous to, for instance, software that would require all usernames to be the same length. Modern programming languages such as PYTHON and C++, as well as databases such as `PostgreSQL`, handle data of non-uniform sizes extremely well. Therefore, this makes scaling the solution of all IVPs to the same time interval completely unnecessary. In addition, this type of restriction would severely limit the potential growth of any software package that did this because this type of scaling would not be straightforward with all IVPs.

5.3.2 The non-stiff IVPs from Hairer et al.

The non-stiff IVPs that feature prominently in the book by Hairer et al. [72] are implemented in `pythODE`. These IVPs are important because they are closer to “real-world problems” than many of the “toy problems” in the “DE test sets”. Although details are best examined in the references, the IVPs demonstrated in the book by Hairer et al. are:

- The “Arenstorf orbit” IVP (2.15) has already been described in Section 2.2 [72, pgs.129–131].
- The “Euler rigid body” IVP describes the rotation of a rigid body using DEs originally found by Euler [72, pgs.244–245]. Many of the IVPs that can be derived from the mechanics of rigid bodies are an important application of ERK pairs. This “Euler rigid body” IVP is nearly identical to the “B5” IVP of the “non-stiff DE” test set, except for having different coefficients, initial conditions, and a time-dependent component to the ODE that simulates the application of a force to the rigid body.
- The “Brusselator” IVP gives a semi-discretized 2D reaction-diffusion PDE using similar finite differences to those described in Section 2.2 [72, pgs.248–249]. Although the semi-discretized 3D analogue of this IVP is stiff [73, pg.148], the 2D version [72, pgs.248–249] used in this thesis is non-stiff and a suitable semi-discretized PDE for testing ERK pairs.
- The “Lorenz” IVP is the system that was originally used to demonstrate chaotic behaviour in atmospheric models [72, pg.245]. Chaotic behaviour is a common difficulty for applications that are simulated by IVPs and therefore an important class of IVP for testing non-stiff IVP solvers. This IVP is difficult to meaningfully use for the performance testing found in this thesis because the relatively sudden onset of chaotic behaviour gives extremely noisy performance data, which is difficult to draw specific conclusions from.
- The “Pleiades” IVP (2.16) has already been described in Section 2.2 [72, pgs.245–246][115].
- The “hanging string” IVP simulates a hanging string moving under a horizontal force and is modelled by similar mechanics to a rigid beam, except neither a string nor the IVP that models it is “stiff” [72,

pgs.247–248]. The similar "rigid beam" IVP, described in Hairer et al. [73, pgs.8–11], is also practically solvable by ERK methods, albeit inefficiently compared to IRK methods.

5.3.3 The non-stiff IVPs from the "Bari" test set

Several non-stiff IVPs are used from the "Bari" test set [115], where the associated software release is already described in Section 5.2. These IVPs are:

- The "HIRES" IVP describes a photochemical reaction from the study of plant physiology [115]. This IVP is mildly stiff. Because the "HIRES" IVP can still be solved reasonably well using solvers based on ERK methods, it is a good candidate to test performance solving mildly stiff IVPs.
- The "Pleiades" IVP (2.16) has already been described in Section 2.2 because this overlaps with IVPs published by Hairer et al. [72, pgs.245–246][115]. It is important to include non-stiff IVPs in test sets for stiff IVP methods such as the "Bari" test set because stiff IVP methods should still be able handle non-stiff IVPs without issue. This should be the case even if the stiff IVP methods are generally not as efficient as explicit non-stiff IVP methods when solving non-stiff IVPs.
- The van der Pol ODE (2.14) and corresponding IVPs are already described in Section 2.2. The implementation used in the "Bari" test set allows choosing ϵ to change the characteristics of the IVP, giving either a stiff or non-stiff IVP. Both this version from the "Bari" test given in Section 2.2 as well as the "non-stiff E2" IVP are implemented in `pythODE`.

5.3.4 Sharp's test sets of celestial mechanics IVPs

Sharp has published several collections of IVPs from celestial mechanics [157, 158, 159] that are ideal for testing non-stiff IVP methods and that have been implemented in `pythODE`. It is shown below that celestial mechanics problems based directly on the simple Kepler IVP (2.11) have relatively well-defined optimal ERK pairs.

Sharp's restricted three-body test set

The non-stiff "Arenstorf orbit" IVP (2.15), which belongs to the class of restricted three-body problems, has already been introduced in Section 2.2. In 2004 Sharp published a test set of similar restricted three-body problems (not all of which can be classified as Arenstorf orbit problems) [158]. These IVPs are convenient for testing IVP methods because they are 28 IVPs based on only 2 ODEs. One of these ODEs is same ODE used for the "Arenstorf orbit" IVP (2.15) and the second ODE models an analogous 3D system. The many different initial conditions give completely different systems with varying degrees of stability and instability, which are described in detail by Sharp [158]. The last 4 IVPs of the 28 are implemented in `pythODE` but not used by this study because they require at least quadruple-precision arithmetic to use effectively for testing IVP methods.

IVPs from modelling the solar system

Sharp has also published several overlapping test sets of IVPs from celestial mechanics that describe the motion of bodies in the solar system [157, 159, 160]. The IVPs used in this thesis are:

- The "Jovian" IVP is based on the "non-stiff C5" IVP, but does not include Pluto and is integrated on a long timescale to provide a challenging test of IVP methods.

- A variation on the "Jovian" IVP includes a comet that makes six close encounters with Jupiter, with the last being at 6970 days [160]. In this thesis, this variation on the "Jovian" IVP is referred to as the "Jovian asteroid" IVP.

- The "Plutonian" IVP is just the "non-stiff C5" IVP from the "non-stiff DE" test set integrated on a longer time scale. There are important events in the interactions Neptune and Pluto that are important to simulate accurately and provide a challenge for IVP solvers [157, 159]. However, the specifics of these events are beyond the scope of this thesis. Despite this, the long-term integration of the "Plutonian" IVP provides a good test of IVP methods.

- The "nine planets" IVP incorporates all nine planets. This is particularly challenging because Mercury orbits once every 88 days and Pluto orbits once every 248 years. These different time scales and the mutual interactions make accurate long-term simulation a challenge for IVP solvers.

- The "DE 102" IVP that Sharp presents in his publications [157, 159] incorporates *general relativity* to accurately model the configuration of the solar system, which includes accurately simulating the Moon from 1411 BC to 3002 AD [122]. Sharp does not provide a detailed description of the "DE 102" family of problems or their implementation, but extensive discussion and links can be found in the references [122] and on the internet.[21] No published code could be found for the "DE 102" IVP that Sharp used. However, code is available for a revision called "DE 118" [122].[22] What is often available from organizations such as JPL (Jet Propulsion Labratory) for problems similar to "DE 102" are several gigabytes of *ephemerides* that can be interpolated to give positions of bodies in the solar system in the past or the future, rather than the simulation code itself. Incorporating general relativity makes the "DE 102" family of problems *post-Newtonian*, i.e., they can no longer be described as a system of ODEs with only second-order derivatives. This means some specialized methods, such as *Runge–Kutta–Nyström* methods or *symplectic* methods (a type of structure preserving method), can no longer be used [157, 159]. Future work could include implementing the ODE from "DE 118" or similar models

[21] https://www.projectpluto.com/jpl_eph.htm/
[22] http://www.moshier.net/

for examining whether the results in this thesis continue to apply for ODEs that cannot be formulated as second-order ODEs.

Although Sharp tested much higher-order methods than fifth-order and used an extensive number of integrators other than ERK, these IVPs are all severe tests and give insights into the construction of ERK methods even when solved with just fifth-order ERK methods. Unlike some of the previous IVPs discussed, these IVPs do not have published reference solutions, and therefore reference solutions must be generated. Using `pythODE`, the reference solutions are generated by using a constant stepsize with a procedure that is be discussed below in Section 5.6.

5.3.5 Non-stiff advection IVPs

Semi-discretized advection PDEs, such as (2.17), are also implemented in `pythODE` because ERK methods are commonly used for solving semi-discretized PDEs when the resulting IVP is non-stiff. The ODEs resulting from semi-discretization of (2.17) are linear when the semi-discretization is done with methods such as (2.18) or (2.19). However, other PDEs can have terms that result in the ODE found from semi-discretization being nonlinear. For example, the shallow water PDE implemented in FATODE [202, 203] has terms containing the spatial derivatives non-linearly. The semi-discretized shallow water equations are non-linear and an example of a PDE, which when semi-discretized, is often better solved with ERK methods than IRK methods [202, pgs.45–48]. In addition, there are also semi-discretization methods that give a non-linear ODE from linear PDEs. However, because only semi-discretized linear PDEs were implemented, no easily generalizable conclusions about either more efficient or more "optimal" methods for semi-discretized PDEs were reached in this study. Given that there are often many PDEs and semi-discretization methods with a similar mathematical foundation, analogous to the wide range of celestial mechanics IVPs based on the Kepler IVP (2.13), there is significant potential to apply similar techniques to those introduced in this study in order to find better ERK pairs for semi-discretized PDEs. Although a study similar to this one involving semi-discretized PDEs would push the limits of both the software and available hardware, it is definitely possible in the future as `pythODE` develops through use in more studies.

5.4 Overview of the `pythODE` package

The version of the `pythODE` package used for this study is about 13,000 lines of code not including auto-generated code or comments. It is unnecessary to describe within this thesis all of the functional components of `pythODE`. This is especially because `pythODE` is likely to evolve substantially as it gets used for future studies. Furthermore, it is undesirable to describe unimportant software or implementational details that are likely to be changed in the future or be different for other studies subsequent to the one in this thesis. However, the general form of many components has remained the same over the many versions. Similar to how `OCSage` is presented in Section 4.4, in the following overview the files of the `pythODE` package are described as functional units with respect to the main package directory.

- The source file that provides the `IVPSolver` class to coordinate the IVP solution is:

  `solver.py`

  Most of the numerical functionality of an IVP solver is provided by modules that the `IVPSolver` class coordinates and that are described below. The `ConstantIVPSolver`, `EmbeddedIVPSolver`, and `StepDoublingIVPSolver` sub-classes of the `IVPSolver` class coordinate IVP solvers that use a constant stepsize, an embedded error estimate (2.52), and a step-doubling error estimate (2.57), respectively.

- Various constants and user-configurable variables are stored in the source file:

  `experiment_common.py`

- Some functionality is used repeatedly by both `pythODE` itself and its supporting scripts. Some of this functionality to help quickly write these scripts and extend `pythODE` is included in the source file:

  `pymathdb/pymath_common.py`

  A single function call from `pymath_common.py` imports a large number of PYTHON packages and other functions commonly used by `pythODE` (and by the author for other projects). Functions are included for widely used but simple functionality found throughout `pythODE`, such as argument parsing, timing parts of scripts, exception handling and reporting, etc. This greatly simplifies maintenance of `pythODE`, experimentation, and creating the many specific scripts that are used to arrive at the results presented in this chapter. The `pymath_common.py` source file was originally started as a standalone component by the author and is still maintained as such because it continues to be useful as a starting point for new projects.

- The source files containing most of the functional components of an solver are:

  `modules/basemodules.py`
  `modules/erk.py`
  `modules/stats.py`
  `modules/stepcontrol.py`

  The `basemodules.py` source file contains the base class for all other modules. The `erk.py` source file contains the `ERK` and `ERKEmbeddedEstimate` classes with modules that implement ERK formulae. The `stats.py` source file contains modules that record information, such as the number of steps, or write the solution to a file or data structure. The `stepcontrol.py` source file contains modules for tasks such as estimating the error and determining whether a step should be rejected or not. Even when using a constant stepsize, modules that do not directly control the stepsize are sometimes required to reject a step when there are invalid values or other failures in the solution process. Rejecting steps as soon as there is a problem avoids excessive cost and ensures that the correct reason for solution failure is recorded. Some of these modules are also specific to the type of solver, for example, step-doubling error estimation requires some modules to be specific to it even if they are unrelated to step control.

- The directory that contains the definitions of any IVPs implemented is:

  `odes/*`

  Specific examples of how IVPs are defined is given in Section 5.3.

- The directory that contains the definitions of numerical methods (mostly ERK methods currently) is:

  `schemes/*`

  The code to implement the numerical algorithm, e.g., the code reflecting for RK methods (2.33), itself is not in this directory, only information such as the Butcher tableau (2.34) coefficients, which are then used by the appropriate module.

- The source files that are used to solve the chosen IVPs with the chosen RK methods based on a particularly formatted `PostgreSQL` table are:

  `pymathdb/db_solver.py`

  `pymathdb/db_watcher.py`

  These scripts are in their own directory because they are usable independent of `pythODE` for tasks that run a program focused on updating `PostgreSQL` tables on a small cluster of computers. The format of the `PostgreSQL` table used is further described in Section 5.4.3, but in addition to the required format, it can also contain extra columns with extra information such as data from `OCSage` searches, without causing conflicts. The `pymathdb/db_watcher.py` source file partitions chunks of work to each computer as necessary and the `pymathdb/db_solver.py` source file actually solves the IVPs using the `IVPSolver` class, including partitioning these among the CPU cores on each computer.

- The directories that define, contain the relevant scripts, and allow visualization of the numerical experiments are:

  `akroshko_erk56_study/db_loader.py`

  `akroshko_erk56_study/pythode_standard.sh`

  `akroshko_erk56_study/akroshko_erk56_plot.py`

  `basic_examples/*`

  `reference_solution_generate/*`

The `basic_examples/` directory contains the code used for the figures in Sections 2.2 and 2.3. The scripts serve to demonstrate basic usage of `pythODE` without a `PostgreSQL` database and are further described in Section 5.4.2. The `akroshko_erk56_study/` directory contains the numerical experiments for most of the figures in this chapter. The `reference_solution_generate/` directory has a similar structure and function to the `akroshko_erk56_study/` directory. These directories are structured in such a way that it is easily possible to copy them and create similar ones for different studies or purposes anywhere in the file system.

These directories have many supporting SAGE, PYTHON, and shell scripts used by `pythODE` to support this study. These study-specific scripts were actually a larger amount of development effort than the core functionality of `pythODE` just described and will be extremely important templates for building on `pythODE` and this study for future studies. These include PYTHON scripts for setting up specialized performance experiments, SQL scripts for data manipulation within the `PostgreSQL` database, and PYTHON scripts for plotting the final results. However, these scripts are in a constant state of change, specific to the needs of a particular study, and are not specifically discussed further in this thesis. Accordingly, these scripts specific to this study will be released along with `pythODE` because it is expected that others that want to use `pythODE` will have to build on the scripts supporting this study for future studies. Of course, studies that differ substantially from the one presented in this thesis would require substantially more additional development work to build on this study than ones that are more similar.

5.4.1 The `IVPSolver` class and modules

The `pythODE` package has a modular structure that is a direct descendent of that used and described in the paper on `odeToJava` [106]. However, the specific software design presented in this chapter is based heavily on the insights gained not only from building the `odeToJava` package [106], but also the author's Master's thesis [103], the publications that `pythODE` was used to support [104, 105], and the specific needs of this study.

Before discussing the `IVPSolver` class itself, it is important to introduce the `properties` attribute of that class. The different pieces of information needed for an IVP solver are stored in a PYTHON dictionary that is the `properties` attribute of the `IVPSolver` class. The keys indexing this dictionary are strings that are globally unique within `pythODE` and correspond to specific pieces of information used by an IVP solver implemented within `pythODE`. The `properties` attribute is passed to all modules, after which these modules can read or modify any necessary values. Using a PYTHON dictionary leads to much simpler code in comparison to the more complex data structures described for `odeToJava` [106]. Using a simple PYTHON dictionary lacks some type safety (but also some of the additional complexity sometimes associated with type safety) that many JAVA data structures often enforce and was present in the data structure analogous to `properties` in `odeToJava`. However, strict type-safety has been found unnecessary for a research-focused software package of the relatively small size of `pythODE`.

Object-relational managers (ORMs) such as `SQLAlchemy`[23] were examined briefly by the author. However, it was decided that at this time that full-featured ORMs had many features that were unnecessary, would probably have negative impact on performance and increase software system complexity, make it harder to make ad-hoc queries of the database that proved so essential, and make it harder to write tools in languages other than PYTHON that want to interact with the `PostgreSQL` tables from `OCSage` and `pythODE`. Accordingly, it was decided to use only built-in `PostgreSQL` datatypes with the simpler `psycopg` library[24] that was already mentioned in Section 4.1. However, ORMs may be useful for future numerical computing projects incorporating RDBMSs.

Instead, for easy serialization or storage in a database, objects in `pythODE` are often stored using keys that incorporate the same notation of the code placeholders in this thesis, namely, by using double angle brackets `‘«...»’` around the name of the variable. These placeholders are referred to as *string placeholders* in this thesis. This translation to string placeholders became important for avoiding some issues with serializing some objects with the combinations of packages that the author and Adam Preuss encountered, for example, incorporating code to support cluster computing with packages such as the `multiprocessing` and CYTHON. For using a database, string placeholders remain important because rather than trying to serialize PYTHON objects directly in the database, only the human readable string placeholder is stored. This can save space over storing PYTHON objects directly and is beneficial for easily doing many operations in the database, such as comparisons, sorting, etc.. The only drawbacks are that there need to be non-interactive passwordless SSH logins on every machine in a cluster and each machine requires an identical copy of the `OCSage` and `pythODE` source code. However, if a research group has a sufficiently advanced setup (for instance, easy to configure and reliable non-interactive SSH logins between different computers in combination with modern version control tools), this is not a significant burden.

An example of a string placeholder is a reference to the `IVPSolver` object that is stored in a `PostgreSQL` database as the text string `‘«IVPSolver»’`. After being read by `pythODE` from the database, each string placeholder is replaced with the actual `pythODE` object that is global to the `pythode` package.

Some examples of information stored in `properties` and the dictionary keys used to index them for the actual RK steps when solving an IVP are:

- The values stored using the `‘initial time’` and `‘initial values’` keys define the initial conditions (2.1b) associated with a particular ODE (2.1a), which together define the IVP (2.1).

- The values stored using the `‘current t’` and `‘dt’` keys keep track of the time variable t at the beginning of the current step and the stepsize of the current step respectively.

- The values stored using the `‘current’` and `‘next’` keys are the numerical solution stored as NUMPY

[23]https://www.sqlalchemy.org/
[24]http://initd.org/psycopg/

arrays at the beginning and end of the current step respectively. Of course, the solution at the end of the current step is not necessarily present or defined until the appropriate module computes it.

- The values stored using the RK method `'stages'` key are the stages $\mathbf{k}_i$ from (2.33) as a two-dimensional NumPy array where each stage corresponds to a row of this array.

- The values stored using the `'order'` and `'embedded order'` keys are the orders of the two components of an RK pair respectively. When `pythODE` is used in step-doubling or constant-stepsize mode, the `'embedded order'` key may not be present. It should be noted that these values do not necessarily correspond to a higher- or lower-order component of an embedded pair. The value stored by the `'order'` key is the order of the component of an RK pair used to advance the solution, and the value stored by the `'embedded order'` key is the order of the component of an RK pair used only to provide an error estimate.

Some examples of information stored in `properties` used for the step controller are:

- The values stored using the `'relative tolerance'` and `'absolute tolerance'` keys are the relative and absolute tolerances from (2.58), respectively. The `pythODE` package transparently handles having single floating point values or NumPy arrays, so individual tolerances can be specified for each component as described in Section 2.6.

- The values stored using the `'maximum stepsize'`, `'minimum stepsize'`, and `'safety factor'` keys correspond to $a_{\max}$, $a_{\min}$, and α given by (2.60).

Some examples of properties used for output from the modules in `modules/stats.py` are:

- The values stored using the `'statistics accepted steps'` and `'statistics rejected steps'` keys are the number of steps accepted and rejected by the step controller.

- The value stored using the `'rejected times'` keys are a list of tuples of the form `(properties['current t'],properties['step control eps'])` that give the times steps were rejected and the value of $\grave{\kappa}_{n+1}$ from (2.59).

- The values stored using the `'statistics final reference solution'`, `'statistics final error'`, and `'statistics final error norm'` keys are, respectively, a reference solution at the final time as a NumPy vector, the error at the final time based on this reference solution as a NumPy vector, and the root mean square (RMS) of this error at the final time.

- The values stored using the `'full solution interval'` and `'full solution interval times'` keys represent a numerical solution at a user-specified t found by interpolation if an optional module for dense output is present. As mentioned before, dense output by interpolation is not discussed in this thesis but more details can be found in the source code of `pythODE`.

In addition, there are properties that are also used to control `IVPSolver` itself. Some examples are:

- The lists of modules to run by `IVPSolver` are stored in Python lists using the `'integration modules'`, `'step control modules'`, and `'statistics modules'` keys. The modules in each list are run in the same order that keys are given in the lists of modules. The `odeToJava` package ordered modules automatically for assisting with user interfaces that were originally planned. However, this added complexity can detract from flexibility as a research platform and it is generally obvious the order the modules should be in anyways. An example of why separating the modules into different lists is important is step-doubling error control (2.57), where it is important to ensure the modules in `'step control modules'` and `'statistics modules'` are only run once, whereas the modules in `'integration modules'` are run three times per step. The small amount of additional complexity that comes from separating the modules into lists also allows significant code-sharing between different types of solvers.

- The value stored by the `'reject'` key is `False` when beginning a step. Any module or part of the solution process can set this property to `True` when appropriate conditions force rejection of the current step. Most modules check the `'reject'` property for a rejected step before their main functionality is used. This is especially important if invalid values could cause runtime errors. If the modules themselves do not handle some exceptions, the `pythODE` `IVPSolver` class can appropriately handle many common mathematical exceptions, such as divide-by-zero or invalid values, that otherwise might occur in the modules as a rejected step. However, experience has shown that unnecessarily frequent exception handling can be significantly less efficient than a simple check of whether the step has already been rejected. However, exception handling is often more efficient than checking the validity of all of the values in a large NumPy array. Therefore, specific choices must be made for what is most appropriate for a particular part of the code. In addition, for modules that implement tasks such as file operations, it is important to know whether a particular step is rejected before potentially irreversible tasks such as changing files on disk, where relying on the exception handling in the `IVPSolver` class may be risky.

- The value stored by the `'abort'` key remains with the value of `False`, except when an unrecoverable error or condition has occurred. When a module sets the `'abort'` property to a Python object other than `False`, the solver terminates. A string giving the reason for aborting is often the non-Boolean value corresponding to the `'abort'` key. This reason can then be returned to the user, stored in the `PostgreSQL` database, or otherwise used further. Common reasons for aborting an IVP solution are too many rejected steps in a row, exceeding a specified maximum number of steps, mathematical conditions such as a singular linear system, or the presence of invalid or undefined values that indicate an unrecoverable state.

- The value stored by the `'successful'` key is only set to `True` at the end of the `run` method (described below) of the `IVPSolver` class and indicates the solver has run properly. Note that a solver that has aborted with the `'abort'` key corresponding to `True` will still have the `'successful'` key set as

True. The 'successful' key therefore corresponds to successful completion of the IVPSolver (without unhandled exceptions) rather than a successful solution of the IVP, which is instead quickly indicated by 'abort' as False and 'successful' as True. The 'successful' key is extremely important when solving hundreds of thousands of combinations of numerical method, IVP, tolerance, or other parameters because it is important to readily determine from the data in a PostgreSQL database if there were any unhandled exceptions without checking the output manually.

It can be clearly seen that nearly all information that must be queried or otherwise communicated is stored in the dictionary corresponding to the properties attribute of the IVPSolver class. This greatly contributes to the flexibility of pythODE through modules and makes integration with a database straightforward.

Because all modules have read/write access to the properties attribute and because the use of certain properties can depend on context, this means that careful design and appropriate use of modules is still required. However, it has been found through experience that the properties attribute is much easier to work with than the more complex data structures and safeguards that are described in the paper for the odeToJava package [106].

In order to set up the IVPSolver class, the appropriate information must be provided that defines the IVP to be solved, the IVP method to be used, and any other information the solver requires. An example of initializing an IVPSolver class as the solver_object variable and then solving the IVP as done in pymathdb/db_solver.py source file is given in Listing 5.1 where examples of the arguments are given below.

Listing 5.1: An example of calling IVPSolver

```
outgoing_properties = solver_object(method_default_properties=method_properties,
                                    ode_default_properties=ode_properties,
                                    incoming_properties=incoming_properties_dict).run(globals())
```

An example of a value for the method_default_properties keyword argument using the DP5(4)$_{6(7)}$ ERK pair (2.79) is given in Listing 5.2 where all of the values in this dictionary end up being merged into the properties attribute. The 'dense' property that has the value '«DOPR54Dense»' that is a string placeholder for a class providing dense output using interpolation, which is not further discussed in this thesis.

Listing 5.2: An example of calling IVPSolver

```
# Dormand-Prince 5(4) method
dopr54_tableau =
  {'ERK A':sp.array([[0.,           0.,           0.,           0.,        0.,           0.,
      ↪ 0.],
                     [1./5.,        0.,           0.,           0.,        0.,           0.,
                         ↪ 0.],
                     [3./40.,       9./40.,       0.,           0.,        0.,           0.,
                         ↪ 0.],
                     [44./45.,     -56./15.,      32./9.,       0.,        0.,           0.,
                         ↪ 0.],
                     [19372./6561.,-25360./2187.,64448./6561.,-212./729., 0.,             0.,
                         ↪ 0.],
                     [9017./3168., -355./33.,     46732./5247., 49./176., -5103./18656.,  0.,
                         ↪ 0.],
                     [35./384.,     0.,           500./1113.,   125./192.,-2187./6784.,   11./84.,
                         ↪ 0.]]),
   'ERK b':sp.array([35./384.,     0.,           500./1113.,   125./192.,-2187./6784.,   11./84.,
       ↪ 0.]),
   'ERK b embedded':sp.array([ 5179./57600., 0., 7571./16695.,
       ↪ 393./640.,-92097./339200.,187./2100.,1./40.]),
   'dense':'<<DOPR54Dense>>',
   'order':5,
   'embedded order':4,
   'ERK FSAL':True}
```

Implementing IVPs in `pythODE`

The `ode_default_properties` keyword argument from Listing 5.1 contains the information required to define the IVP being solved. An example that gives an implementation of the "Arenstorf orbit" IVP (2.15) in `pythODE` is

```
@All(globals())
def arenstorf_orbit(t,y):
    yp = sp.empty_like(y)
    mu = 0.012277471
    muhat = 1.0 - mu

    d1 = ((y[0] + mu)**2. + y[1]**2.)**1.5
    d2 = ((y[0] - muhat)**2. + y[1]**2.)**1.5

    yp[0] = y[2]
    yp[1] = y[3]
    yp[2] = y[0] + 2.*y[3] - muhat*(y[0]+mu)/d1 - mu*(y[0] - muhat)/d2
    yp[3] = y[1] - 2.*y[2] - muhat*y[1]/d1 - mu*y[1]/d2
    return yp

__all__.append('arenstorf_orbit_default_properties')
arenstorf_orbit_default_properties = {'ivp name':'Arenstorf orbit',
                                      'rhs': '<<arenstorf_orbit>>',
                                      'abbreviation':'aren',
                                      'initial time':0.,
                                      'initial values':sp.array
                                          ↪ ([0.994,0.,0.,-2.00158510637908252240537862224]),
                                      'final time':17.0652165601579625588917206249,
                                      'final reference solution':sp.array
                                          ↪ ([0.994,0.,0.,-2.00158510637908252240537862224])}
```

The `'rhs'` property of `arenstorf_orbit_default_properties` has the value of `'«arenstorf_orbit»'`, which is a string placeholder for function `orbit` that defines the RHS. The `'abbreviation'` property gives a relatively compact string for referring to a particular IVP for tasks such as generating figure titles, figure legends, filenames, etc. The `'final reference solution'` property is a known reference solution at the final time that is useful for performance evaluation, which for the "Arenstorf orbit" IVP (2.15) is identical to the initial values.

Although not the subject of the efficient ERK pairs constructed in this thesis, `pythODE` has features to make the definition of PDEs of different sizes and sets of parameters relatively easy. An example giving an implementation of the 1D advection IVP (2.17) with third-order finite differences (2.19) in `pythODE` is

```
@All(globals())
@parametrizable
def advection_initial_values(p):
    initial_values    = sp.zeros(p['n'])
    window = int(round(0.2/p['final time']*p['n']))
    sd     = int(round(window/10))
    initial_values[0:window] = signal.gaussian(window,sd)
    return initial_values

@All(globals())
@parametrizable
class Advection1DThirdOrderUpwind(object):
    def __init__(self,properties):
        self.n = properties['n']
        lower2  = sp.arange(2,self.n)
        lower   = sp.arange(1,self.n)
        central = sp.arange(0,self.n)
        upper   = sp.arange(0,self.n-1)
        self.M                     = sp.zeros((properties['n'],properties['n']))
        self.M[lower2,  lower2-2]  =  1./6.
        self.M[lower,   lower-1]   = -6./6.
        self.M[central, central]   =  3./6.
        self.M[upper,   upper+1]   =  2./6.

    def __call__(self,t,y):
        return -sp.dot(self.M,y)*float(self.n)

__all__.append('thirdorder_upwind_default_properties')
thirdorder_upwind_default_properties = {'rhs':'<<Advection1DThirdOrderUpwind>>',
                                        'ivp name':'Advection 1D third order upwind',
                                        'abbreviation':'advection-1D-3rd',
                                        'n':100,
                                        'initial time':0.,
                                        'initial values':'<<advection_initial_values>>',
                                        'final time':0.5}
```

The Python code defining the other semi-discretization methods described in Section 2.2 for the advection IVP (2.17) are similar. Better efficiency could be obtained by eliminating explicit storage of arrays, but the code above is sufficient for the purposes of demonstration. Objects referenced in the `properties` dictionary that should be initialized with the `properties` dictionary itself are indicated with the decorator `@parametrizable`. When changing the value of `'n'` in the `thirdorder_upwind_default_properties` dictionary, the `Advection1DThirdOrderUpwind` class is always initialized with the correct value. Using the `advection_initial_values` function ensures the `'initial values'` property is a NumPy array of the correct size that reflects the same initial conditions for the underlying PDE regardless of the value of the `'n'` property. Using objects marked `@parametrizable` helps immensely for IVPs that have many parameters dependent on each other, such as semi-discretized PDEs, and where it is desirable to do considerable experimentation with these parameters. Notice that the above example does not have `'final reference solution'` because there is no known exact or analytic reference solution for many semi-discretized PDEs, and therefore an appropriate value or `@parametrizable` function must be implemented on a case-by-case basis. For instance, if the `'final reference solution'` property is required but there are many different values of `'n'`, precomputed values of the reference solution could be stored in the `PostgreSQL` database instead.

IVPs from celestial mechanics, which are the focus of the optimal ERK pairs constructed in this thesis, are also amenable to vectorization for all but the smallest IVPs. Knowing how to take advantage of NumPy array operations and using good implementational practices, which are further discussed in Section 5.5 below, are essential for many aspects of this study. In particular, the vectorized code in Listing 5.4 is 20–30 times faster than the basic unvectorized code in Listing 5.3. This is especially important because for the figures in Section 5.10 involving the "nine planets" IVP, they often took 24 hours each to generate, and the "nine planets" IVP itself was about 70% of the computational cost of these experiments.

Listing 5.3: Unvectorized Python code for the "nine planets" IVP.

```
  # mass of sun in AU^3 d^-2 mu_Sun=(0.01720209895)**2.

nine_planets_constants = {'mu':[mu_Sun,             # mass of Sun
                                mu_Sun/6023600.,    # mass of Mercury
                                mu_Sun/408523.5,   # mass of Venus
                                mu_Sun/328900.53,  # mass of Earth
                                mu_Sun/3098710.0,  # mass of Mars
                                mu_Sun/1047.355,   # mass of Jupiter
                                mu_Sun/3498.5,     # mass of Saturn
                                mu_Sun/22869.,     # mass of Uranus
                                mu_Sun/19314.,     # mass of Neptune
                                mu_Sun/3000000.]}  # mass of Pluto

@All(globals())
def nine_planets_rhs_basic(t,y):
    yp           = sp.zeros(60)
    y_pos        = sp.zeros((3,10))
    ypp          = sp.zeros((3,10))
    r_norm       = sp.zeros((10,10))
    r_norm_cubed = sp.zeros((10,10))

    for j in range(10):
        for k in range(3):
            y_pos[k][wget: missing URL
    for i in range(10):
        for j in range(10):
            for k in range(3):
                if i != j:
                    r_norm[j][i] += (y_pos[k][j] - y_pos[k][i])**2.0
    for i in range(10):
        for j in range(10):
            if i != j:
                r_norm_cubed[j][i] = r_norm[j][i]*sp.sqrt(r_norm[j][i])
    for i in range(10):
        for j in range(10):
            for k in range(3):
                if i != j:
                    ypp[k][i] += nine_planets_constants['mu'][j]*((y_pos[k][j] - y_pos[k][i])/
                        ↪ r_norm_cubed[j][i])
    # i is just an index here
    for i in range(30):
        yp[i] = y[30 + i]
    for k in range(3):
        for i in range(10):
            yp[30 + i*3+k] = ypp[k][i]
    return yp
```

Listing 5.4: Vectorized PYTHON code for the "nine planets" IVP.

```
  NINEPLANETS_CONSTANTS =
  sp.repeat(sp.repeat(sp.array(nine_planets_constants['mu'])[sp.newaxis,:],3,axis=0)[:,:,sp.newaxis
      ↪ ],10,axis=2)

NINEPLANETS_ZERO      = [(0,0,0,0,0,0,0,0,0,0,
                          1,1,1,1,1,1,1,1,1,1,
                          2,2,2,2,2,2,2,2,2,2),
                         (0,1,2,3,4,5,6,7,8,9,
                          0,1,2,3,4,5,6,7,8,9,
                          0,1,2,3,4,5,6,7,8,9),
                         (0,1,2,3,4,5,6,7,8,9,
                          0,1,2,3,4,5,6,7,8,9,
                          0,1,2,3,4,5,6,7,8,9)]

@All(globals())
def nine_planets_rhs(t,y):
    y_pos                          = y[0:30].reshape((3,10,1),order='F')
    y_pos_diff                     = y_pos.repeat(10,axis=2)
    y_pos_diff                    -= y_pos_diff.swapaxes(1,2)
    y_pos_diff_square              = y_pos_diff*y_pos_diff
    r_norm                         = y_pos_diff_square.sum(axis=0)
    r_norm                       **= 1.5
    y_pos_diff                    *= NINEPLANETS_CONSTANTS
    y_pos_diff[:]                 /= r_norm
    y_pos_diff[NINEPLANETS_ZERO]   = 0.
    ypp = y_pos_diff.sum(axis=1)
    yp=SP_CONCATENATE((y[30:60],ypp.flatten('F')))
    return yp
```

Finally, for the `IVPSolver` class, the property `'incoming_properties'` refers to all other properties required and overrides keys in `'method_default_properties'` and `'ode_default_properties'`. The values in `'ode_default_properties'` also override keys in `'method_default_properties'`. Although this feature is not current used, it could be useful for performance experiments where some of the parameters associated with a numerical method, such as step control parameters, need to have specific values for a particular IVP. It could become especially important when studying IRK methods that often require specific tuning of the parameters for routines, such as non-linear equation solvers, to both the integration method used and the parameters of the IVP being solved.

Performance measurement of `pythODE`, discussed in detail in Section 5.5 below, shows that much of the computational cost of `pythODE` for all but the smallest IVPs occurs in the RHS evaluations. Therefore, in the future, despite some significant optimizations possible for PYTHON code implementing the RHS evaluations and the significant possibility of leveraging the performance of low-level libraries, it will likely be important to allow efficient implementations of numerical code itself. Numerical code rarely requires the dynamic typing and automatic memory management that account for much of the performance lost for languages such as PYTHON, in comparison to lower-level languages. Calling C directly from PYTHON is fairly straightforward, although the need to recompile modules corresponding to numerical code can add significantly to software system complexity and increase the barriers to usability. The better possibility of CYTHON has already been mentioned, and CYTHON is actually used widely in projects such as SAGE. Although not used for this study, for simply implementing RHSs or other numerical formulae CYTHON can be relatively straightforward and a limited usage for specific numerical formula, where dynamic typing is not beneficial, would not substantially add to the overall complexity of the software system. It should be noted that solvers from the `scipy.integrate` package for integration of quadrature problems (2.2) allow the RHS to written as a callback in a low-level language using the `scipy.LowLevelCallable` class.[25] The `scipy.LowLevelCallable` class or an adaptation are another option for future versions of `pythODE`. Once an efficient methodology is implemented for writing numerical code in a low-level language, the overhead of `pythODE` solver itself may be significant enough that it would need to be improved. Careful design would be required to implement key components in a low-level language while not making the software system too complex and inhibiting usability.

Calling the `IVPSolver` class and using `IVPSolverModules`

The methods of an `IVPSolver` class that interest a user of the class are given in Listing 5.5.

Listing 5.5: Overview of the internals of `IVPSolver`.

```
class IVPSolver(object):
    def set_defaults(self):
        <<...>>

    def initialize(self):
        <<...>>

    def step(self):
```

[25] https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.quad.html

```
            <<...>>

        def finalize(self):
            <<...>>

        def run(self,theglobals,queue=None):
            <<...>>
```

The `set_defaults` method allows setting some default (but able to be overridden) values from `properties`, `initialize` sets up the solver, `step` is repeatedly run for each step, and `finalize` performs tasks required after the numerical solution is finished such as updating `properties` to reflect this or closing open files. The `run` method is run by the user or script, such as seen in Listing 5.1, and generally needs access to the global variables from where the solver is run in order to be able to substitute appropriate placeholder strings '`«...»`' with the appropriate object.

An example of an `IVPSolverModule` subclass to count the total steps is given in Listing 5.6.

Listing 5.6: An example of subclassing `IVPSolverModule`.

```
@All(globals())
class TotalSteps(IVPSolverModule):
    def set_defaults(self,properties):
        pass

    def initialize(self,properties):
        self.count = 0

    def step(self,properties):
        self.count += 1

    def finalize(self,properties):
        properties['statistics steps'] = self.count
```

The `set_defaults` method allows defaults to be set that incorporate information from the `properties` dictionary. Each of the methods `initialize`, `step`, and `finalize` in an `IVPSolverModule` are called from the corresponding method in the `IVPSolver` subclass.

5.4.2 Basic usage of `pythODE` for solving simple IVPs and plotting the results

In the `basic_examples/` directory, there are relatively simple programs that generate the figures for the examples of the solutions of IVPs from Sections 2.2 and 2.3. These relatively simple programs also serve as basic usage examples for `pythODE`. An example of a function to solve the "Arenstorf orbit" IVP (2.15) is given by Listing 5.7.

Listing 5.7: An example of basic `pythODE` usage for solving the "Arenstorf orbit" IVP (2.15).

```
def arenstorf_orbit():
    solver_object            = ERKEmbeddedEstimateWritingIVPSolver
    method_properties        = dopr54_tableau
    incoming_properties_dict = {'relative tolerance':1e-10,
                                'absolute tolerance':1e-10,
                                'full solution write interval':0.01}
    outgoing_properties_keys = {'statistics time',
                                'statistics rhs count',
                                'statistics steps',
                                'statistics accepted steps',
                                'statistics rejected steps',
                                'statistics final error norm',
                                'successful',
                                'solution times values',
                                'full solution interval',
                                'full solution interval times'}
    ode_properties              = arenstorf_orbit_default_properties
    outgoing_properties         = solver_object(method_default_properties = method_properties,
                                                ode_default_properties    = ode_properties,
                                                incoming_properties       = incoming_properties_dict
                                                ).run(globals())
    outgoing_properties_dict = {k:outgoing_properties[k] for k in outgoing_properties_keys}
    pprint(outgoing_properties_dict)
    x = zip(*outgoing_properties_dict["full solution interval"])[0]
    y1 = [e[0] for e in zip(*outgoing_properties_dict["full solution interval"])[1]]
    y2 = [e[1] for e in zip(*outgoing_properties_dict["full solution interval"])[1]]
    fig = plt.figure()
    ax = plt.subplot(111)
    ax.plot(y1,y2)
    ax.plot(0.0,0.0,'ok')
    ax.plot(1.0,0.0,'.k')
    ax.set_xlabel('$x(t)$',size=16)
    ax.set_ylabel('$y(t)$',rotation=0,size=16)
    ax.tick_params(labelsize=12)
    ax.set_xlim([-1.5,1.5])
    ax.set_ylim([-1.5,1.5])
    ax.axis('equal')
    ax.grid(True)
    plt.show(ax)
    fig.savefig('example_orbit.pdf', bbox_inches='tight')
    plt.close(fig)
```

This code solves the "Arenstorf orbit" IVP (2.15) and shows a solution corresponding to Figure 2.7 in a window in addition to writing it to a `pdf` file. The `ERKEmbeddedEstimateWritingIVPSolver` object is a helper class in `experiment_common.py` that sets up the appropriate modules for this usage.

5.4.3 Using the `PostgreSQL` database for setup and coordination of `pythODE`, along with the analysis of performance data

As mentioned in the introduction to this chapter, in order to do thorough performance testing and support detailed analysis of the resulting performance data comparing RK formulae and pairs, `pythODE` is integrated with a `PostgreSQL` database.

Some performance testing done for this study can generate up to several gigabytes of data, for example, when several thousand $5(4)_6$ ERK pairs are being used to solve all of the IVPs that are described below along with a wide range of tolerances and step control parameters. As discussed with `OCSage` in Chapter 4, although working with this amount of data has now become commonplace, when the data possesses a high degree of complexity and heterogeneity (as opposed to simple numerical arrays that are a uniform data type) and requires further manipulation, simple programs in imperative languages can be error-prone and inefficient, meaning a database should be used.

In principle, many of the manipulations commonly done to data within a database can be accomplished by imperative programming in common languages such as FORTRAN, C++, PYTHON, or MATLAB. However, even when dealing with only a limited range of methods, e.g., only ERK methods as this thesis does, imperative code that supports the management of data becomes extremely brittle and error prone as an experiment proceeds from `OCSage`, to `pythODE`, to performance data, and finally to analysis and plotting. A huge advantage to `PostgreSQL` (and other relational databases) is that they use the domain-specific and declarative language SQL (structured query language) [43, 44, 92]. By using a declarative language to specify only the result required from a particular manipulation of data, this eliminates many potential issues that would arise from manipulating data in imperative languages. Languages such as SQL also make it extremely easy to add additional columns of data in a way that often does not require modifying existing code. During the course of this study, `PostgreSQL` was often used for preliminary study of data in ways that are not directly incorporated into the figures in this chapter.

A principal disadvantage to using an SQL database can be relatively poor performance for some usage cases, which is a common issue with declarative languages. Routine data manipulations done on several thousand data points supporting many components of this study often took several milliseconds to several seconds. Some manipulations on data from searches done by `OCSage` in Chapter 4 can take tens or hundreds of seconds. However, good SQL programming practices can often greatly speed up database operations that are extremely slow when implemented naively. In addition, code written ad-hoc to manage data in languages such as C or PYTHON can easily end up being much slower than the optimized and well-tested algorithms, queries, and planning that RDBMSs employ.

Due to these performance limitations, an SQL database can only play a complementary role to operations on numerical arrays provided by more conventional libraries in languages such as FORTRAN, C++, PYTHON, or MATLAB. These numerical array operations can easily reach dozens of GFLOPs (billion floating-point operations per second) on common desktop computers,[26] whereas typical uses of SQL databases often process complex and heterogeneous data types at a much lower rate but using much higher-level operations than just simple arithmetic. Obviously, SQL or database operations in general do not belong in the inner loops of numerical algorithms. Despite the performance drawbacks in some cases, many of the common applications where SQL is used also require a great deal of flexibility and adaptability for the data handling and processing. This is where in comparison to imperative languages, a declarative language such as SQL means the ever-changing requirements of experimental and research computing require much less development and troubleshooting time, which is typically much more expensive in terms of dollars than high-level operations that require seconds to run, but may only be executed a small number of times each time a software package is run.

Although the IVP solver in `pythODE` just described in Section 5.4.1 can be used without a `PostgreSQL` database, the database quickly became an indispensable software component for conducting this study. When dealing with disparate requirements and multiple software components, it is often impossible to have the same data structures across projects or even for different studies or components of the same project. Therefore, some sort of more advanced data management and manipulation is required. For instance, data may be required in significantly different configurations for searches of the space of free parameters such as those done with `OCSage`, performance testing with `pythODE`, or when used further for analysis or visualization. As this study unfolded, data manipulations that are extremely convenient using an RDBMS but that would be more difficult, brittle, and error-prone in an imperative language often became necessary. In addition,

[26] https://www.tcm.phy.cam.ac.uk/~mjr/linpack/

the ease of manipulating data allows quick ad-hoc data manipulations to explore the data before the final presentation of the results of a study. This versatility and applicability of SQL databases can be seen by the popularity of the `SQLite` database as an internal component in many well-known applications.[27]

Although the many complex features of an SQL RDBMS may be more than necessary for simple numerical data in a tabular format, many of the more complex features proved useful, and it is fortunate they were available at some stages of this study. Although there are types of databases other than SQL-based RDBMSs available, in a book on PYTHON data analysis in astronomy from 2014 by several prominent authors [89, pg.44] it was noted that `Nosql` databases are currently inefficient for tabular and array-based data, as well as being immature compared to SQL-based RDBMS. Considering that the field of astronomy deals with enormous amounts of numerical data, it is a good place to seek guidance on appropriate techniques for numerical data. There are also databases focused on the needs of scientific computing and data such as `SciDB`,[28] which was noted as an alternative to SQL RDBMs because the latter may not be entirely suitable for extremely large amounts of scientific data [89, pg.44]. However, for `OCSage` and `pythODE` it was decided that it is currently better to build the software system out of well-known components that many people are likely to have experience with.

The `db_solver.py` program and its helpers

The `pymathdb/db_solver.py` program is what actually calls the appropriate `IVPSolver` subclasses to solve the desired IVPs once the appropriate table in the `PostgreSQL` database is set up (the format of this database table is described below). Another script used is the `pymathdb/db_watcher.py` program, which automatically assigns chunks of new work, i.e., the sets of IVPs to be solved at one time by a single computer, when each computer is done with its current chunk of work.

Using the support for concurrent operations over the network that `PostgreSQL` provides for cluster computing has a much lower throughput than commonly used cluster computing tools for scientific computing, such as the OPENMPI implementation of MPI [63, 65]. However, when performance testing IVPs, for each numerical IVP solution that typically takes up to several seconds, only a small amount of data usually needs to be communicated to determine the particular IVP to be solved, the numerical method to be used, the solution at the final time, and the statistics about the solution process. This relatively low rate of data communication is well within the capabilities of a `PostgreSQL` database operating over a typical LAN (local area network). That `PostgreSQL` (and most other SQL) databases are specifically designed to maintain consistency despite multiple concurrent transactions from many different programs that may be on many different computers is an important reason why using `PostgreSQL` gives such an enormous benefit to this study. Other approaches such the socket-based programming that `pythODE` originally used or the many available software components for cluster computing, such as OPENMPI, would require additional complexity to achieve the

[27] https://www.sqlite.org/mostdeployed.html
[28] https://www.paradigm4.com/

same functionality anyways.

However, it should be noted that despite this use case being well within the capabilities of a PostgreSQL database and data rates on typical LANs, good SQL programming practices are still required to avoid bottlenecks. This includes ensuring that many SELECT and UPDATE operations are done at once (the defaults are 32768 and 4096, respectively). In experiment_common.py appropriate and configurable delays are built into pymathdb/db_solver.py to avoid overloading the database while ensuring virtually no efficiency is lost (by default each computer checks for new work every 10 seconds once it has run out of work). If a particularly long running IVP solution is one of the last ones in a batch of work and fewer than the available cores are used, the pymathdb/db_solver.py script starts on the next chunk of work with the free cores. Because interesting performance experiments invariably seem to have at least some combinations of IVPs and IVP methods that take inordinately long, assigning new work once the cores are underutilized often gave 2–3 times better performance in comparison to waiting for all work to finish on a computer. With all of these details taken care of, all cores on all of the machines used are typically at least utilized to at least 90% and the utilization is usually greater than 95%. For using pythODE on different clusters of computers other than the one used for this study or for different types of experiments some trial, error, and iterative adjustment to the tuning parameters contained in experiment_common.py would likely be required. It is also likely that for some use cases, pymathdb/db_solver.py could easily be modified for use on clusters of computers connected at network speeds lower than a typical LAN, allowing geographically distributed computers to be used.

There is currently no provision to manage computers that go offline or come online while pythODE is running. Because the pymathdb/db_solver.py script is just over 300 lines of code, more sophisticated features can probably be implemented in a straightforward manner. However, the time spent testing the additional code (several thousand lines may be required) to obtain this next level of robustness should not be underestimated either.

An important contribution that OCSage and pythODE make (OCSage can be used over a LAN using the PostgreSQL database connection but cannot yet run a single search over a cluster of multiple computers) is that they demonstrate how using robust and commonly available software tools (the Python and scientific Python ecosystems, PostgreSQL, etc.) can be used to increase the utility of available but not formally organized computing resources. By leveraging robust and available software tools, only several hundred lines of code beyond the basic numerical functionality of a package such as pythODE is required in order to take advantage of available computing resources.

The code in pythODE can be used as a template that could have wide applicability to compliment tools such as OpenMPI [63, 65], which is better for high data transfer rates but does not include concurrent data storage. The BOINC project, which is used for volunteer grid computing for projects like the well-known SETI@home project, is definitely overly complicated for a single project that needs to leverage up to several dozen computers on a LAN to which an individual or research group may have easy access [5]. The complexity of the BOINC project comes from requirements such as automatically distributing binary versions of the appropriate software to all participants, managing different versions of software, managing computers that go online and offline, automatically adapting to different network speeds and hardware, etc. Although these are all nice features to have, they also substantially increase the complexity of any setup and configuration to far beyond what is practical for a single study on up to a few dozen computers.

OpenMPI is an excellent set of tools for distributing computations, especially for tasks that are not embarrassingly parallel. However, OpenMPI is a single component for facilitating cluster computing and does not specifically address aspects such as the data format and storage. Therefore, OpenMPI would still have to be integrated into a complete software system to manage disk storage and the data structures required, including the important feature of concurrent access to the database. Where OpenMPI could be useful for the future development of `pythODE` is if larger amounts of data communication becomes required for larger problems like PDEs or for experiments where the solutions of larger problems are being saved at every step. In this case, a platform such as OpenMPI could be used in addition to a `PostgreSQL` database for transferring large numerical arrays, although this would definitely add to the complexity of the software system. If storing these larger numerical arrays would be unwieldy for a `PostgreSQL` database, there are databases specifically for storing numerical data that could be used in addition to `PostgreSQL`. However, the author feels that for studies similar to this study, but at a larger scale, that these additional software components would compliment the `PostgreSQL` database rather than replace it because being able to use SQL to manage the experiments was found to be too essential.

For the reasons just described, many existing tools for clustering such as OpenMPI or `BOINC` are not necessarily a good fit to a particular project, even if in principle all the tools under consideration accomplish the same general task. In fact, one of the reasons why the author made `pymathdb` a separate directory is to be able to quickly reuse it for future projects that require a database and cluster computing.

Another reason for using relatively simple programs built on popular tools is that all computers in the cluster only need a standard Python/SciPy, or preferably Sage, installation and a `PostgreSQL` client. All of the software libraries and tools used for this study track the current versions available from the current version of Sage. However, the Sage platform has normally been restricted to Linux or Apple operating systems. Although not tested for this thesis, both a standard Python/SciPy installation and `PostgreSQL` installation can be installed on Windows, and, with minor modifications, Windows computers could be included in a cluster. During the final stages of this study, the announcement came that Sage can now be installed natively on Windows.[29] Given that the vast majority of the libraries used by `OCSage` and `pythODE` are included in a Sage installation, once the Windows version of Sage is robust enough if it is not already, this only increases the number of computers that can easily be incorporated into a cluster.

[29]http://www.sagemath.org/download-windows.html

The details of `db_loader.py`, `db_watcher.py`, and `db_solver.py`

A Python script particular to each type of study is the `db_loader.py` script, which loads the data on the experiments to be run into a database table. The `db_loader.py` script does not need to have that name and can include any functionality the user desires, as long as a `PostgreSQL` table of an appropriate format is created by the script.

The `pymathdb/db_watcher.py` script is run on a designated primary computer of the cluster and controls the partitioning of the IVPs to be solved into chunks of work for each computer. Currently, the maximum chunk size is 32,768 IVPs for each computer to solve before getting more work, but this can be changed easily. The default initial chunk size is chosen so there are initially about four chunks of work per core available, but this can also be configured easily. As the solutions proceed, once each computer has fewer IVPs to solve than its number of cores, a new chunk of work is assigned. With some provisions for load balancing about to be described, this proceeds until no more work can be assigned and all IVPs are solved.

The individual IVPs are currently partitioned into the chunks randomly. Although a publication by the author and supervisor of this thesis has shown that similar operating parameters give similar solution costs [105] and that this can be used to help with parallelism when solving many similar IVPs. However, there is too much variability across the many IVPs tested in this study to currently make effective use of this idea. The correlation between specific parameters and solution costs is probably better used when scaling up the solution of many closely related IVPs in a production setting.

There is significant variability in the solution time of IVPs, especially when doing research into numerical methods, where methods that perform poorly or lose stability for some IVPs often need to be tested for a particular study. Due to this variability, at least a minimal amount of adaptive load balancing is required. When there are fewer than the number of computers multiplied by the remaining number of IVPs to be solved, then the chunk size is recalculated from the remaining number. The default is to allow up to two reductions to occur in order to avoid many iterations of assigning extremely small numbers of IVPs to a computer, and this default has been found to be adequate for the experiments done for this study. For different numbers of computers or if some IVP solutions take particularly long, choosing different default values for chunk size and number of reductions may be necessary.

Listing 5.8: Three sample rows of an input table for `pythODE` split into four groups of columns.

```
   solve_number     |      solver_object        |     method_properties     |            ode_properties
---------------------+--------------------------+--------------------------+--------------------------------------
 3590889128163540992 | <<ERKEmbeddedEstimateIVPSolver>> | <<create_ocsage_tableau>> | <<nonstiff_d1_default_properties>>
 2077833795448340480 | <<ERKEmbeddedEstimateIVPSolver>> | <<create_ocsage_tableau>> | <<nonstiff_d3_default_properties>>
 1609390315293114368 | <<ERKEmbeddedEstimateIVPSolver>> | <<create_ocsage_tableau>> | <<nonstiff_a1_default_properties>>

                                          incoming_properties_keys
---------------------------------------------------------------------------------------------------------------
 {"initial stepsize","relative tolerance","absolute tolerance","ocsage label","ocsage tableau parameters","maximum steps"}
 {"initial stepsize","relative tolerance","absolute tolerance","ocsage label","ocsage tableau parameters","maximum steps"}
 {"initial stepsize","relative tolerance","absolute tolerance","ocsage label","ocsage tableau parameters","maximum steps"}

                                                              outgoing_properties_keys
---------------------------------------------------------------------------------------------------------------------------------------------------
 {"statistics time","statistics rhs count","statistics steps","statistics accepted steps","statistics rejected steps","statistics final error norm",successful}
 {"statistics time","statistics rhs count","statistics steps","statistics accepted steps","statistics rejected steps","statistics final error norm",successful}
 {"statistics time","statistics rhs count","statistics steps","statistics accepted steps","statistics rejected steps","statistics final error norm",successful}

 initial stepsize | maximum steps | relative tolerance | absolute tolerance |  ocsage label   | ocsage sublabel |     ocsage tableau parameters      |   statistics time   | statistics rhs count
------------------+---------------+--------------------+--------------------+-----------------+-----------------+------------------------------------+---------------------+----------------------
            0.001 |       1000000 |                0.1 |              0.001 | erk5sVVIs6emb   | 66-A5           | {-0.05,0.23,0.3,0.92,1}            |                NULL |                 NULL
            0.001 |       1000000 |              1e-09 |              1e-11 | erk5sVVIs6emb   | 66-A4           | {0.225,0.25,0.31,1,0.93}           |                NULL |                 NULL
            0.001 |       1000000 |                0.1 |              0.001 | erk5sVVIs6emb   | 66-A4           | {0.225,0.25,0.31,1,0.93}           |                NULL |                 NULL

 statistics steps | statistics accepted steps | statistics rejected steps | statistics final error norm | successful
------------------+---------------------------+---------------------------+-----------------------------+------------
             NULL |                      NULL |                      NULL |                        NULL | NULL
             NULL |                      NULL |                      NULL |                        NULL | NULL
             NULL |                      NULL |                      NULL |                        NULL | NULL
```

Listing 5.9: The output of `pymathdb/db_solver.py` that updates the last two groups of columns from Listing 5.8 with performance data.

```
 initial stepsize | maximum steps | relative tolerance | absolute tolerance |  ocsage label   | ocsage sublabel |     ocsage tableau parameters      |   statistics time   | statistics rhs count
------------------+---------------+--------------------+--------------------+-----------------+-----------------+------------------------------------+---------------------+----------------------
            0.001 |       1000000 |                0.1 |              0.001 | erk5sVVIs6emb   | 66-A5           | {-0.05,0.23,0.3,0.92,1}            | 0.00419402122497559 |                   78
            0.001 |       1000000 |              1e-09 |              1e-11 | erk5sVVIs6emb   | 66-A4           | {0.225,0.25,0.31,1,0.93}           |   0.165098905563354 |                 3234
            0.001 |       1000000 |                0.1 |              0.001 | erk5sVVIs6emb   | 66-A4           | {0.225,0.25,0.31,1,0.93}           | 0.00392293930053711 |                   78

 statistics steps | statistics accepted steps | statistics rejected steps | statistics final error norm | successful
------------------+---------------------------+---------------------------+-----------------------------+------------
               13 |                        13 |                         0 |            29.4332548826652 | t
              539 |                       520 |                        19 |        1.21985261080538e-07 | t
               13 |                        12 |                         1 |        3.20337426914803e-05 | t
```

The `pymathdb/db_solver.py` script is what actually runs the `pythODE` solver described in Section 5.4.1 on each core by using the function call given by Listing 5.1. An instance of `pymathdb/db_solver.py` is run on each computer and checks the database table for newly assigned work when the number of IVPs to be solved is less than the number of cores being used, by default every 10 seconds. The `pymathdb/db_solver.py` script reads the columns of a row corresponding to assigned work, in order to initialize the inputs shown in Listing 5.1 that are used by the `IVPSolver` object to construct the `properties` attribute. The PYTHON `multiprocessing` library, which is built into the PYTHON standard library, is used to manage the work among the individual cores of a computer. After the solution of an IVP is finished, the newly updated `properties` are used to update the database row corresponding to that IVP with properties such as the final values, execution time, number of steps taken, and any other information about the solution process. If the solution failed, the updated database row reflects this.

The details of the `PostgreSQL` tables supporting `pythODE`

The `pymathdb/db_solver.py` source file requires the `PostgreSQL` table that gives the information describing the experiments to be run to be in a particular format. Some columns that are used to set up subclasses of `IVPSolver` are:

- The `‘solve_number’` column is a unique 64-bit integer (randomly generated and verified to be unique) used as a *primary key* [92, pg.11] that identifies a particular combination of IVP method, IVP, and any other properties. Having this unique identifier helps when managing cluster computing, as well as many other tasks conducted within the database because it avoids the need to compare floating-point values in many cases.

- The `‘solver_object’` column contains a string placeholder for the subclass of the `IVPSolver` class or any other `pythODE` object of similar functionality.

- The `‘method_properties’` column contains a string placeholder that refers to the dictionary that defines the method to be used, such as the example of `dopr54_tableau` given in Listing 5.2. The object referenced in this column is passed to the `run` method using the `‘method_properties’` keyword in Listing 5.1.

- If the value for the `‘method_properties’` column is the string placeholder `‘«create_ocsage_tableau»’` for the corresponding function, then there are additional columns hold data to get the tableau from `OCSage` generated code described in Chapter 4. The `‘ocsage_properties’` column contains the label defined in Chapter 4 that indicates the family, and the `‘ocsage tableau parameters’` column contains an array giving the parameters that the generated code for the family requires; see the example of code `OCSage` generates in Listing 4.3. The `‘ocsage sublabel’` column is not used by `pythODE` when actually solving IVPs, but it is a unique user-defined string for each method that is useful for plotting, analysis, and avoiding floating-point comparisons of the arrays from the `‘ocsage tableau parameters’` column that would otherwise be necessary when doing database operations that do comparisons between methods.

- The `‘ode_properties’` column contains a string placeholder for a `pythODE` object representing the IVP to be solved. The object referenced in this column is passed to the `run` method using the `‘ode_properties’` keyword in Listing 5.1.

- The `‘incoming_properties_keys’` column contains an array of other columns to be read from the database in order to construct an appropriate `properties` dictionary that gets passed to the `run` method using the `‘incoming_properties’` keyword in Listing 5.1.

- The `‘outgoing_properties_keys’` column is an array of the values from the `properties` attribute that are put back into the database from the updated `properties` attribute of the `IVPSolver` class. Using the values of `‘outgoing_properties_keys’` to select only certain values from `properties` is important because many PYTHON objects used in `properties` do not lend themselves to easy representation in the database. In addition, storing all information represented by all `properties` attributes for all IVPs solved also wastes storage in the database and network bandwidth unnecessarily.

The names of the columns of the `PostgreSQL` table correspond directly to the keys in `properties` attribute of the `IVPSolver` subclasses that are used throughout `pythODE`. In more complex studies, a little care is required to make sure column names used for other purposes than `pythODE` do not conflict with keys in the `properties` attribute of the `IVPSolver` subclasses. Like much of both mathematics and computer programming, care can be required to avoid naming and notation conflicts while keeping names and notation meaningful. For instance, to use the module `ERK`, which is a subclass of `IVPSolverModule` and implements ERK formulae (2.33), the $\mathbf{b}$ vector of the Butcher tableau (2.34) is referred to by the key `‘RK b’` instead of `‘b’`. This is because `‘b’` or other single letters are often the well-understood notation for some parameters of some IVPs. The `pythODE` package is not complex enough to require implementing namespaces and the decision was made to make naming parameters for new IVPs as easy and intuitive as possible. An example of a `PostgreSQL` table to be read by `pythODE` is given by Table 5.8. This `PostgreSQL` table updated with performance data is given by Table 5.9.

5.4.4 Setup of the `PostgreSQL` database

To actually run the `pymathdb/db_solver.py` program just described on all computers in a cluster `pythODE` uses the `OpenSSH` tool,[30] which is the standard for secure remote shell logins. For security reasons, the default for many Linux distributions is to have the `PostgreSQL` server only listen for connections from `localhost`. Many firewalls at institutions also block the default port for `PostgreSQL`, i.e., port 5432. Therefore, on

[30] https://www.openssh.com/

computers other than the one running the PostgreSQL database server, an ssh tunnel is used so that connections to localhost port 5433 go to localhost port 5432 of the computer the PostgreSQL database server is on. This different port, i.e., port 5433 rather than 5432, is chosen for computers other than the one the PostgreSQL database server is on, just in case they have their own PostgreSQL database server installation that is unused by pythODE.

5.4.5 Plotting from the PostgreSQL database

Plotting and data analysis are generally done with Python scripts that read directly from the PostgreSQL database, but that are specific to the study being done. Reading from the PostgreSQL database is extremely convenient because plotting and data analysis often require data in completely different configurations than the table in Listing 5.8. A declarative language such as SQL makes it straightforward to manipulate data into the desired configuration and to combine data from different experiments within the database into the desired format. Additionally, functions from OCSage are called from these scripts to add information such as characteristic numbers to the PostgreSQL table for the purposes of data analysis or plotting.

5.5 Comparing the output and performance of pythODE to published IVP solvers

Before performance testing of the newly derived ERK pairs from Chapter 4, it is important to compare and verify the ERK-based solvers in pythODE with existing IVP solvers. Implementational and even language differences (experience has shown the author that despite identical expressions, floating-point arithmetic can sometimes give slightly different values between all of Fortran, C, Matlab, Java, and Python) mean different implementations of the same algorithm do not give exactly the same behaviour or result. However, implementing the same algorithm should still give reasonably similar behaviour. The code released by other researchers, such as Ernst Hairer on his website,[31] can provide excellent reference implementations for many important IVP methods as well as the IVPs themselves. Table 5.1 shows the tolerance, number of total steps, number of accepted and rejected steps, accuracy, and solution time for solving the "Arenstorf orbit" IVP (2.15) for several methods based on existing codes in comparison to their analogous pythODE implementations.

The execution times for Fortran, which are measured using the built-in shell command time, are the minimum from running several dozen trials of the executable from a Linux command prompt. Although better timings could be made by modifying the code to measure execution time between internal points within the code, modifying published code is not necessary for a crude demonstration of the relative execution time using different languages. Measured execution time in pythODE starts at the beginning of the IVPSolver.initialize method and ends at the end of the IVPSolver.finalize method. Because pythODE

[31]http://www.unige.ch/~hairer/software.html

needs to load many libraries and initialize many data structures when individual IVPs are solved and there can be significant overhead due to the platform itself when `pymathdb/db_solver.py` is used, the two locations just described are the best choice for measuring execution time of an algorithm. The timings for OCTAVE are measured using standard built-in methods, i.e., the `tic` and `toc` commands, at similar points within the script.

In Table 5.1, it can be seen that actual solution and numerical behaviour based on the number of steps of `pythODE` solvers is generally within about one percentage point of the codes available from Hairer. It should also be noted that the self-reported steps, accepted steps, and rejected steps from Hairer's code does not add up properly in some cases, whereas the ones from `pythODE` always do, indicating that Hairer's codes have bugs with respect to reporting statistics.

5.5.1 Discussion of the performance of `pythODE`

The solution times in Table 5.1 for `pythODE` are about five times faster than the first naive implementations because `pythODE` has seen considerable optimization in order to make the study in this thesis possible. Despite the advantages of PYTHON in constructing many types of software, a side effect of these advantages can be an extremely high overhead for many operations or library functions. It is especially easy to naively write inner loops, such as the individual steps of the numerical method, with extremely poor performance.

Based on specific timing and optimization work, some general strategies to get the best performance for building software in pure PYTHON or extending `pythODE` are:

- Dictionary key lookups or SCIPY array indexing, e.g., `x[3]`, are fast individually but can severely slow down inner loops. Therefore ensuring individual lookups were only done once by assigning the result to a variable, e.g., `x3=x[3]`, noticeably sped up `pythODE`.

- SCIPY functions are generally faster than `for` loops and individual arithmetic operations. However, many SCIPY functions have considerable overhead when used for relatively small arrays. For example, adding two arrays of size 4 take about 700 nanoseconds and adding two arrays of size 1000 takes about 2000 nanoseconds. Due to non-intuitive relative execution times, it is highly recommended to use the `iPython %timeit` command to allow quickly determining the most effective methodologies. Some specific but not necessarily intuitive examples of performance optimization found while developing `pythODE` are:

    - The scalar functions built into PYTHON such as `math.sqrt` are faster for scalar operations than the corresponding SCIPY functions such as `scipy.sqrt` (in this case for taking square roots). This is because the latter is heavily polymorphic in order to accommodate elementwise square roots on multi-dimensional arrays as well as several different scalar datatypes. This is a general trend that affects a variety of other mathematical functions, such as trigonometric functions.

  - The `numpy.dot` function is 2–3 times faster for multiplying a NumPy array by a scalar than to use the built-in multiplication operator (or `numpy.multiply` function). This is because the `numpy.dot` function wraps `BLAS` without the type-checking that the heavily polymorphic and overloaded multiplication operator requires.
  - The NumPy and SciPy functions providing many linear algebra operations, such as norms, can have considerable overhead due to being polymorphic. If the size of the operation does not change during the solution process, such as for error estimation calculations, e.g., the calculation (2.53) or other operations that depend on the size of a specific IVPs, it can be faster to square and sum explicitly. For example, `a.sum()/length_of_array` is about 3–4 times faster than `scipy.mean(a)` or `a.mean()` for arrays up to a length of at least 60 if the length of the array `a`, i.e., `length_of_array`, is always known.
  - It is faster to add two arrays of the same size where the second array has the same value in every element, rather than the equivalent operation of adding a scalar to an array using the addition operator. Due to this, a considerable speedup of `pythODE` was obtained by always converting scalar tolerances to vector tolerances of uniform size before starting an IVP solution.
  - Slicing operations on arrays, e.g., `a[1:4]`, are extremely useful operations, but more complex slicing can be quite slow. Therefore, if similar slices are to be made in an inner loop, e.g., a selection of rows of the (currently) unchanging $\mathbf{A}$ matrix of the Butcher tableau (2.34), it can be much faster to pre-compute these slices and do a look up instead.

- Python can have relatively high function call overhead compared to other languages. Therefore, once a particular study has developed and become focused enough, it can be advantageous to combine several modules needed for that particular study (for instance measurement statistics such as seen in Table 5.1 or the various components of error estimation and step control) into a single module. However, the original modules used as the basis for the combined module still exist as separate modules. This means that that in future studies, they can be used separately until that study progresses enough that a single optimized study-specific module becomes required. Being able to customize modules in a way that is most efficient for a specific study is an enormous advantage to a modular IVP solver.

In general, even on relatively small IVPs, 90–95% of the measured solution time for `pythODE` comes from the actual time-stepping itself, with most of the rest being the initialization cost for objects from the `IVPSolver` class. For more costly IVPs, such as the "nine planets" IVP integrated for 7200 days in problem time, over 99% of the time is spent doing the time-stepping. Despite this computational cost, there are already mentioned advantages to remaining in pure Python in terms of software system complexity, usability, and allowing straightforward experimentation. However, because so much of the computational cost comes from inner loops, there are some enhancements to SciPy that would be make it much easier to write most efficient pure Python code for implementing numerical methods. Some potential enhancements that would make software like `pythODE` significantly more efficient include:

- Optimized functions for arithmetic involving an array and scalar, in order to avoid using functions like `numpy.dot` or creating special arrays for this purpose.
- Specific functions that efficiently combine operations such as multiplications and additions that appear repeatedly in the inner loops of implementations of numerical methods. For RK methods, the combined operation of `numpy.dot(v,K)*t+y`, where `v` is a vector, `K` is a matrix, `t` is a scalar, and `y` is a vector, appears in many places.
- Optimized functions for common norms and averages, such as relatively basic equivalents to `scipy.mean`. Notably lacking is an efficient function to calculate the widely used root-mean-square.
- Efficient functions to check that an array contains only finite values, not `inf` (infinity) or `nan` (not-a-number). The current best solution is `all(sp.isfinite(y))`, which is two function calls that check the vector `y`. This operation of checking for finite values takes about 4 microseconds for a NumPy array of size 60, which is often a longer time than many arithmetic operations on the same vector `y`.
- A large amount of overhead in the step-controller comes from the code `sp.maximum(abs(yn),abs(yn1))`. Each of the three component functions has a significant execution time. Therefore, a specialized function for maximum absolute value would easily eliminate this enormous source of computational cost.

This list of improvements would greatly help making prototyping inner loops or research-focused tools as efficient as possible. Ultimately once an idea is prototyped, then eventually Cython would be necessary to further improve performance to where performance could be competitive with lower-level languages.

5.5.2 Putting into perspective the relative performances of software based on low-level languages and software based on Python

Ultimately, despite the performance disparities between `pythODE` and the C/Fortran solvers in Table 5.1, it is important to consider how individual software packages are to be used if an appropriate assessment is to be made of their relative performances. In order to support specialized research into numerical methods, `pythODE` stores and then makes available all solution information to each module at each step, which does have an additional performance cost. A similar package to `pythODE` written in C++ could lead to a package with similar performance to the C and Fortran solvers in Table 5.1. However, to be an effective research tool, this hypothetical software package could easily be at least an order of magnitude more complex to develop and use than `pythODE`. At the present time, if it is desirable to remain in Python, it is important to minimize the performance bottlenecks with an appropriate choice of library functions, such as those provided by SciPy. Another good choice if at some point performance becomes extremely critical, would be to incorporate the static typing and low-level performance provided by platforms such as Cython.

Table 5.1: Comparing the behaviour of `pythODE` to published code. Notice how sometimes the reported **steps** $\neq$ **accepted** + **rejected** for Hairer's code indicating a previously undiscovered bug.

IVP	platform	method & controller	atol=rtol	steps	accepted	rejected	accuracy	time(s)
Arenstorf orbit (2.15)	`pythODE`	DP5(4)$_{6(7)}$ (2.79) I	1e-7	229	204	25	3.14357e-04	0.023
Arenstorf orbit (2.15)	F77 from Hairer	DP5(4)$_{6(7)}$ (2.79) I	1e-7	229	204	24	3.14358e-04	0.006
Arenstorf orbit (2.15)	C from Hairer	DP5(4)$_{6(7)}$ (2.79) I	1e-7	229	204	24	3.14358e-04	0.008
Arenstorf orbit (2.15)	OCTAVE from Hairer	DP5(4)$_{6(7)}$ (2.79) I	1e-7	229	204	24	3.14362e-04	0.486
Arenstorf orbit (2.15)	`pythODE`	DP5(4)$_{6(7)}$ (2.79) I	1e-13	3168	3165	3	2.15494e-09	0.325
Arenstorf orbit (2.15)	F77 from Hairer	DP5(4)$_{6(7)}$ (2.79) I	1e-13	3168	3165	0	2.23433e-09	0.010
Arenstorf orbit (2.15)	C from Hairer	DP5(4)$_{6(7)}$ (2.79) I	1e-13	3168	3165	0	2.16876e-09	0.009
Arenstorf orbit (2.15)	OCTAVE from Hairer	DP5(4)$_{6(7)}$ (2.79) I	1e-13	3168	3165	0	2.36980e-09	6.615
Arenstorf orbit (2.15)	`pythODE`	DP5(4)$_{6(7)}$ (2.79) PI	1e-7	240	219	21	7.53846e-04	0.022
Arenstorf orbit (2.15)	F77 from Hairer	DP5(4)$_{6(7)}$ (2.79) PI	1e-7	240	216	22	7.45339e-04	0.007
Arenstorf orbit (2.15)	C from Hairer	DP5(4)$_{6(7)}$ (2.79) PI	1e-7	240	216	22	7.45339e-04	0.007
Arenstorf orbit (2.15)	OCTAVE from Hairer	DP5(4)$_{6(7)}$ (2.79) PI	1e-7	240	216	22	7.45344e-04	0.512
Arenstorf orbit (2.15)	`pythODE`	DP5(4)$_{6(7)}$ (2.79) PI	1e-13	3353	3350	3	1.38915e-09	0.316
Arenstorf orbit (2.15)	F77 from Hairer	DP5(4)$_{6(7)}$ (2.79) PI	1e-13	3353	3350	0	1.68220e-09	0.009
Arenstorf orbit (2.15)	C from Hairer	DP5(4)$_{6(7)}$ (2.79) PI	1e-13	3353	3350	0	1.69841e-09	0.011
Arenstorf orbit (2.15)	OCTAVE from Hairer	DP5(4)$_{6(7)}$ (2.79) PI	1e-13	3353	3350	0	1.45063e-09	7.239

5.6 Performance testing with pythODE and generating reference solutions using published ERK pairs

Performance testing of IVP methods in the literature is often presented in the form of *work-precision diagrams*, examples of which are given in this section and can be found extensively in the references [2, 21, 29, 37, 46, 47, 48, 49, 50, 56, 94, 96, 128, 136, 142, 157, 158, 159, 161, 164, 166, 187, 189, 190, 203]. Typically, the quantity measuring the work, either CPU time or using a proxy for performance such as counting RHS evaluations, is indicated by the y-axis of a plot where numerical solutions requiring a greater amount of work are towards the top. Typically, the so-called "precision" (that is actually the accuracy or a calculation of the global error (2.29) of the numerical solution) of the solution is given on the x-axis of a plot where greater accuracy can be at either the left or right, depending on an author's preference. Lines that are lower than others on the work-precision diagram indicate numerical methods that are more efficient. The execution time of well-optimized implementations of different algorithms is often accepted as the "ultimate" standard in assessing the performance of numerical algorithms [56]. However, due to the relative newness of `pythODE`, in combination with all of the potential sources of overhead and spurious timings that using a dynamically typed high-level language such as PYTHON can lead to, the number of RHS evaluations is instead used as a proxy for the relative performance of IVP methods. In addition, with the large and sometimes surprising discrepancies in performance between the performance of SCIPY functions (as well as other PYTHON statements and library functions) discussed in Section 5.5, implementations of different numerical methods that leverage different functionality in different ways may not have relative performances that correlate well with implementations in low-level languages. Implementation-independent performance assessment, such as counting RHS evaluations or the number of steps if there is a fixed number of RHS evaluations per step, without actually timing the IVP solutions, has been used for many publications [2, 16, 37, 46, 47, 48, 49, 50, 128, 136, 142, 162, 155, 156, 158, 160, 164, 166, 187, 189, 190]. This is because implementation-independent performance measurement contributes to the reproducibility of results if optimized implementations are not available; i.e., there is no "noisy" variation that happens in actual timings and it allows exploration of numerical behaviour that is not necessarily present in methods suitable for practical use or well-optimized implementations. In fact, for most of the results presented from Section 5.8 and onward, the sometimes substantial variation in timings would be too large to reliably give the insights seen. In addition, the combination of multiple trials at the multiple accuracies, which are often aggregated into single data points below, would make it extremely expensive to eliminate noise due to variation in timings. Ultimately, many IVPs from practice have expensive RHSs and it can be expected that for these IVPs the computational cost of the expensive RHS evaluations will dominate in implementations for practical purposes.

The work-precision diagram of classic $5(4)_6$ ERK pairs for the "non-stiff A3" IVP (2.11) is shown in Figure 5.1. Also included in Figure 5.1 are the eighth- and seventh-order Dormand–Prince ERK pair [136] and the third- and second-order by Bogacki–Shampine ERK pair [15] that MATLAB uses for its `ode23`. On this extremely simple IVP, all of these pairs show the generally expected asymptotic behaviour except when solving to coarse accuracies. What can clearly be seen in Figure 5.1 is that at coarse accuracies the third- and second-order Bogacki–Shampine ERK pair has relatively smooth and predictable behaviour, whereas higher-order pairs often do not.

Stiff IVPs, which are discussed in Section 2.5.4, do not show the expected asymptotic behaviour when typical accuracies are sought because of the stepsize restrictions for numerical stability described in Section 2.5.4. This is demonstrated by Figure 5.2 for the “stiff B1” IVP (2.12). It can clearly be seen on this mildly stiff IVP that until the asymptotic region is reached, there is nearly the same cost for a wide range of accuracies. Although not directly relevant to the pairs constructed for IVPs from celestial mechanics, performance on stiff IVPs and the characteristics of the linear stability region can be extremely important for ERK pair construction. In addition to explicit methods being a simpler choice for mildly stiff IVPs, explicit methods should be able to gracefully handle some stiffness if it unexpectedly occurs.

The performance of the “Brusselator” IVP described in Section 5.3.2 is shown in Figure 5.3. The numerical behaviour of ERK pairs solving the “Brusselator” IVP is in between non-stiff IVPs and IVPs that are definitely stiff, such as the “stiff B1” IVP (2.12). The CK4$(5)_{6(6)}$ pair (2.83) outperforms most, except for extremely high accuracy where the DP5$(4)_{6(7)}$ method (2.79) and the eighth- and seventh-order Dormand–Prince ERK pair is most efficient. Possible justification for lower-order numerical methods still being widely used for IVPs from semi-discretized PDEs is also clearly seen from Figure 5.3. At coarse tolerances and poor accuracies, the third- and second-order Bogacki–Shampine ERK pair [15] incurs less computational cost and behaves more predictably than any of the higher-order pairs tested. The author’s best hypothesis is that higher-order methods have less leeway for the actual asymptotic behaviour to be different from the asymptotic behaviour assumed by the step controller, which can lead to the instability discussed in Section 2.6. Given the regularity that calls appear in the literature for higher-order numerical methods to be adopted more widely by practitioners [36, 74, 169, 198, 199], this phenomenon (including the widespread use of low-order methods without stepsize control) should be more carefully examined in the future, especially with large amounts of performance data and fine-grained experimentation.

5.6.1 Performance tradeoffs of local extrapolation

Recall from Section 2.6 that local extrapolation is when an embedded pair advances the solution using the higher-order component, despite the error estimate actually being for the lower-order component. Except for several important IVPs, in the course of this study local extrapolation usually provided a clear advantage for all $5(4)_6$ ERK pairs at most tolerances tested. These exceptions are important because although pairs such as the RKF4$(5)_{6(6)}$ pair (2.78) were originally designed to be used without local extrapolation by minimizing the leading error coefficient A^5 of the fourth-order component, subsequent studies have generally concluded that they usually work best using local extrapolation anyways [47, 146].

The advantage of local extrapolation for solving the van der Pol IVP (2.14), the “Arenstorf orbit”

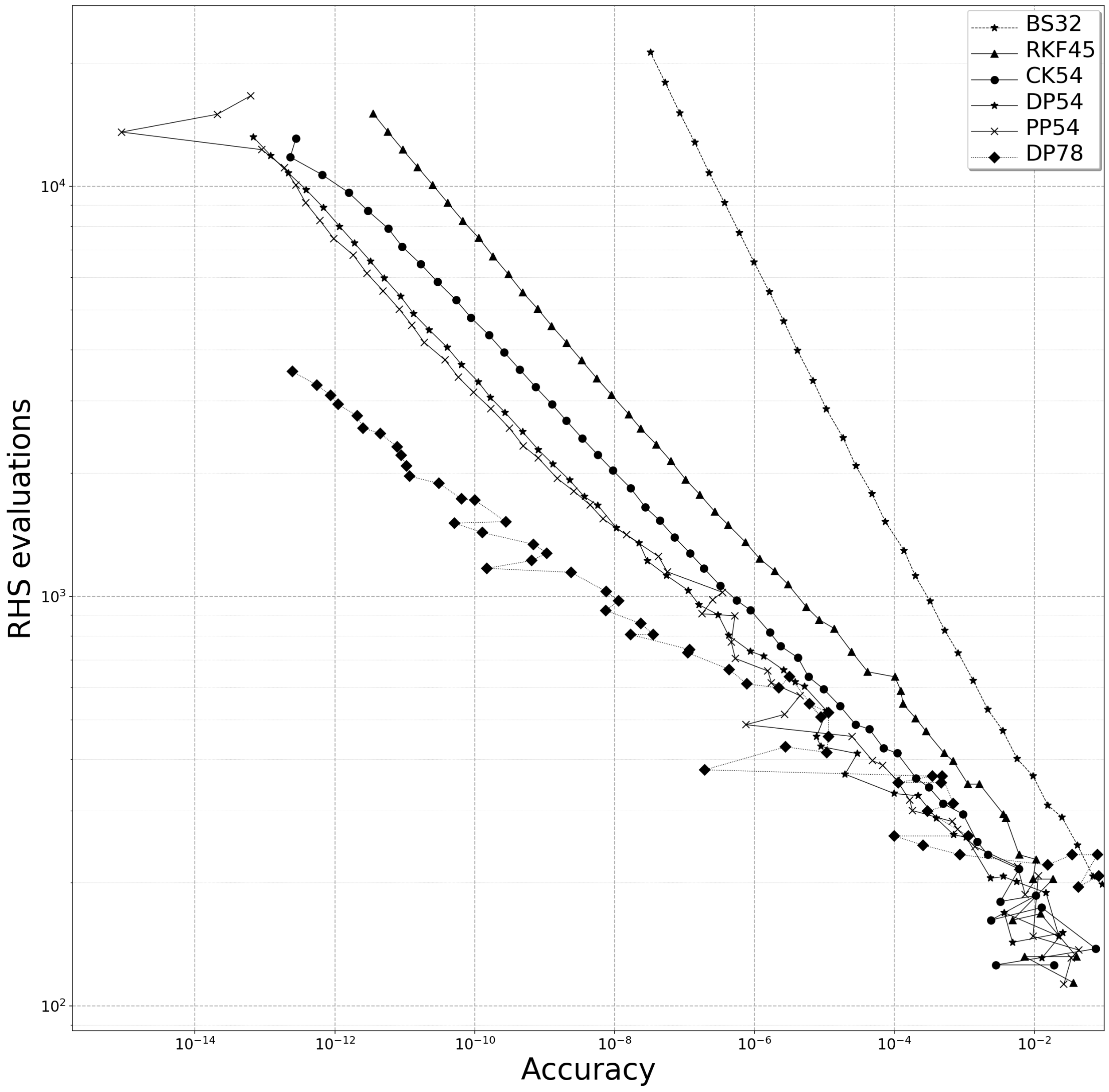


Figure 5.1: A work-precision diagram from solving the "non-stiff A3" IVP (2.11). BS32 represents the third- and second-order Bogacki–Shampine ERK pair [15], RKF45 represents the RKF4(5)$_{6(6)}$ pair (2.78), CK54 represents CK4(5)$_{6(6)}$ pair (2.83), DP54 represents the DP5(4)$_{6(7)}$ pair (2.79), PP54 represents the PP5(4)$_{6(7)}$ pair (2.84), and DP78 represents the eighth- and seventh-order Dormand–Prince ERK pair [136].

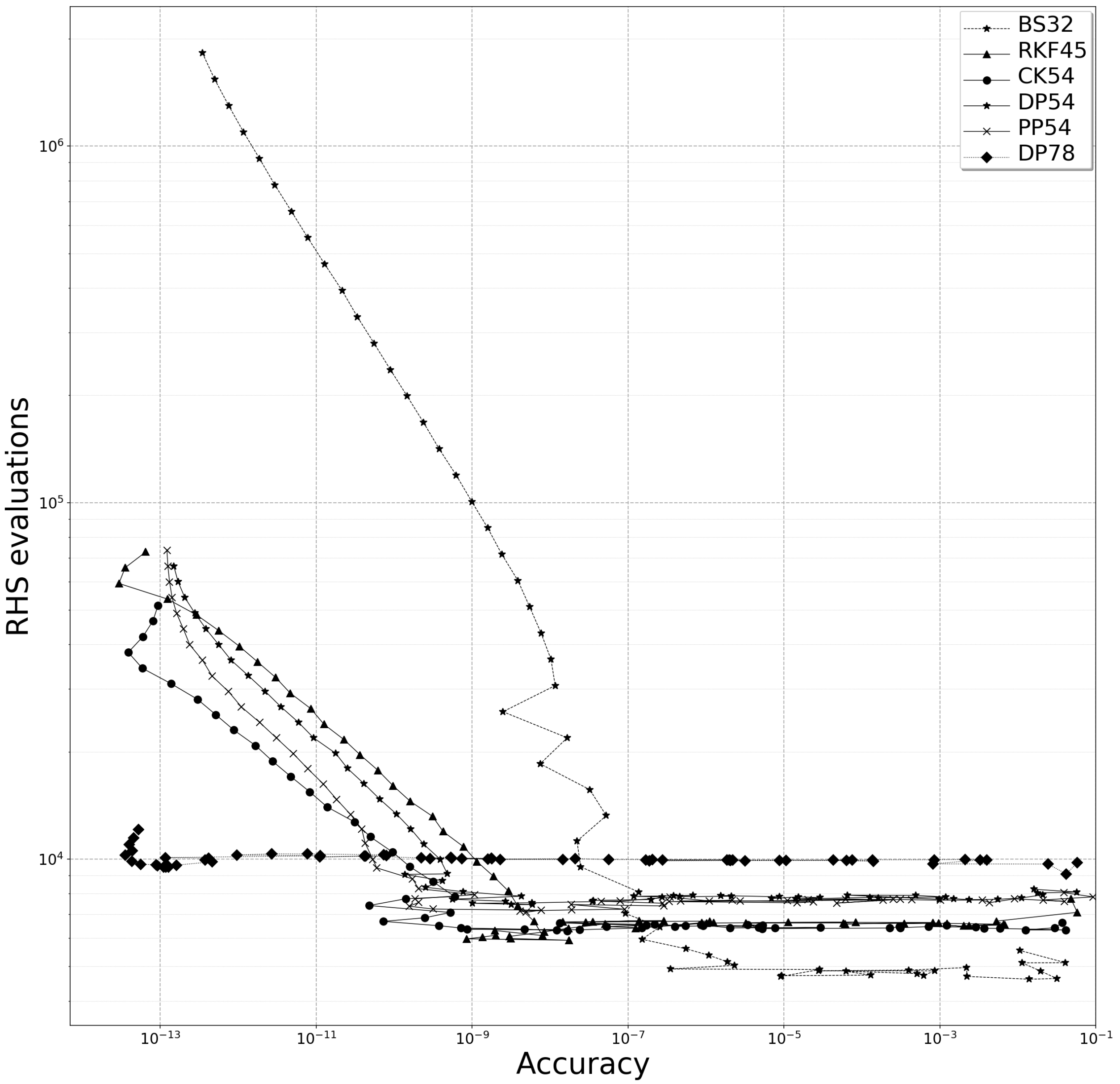


Figure 5.2: A work-precision diagram from solving the "stiff B1" IVP (2.12). BS32 represents the third- and second-order Bogacki–Shampine ERK pair [15], RKF45 represents the RKF4(5)$_{6(6)}$ pair (2.78), CK54 represents CK4(5)$_{6(6)}$ pair (2.83), DP54 represents the DP5(4)$_{6(7)}$ pair (2.79), PP54 represents the PP5(4)$_{6(7)}$ pair (2.84), and DP78 represents the eighth- and seventh-order Dormand–Prince ERK pair [136].

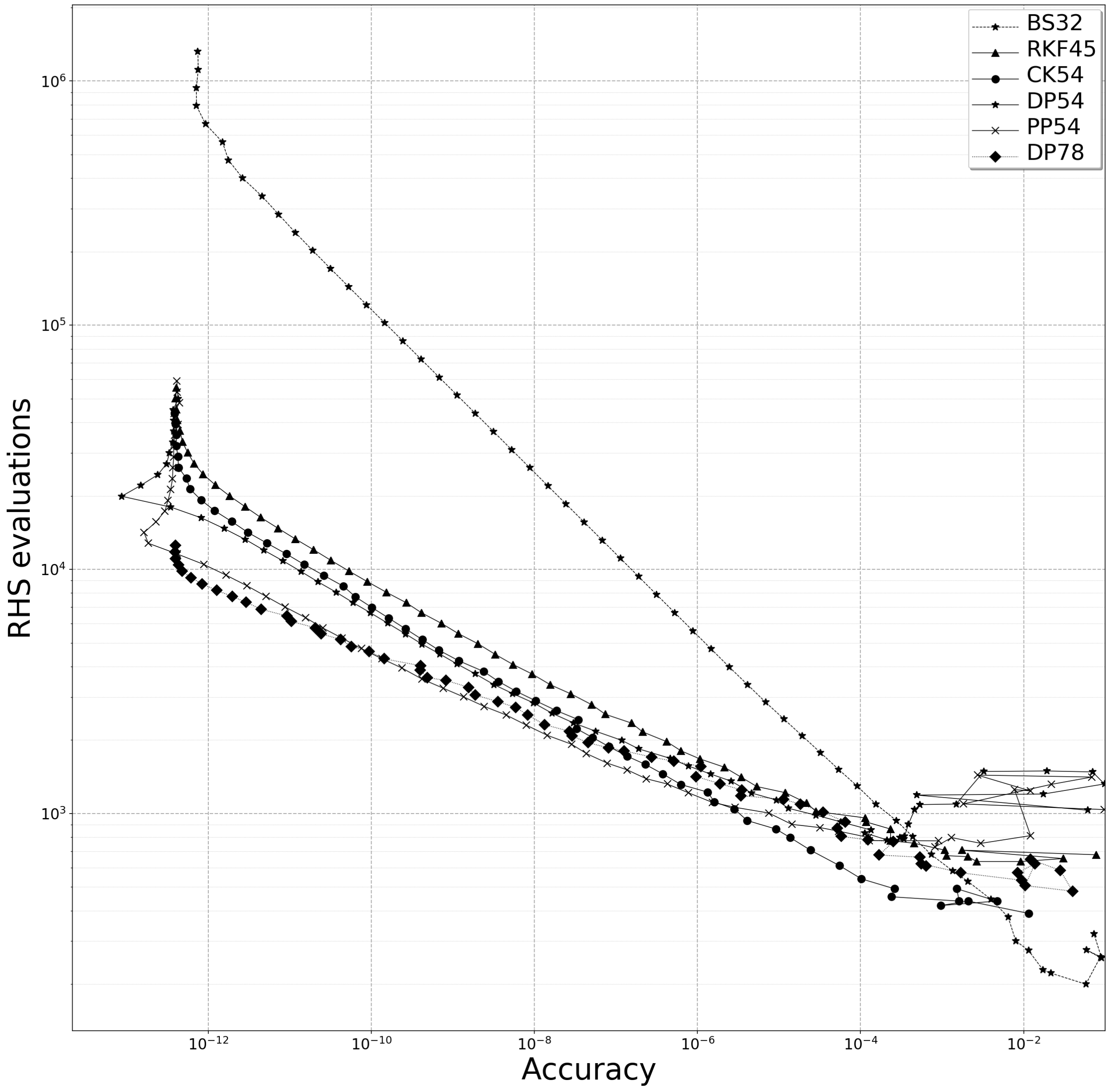


Figure 5.3: A work-precision diagram from solving the "Brusselator" IVP described by Hairer et al. [72, pg.248]. BS32 represents the third- and second-order Bogacki–Shampine ERK pair [15], RKF45 represents the RKF4(5)$_{6(6)}$ pair (2.78), CK54 represents CK4(5)$_{6(6)}$ pair (2.83), DP54 represents the DP5(4)$_{6(7)}$ pair (2.79), PP54 represents the PP5(4)$_{6(7)}$ pair (2.84), and DP78 represents the eighth- and seventh-order Dormand–Prince ERK pair [136].

IVP (2.14), the "non-stiff D2" IVP (2.13), and the "non-stiff D5" IVP (2.13) is shown in Figures 5.4, 5.5, 5.6, and 5.7, respectively. It can clearly be seen that, except for specific ranges of accuracy on specific IVPs, local extrapolation gives clearly superior performance overall. This is what was also observed for the overwhelming majority of IVPs described in Section 5.3. The only exception in the just-mentioned figures is when solving the "Arenstorf orbit" IVP (2.14) with the CK4(5)$_{6(6)}$ pair (2.83), where local extrapolation is not more efficient when solving to typical accuracies. This warrants further investigation, but the author's best hypothesis is that because the "Arenstorf orbit" IVP (2.14) is a severe test of step control, an error estimate more representative of the actual local error may be an advantage in this case. This hypothesis could be verified by examining the behaviour of a more accurate method for error estimation, such as the step-doubling error estimation described in Section 2.6.1.

However, given that the focus of efficient pairs in this thesis are IVPs from celestial mechanics, it is important to further examine whether local extrapolation might be a drawback when solving some practical IVPs from celestial mechanics. In Figures 5.8 and 5.9 respectively, the "Jovian asteroid" IVP and "nine planets" IVP are solved with the 5(4)$_{6(6)}$ ERK pairs described in Section 2.7, both with and without local extrapolation. What can easily be observed, is that the pairs designed with $\hat{A}^5 < A^6$, i.e., the CK4(5)$_{6(6)}$ pair (2.83) and the RKF4(5)$_{6(6)}$ pair (2.78), perform best without local extrapolation on these two IVPs. The pair designed with $\hat{A}^5 > A^6$, i.e., the DP5(4)M$_{6(6)}$ pair (2.82), performs best with local extrapolation on these two IVPs. In addition, the CK4(5)$_{6(6)}$ pair (2.83) without local extrapolation performs unexpectedly well on the "Jovian asteroid" IVP and the RKF4(5)$_{6(6)}$ pair (2.78) without local extrapolation performs unexpectedly well on the "nine planets" IVP. Because local extrapolation is currently the standard technique, the pairs constructed in this thesis are designed to use local extrapolation. However, based on this observation of longer-term integration of these two celestial mechanics IVPs, future work should examine whether, especially when scaling up from simple test sets of IVPs to real-world IVPs, local extrapolation is in fact the ideal strategy in many situations. In addition, if it is not ideal, the standard IVP test sets should be updated to reflect this. Therefore, given these observations of the performance data, only local extrapolation is used for the remainder of this study. However, given the historical discussion and that the error estimate in local extrapolation is less likely to reflect the local error, a detailed study of local extrapolation should be done in the future. In particular, if high-order methods are to be more predictable at coarse tolerances, this is one aspect that should be examined.

5.6.2 The performance of the PP5(4)$_{6(7)}$ pair and Tsitouras' pairs

After finding correct coefficients for Tsitouras' pairs, as described in Section 4.15.1, despite the issue discussed with their claimed properties not being able to exactly satisfy the order conditions, on most IVPs tested the performance is fairly similar to other 5(4)$_6$ ERK pairs if the properties are also fairly similar. Only for the numerical solution of a few IVPs tested do potential issues appear, specifically for several IVPs from Sharp's restricted 3-body test set described in Section 5.3.4.

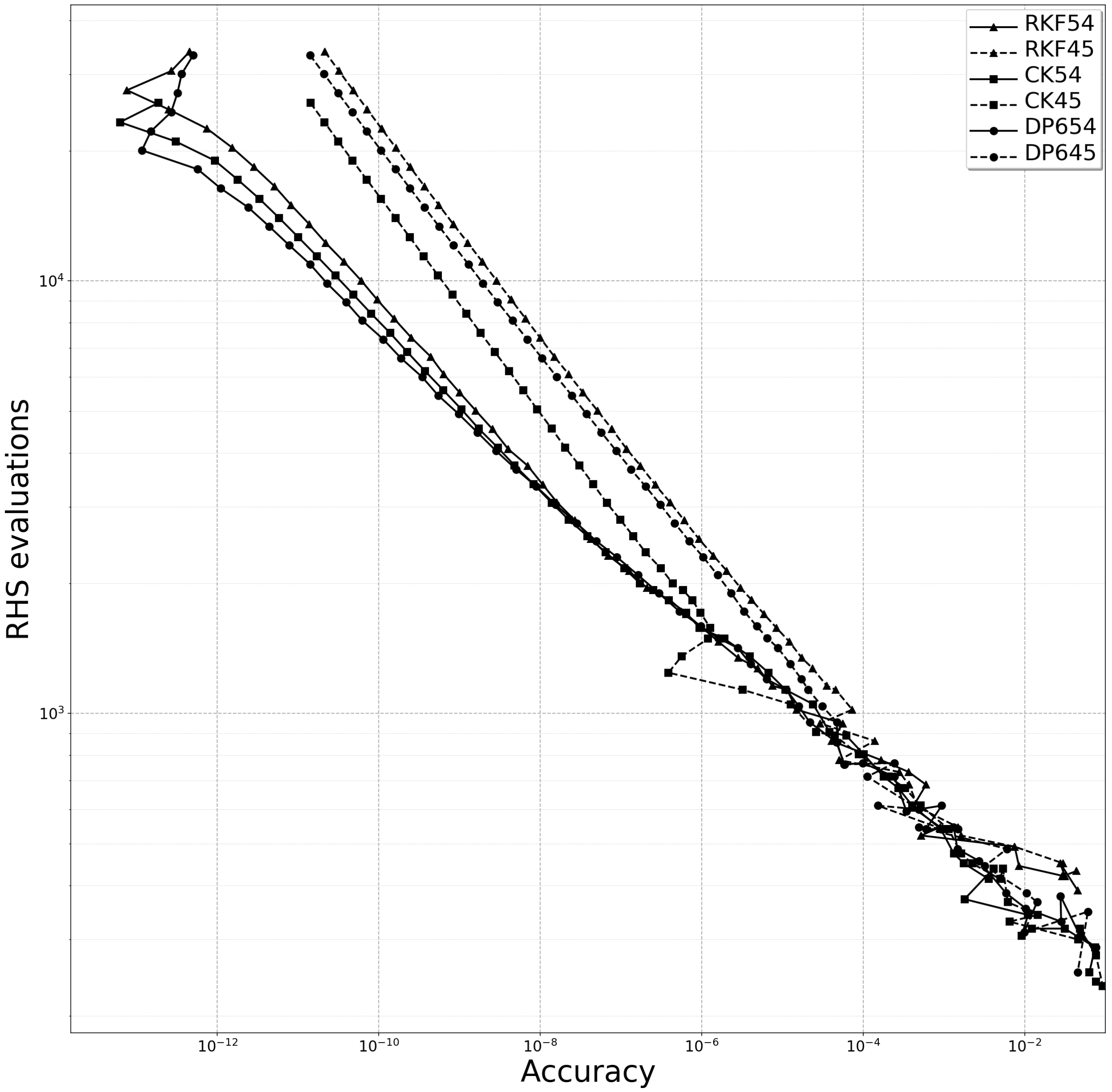


Figure 5.4: A work-precision diagram demonstrating local extrapolation when solving the van der Pol equation (2.14). RKF54 and RKF45 represent the RKF4(5)$_{6(6)}$ pair (2.78) with and without local extrapolation, respectively. CK54 and CK45 represent the CK4(5)$_{6(6)}$ pair (2.83) with and without local extrapolation, respectively. DP654 and DP645 represent the DP5(4)M$_{6(6)}$ pair (2.82) with and without local extrapolation, respectively.

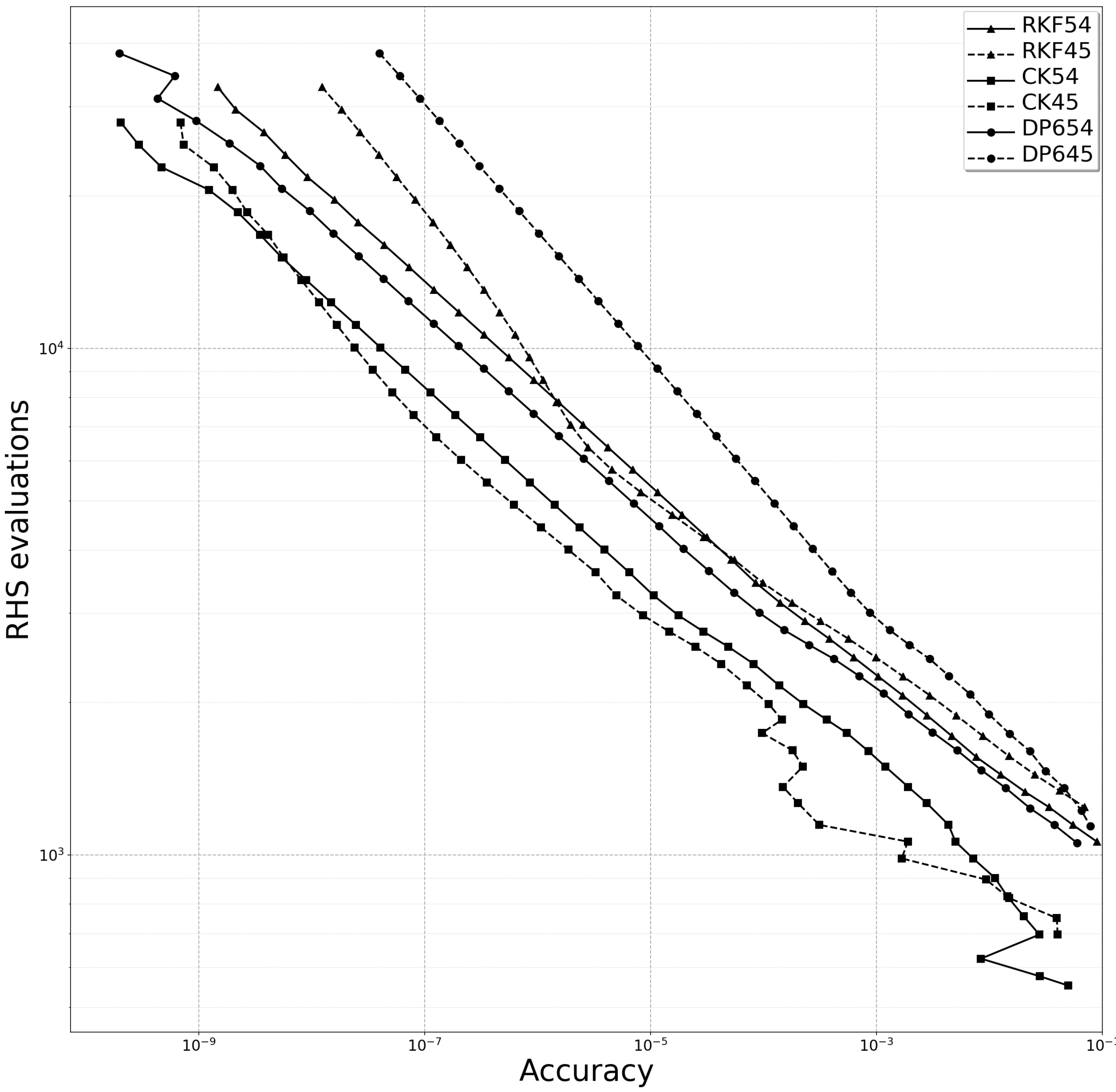


Figure 5.5: A work-precision diagram demonstrating local extrapolation when solving the "Arenstorf orbit" IVP (2.15). RKF54 and RKF45 represent the RKF4(5)$_{6(6)}$ pair (2.78) with and without local extrapolation, respectively. CK54 and CK45 represent the CK4(5)$_{6(6)}$ pair (2.83) with and without local extrapolation, respectively. DP654 and DP645 represent the DP5(4)M$_{6(6)}$ pair (2.82) with and without local extrapolation, respectively.

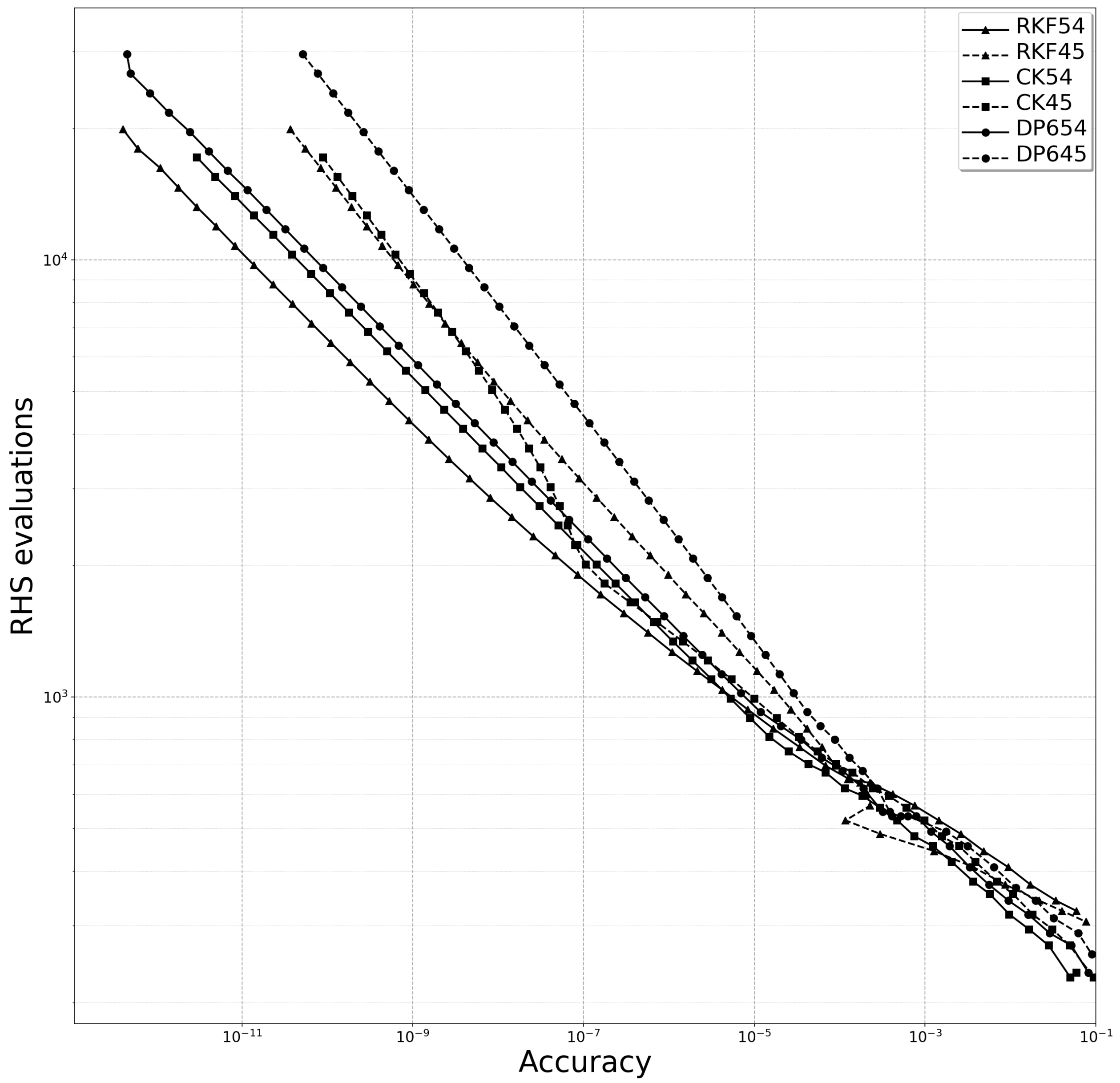


Figure 5.6: A work-precision diagram demonstrating local extrapolation when solving the "non-stiff D2" IVP (2.13). RKF54 and RKF45 represent the RKF4(5)$_{6(6)}$ pair (2.78) with and without local extrapolation, respectively. CK54 and CK45 represent the CK4(5)$_{6(6)}$ pair (2.83) with and without local extrapolation, respectively. DP654 and DP645 represent the DP5(4)M$_{6(6)}$ pair (2.82) with and without local extrapolation, respectively.

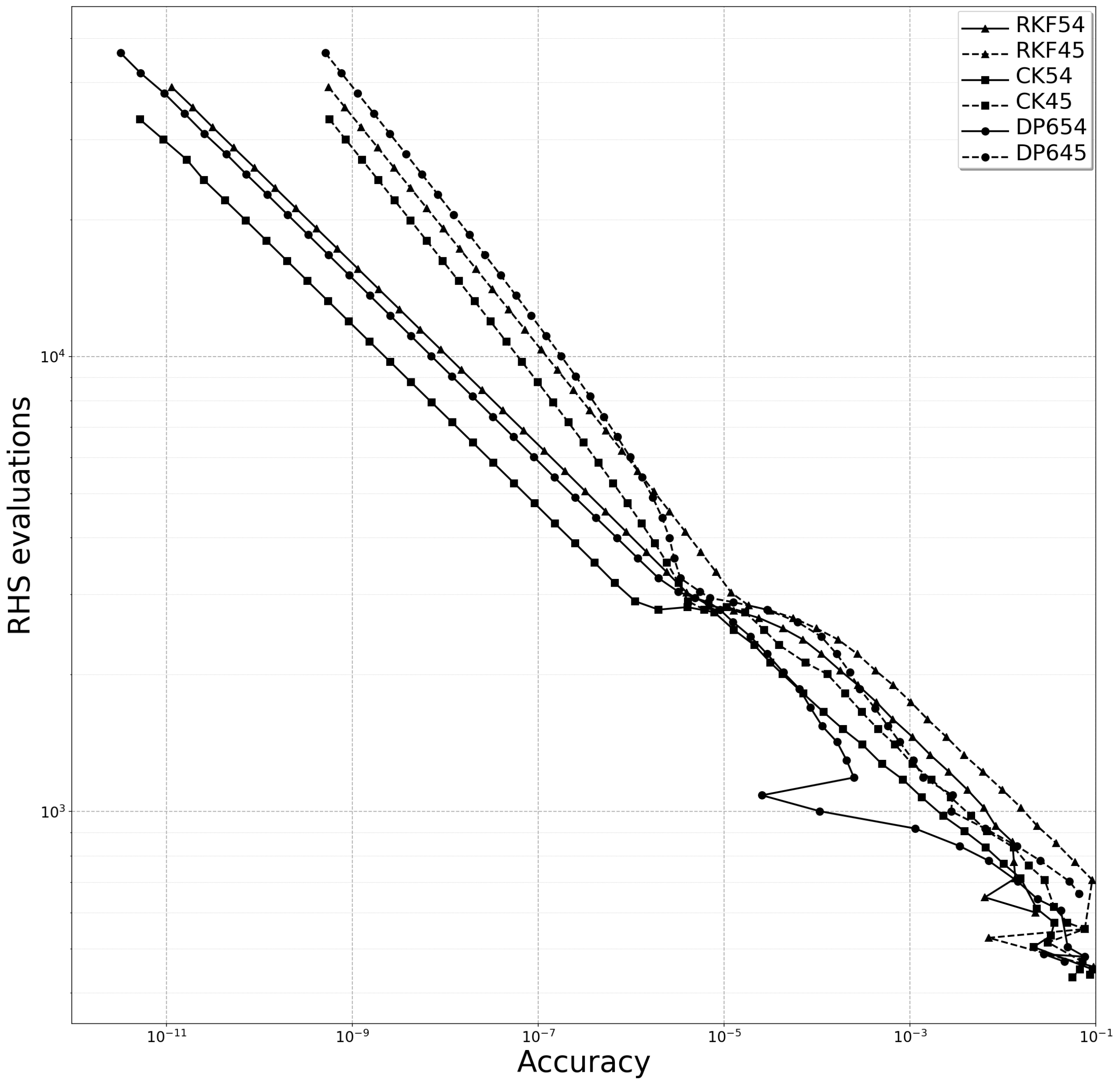


Figure 5.7: A work-precision diagram demonstrating local extrapolation when solving the "non-stiff D5" IVP (2.13). RKF54 and RKF45 represent the RKF4(5)$_{6(6)}$ pair (2.78) with and without local extrapolation, respectively. CK54 and CK45 represent the CK4(5)$_{6(6)}$ pair (2.83) with and without local extrapolation, respectively. DP654 and DP645 represent the DP5(4)M$_{6(6)}$ pair (2.82) with and without local extrapolation, respectively.

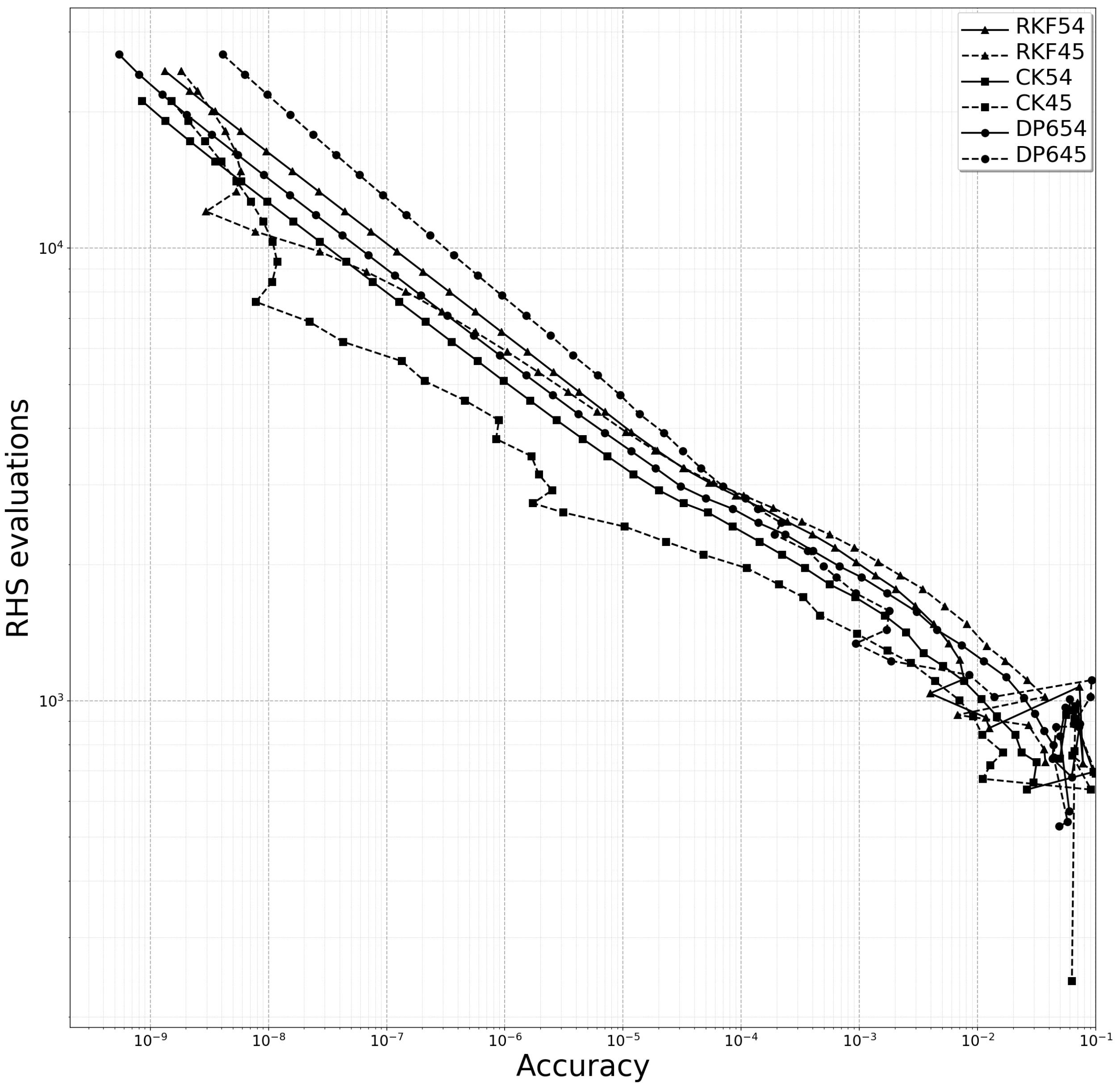


Figure 5.8: A work-precision diagrams demonstrating local extrapolation when solving the "Jovian asteroid" IVP. RKF54 and RKF45 represent the RKF4(5)$_{6(6)}$ pair (2.78) with and without local extrapolation, respectively. CK54 and CK45 represent the CK4(5)$_{6(6)}$ pair (2.83) with and without local extrapolation, respectively. DP654 and DP645 represent the DP5(4)M$_{6(6)}$ pair (2.82) with and without local extrapolation, respectively.

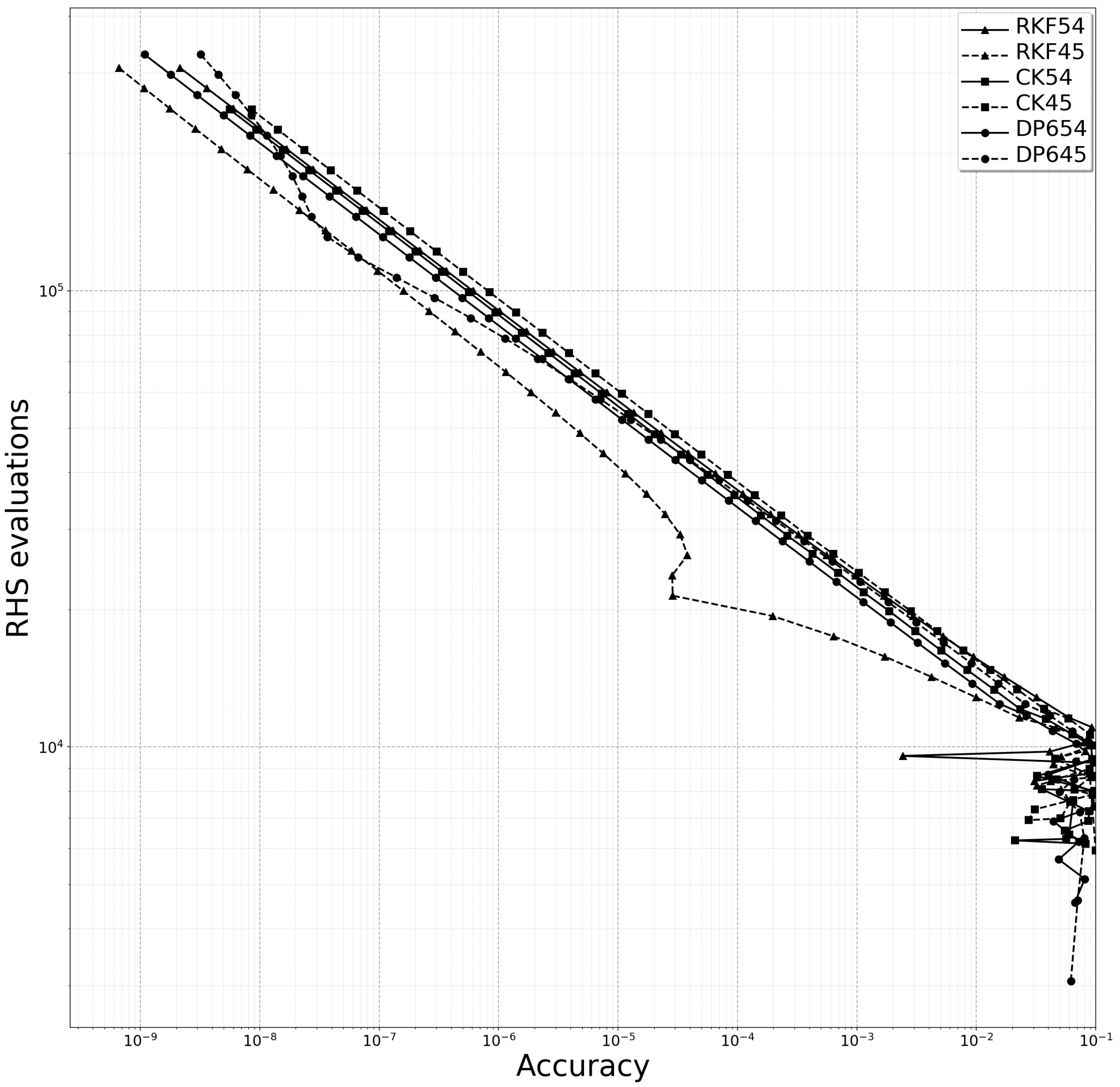


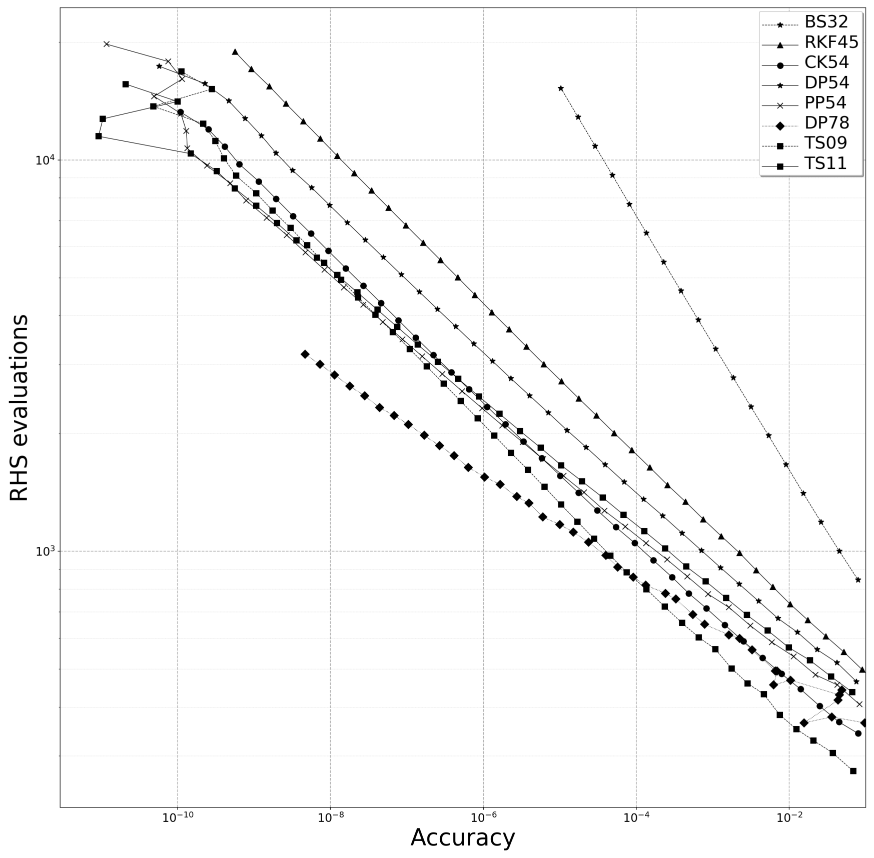
Figure 5.9: Work-precision diagram demonstrating local extrapolation when solving the "nine planets" IVP. RKF54 and RKF45 represent the RKF4(5)$_{6(6)}$ pair (2.78) with and without local extrapolation, respectively. CK54 and CK45 represent the CK4(5)$_{6(6)}$ pair (2.83) with and without local extrapolation, respectively. DP654 and DP645 represent the DP5(4)M$_{6(6)}$ pair (2.82) with and without local extrapolation, respectively.

In Figure 5.10, two IVPs from Sharp's restricted 3-body test set no longer steadily converge when solved to high accuracy by Tsitouras' pairs (corrected by Stone). In addition, the PP5(4)$_{6(7)}$ pair (2.84) with an extremely small leading error coefficient of $A^6 = 0.000065497266725264854$ shows the same behaviour and the DP5(4)$_{6(7)}$ pair (2.79) shows the behaviour to a small degree. However, Tsitouras' pair from 2009 (corrected by Stone) also shows the same behaviour of not converging steadily at the highest accuracies with $A^6 = 0.00052322700318498973$ (an A^6 that is slightly larger than the DP5(4)$_{6(7)}$ pair (2.79)). The author's hypothesis is that this inconsistent convergence at the highest accuracies may be a combination of any of: roundoff error due to somewhat large Butcher tableau (2.34) coefficients, inexactly satisfying the order conditions in the case of Tsitouras' pairs, and an extremely small leading error coefficient that gives inconsistent asymptotic behaviour and might even lead to instability in some cases.

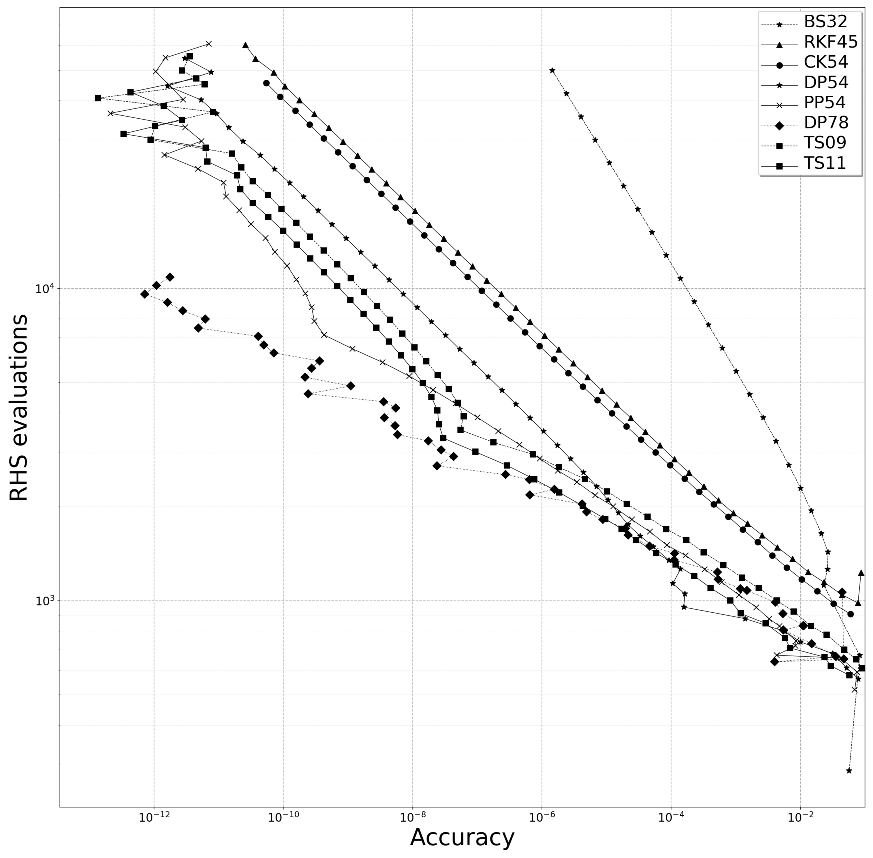
These observations indicate that caution should be used for ERK pairs with aggressively minimized leading error coefficients or ERK pairs from families that have properties that do not allow them to exactly satisfy the order conditions. These types of pairs can perform extremely well in terms of average performance for test sets such as the "non-stiff DE" test set, but occasionally show unexpectedly inconsistent or otherwise poor behaviour. In Section 5.11 below, it is shown that more aggressively minimized leading error coefficients often mean that performance can be extremely sensitive to the small changes in the properties of the ERK pair and coefficients of the Butcher tableau (2.34).

5.6.3 Generating reference solutions for IVPs

The vast majority of IVPs derived from real-world applications do not have known exact solutions and high-precision numerical solutions are not published or otherwise available. Therefore, in order to evaluate the performance of numerical methods solving these IVPs, *reference solutions* must be generated. Reference solutions are high-precision numerical solutions that are assumed to be much more accurate than the numerical solutions found while performance testing numerical methods. The reference solutions for particular IVPs can often be found using high-accuracy methods that are impractical for most efficiently solving the IVPs to reasonable tolerances, such as high-order RK methods using a small constant stepsize or the Taylor series methods mentioned in Section 2.4.2. Many authors also use quadruple-precision arithmetic to generate reference solutions despite a huge performance loss on widely available computing hardware compared to double-precision arithmetic.



(a) Work-precision diagram for the G2 12 IVP from Sharp [158].



(b) Work-precision diagram for the G3 14 IVP from Sharp [158].

Figure 5.10: Work-precision diagrams showing potential performance issues at high-accuracy of the $PP5(4)_{6(7)}$ pair (2.84) and Tsitouras' pairs (corrected by Peter Stone) [182, 183]. BS32 represents the third- and second-order Bogacki–Shampine pair [15], RKF45 represents the $RKF4(5)_{6(6)}$ pair (2.78), CK54 represents $CK4(5)_{6(6)}$ pair (2.83), DP54 represents the $DP5(4)_{6(7)}$ pair (2.79), PP54 represents the $PP5(4)_{6(7)}$ pair (2.84), DP78 represents the eighth- and seventh-order Dormand–Prince pair [136]. TS09 represents Tsitouras' (corrected) pair from 2009 [182]. TS11 represents Tsitouras' (corrected) pair from 2011 [183].

For this study, all tested IVPs can be solved relatively quickly (a few seconds to a few minutes) when used for performance testing. Therefore, the extra effort to generate reference solutions for those IVPs that do not have a published one is fairly minor. Happily, `pythODE` readily allows generating reference solutions with many different ERK pairs and allows easily comparing them in order to increase the confidence level in the reference solution used. Neither quadruple- or arbitrary-precision arithmetic is required to generate reference solutions that effectively support this study. Reference solutions are generated that minimize the error when used in double-precision computations by looking at the RMS of the change in successive solutions at the final time t_f. Seeking too small of a stepsize (when using a constant stepsize) or too small of a tolerance (when using embedded error estimation) can lead to a reduction in accuracy due to excessive round-off error. Therefore, in order to get a better picture of the behaviour of the reference solutions, candidate reference solutions are generated using many of the same published ERK pairs that are also used in the performance testing below. A reduction in accuracy is diagnosed by a steadily increasing change in the solutions between successive reductions in either stepsize (when using a constant stepsize) or tolerances (when using embedded error estimation). Both a constant stepsize and a variable stepsize were used as candidates for a reference solution, and the one judged to be highest accuracy is used as the reference solution. The most commonly used pair for the reference solutions is the eighth- and seventh-order ERK method constructed by Dormand and Prince [136]. As an example of these different stepsize strategies, the reference solution for the "nine planets" IVP was highest quality if a constant stepsize was used, whereas for the reference solution for the "Jovian asteroid" IVP, the highest quality reference solution utilized a variable stepsize.

5.7 Studying the relative performance of large numbers of ERK pairs with `pythODE`

In order to determine specific criteria for constructing ERK pairs, it is important to examine the overall importance and effect on performance of specific properties and associated characteristic numbers. In fact, the performance data presented in Section 5.11 shows that the specific values of specific $\overline{\text{PEC}}$s (2.48) are the largest factor in determining the performance of individual ERK pairs for at least one class of IVPs, i.e., IVPs found from celestial mechanics. Additionally, it can be seen from Figures D.1 and D.2 that different values of the $\overline{\text{PEC}}$s (2.48) than those found for celestial mechanics IVPs may be a principal determining factor for other classes of IVPs. This importance of specific PECs,including that it can be used to help find better ERK pairs for some classes of IVPs, is an aspect of the performance of RK methods that has not been previously discussed in the literature.

To provide empirical support to specific guidelines for construction of ERK pairs, a large number of randomly selected sets of free parameters (up to several thousand) from searches done with `OCSage` are performance tested in order to observe whether better performing pairs have certain values of characteristic numbers and PECs. Initial observations were often made based on a sample of free parameters selected uniformly and randomly from the `PostgreSQL` tables described in Chapter 4 containing the searches of the free parameters, whereas in many of the figures in the remainder of this chapter, a fixed number of randomly selected pairs for each of a set of identically sized ranges of values of the specific property being investigated is used. For example, if the property under investigation spans the interval $[-0.040, 0.040]$, rather than 6400 randomly selected pairs, 16 sub-intervals of 0.05 are used with 400 randomly selected pairs in each sub-interval. It was found that this careful sampling enormously reduced the possibility of bias and false conclusions that could be drawn, for instance, due to some values of the parameter under investigation being much more common than other values.

Although insights gained empirically from performance testing randomly selected sets of numerical methods are not as rigorous as those found by more theoretical studies, performance testing can provide new and valuable insights into the specific tradeoffs inherent in the coefficient selection process. As long as the possible limitations of larger-scale empirical performance testing are considered when interpreting the performance data, a better idea of the effect of specific properties of ERK pairs than what is currently known can be found. A possible issue that must be considered are that false conclusions could be drawn about the importance of characteristic numbers that are only correlated with other properties actually responsible for the observed behaviour, which may not currently correspond to or be obvious from known characteristic numbers. However, despite the inherent limitations of empirical studies, the procedure developed for this study gives insights into the construction of ERK pairs that would simply not be apparent from only limited performance testing.

It is shown below that the standard error coefficients and other characteristic numbers, e.g., (2.56), are not always the dominant factors in performance differences between ERK pairs. When researchers have presented numerical methods in the literature that are genuinely competitive with what has been available previously, they have invariably experimented with many combinations of free parameters before choosing the specific numerical methods that they actually published. It is understandable that in the past researchers could not easily report all of the ad-hoc experimentation done. However, this means many published and widely used numerical methods are not published with much justification as to why a specific set of coefficients were selected, it remains unknown how "optimal" they actually are for a particular purpose, and it is usually impossible for anyone to reproduce the steps that lead to the specific numerical methods that were published. This is especially the case if the research is decades old. What the performance data generated from a software system such as `OCSage` and `pythODE` used together allows, is expanding on the ad-hoc procedures of the past to give a well-defined, systematic, and reproducible approach to choosing the best numerical methods, or at least knowing the circumstances upon which a method is chosen.

5.7.1 The specific procedure for evaluating large numbers of ERK pairs

As seen from Figures 5.1–5.10, the work-precision diagrams from solving IVPs with a few ERK pairs can be extremely noisy and it can be difficult draw specific conclusions by visual inspection alone. Because the author feels that the CK4(5)$_{6(6)}$ pair (2.83) is the most consistently performing of the 5(4)$_6$ ERK pairs from Section 2.7 across a wide range of accuracies and IVPs, in this study performance is normalized with respect to the CK4(5)$_{6(6)}$ pair (2.83). For example, an ERK pair that has a normalized performance of 1.0 at a particular accuracy on a particular IVP would have exactly the same computational cost (determined by the number of RHS evaluations) as the CK4(5)$_{6(6)}$ pair (2.83) to obtain that accuracy. For overall performance on a particular IVP or set of IVPs, the normalized performances at coarse, medium, and fine accuracies is averaged. Therefore, an ERK pair with an overall averaged performance of 1.0 on a particular IVP or set of IVPs would have the same average computational cost as the CK4(5)$_{6(6)}$ pair (2.83) when the performance at coarse, medium, and fine accuracies are averaged. The accuracies chosen are found by the author's examination of a work-precision diagram for each IVP and a subjective determination of what typical coarse, medium, and fine accuracies are from the right-side, middle, and left-side, respectively, of that work-precision diagram.

In general, it is not necessarily known whether the IVPs of interest to practitioners solved to the desired accuracy correspond to the behaviour at coarse, medium, and fine accuracies, or even some combination of these in different regions of the ODE being solved. Therefore, the average of the normalized performance at these three accuracies is the overall performance measurement used to judge the performance of ERK pairs, although performance at the individual accuracies is always strongly considered. A consistent and normalized methodology of assessing performance provides much better information than is often presented in the literature. In many publications, performance is indicated without specifying the accuracy sought, only a single point of accuracy is indicated, or only a small number of pairs and IVPs are tested in a way that appears to give an obvious advantage to the newly constructed numerical method.

The tolerances used for the I step controller described in Section 2.6.2 are 12 values of relative tolerance T_{rel} (2.58) from 10^{-1} to 10^{-12}, each spaced by a factor of 10. A relative tolerance of 10^{-1}, which is only about 10 times smaller than the solution values themselves. A relative tolerance of 10^{-12} is where excessive computational cost due to issues such as floating-point roundoff start to become noticeable. For this study, the corresponding absolute tolerance T_{abs} (2.58) is 100 times less than the relative tolerance used, i.e., 12 values from 10^{-3} to 10^{-14} each spaced by a factor of 10. It is ideal that the absolute tolerances remain relatively small because they are usually more of a safety factor to protect against if one of the solution components begins to vanish. These tolerances do not generally give the performance at precisely the desired values of coarse, medium, or fine accuracies. Therefore, interpolation between the actual data points representing computational cost is used to find the expected computational cost for a particular accuracy. Although individualizing the selection of relative and absolute tolerances for each IVP could be done for future studies, experimentation by the author with different schemes for selecting absolute and relative tolerances did not affect the overall conclusions from the performance experiments done for this study.

Other step controllers such as the PI step controllers were also tested in order to make sure that they do not change the general conclusions made. For all IVPs tested in this study, rejected steps were rare at fine tolerances, an example of this can clearly be seen in Table 5.1. However, incorporating PI or other more advanced step controllers into this study would be a complication to the focus on the families and coefficient selection process for ERK pairs. Therefore, experimentation with different schemes for stepsize control must be left to future work.

In Figures 5.11–5.45 below, which are described in detail where they appear, the scatter plots in first column give the normalized computational cost required at coarse tolerances, the scatter plots in the second column give the normalized computational cost required at medium tolerances, the scatter plots in the third column give the normalized computational cost required at fine tolerances, and the scatter plots in the fourth column give the average of these three normalized computational costs. Table 5.2 gives symbols that appear on many of these scatter plots to show the values that classic $5(4)_6$ ERK pairs from Section 2.7 correspond to.

Table 5.2: Symbols used by some figures, where indicated, in this chapter to represent published $5(4)_6$ ERK pairs.

Symbol	Method
■	The RKF4$(5)_{6(6)}$ pair (2.78)
●	The CK4$(5)_{6(6)}$ pair (2.83)
▲	The DP5(4)M$_{6(6)}$ pair (2.82)
◆	The PP5$(4)_{6(7)}$ pair (2.84)
★	The DP5$(4)_{6(7)}$ pair (2.79)
◀	The DP5(4)C$_{6(7)}$ pair (2.81)
▶	The DP5(4)S$_{6(7)}$ pair (2.80)

5.8 Observations of the relative performance of different families and leading error coefficients

The effect of the leading error coefficient of $5(4)_6$ ERK pairs from the $5(4)_{6(6)\mathrm{C}(2)}$ and $5(4)_{6(7)\mathrm{C}(2)}$ families on various IVPs is demonstrated in this section. This shows that, although important, a small leading error coefficient is not sufficient to ensure the best performance. For the families without the C(2) simplifying assumptions (2.44b), the figures lead to similar conclusions to those from the families with the C(2) simplifying assumptions (2.44b). Therefore, these families without the C(2) simplifying assumptions (2.44b) are only addressed starting in Section 5.10.

The "A3" IVP

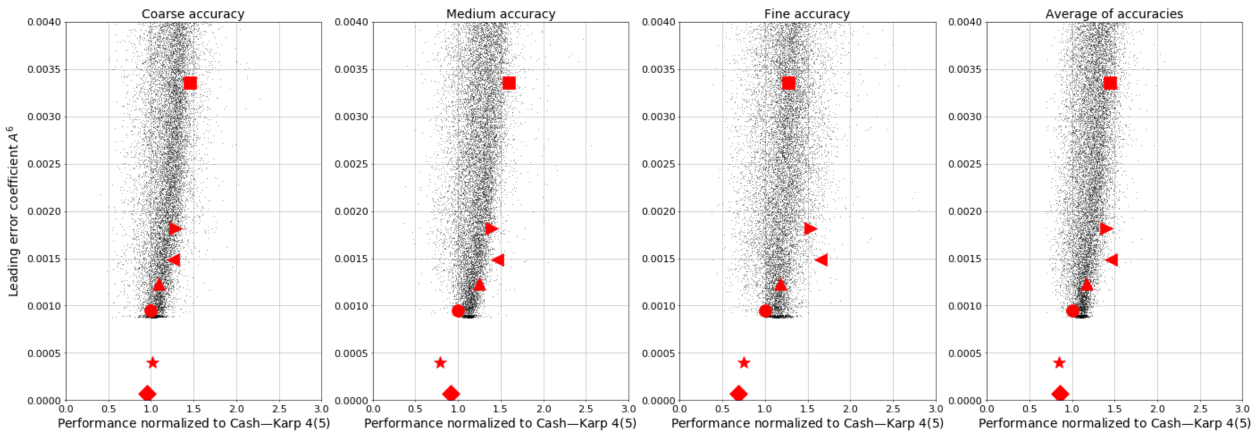
In Figure 5.11, the performance solving the "A3" IVP (2.11) from the "non-stiff DE" test set in relation to the leading error coefficient A^6 is shown at coarse, medium, fine, and averaged accuracies. What can clearly be seen is that, for this relatively simple IVP, the best-performing pairs have the smallest leading error coefficient and that this relationship holds between families, with the $5(4)_{6(7)\mathrm{C}(2)}$ family having pairs with the best performance overall because the leading error coefficient A^6 can be minimized arbitrarily.

In Figure 5.11, at each of the coarse, medium, and fine tolerances there are a small number of pairs that perform at a normalized performance of 0.5, i.e., only half the computational cost to obtain the specified accuracy as $\mathrm{CK}4(5)_{6(6)}$ (2.83). However, when the coarse, medium, and fine tolerances are averaged there are no pairs found that have a performance near 0.5 averaged accuracy. This demonstrates a more general observation by the author that there are pairs that are "optimal" for specific accuracies on individual IVPs but often this "optimal" performance does not translate to the IVP as a whole or to related IVPs. This is why the decision was made to use three different accuracies. This also supports the general observation made that any specific conclusions drawn should apply to at least several different accuracies sought and ideally to more than one IVP. During the initial explorations of performance data in support of the study in this thesis, the author found it relatively easy to find ERK pairs that performed extremely well at a particular tolerance on a particular IVP, but had at best had average performance at different tolerances or on closely related IVPs.

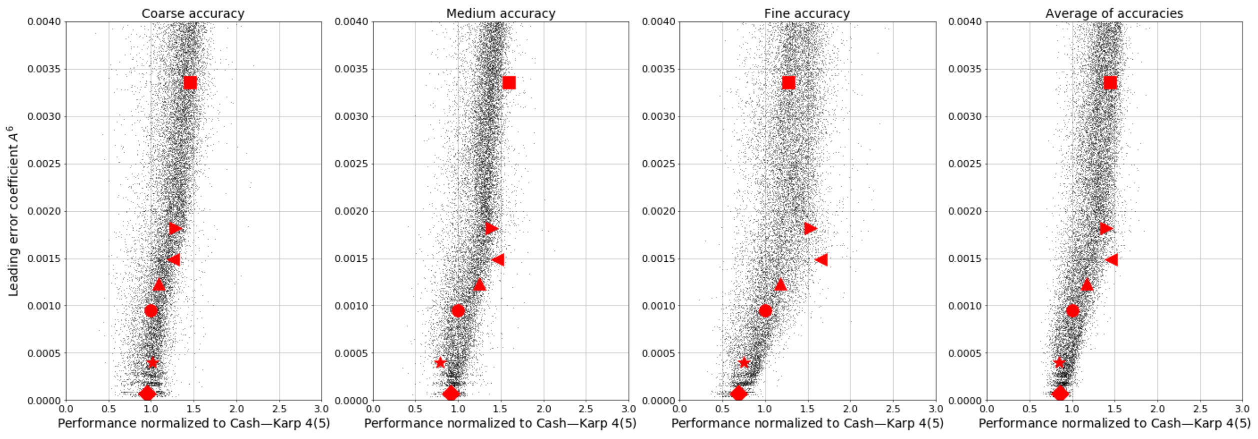
This also clearly illustrates the source of an issue that has been discussed for both numerical methods [111, 166, 186] and extensively for other areas of research into algorithms and computational methodologies [22, 42, 45, 88, 90], that is that many studies compare relatively few numbers of methods that are either deliberately or incidentally "tuned" on a small number of problems. "Good candidates" for these "tuned" methods are clearly visible in Figure 5.11, in addition to many other figures throughout this chapter. This clearly indicates the importance of using larger amounts of performance data than is typically used to support most studies into numerical methods.

The "Arenstorf orbit" IVP

The "Arenstorf orbit" IVP (2.15) is an IVP where the magnitude of the leading error coefficient A^6 is still important, but is not necessarily the most dominant factor governing performance between different pairs or families. In Figure 5.12, the performance solving the "Arenstorf orbit" IVP (2.15) is shown in comparison to the leading error coefficient A^6. It can be seen that nearly the best overall pair for the "Arenstorf orbit" IVP (2.15) is in fact the $\mathrm{CK}4(5)_{6(6)}$ pair (2.83). In particular, on this challenging IVP for variable-stepsize solvers, there are pairs from the $5(4)_{6(7)\mathrm{C}(2)}$ family that perform as well as the $\mathrm{CK}4(5)_{6(6)}$ pair (2.83), even though many published pairs perform quite poorly overall. Notice that the $\mathrm{PP}5(4)_{6(7)}$ method (2.84) performs relatively better at low accuracy and performs progressively worse at fine accuracy. Even though the $\mathrm{PP}5(4)_{6(7)}$ method (2.84) often performs extremely well on many tested IVPs when seeking high accuracy due to the extremely small value of A^6, this observation of poor performance at high accuracy on a challenging IVP should raise more questions about the real-world performance and robustness of an aggressively minimized leading error coefficient. Observe that pairs in the $5(4)_{6(6)\mathrm{C}(2)}$ family often perform as well as pairs in the $5(4)_{6(7)\mathrm{C}(2)}$ family even when the former have a much larger leading error coefficient. This clearly indicates

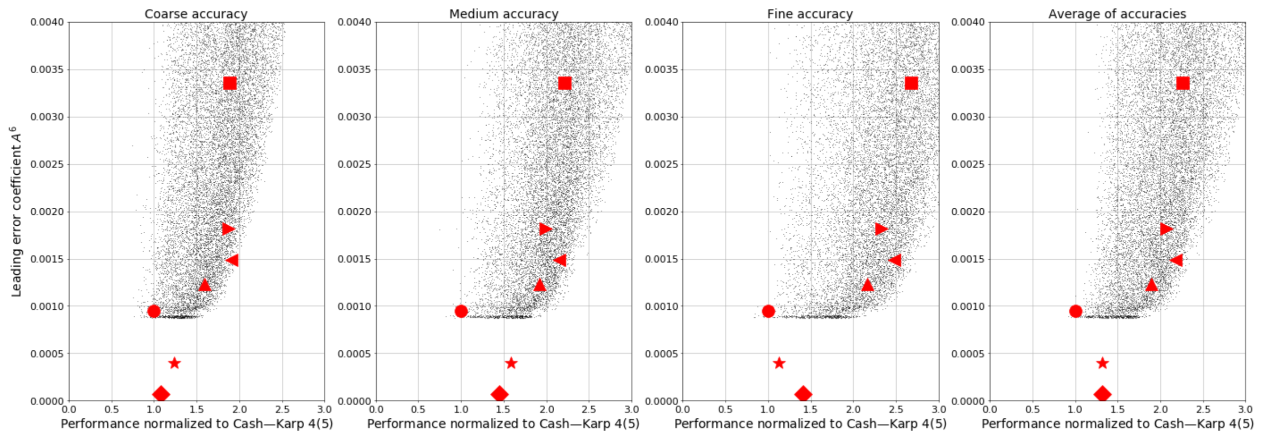


(a) The $5(4)_{6(6)\mathrm{C}(2)}$ family.

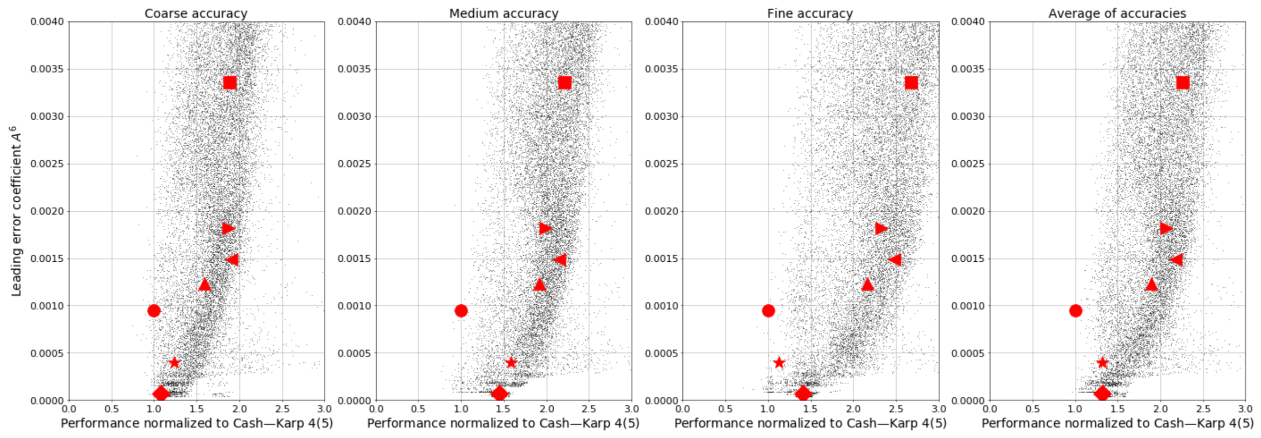


(b) The $5(4)_{6(7)\mathrm{C}(2)}$ family.

Figure 5.11: The performance solving the "non-stiff A3" IVP (2.11) in comparison to the leading error coefficient. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



(a) The $5(4)_{6(6)\mathrm{C}(2)}$ family.



(b) The $5(4)_{6(7)\mathrm{C}(2)}$ family.

Figure 5.12: The performance solving the "Arenstorf orbit" IVP (2.15) in comparison to the leading error coefficient. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

that a more thorough understanding is needed of the relationship between the properties of ERK pairs and performance solving IVPs.

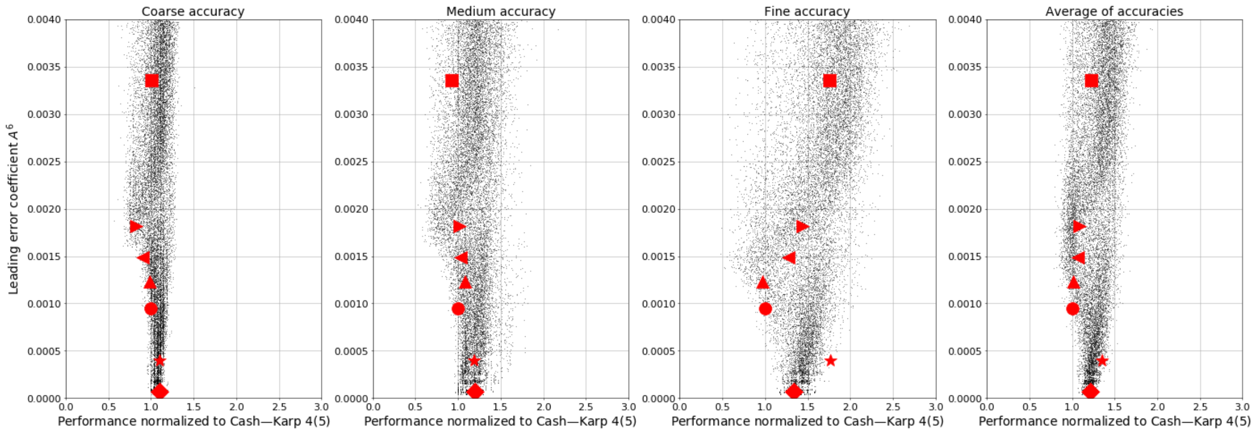
The "non-stiff B2" and "non-stiff B3" IVPs

In Figure 5.13, the performance in relation to the leading error coefficient is shown for the "non-stiff B2" and "non-stiff B3" IVP, from the "non-stiff DE" test set that each model a chemical reaction. The "non-stiff B2" IVP is a linear IVP and the "non-stiff B3" IVP is a non-linear IVP. Only the $5(4)_{6(7)\mathrm{C}(2)}$ family is shown because this demonstrates the numerical behaviour most clearly. Observe that minimizing the leading error coefficient to the extent possible with the $5(4)_{6(7)\mathrm{C}(2)}$ family is actually detrimental for these IVPs. This is a phenomenon that has also been observed for all other linear IVPs that were tested in the course of this study. It was further observed that the leading error coefficient rarely had relevance to the performance of ERK pairs for solving linear IVPs. This is because, as discussed in Section 2.5.1, the only non-zero elementary differentials for IVPs with linear RHSs are those corresponding to the tall trees. In addition, despite being relatively easy to study, observations made by the author throughout this study indicate that non-stiff linear IVPs may only have limited applicability for studying the performance of many non-stiff non-linear IVPs. In particular, in many studies phenomenon such as dispersion, dissipation, and linear stability are examined through the scalar test equation (2.10) with appropriate λ when studying the numerical solution of non-linear IVPs [4, 82, 160, 179]. However, this practice should be revisited.

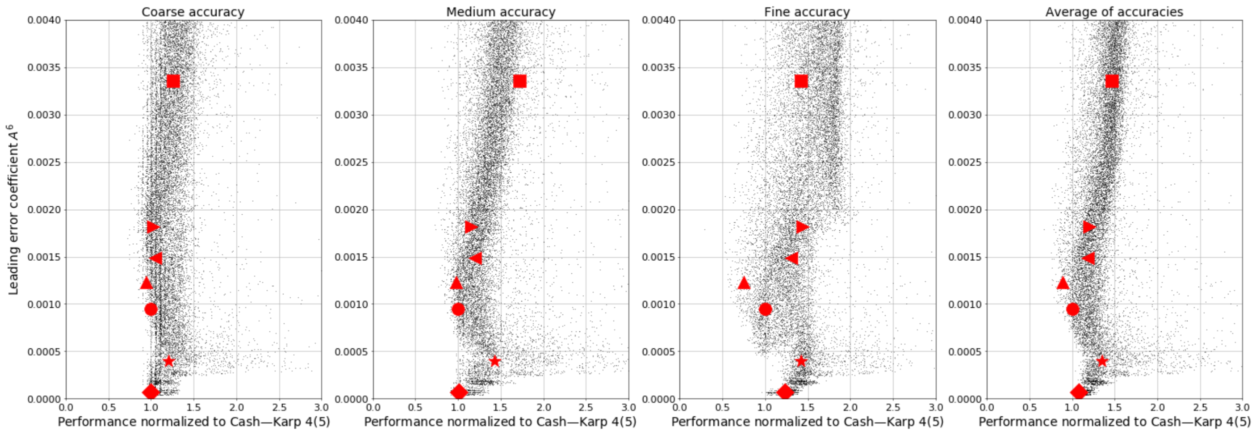
For IVPs with quadratic RHSs, such as the "non-stiff B3" IVP, the only elementary differentials (2.36) that are non-zero are those corresponding to trees that have no more than 2 branches extending from any vertex, which can easily be seen by examining the expressions for elementary differentials (2.36). This means for IVPs that have quadratic RHSs, at most only 11 of the 20 PECs of the sixth-order error term are non-zero. This helps explain why performance testing by the author, similar to that shown in Section 5.10 below, showed that the best performing pairs for each of the "non-stiff B2" IVP and "non-stiff B3" IVP were not well correlated. This and other observations by the author indicate that caution should be used in assuming that the performance of different IVPs will be correlated based on similar origin or similar mathematical classification. Although these two IVPs and computational chemistry in general are not the focus of this study, this example is included to illustrate the potentially severe loss of performance that could occur with a blind overemphasis on leading error coefficients. Considering that many of the examples the author has seen in the literature and text books discuss using the MATLAB `ode45` solver for solving IVPs from chemistry [53, 109, 170], future work could include implementing a more IVPs from computational chemistry into `pythODE` to see if general conclusions about better ERK pairs for this important application could be found.

The average performance solving the IVPs from the "non-stiff DE" test set

Although finding improved average performance solving the IVPs from the "non-stiff DE" test set or any other arbitrary set of IVPs is not alone considered to be a notable research result in the contemporary literature, the average performance on test sets does have both illustrative and historical value. Publications that have measured the average performance of IVP methods on IVPs from the "non-stiff DE" test set include many in the references [30, 155, 161, 162, 180, 182, 183]. In Figure 5.14, the average performance solving the IVPs of "non-stiff DE" test set in relation to the leading error coefficient is shown for the family of $5(4)_{6(6)}$ ERK pairs with the C(2) simplifying assumptions (2.44b) and the family of $5(4)_{6(7)}$ ERK pairs with the C(2) simplifying assumptions (2.44b). It is interesting to observe that the CK4$(5)_{6(6)}$ pair (2.83) (with $A^6 = 0.000948$) and the DP5(4)M$_{6(6)}$ pair (2.82) (with $A^6 = 0.00123$) have better average performance solving the "non-stiff DE" test set at all accuracies than the DP5$(4)_{6(7)}$ pair (2.79) (with $A^6 = 0.000399$). This is despite both $5(4)_{6(6)}$ ERK pairs mentioned having a leading error coefficient that is 2 to 3 times larger than the DP5$(4)_{6(7)}$ pair (2.79). The CK4$(5)_{6(6)}$ pair (2.83) has better performance than even the PP5$(4)_{6(7)}$ pair (2.84) (with $A^6 = 0.0000654$, i.e., an extreme minimization of A^6) at all but fine accuracies. In addition, observe that there are pairs from the $5(4)_{6(7)\mathrm{C}(2)}$ family in Figure 5.14b that perform extremely poorly, which are explained

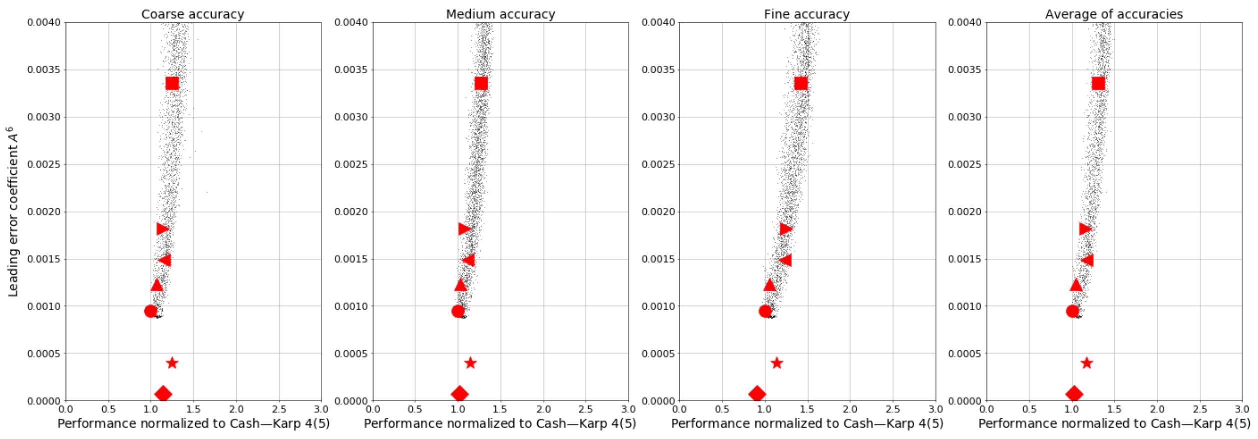


(a) The "non-stiff B2" IVP.

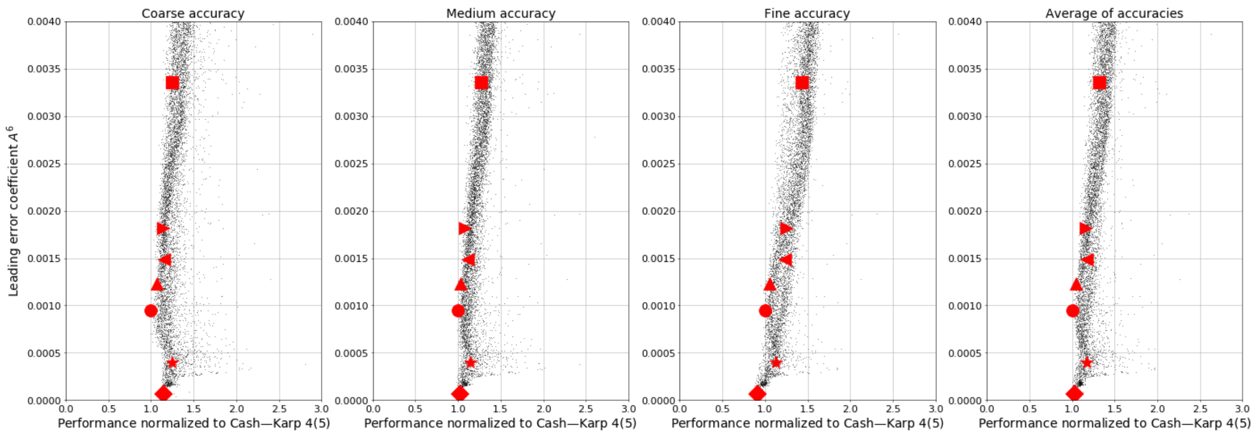


(b) The "non-stiff B3" IVP.

Figure 5.13: The performance solving the "non-stiff B2" and "non-stiff B3" IVPs in comparison to the leading error coefficient using the $5(4)_{6(7)\mathrm{C}(2)}$ family. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



(a) The $5(4)_{6(6)\mathrm{C}(2)}$ family.



(b) The $5(4)_{6(7)\mathrm{C}(2)}$ family.

Figure 5.14: The average performance solving the "non-stiff DE" test set in comparison to the leading error coefficient. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

in the next section. In this chapter, it is repeatedly seen that even when an aggressively optimized leading error coefficient is advantageous, it does not provide the performance advantage expected from the just the "relative efficiency", see the discussion about (5.1) below. The dominant factors in the performance of ERK pairs for celestial mechanics IVPs are addressed in Section 5.11. Only then can properties such as the classic characteristic numbers (2.56) be addressed in Section 5.13. More importantly, it is seen in Figures 5.11–5.13 that the performance variation on individual IVPs or specific classes of IVPs (shown below in Figure 5.15 for the "non-stiff D" IVPs (2.13)) can be much greater than for large and diverse sets of IVPs, such as the "non-stiff DE" test set.

The "non-stiff D" IVPs

In order to reach informative and quantitative conclusions about performance of RK pairs or the importance of characteristic numbers on the IVPs under study, not only is investigation at multiple accuracies required, but it is also necessary to consider at least several IVPs representative of the application that a numerical method is being constructed for. As an example, the “D1”–“D5” IVPs (2.13) in the “non-stiff DE” test set are the same ODE but with different initial conditions based on a single parameter, i.e., the eccentricity ϵ of the orbit. Therefore, these make an excellent set of IVPs for demonstrating the difference between performance on an individual IVP and averaging the performance of closely related IVPs. In particular, the average performance of the “D1”–“D5” IVPs (2.13) at coarse, medium, and fine accuracies shown in Figure 5.15 is never quite as good as that on the individual IVPs (not shown). This additionally demonstrates that numerical methods can be “tuned” for best performance on a particular IVP at multiple accuracies, but that the “tuning” may not carry over to closely related IVPs. In general, only examining the performance solving individual IVPs or too few IVPs means that conclusions reached are unlikely to be generalizable. Conversely, averaging the performance of too many IVPs that are unrelated in their numerical behaviour means that after pairs with obviously poor properties are excluded, it is difficult to make further quantitative and well-defined improvements. In Section 5.10 below, it is shown that there is a significant but not a perfect correlation in the performance of ERK pairs solving the “D1”–“D5” IVPs (2.13), which are also fairly well correlated to performance solving more complex IVPs from celestial mechanics.

5.8.1 Mathematical demonstrations

The wide range of performance seen in the figures in this section (as well as many others in this chapter) support the observation by Söderlind (the quotation at the beginning of this chapter) that most evaluations of numerical methods in the literature are in fact “mathematical demonstrations” rather than rigorous empirical experiments [166]. Of course, “mathematical demonstrations” are extremely important because they verify that any theoretical results of a study are correct, that there exists at least one case where the results of a study are visible as numerical behaviour and that the study may lead to an alternative to existing numerical methods. However, “mathematical demonstrations” are rarely sufficient to draw well-defined and generalizable conclusions about the difference between specific IVP methods, families, or even distinctly different types of numerical methods. In particular, with enough experimentation “tuning” ERK pairs, the author’s experience has shown it would be possible to argue that many ERK pairs from the families constructed in Chapter 3 are better than all of the classic pairs from Section 2.7 on the Kepler IVP (2.13). It would be even easier to argue based on limited experimentation with “tuned” pairs that many of the families from Chapter 3 are the best for the “Arenstorf orbit” IVP (2.15) because of the factor of two variation in performance shown in Figure 5.12 for similar values of the leading error coefficient. Even for the average performances solving the “non-stiff DE” test set shown in Figure 5.14, there is enough variation in performance that it would often be possible to “tune” pairs to obtain a result that was expected a priori.

Relative efficiency and leading error coefficient

An aggressively minimized leading error coefficient gives a narrow range of performance because the aggressively minimized leading error coefficient occurs within a relatively small region of parameter space. Due to the smoothness of the space of free parameters for most families of ERK pairs, shown in Chapter 4, all of the pairs found from small regions of space of free parameters often have similar properties. In Figure 5.12, the performance of the "Arenstorf orbit" IVP (2.15) is an example where an aggressively minimized leading error coefficient, such as that found by using numerical minimization to find the minimum value possible, would be severely detrimental to performance.

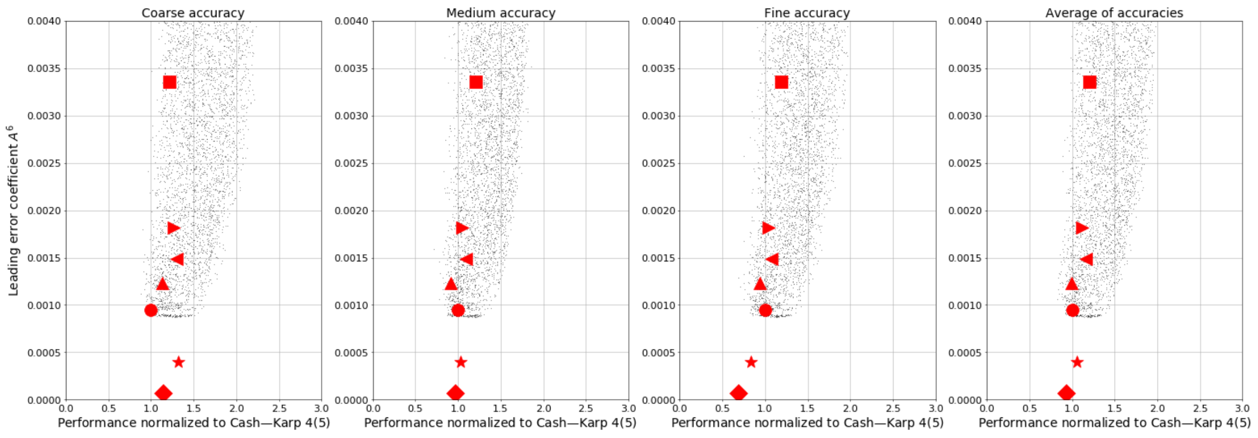
The simplistic ratio implying that the leading error coefficient is directly related to the global error was given the name "relative efficiency" by Shampine as a way of evaluating $5(4)_6$ ERK pairs in 1986 [146]. An example of calculating relative efficiency for comparing the RKF$4(5)_{6(6)}$ method (2.78) and the DP$5(4)_{6(7)}$ method (2.79) based on using the fifth-order component to advance the solution is

$$\text{"relative efficiency"} \approx \left(\frac{0.003357}{0.000399}\right)^{\frac{1}{5}} \approx 1.53, \tag{5.1}$$

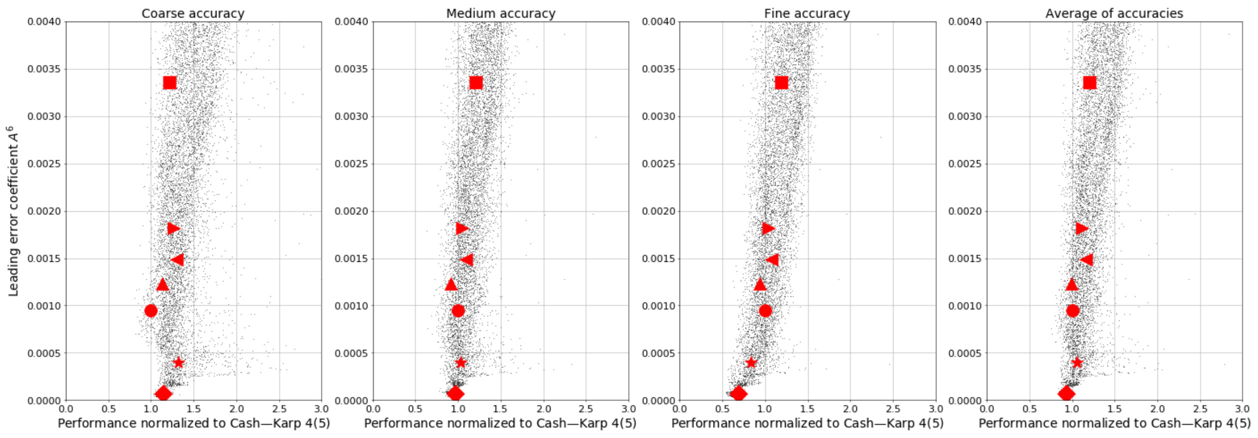
implying that the DP$5(4)_{6(7)}$ method (2.79) on average have 1.53 times less computational cost than the RKF$4(5)_{6(6)}$ method (2.78). However, the only IVPs solved in this section where the ratio shown by (5.1) holds true for the two aforementioned ERK pairs are the "A3" IVP (2.11) and the "Arenstorf orbit" IVP (2.15). Close inspection of Figure 5.12 for the "Arenstorf orbit" IVP (2.15) shows the discrepancy between the DP$5(4)_{6(7)}$ method (2.79) and RKF$4(5)_{6(6)}$ method (2.78) does not have to do with the magnitude of the leading error coefficient alone. Many authors, both before and since Shampine explicitly discussed "relative efficiency" in 1986 [146], have used at the magnitude of the leading error coefficient as an attempt to predict the potential performance of numerical methods for IVPs. However, in this study by using software packages such as `OCSage` and `pythODE`, a more complex appraisal is made using the larger amount data now available. Hence, the term "relative efficiency" is not used further in this thesis. However, it can also be seen from the preceding Figures 5.11–5.15 that pairs with smaller leading error coefficients do have better potential for the best performance as long as other criteria are considered. As long as the minimization is not too aggressive and appropriate performance testing of candidate IVP methods is done, if no other criteria for coefficient selection are obvious, then presenting the pair with smallest leading error coefficient is still justified.

Unexpectedly good performance on some IVPs of the CK$4(5)_{6(6)}$ pair

In Figure 5.12 and Figure 5.14, for both the "Arenstorf orbit" IVP (2.15) and the "non-stiff DE" test set [83], respectively, the CK$4(5)_{6(6)}$ pair (2.83) performs among the absolute best (in one experiment the author found this was the 99th percentile of pairs found in a search by `OCSage`) of the randomly chosen pairs with leading error coefficients $A^6 \leq 0.0010$ from the $5(4)_{6(6)C(2)}$ family. Although not mentioned in the original paper [30] or any other publications, the author feels that this observed performance is too coincidental. When the CK$4(5)_{6(6)}$ pair (2.83) was constructed it was probably "tuned" for best performance on both the DE test set and the "Arenstorf orbit" IVP (2.15) together. In fact, although the CK$4(5)_{6(6)}$ pair (2.83) does in fact perform extremely well on many IVPs, it does not perform among the absolute best of randomly



(a) The $5(4)_{6(6)\mathrm{C}(2)}$ family.



(b) The $5(4)_{6(7)\mathrm{C}(2)}$ family.

Figure 5.15: The average performance solving the "non-stiff D" IVPs (2.13) in comparison to the leading error coefficient. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

selected pairs for any other of the many IVPs tested in this study, other than a few contained in the "non-stiff DE" test set [83]. The "non-stiff DE" test set [83] has been around since 1972 and the "Arenstorf orbit" IVP (2.15) appeared in the first edition from 1987 [71] of the book by Hairer, Nörsett, and Wanner [72] that is extensively referenced throughout this thesis. Therefore, it is highly probable that CK4(5)$_{6(6)}$ pair (2.83) was found by "tuning" using the "Arenstorf orbit" IVP (2.15) and "non-stiff DE" test set [83].

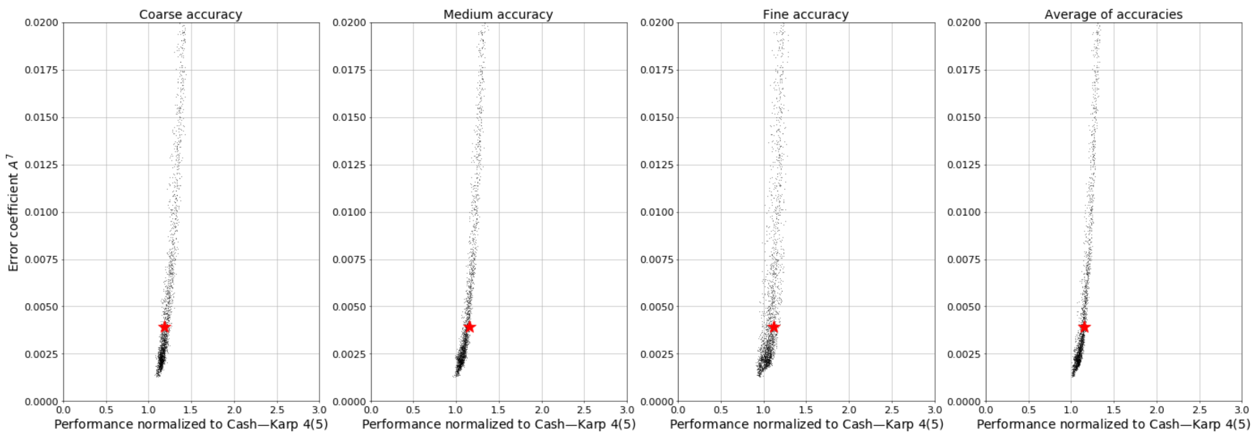
5.9 The norms used for error coefficients and the need to avoid large higher-order error coefficients

Higher-order error coefficients, e.g., the error coefficients $A^q, q \geq 7$ that are of higher order than the leading error coefficient A^6 for $5(4)_6$ ERK pairs, can substantially affect the performance of ERK pairs, although the effects are visible to a much greater degree for some families. The ratios of the leading error coefficient to the next error coefficient, such as $\frac{A^7}{A^6}$ for $5(4)_6$ ERK pairs, has been mentioned by other authors [16, 30, 96, 128, 161]. Several authors have addressed $\frac{A^7}{A^6}$ and proposed limits so it does not adversely affect performance for six-stage fifth-order ERK: Sharp and Smart suggest $\frac{A^7}{A^6} < 2.5$ [161], Bogacki and Shampine suggest $\frac{A^7}{A^6} < 10.0$ (2.54) [16], and Papakostas and Papageorgiou suggest $\frac{A^7}{A^6} < 25.0$ [128]. However, experimentation by the author shows that limiting the actual magnitude of higher-coefficients such as A^7 is a better indicator than the just-shown ratios of actual performance solving IVPs.

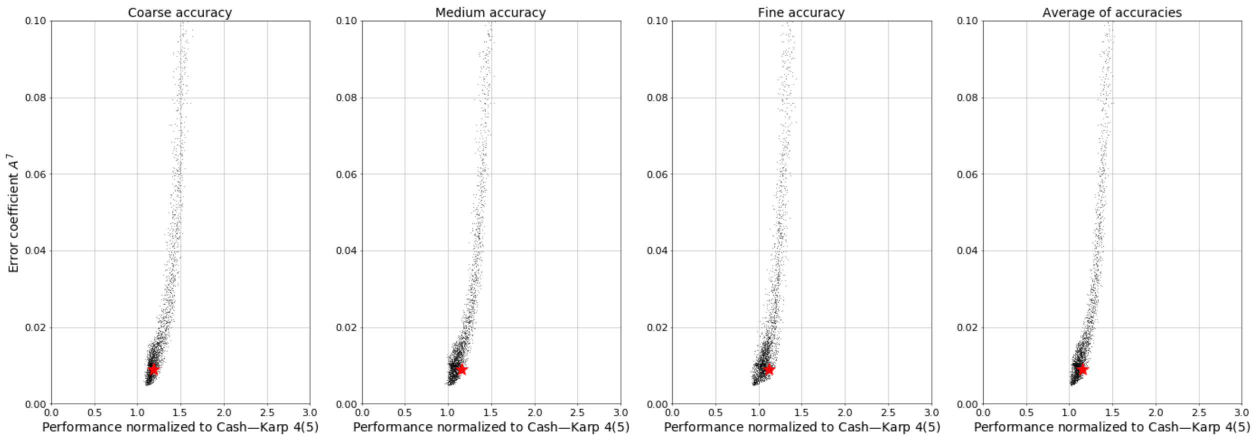
In particular, the $5(4)_{6(7)\mathrm{C}(2)}$ family has much larger ranges of many properties than other families and this means the performance difference between good and poor properties is often more obvious. This is in part because of the singularity for the $5(4)_{6(7)\mathrm{C}(2)}$ family discussed in Section 4.12.2, i.e., where $A^6 \to 0$ and $D \to \infty$ as $c_4 \to 1$ and $c_5 \to 1$ at a particular value of c_3, that seems to allow other error coefficients and properties to sometimes have a much wider range. The just described singularity and the wider ranges of other properties seem to be because the $5(4)_{6(7)\mathrm{C}(2)}$ family has all of c_3, c_4, c_5 as free parameters, whereas for nearly all of the other families studied in Chapter 3, with the exception of the family using Case I from Section 3.5.2 that uses c_5 instead, c_4 must be used for satisfying the order conditions of the fourth-order component.

The average performance solving the "non-stiff DE" test set is useful for illustrating the effect of higher-order error coefficients and showing the differences from different norms used for the error coefficients. The use of the ∞-norm in addition to the 2-norm appears to have been first discussed in the context of embedded RK pairs by Verner [189, 190]. For the IVPs tested in this thesis, the 2-norm was always observed to be the most predictive for evaluating potential performance based on the leading error term. However, the different possible norms of the PECs (2.47) defined by (2.55) were found to be useful for studying the contribution of the seventh-order error term. The average performance of 3200 randomly selected pairs from the $5(4)_{6(7)\mathrm{C}(2)}$ family with $A^6 \in [0.00035, 0.00040]$ solving the "non-stiff DE" test set in comparison to different seventh-order error coefficients is shown in Figure 5.16. The performance of the $5(4)_{6(7)\mathrm{C}(2)}$ family in relation to the value of A^7 is shown in Figure 5.16a. In Figure 5.16a, it can clearly be seen that a large A^7 is detrimental to overall performance because the pairs with the worst performance are almost always associated with relatively large A^7, whereas the best performing pairs always have an A^7 somewhat smaller than the $A^7 = 0.003956$ of the DP$5(4)_{6(7)}$ pair (2.79). This is particularly obvious at low accuracies, but is seen even at high accuracies where it would commonly be assumed that the leading error term would completely dominate performance. However, $|A^7|_1$ is clearly seen in Figure 5.16b to be even better at indicating poorly performing pairs because poorly performing pairs always have larger values of $|A^7|_1$ than is strictly necessary for the specific range of A^6 under study. The effectiveness of $|A^7|_1$ is probably because, in comparison to A^7, the 1-norm used by $|A^7|_1$ gives more weighting to individual PECs that are larger than strictly necessary for the specific range of A^6 under study. Although $|A^7|_\infty$ should still be examined when constructing new $5(4)_6$ ERK pairs, based on the experience with the IVPs examined during this study it seems to be less useful than $|A^7|_1$ and A^7.

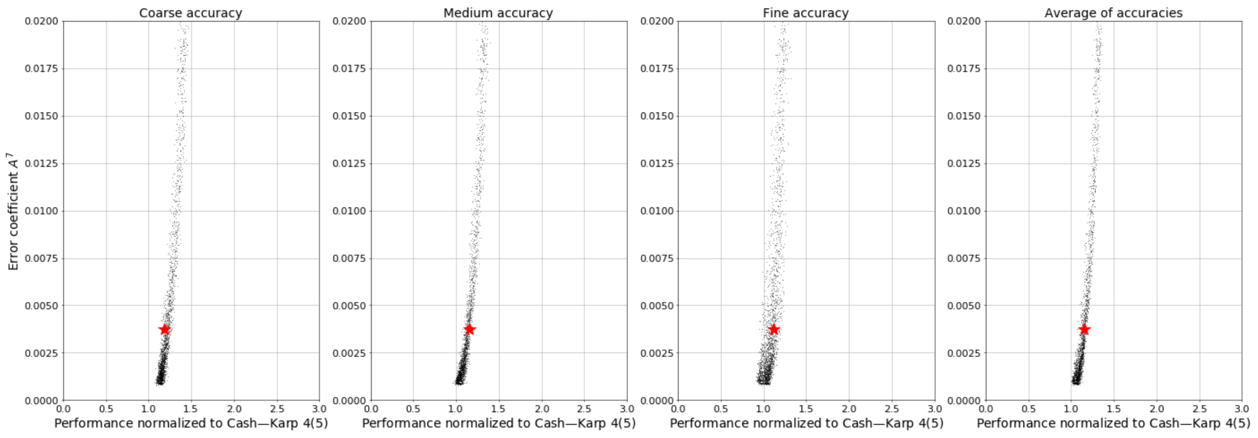
Error coefficients based on the $\overline{\text{PEC}}$s (2.48), which can be seen as the PECs (2.48) normalized to what their value would be if the numerical method was a simple truncation of the Taylor series of the local



(a) Using A^7 (2.55b).



(b) Using $|A^7|_1$ (2.55a).



(c) Using $|A^7|_\infty$ (2.55c).

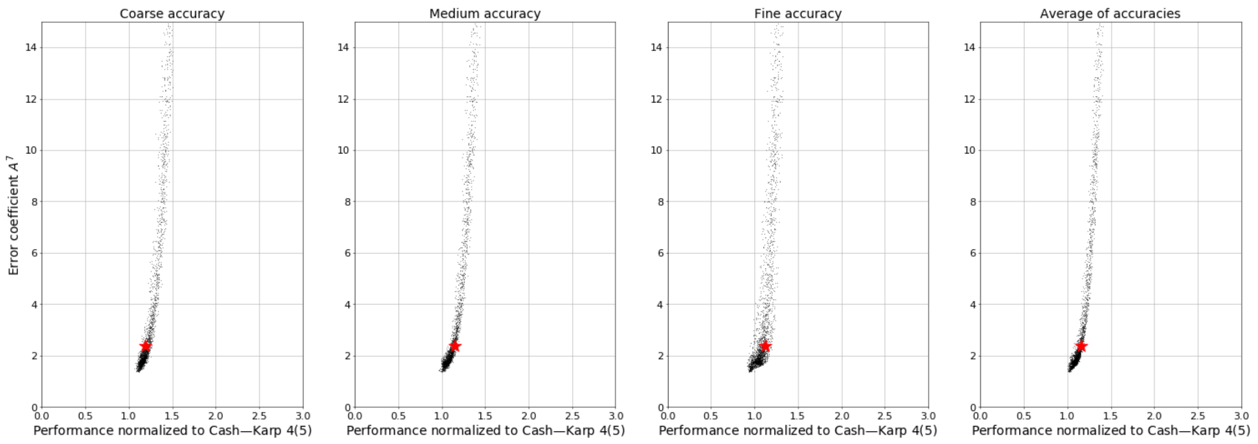
Figure 5.16: The performance solving the "non-stiff DE" test set of 3200 pairs with $A^6 \in [0.00035, 0.00040]$ from the $5(4)_{6(7)C(2)}$ family in comparison to different error coefficients that use the PECs (2.47) of the seventh-order error term. The red ★ symbol indicates the value of the seventh-order error coefficient and the performance of the DP$5(4)_{6(7)}$ pair (2.79).

solution (2.38), can also be examined. In Figure 5.17, the overall trend is similar to Figure 5.16 except that $|\bar{A}^7|_1$ appears to be an even better indicator of performance than $|A^7|_1$ for the $5(4)_{6(7)\mathrm{C}(2)}$ family. This is probably because individual PECs (2.48) may be under-represented due to the scaling by $\frac{1}{\sigma(\mathbf{F})}$ in (2.47). However, using the normalized definition, i.e., the $\overline{\text{PEC}}$s (2.48), ensures that PECs that are otherwise under-represented by (2.47) and have larger than appropriate magnitudes, now have a greater contribution when calculating the norm being used for the error coefficient.

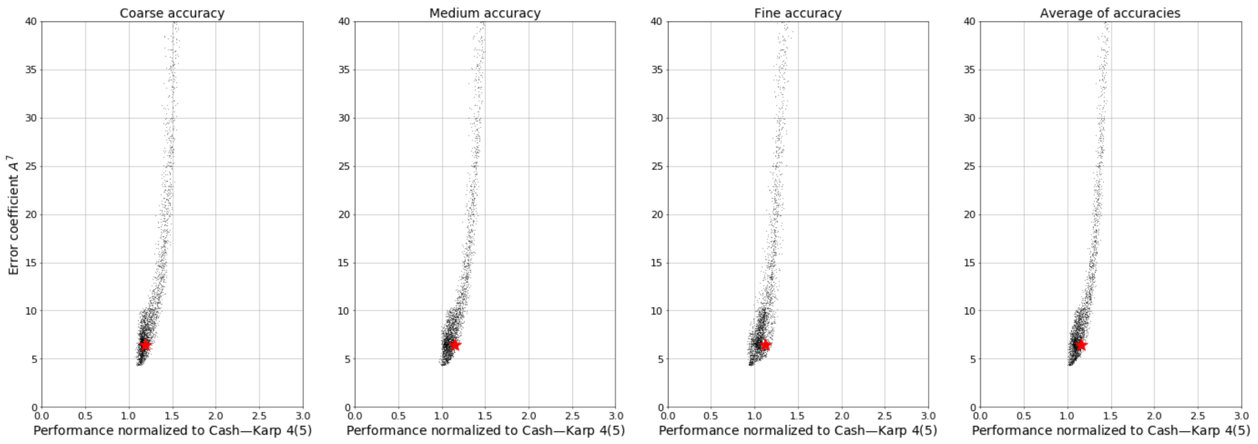
However, despite Figures 5.16 and 5.17 showing the importance of $|A^7|_1$ and $|\bar{A}^7|_1$ to the average performance solving the "non-stiff DE" test set, detailed examination of the performance data by the author (not shown) indicates that the most informative norm of the seventh-order error term is often IVP dependent. Further experimentation by the author (not shown) also showed that overly minimizing A^7, for example, by making $A^7 \leq 0.0020$ for $A^6 \in [0.00035, 0.00040]$ for the $5(4)_{6(7)\mathrm{C}(2)}$ family, is severely detrimental to performance solving some IVPs but not others. An additional guideline is that limiting the D characteristic number (2.56c) as much as possible is important because large PECs for error terms greater than seventh-order were observed to be correlated with large values of the D characteristic number (2.56c).

An investigation by the author (not shown) of the performance data on the "non-stiff DE" test set summarized in Figure 5.14 also supports Dormand and Prince's observation from 1986 [48], that is when the "non-stiff D" IVPs (2.13) are removed, the DP5(4)C$_{6(7)}$ pair (2.81) with $A^6 = 0.001489$ outperforms the DP5(4)$_{6(7)}$ pair (2.79) with $A^6 = 0.000399$. The author's hypothesis for Dormand and Prince's observation is that $A^7 = 0.002064$ for the DP5(4)C$_{6(7)}$ pair (2.81) is much smaller than $A^7 = 0.003956$ for the DP5(4)$_{6(7)}$ pair (2.79). This possible explanation of this historical observation must remain a hypothesis because it is difficult to compare the overall effect of the seventh-order error coefficients when A^6 is not held within a narrow range, such as what was done for Figures 5.16 and 5.17. However, it is important to mention because it indicates that despite the strong emphasis in the literature on the effect of the leading error term, there is historical precedent of observing that the leading error term may not be the only important factor.

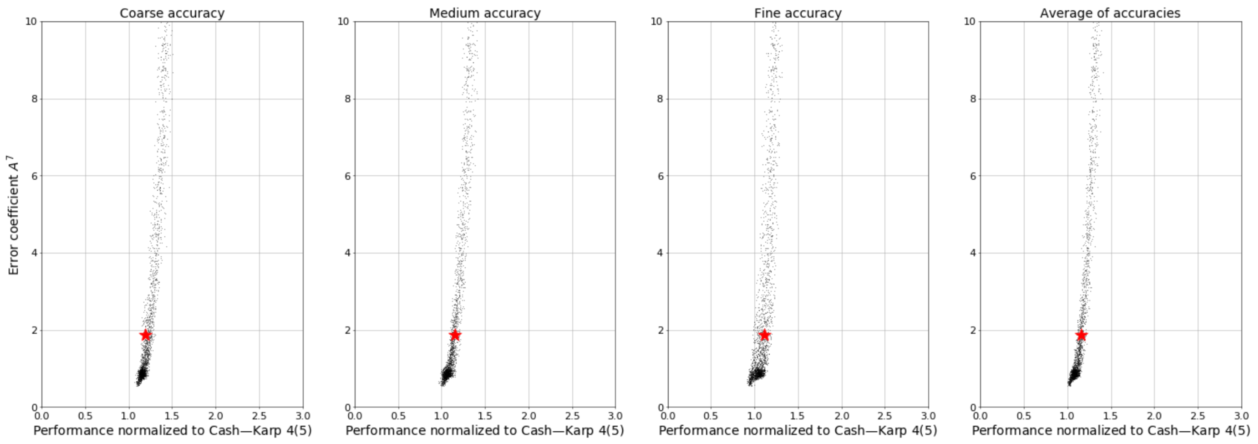
The popular DP5(4)$_{6(7)}$ pair (2.79) is definitely not "optimal" because it has $A^7 = 0.003956$ and there are pairs with $A^6 \leq 0.000400$ and A^7 as small as 0.00120, even though A^7 that small is definitely over-minimized for some IVPs. It is clearly seen in Figure 5.17a that the DP5(4)$_{6(7)}$ pair (2.79) performs as well as can be expected given value of A^7 and, with all other things being equal, finding an otherwise similar pair except with $A^7 = 0.0025$ instead could lead to a 20% improvement in average performance solving the "non-stiff DE" test set. However, an A^7 that might be somewhat larger than optimal, such as for the DP5(4)$_{6(7)}$ pair (2.79), seems to only result in a slight but still noticeable loss in performance (up to 40–50%) in comparison to the severe performance loss sometimes seen for pairs with an over-minimized A^7. This is likely why the DP5(4)$_{6(7)}$ pair (2.79) remains in use, the influence of the seventh-order error coefficient is modest and the non-optimality only becomes apparent with a substantial amount of performance data.



(a) Using $|\bar{A}^7|_2$ (2.55e).



(b) Using $|\bar{A}^7|_1$ (2.55d).



(c) Using $|\bar{A}^7|_\infty$ (2.55f).

Figure 5.17: The average performance solving the "non-stiff DE" test set of 3200 pairs with $A^6 \in [0.00035, 0.00040]$ from the $5(4)_{6(7)\mathrm{C}(2)}$ family in comparison to different error coefficients that use the $\overline{\text{PEC}}$s (2.48) of the seventh-order error term. The red ★ symbol indicates the value of the seventh-order error coefficient and the performance of the $\text{DP}5(4)_{6(7)}$ pair (2.79).

5.10 Correlations and tradeoffs in performance on select IVPs for ERK pairs from the same family

In this section, performance data is presented that shows that once the obviously undesirable properties, such as overly large error higher-order coefficients, are removed that it is probably not possible to find the "best" or "optimal" $5(4)_6$ ERK pairs for the extremely diverse classes of IVPs that are commonly solved in practice. Because of the enormous diversity of IVPs solved in practice, there is already a general expectation that there are generally no universally "best" IVP methods or "best" numerical methods. However, it is important to examine performance data to see how this manifests itself in practice and to determine the circumstances under which more optimal IVP methods can be constructed.

Even well-accepted and generally excellently performing pairs, such as the CK$4(5)_{6(6)}$ pair (2.83) or the DP$5(4)_{6(7)}$ pair (2.79), have important representative IVPs that they do not perform comparatively well on, in addition to other important representative IVPs that they do perform comparatively well on. Some of the performance variation of IVP methods with similar numerical properties can be attributed to the discrete number of steps and that quantities based on theoretical considerations, such as the PECs up to a certain order, do not fully represent the numerical solution. In addition, many theoretical properties only hold exactly in an asymptotic limit and they cannot be used to exactly predict performance for solving IVPs in practice. However, the tradeoffs shown below and examining specific PECs in the remainder of this chapter show that the substantial amount of variation in performance for relatively small ranges of the leading error coefficient seen in Figures 5.11–5.14 can be substantially reduced for IVPs based on the Kepler IVP (2.13). More importantly, this can help guide future work to reduce the amount of unexplained performance variation when solving additional classes of IVPs.

To demonstrate that there are $5(4)_6$ ERK pairs that are more "optimal" than others for solving IVPs from celestial mechanics based on the Kepler IVP (2.13), the relative performance of 3200 randomly selected pairs from each of the $5(4)_{6(6)\mathrm{C}(2)}$ family (with $A^6 \in [0.00095, 0.00100]$ that would contain the CK$4(5)_{6(6)}$ pair (2.83)), $5(4)_{6(7)\mathrm{C}(2)}$ family (with $A^6 \in [0.00035, 0.00040]$ that would contain the DP$5(4)_{6(7)}$ pair (2.79) and $A^7 < 0.0050$), and the Case VIb family (with $A^6 \in [0.00100, 0.00110]$ because $A^6 < 0.00100$ was observed to hurt performance and $\mu = 1, \lambda \in [-1, 1]$) are shown in Figures 5.18, 5.19, and 5.20, respectively. In this section, the performance presented for each $5(4)_6$ ERK pair is the average at the coarse, medium, and fine accuracies chosen for the particular IVP or set of IVPs. The author also examined each of the coarse, medium, and fine tolerances individually (not shown), and the tradeoffs appeared nearly the same as seen in Figures 5.18, 5.19, and 5.20. The rows of Figures 5.18, 5.19, and 5.20 correspond to instances of the Kepler IVP from the "non-stiff D" IVPs (2.13) (with eccentricities $\epsilon \in [0.1, 0.3, 0.5, 0.7, 0.9]$) in addition to the "non-stiff D6" IVP (2.13) with $\epsilon = 0.99$. The columns of Figures 5.18, 5.19, and 5.20 correspond to examples of IVPs from celestial mechanics that are more complex than the simple Kepler IVP (2.13), i.e., the "Arenstorf orbit" IVP (2.15), the "Pleiades" IVP (2.16), the "Jovian" IVP (solved to 7200 days) [157], the

"Jovian asteroid" IVP (solved to 7200 days) [160], and the "Nine planets" IVP (solved to 7200 days) [157], in addition to the average performance solving the "non-stiff D" IVPs (2.13).

What can clearly be seen from Figures 5.18, 5.19, and 5.20 is that for each of the more complex IVPs from celestial mechanics, the performance solving them is strongly correlated with at least one of the "non-stiff D" IVPs (2.13). However, the correlation is rarely with the specific "non-stiff D" IVP (2.13) that would be expected, based on the eccentricities present in the individual orbits that constitute the more complex IVP from celestial mechanics. For instance, the "nine planets" IVP (with eccentricities of the individual orbits of at most 0.09) has strongly correlated performance with the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$ but is less correlated with the "non-stiff D1" IVP (2.13) with $\epsilon = 0.1$. The performance solving the Pleiades and the "Jovian asteroid" IVPs are most strongly correlated with the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$, in addition to being reasonably well correlated with the "non-stiff D6" IVP (2.13) with $\epsilon = 0.99$. This indicates that the performance solving simpler Kepler-based IVPs with eccentricities close to 1.0 is correlated to the performance solving more complex Kepler-based IVPs with close approaches. The performance solving the more complex IVPs from celestial mechanics derived from the Kepler IVP (2.13) is also fairly well correlated with the average performance solving the "non-stiff D" IVPs (2.13), which for illustrative purposes is in both a row and a column of Figures 5.18, 5.19, and 5.20.

These correlations in performance are actually contrary to the discussion in a recent paper by Sharp, Castillo-Rogez, and Grazier from 2012 [160]. Where they studied a different class of ERK methods (Runge–Kutta–Nyström methods, which are specific to IVPs based on second order ODEs (2.4)) they observed that performance on the basic Kepler problem (2.13) was often not applicable to larger/more complex problems from celestial mechanics in terms of performance for numerical methods. This was done specifically by Sharp et al. by solving Kepler IVPs (2.13) with only a few numerical methods but many eccentricities $\epsilon \in [0.0, 0.9]$, a range they justified because the maximum eccentricity of the planetary orbits is $\epsilon = 0.09$ and that the vast majority of minor bodies also have eccentricities in this range [160]. However, despite Sharp et al. [160] including authors from JPL (the Jet Propulsion Laboratory),[32] that study was not able to use the large amount of performance data on thousands of individual IVP methods that is now available by using `OCSage` and `pythODE`. Excellent future work requiring significant computational resources would be combining the long integration times of Sharp et el. [160] with the techniques of this study.

The "Arenstorf orbit" IVP (2.15) is somewhat different from other IVPs based on the Kepler IVP (2.13) because the performance solving it is less correlated with the "non-stiff D" IVPs (2.13) in comparison to the other IVPs from celestial mechanics. This somewhat different numerical behaviour might be because the "Arenstorf orbit" IVP (2.15) has an incredibly close pass of the massless satellite (corresponding to an eccentricity extremely close to 1.0) or it might be because the "Arenstorf orbit" IVP (2.15) uses a rotating frame of reference.

[32] https://www.jpl.nasa.gov/

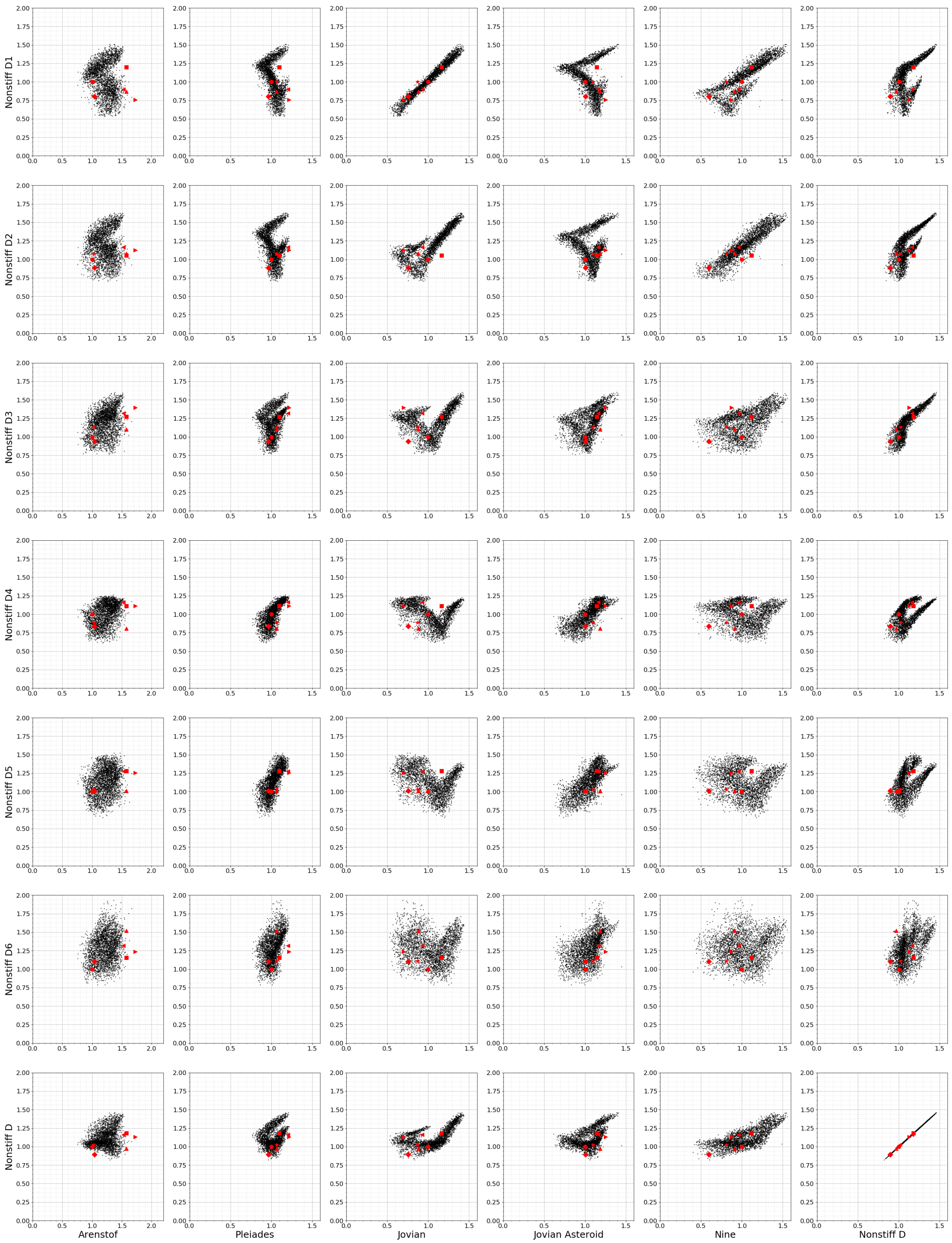


Figure 5.18: The performance solving the "non-stiff D" IVPs (2.13) in comparison to other celestial mechanics IVPs using 3200 randomly selected pairs from the $5(4)_{6(6)\mathrm{C}(2)}$ family with $A^6 \in [0.0010, 0.00110]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

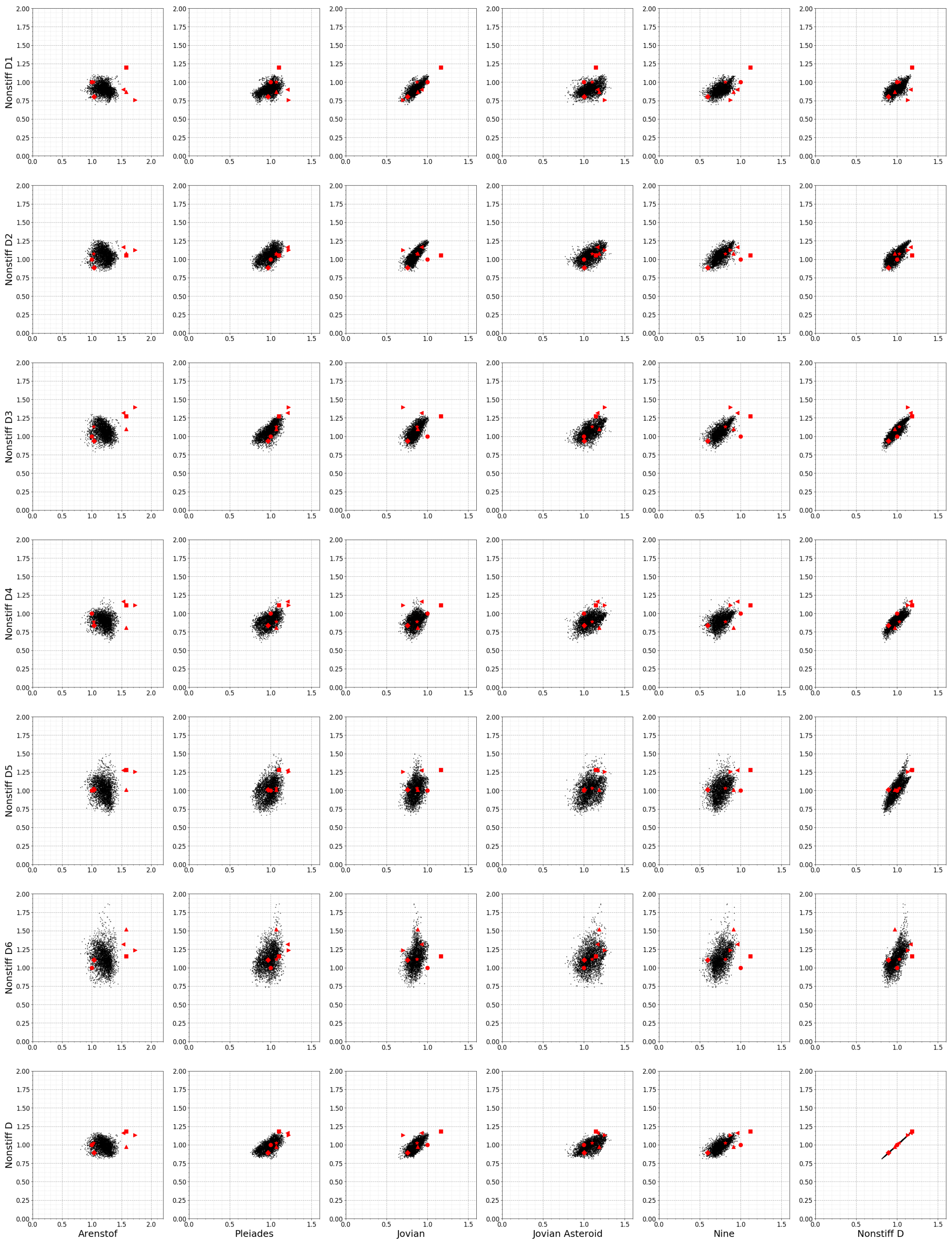


Figure 5.19: The performance solving the "non-stiff D" IVPs (2.13) in comparison to other celestial mechanics IVPs using 3200 randomly selected pairs from the $5(4)_{6(7)C(2)}$ family with $A^6 \in [0.00035, 0.00040]$ and $A^7 \leq 0.0050$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

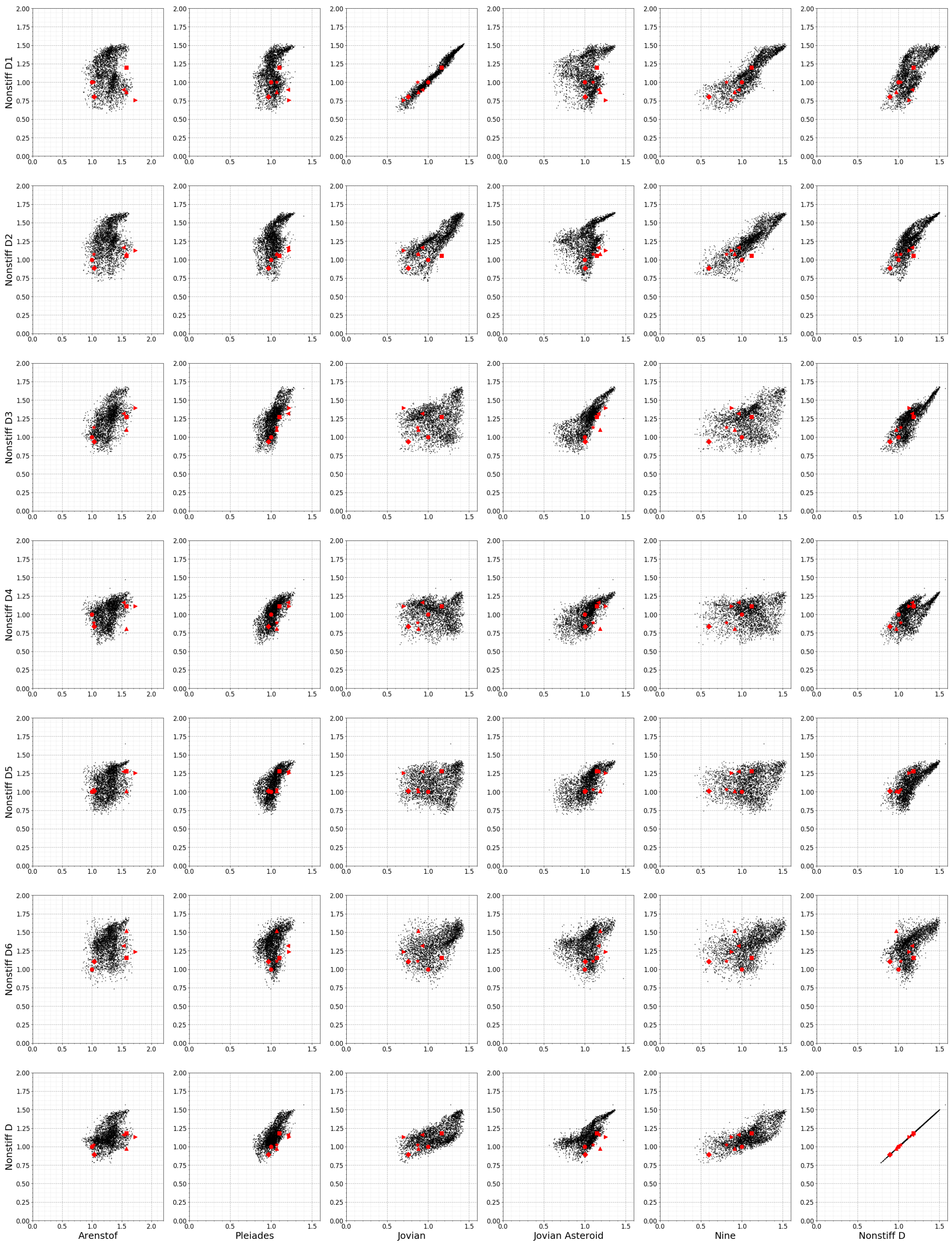


Figure 5.20: The performance solving the "non-stiff D" IVPs (2.13) in comparison to other celestial mechanics IVPs using 3200 randomly selected pairs from the family of Case VIb with $A^6 \in [0.0010, 0.00110]$, $\mu = 1, \lambda \in [-1, 1]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

5.10.1 Tradeoffs in performance when solving IVPs other than celestial mechanics IVPs

In contrast, IVPs from “non-stiff B” class of IVP along with the “Euler rigid body” IVP published by Hairer et al. [72, pgs.244–245], which were discussed in Section 5.3, are not well correlated in performance, either with each other or with the celestial mechanics IVPs just discussed. This is shown by Figures 5.21 and 5.22 for the $5(4)_{6(6)\mathrm{C}(2)}$ and $5(4)_{6(7)\mathrm{C}(2)}$ families, respectively. It can be seen that non-correlated performance and tradeoffs are more common than the performance being correlated. However, this study focus on systematically constructing and finding the reasons that some pairs are more optimal for IVPs from celestial mechanics based on the Kepler IVP (2.11). Therefore, finding systematic construction methodologies and more optimal ERK pairs for other classes of IVPs will have to be left as future work.

5.11 Finding the best values of individual $\overline{\mathrm{PEC}}$s for the higher-order component of $5(4)_6$ ERK pairs for efficiently solving IVPs from celestial mechanics

The primary application of the efficient $5(4)_6$ ERK pairs constructed for this study are IVPs from celestial mechanics that are based on the Kepler IVP (2.13), such as the IVPs that are the subject of Figures 5.18, 5.19, and 5.20. IVPs from celestial mechanics are the obvious class for a deeper study into the differences in performance between pairs because there are several dozen IVPs from celestial mechanics in the IVP test sets implemented by `pythODE`. Furthermore, the author’s experience during this study shows that having many available examples of IVPs of a particular class appears to be essential for reaching well-defined conclusions on the efficiency of ERK pairs based on performance data. IVPs from celestial mechanics were also the class of IVP that most obviously had more optimal $5(4)_6$ ERK pairs. This is probably because many important IVPs from celestial mechanics are based directly on and have a similar mathematical structure to the simple Kepler IVP (2.13). In addition, these pairs that are efficient for IVPs from celestial mechanics generally work well as general-purpose $5(4)_6$ ERK pairs for most other IVPs described in Section 5.3.

The most important factors found in determining the performance on many of the IVPs tested with $5(4)_6$ ERK pairs (when using local extrapolation) are the specific values of individual PECs of the higher-order component. Obviously, the PECs are a concept that has appeared throughout the literature on RK methods because they form the basis of many of the classic characteristic numbers (2.56). However, the overall importance of specific ranges of values for specific PECs has not previously been discussed in the literature. In fact, to find the best performing and avoid the worst performing $5(4)_6$ ERK pairs for solving IVPs from celestial mechanics, certain PECs must have values within certain narrow ranges specific to the individual PEC. In Figures D.1 and D.2, additional performance data is shown indicating that individual examples from other classes of IVPs exhibit a similar phenomenon. Currently, the optimal ranges for the PECs must be

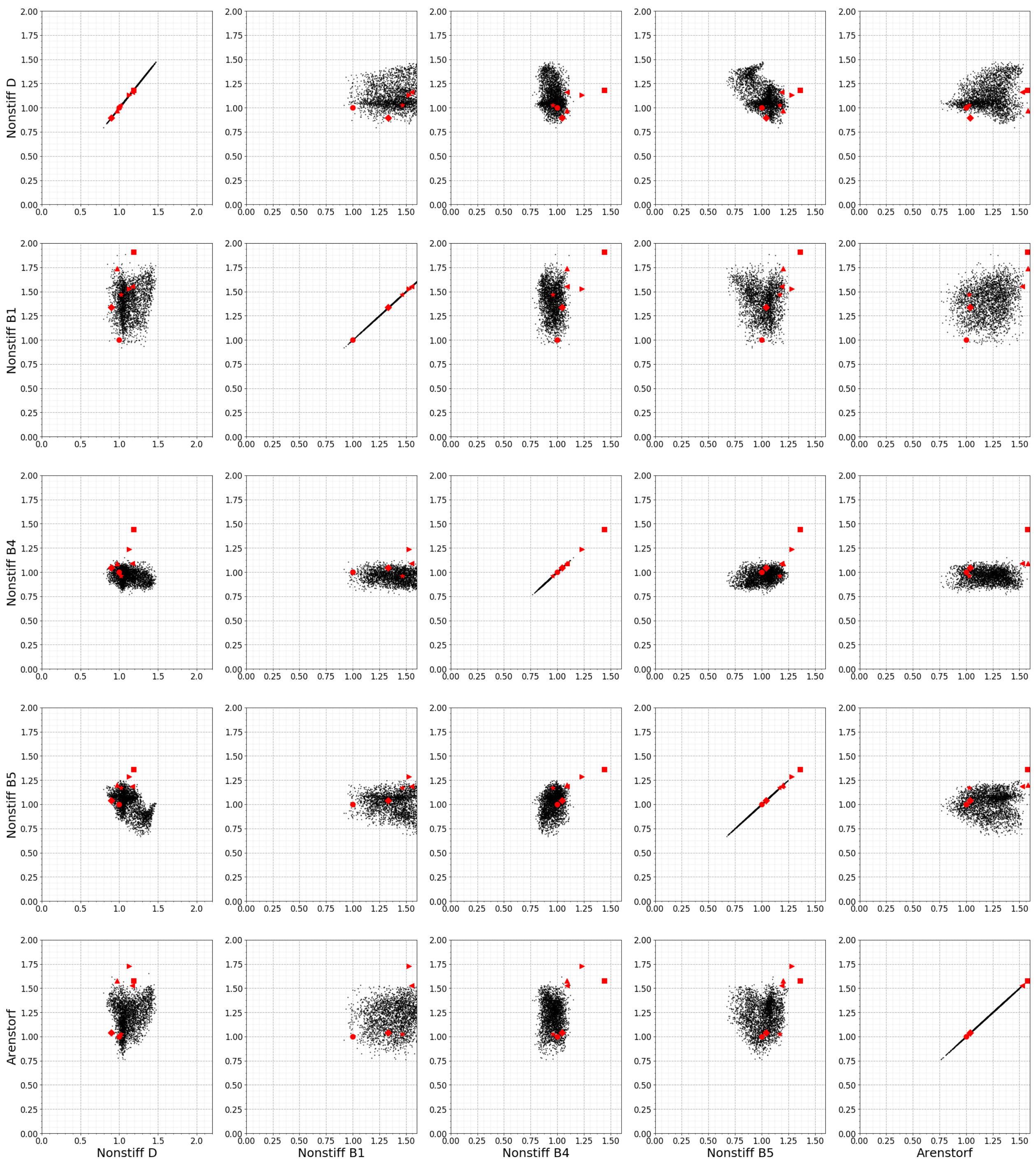


Figure 5.21: The tradeoffs in performance between solving selected IVPs and the "non-stiff D" IVPs (2.13) using randomly selected pairs from the $5(4)_{6(6)\mathrm{C}(2)}$ family. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

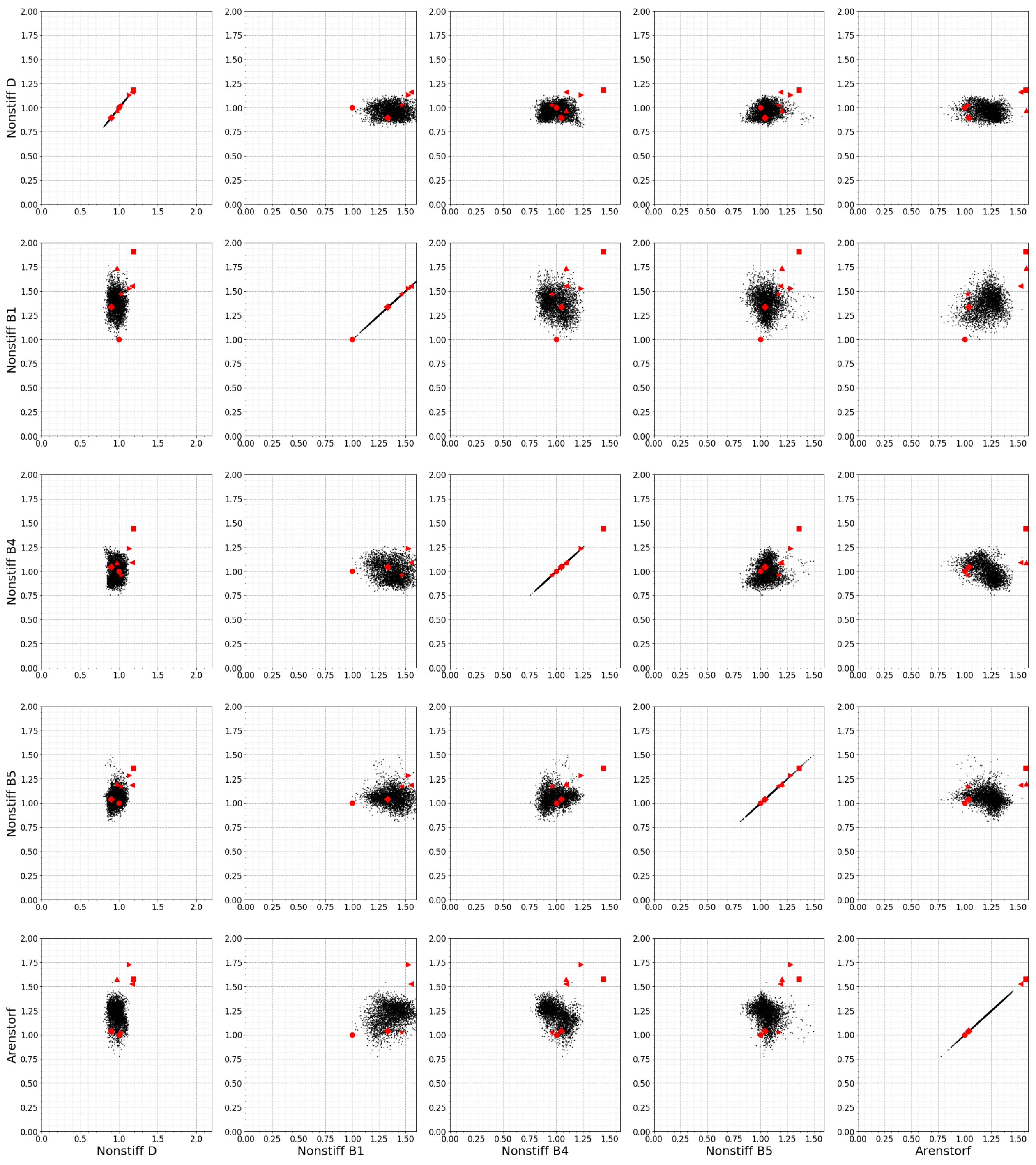


Figure 5.22: The tradeoffs in performance between solving selected IVPs and the "non-stiff D" IVPs (2.13) using randomly selected pairs from the $5(4)_{6(7)\mathrm{C}(2)}$ family and $A^7 \leq 0.0050$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

found empirically without the support of a theoretical explanation. However, now that it is known that ideal ranges of PECs exist for different classes of IVPs, this phenomenon cannot be ignored when constructing new ERK pairs. This is because even if the leading error coefficient is small and other classic characteristic numbers (2.56) are in acceptable ranges, for some applications there can still be a performance difference of up to a factor of two or more between ERK pairs that otherwise appear to have similar properties. If a theoretical explanation of this phenomena is discovered in the future, this will further increase the reliability and confidence that IVP methods that are the most efficient and suitable can be selected for a particular class of IVP.

Important IVPs from celestial mechanics are often solved to an extremely long duration in problem time and therefore can take substantial computational effort to solve. Therefore, the number and density of data points seen in the figures in this section cannot be readily examined for larger celestial mechanics IVPs integrated for longer periods of time, given the current limitations in available hardware and software. However, the author did limited testing (not shown) with IVPs such as the "nine planets" IVP (solved to 7200 days) and found that the behaviour shown in this section is still present. In addition, in the conclusion of this chapter it is shown that representative pairs with the best properties do in fact perform extremely well (and sometimes unexpectedly well on individual IVPs) solving the larger IVPs from celestial mechanics integrated for longer periods of time.

The searches of the space of free parameters that `OCSage` allows, discussed in Chapter 4, mean that examining the values of the individual PECs for extremely large numbers (millions) of ERK pairs is now possible. Testing the performance of smaller subsets (thousands) of these ERK pairs with `pythODE` allows observing whether some PECs are more important than others for some classes of IVPs.

5.11.1 Examining the effect of the $\overline{\text{PEC}}$ corresponding to the rooted tree $[\tau[[\tau^2]]]$

The PEC that most obviously has more optimal values when solving IVPs from celestial mechanics using $5(4)_6$ ERK pairs is the one corresponding to the rooted tree $[\tau[[\tau^2]]]$. For quite a few other IVPs tested (not just IVPs from celestial mechanics) it was also observed that the value of the $\overline{\text{PEC}}$ (2.48) corresponding to the rooted tree $[\tau[[\tau^2]]]$ is the one having the most dominant effect on performance. In Figures 5.23–5.27, the performance of $5(4)_6$ ERK pairs from selected families on selected IVPs is shown for 6400 randomly selected values in the interval $[-0.040, 0.040]$ of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$. This is done by taking 400 randomly selected points in each interval of size 0.005. The family used in each of Figures 5.23–5.27 and the range of leading error coefficients used is given below when the particular figure and family are discussed in detail.

The $5(4)_{6(6)\text{C}(2)}$ family

It can clearly be seen from Figure 5.23 that the best performance for the $5(4)_{6(6)\text{C}(2)}$ family with $A^6 \in [0.00095, 0.00100]$ on IVPs from the "non-stiff D" IVPs (2.13) occurs when the $\overline{\text{PEC}}$ (2.48) corresponding to

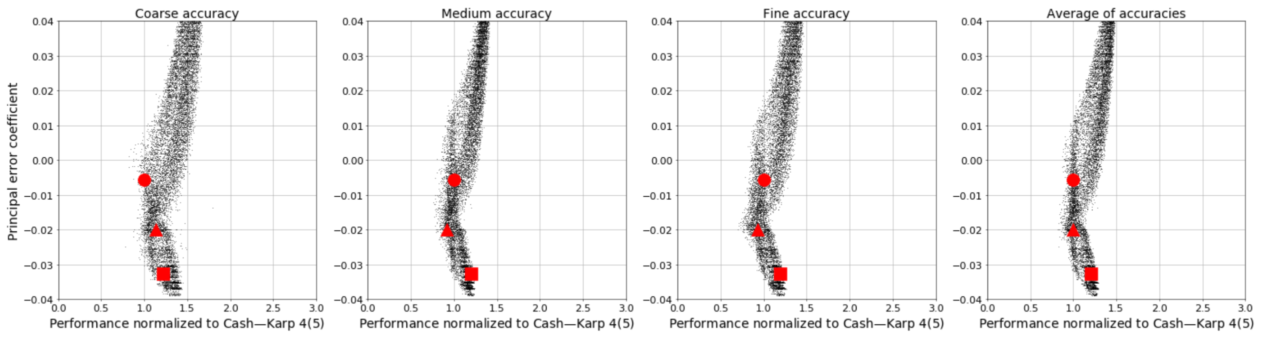
$[\tau[[\tau^2]]]$ has a value of about -0.020. The range of values where the best-performing pairs occur is wider for the individual "non-stiff D2" and "non-stiff D5" IVPs (2.13) in Figures 5.23b and 5.23c, respectively, in comparison to the average performance solving the "non-stiff D" IVPs (2.13) in Figure 5.23a. In addition, in Figure 5.23d the "Arenstorf orbit" IVP (2.15) has an optimal value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ of around -0.010.

The $5(4)_{6(7)\text{C}(2)}$ family

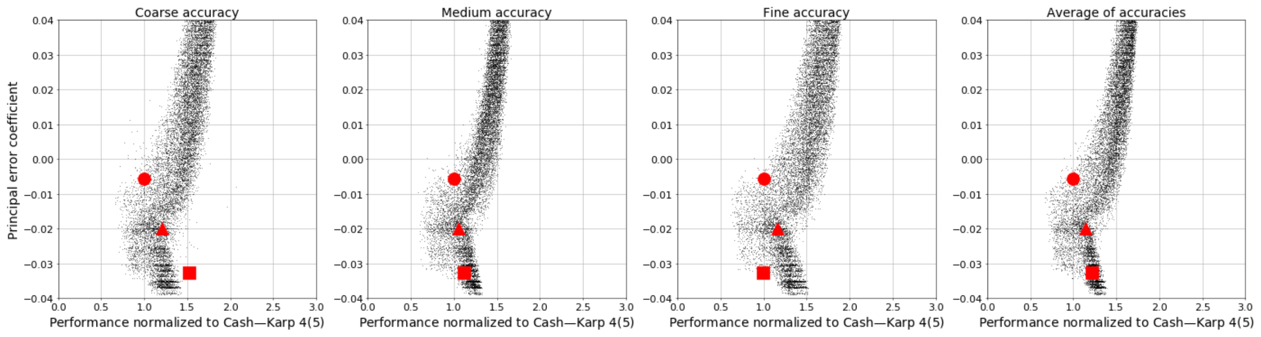
In contrast to the $5(4)_{6(6)\text{C}(2)}$ family, it can clearly be seen from Figure 5.24 that for the $5(4)_{6(7)\text{C}(2)}$ family with $A^6 \in [0.00035, 0.00040]$, the best performance on IVPs from the "non-stiff D" IVPs (2.13) occurs when the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ has a value slightly less than 0.005. The numerical behaviour is also quite different for the "Arenstorf orbit" IVP (2.15) in comparison to IVPs based more directly on the Kepler IVP (2.13) because the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ does not appear to have an optimal value. However, Figure 5.28 shows values of the $\overline{\text{PEC}}$s (2.48) corresponding to $[\tau^2[[\tau]]] \equiv [[\tau][[\tau]]]$ (this specific equivalence between trees holds for any family of ERK pairs that satisfy the C(2) simplifying assumptions (2.44b)) is actually the most dominant factor in performance for the "Arenstorf orbit" IVP (2.15) but does not have a significant impact on performance solving the "non-stiff D" IVPs (2.13). It was also observed (not shown) that the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau^2[[\tau]]] \equiv [[\tau][[\tau]]]$ does not appear to have an optimal value for the "Arenstorf orbit" IVP (2.15) when it is solved with pairs from the $5(4)_{6(6)\text{C}(2)}$ family. This indicates that although there are general trends concerning the PECs that might have optimal values, the specific optimal values of the $\overline{\text{PEC}}$s (2.48) can be dependent on the specific IVP and family of ERK pairs.

The families using Case VI

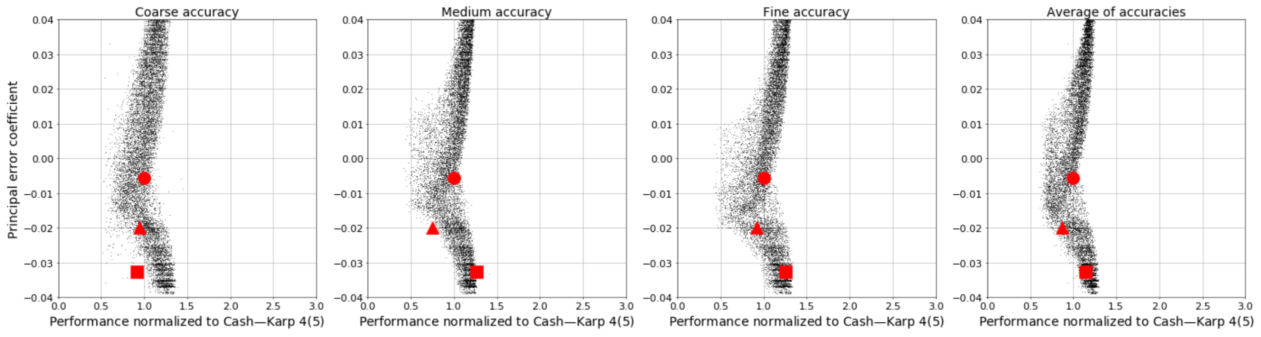
In Figures 5.25 and 5.26, the performance is shown for the families from cases VIa and VIb, respectively, from Section 3.5.1 in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$. The optimal value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ for the family from Case VIa is relatively close to zero for the "non-stiff D" IVPs (2.13), close to -0.01 for the "non-stiff D2" IVP 5.23b, and extremely broad for the "non-stiff D5" IVP (2.13), whereas for the family from Case VIb the optimal is extremely broad for the "non-stiff D" IVPs (2.13), it has a strong optimal value of -0.01 for the "non-stiff D2" IVP 5.23b, and it has extremely broad optimal value for the "non-stiff D5" IVP (2.13) that trends towards positive values. As seen when these families were constructed in Chapter 3 and from the listings of generated code in Section 4.7, the expressions for the Butcher tableau coefficients in terms of the free parameters are much more complex than if the C(2) simplifying assumptions (2.44b) hold. Therefore, it is unsurprising that the correlation between the values of specific $\overline{\text{PEC}}$s (2.48) and performance appears to be more complex for these families. However, although it is not something that could be thoroughly studied here, based on these broader ranges of optimal values and other experimentation done for this study, the author feels that there may be fewer performance tradeoffs between solving different IVPs when using $5(4)_6$ ERK pairs from families without the



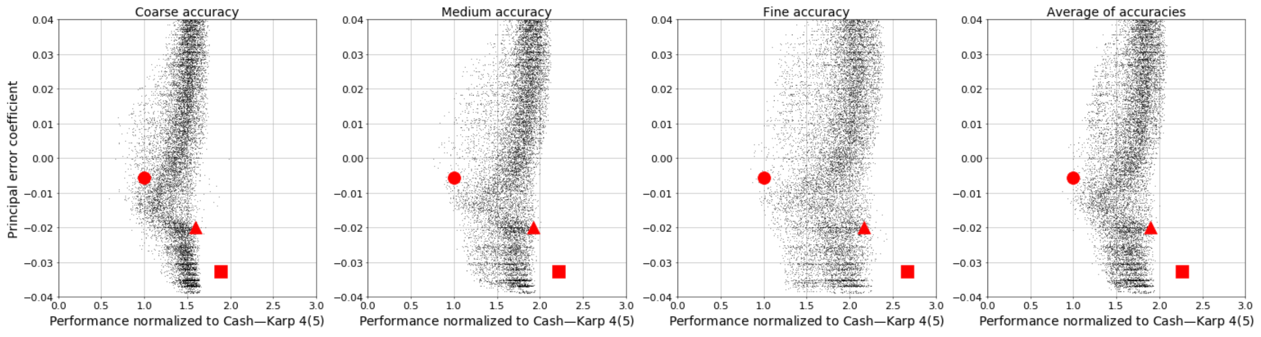
(a) The average performance solving the “non-stiff D” IVPs (2.13).



(b) The performance solving the “non-stiff D2” IVP (2.13) with $\epsilon = 0.3$.

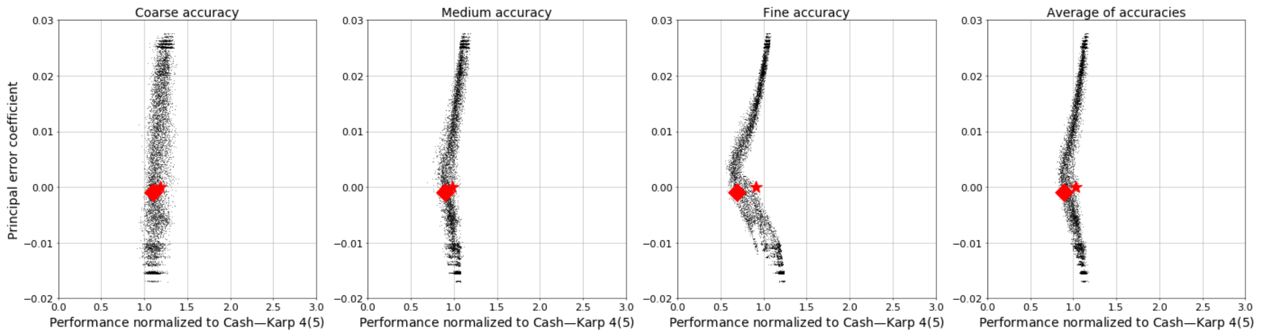


(c) The performance solving the “non-stiff D5” IVP (2.13) with $\epsilon = 0.9$.

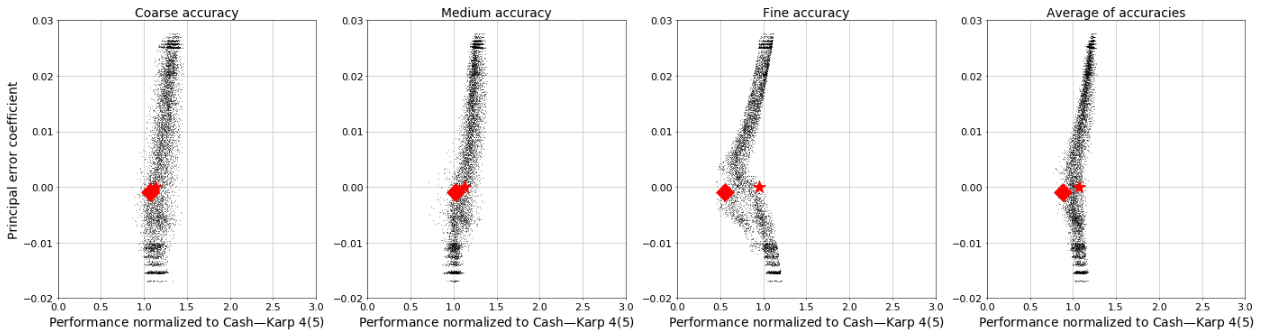


(d) The performance solving the “Arenstorf orbit” IVP (2.15).

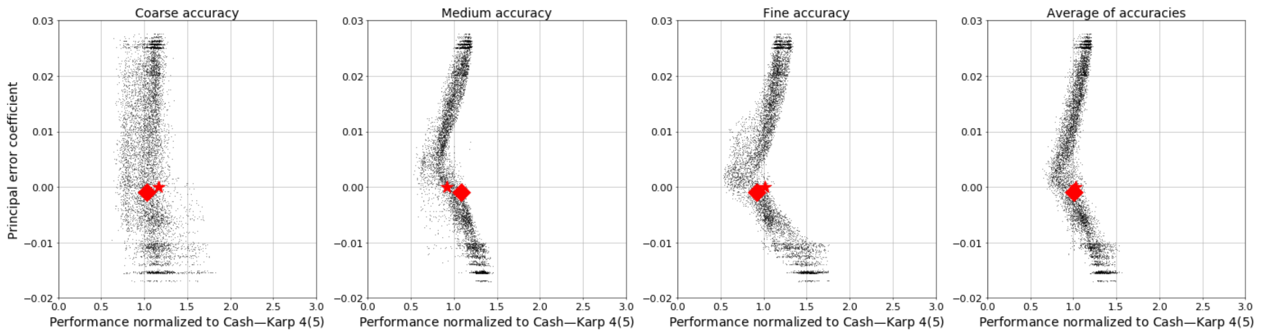
Figure 5.23: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ for the $5(4)_{6(6)\text{C}(2)}$ family with $A^6 \in [0.00095, 0.00100]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



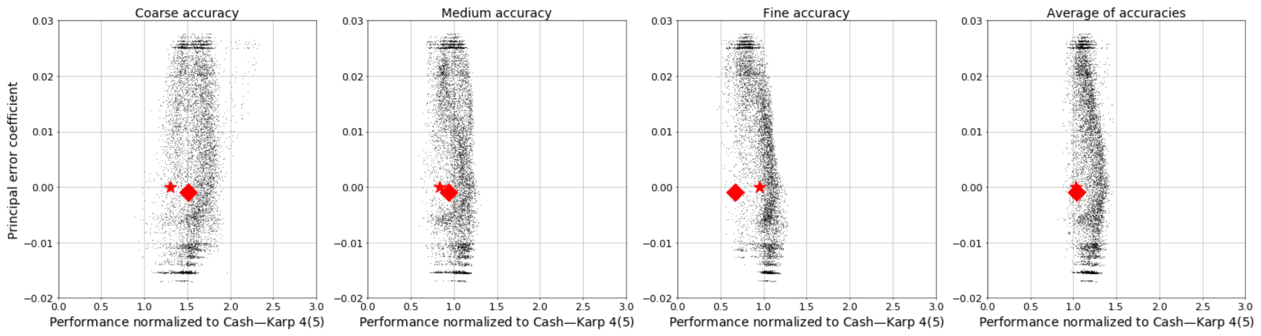
(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.



(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.



(d) The performance solving the "Arenstorf orbit" IVP (2.15).

Figure 5.24: The value of the $\overline{\text{PEC}}$s (2.48) corresponding to $[\tau[[\tau^2]]]$ for the $5(4)_{6(7)\text{C}(2)}$ family. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

C(2) simplifying assumptions (2.44b).

The families using Case V

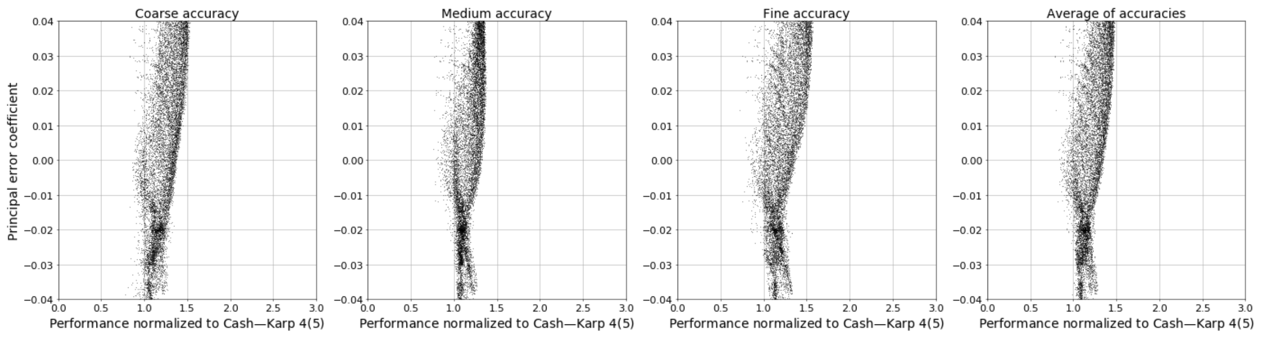
In Figure 5.27 the limitations of families with fewer free parameters, in comparison to other families, can be seen from the performance of Cases Va and Vb from Section 3.5.1. It can clearly be seen that for the "non-stiff D" IVPs (2.13) that Case V does not have enough freedom in its free parameters to find pairs that are competitive with the other families just discussed. Somewhat more freedom in parameter choice can be found by using larger values of A^6. However, this is still not enough for ERK pairs from Case V to be competitive with those from other families.

The requirement of finding families with sufficient free parameters

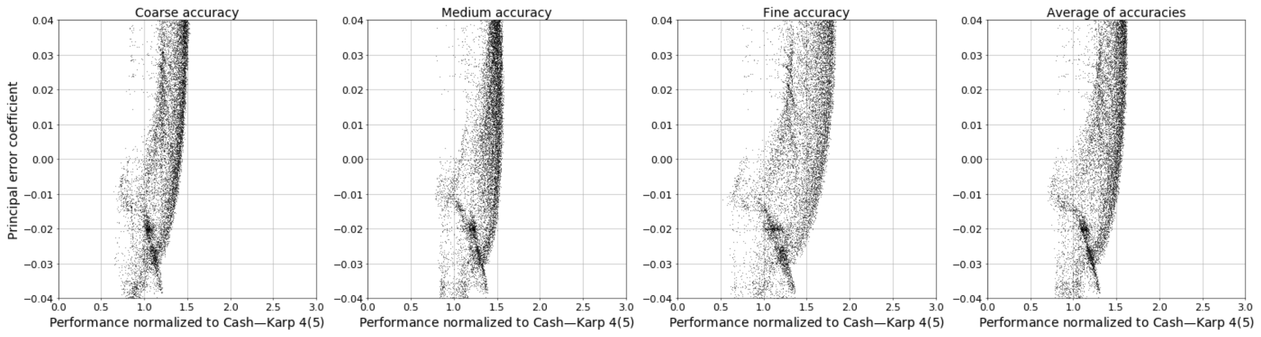
These observations of different families and the need for enough free parameters (degrees of freedom) mean that a significant advantage to finding the complete solution of the order conditions is that it removes uncertainty about not having enough of the correct free parameters or that the best case might not be the one that was solved for. In addition, the sometimes poor performance of an aggressively minimized leading error coefficient was often observed to be due to the often small region of space of free parameters near the minimal value of the leading error coefficient, which usually has $\overline{\text{PEC}}$s (2.48) that deviate from otherwise more optimal values. The disadvantage of an aggressively optimized leading error coefficient can clearly be seen in Figures 5.12 and 5.15 where the smallest leading error coefficients have a limited performance range that is much worse than the best performance from the family overall. This observation of the sometimes poor performance of aggressively minimized leading error coefficients probably applies to other properties. Numerical optimization to find the smallest leading error coefficient [81, 85, 123, 128, 129, 130, 180, 194] or stability properties [62, 97, 107, 123, 129, 130, 177] are sometimes the subject of publications. However, the practical merits of aggressively optimizing properties of IVP methods should be reconsidered.

A hypothesis as to why there are optimal values of PECs

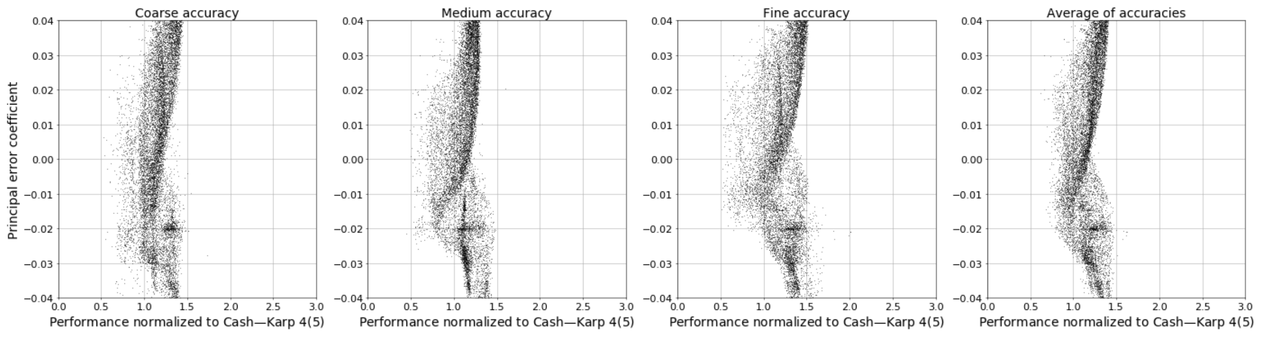
A clue as to why there may be optimal values of some PECs can be seen in Figure 5.29. It can be seen that there is no optimal value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ for the "Jovian" IVP solved for 400 days, where less than 10% of an orbit is completed for any of the simulated planets. In constrast, the "Jovian" IVP solved for 7200 days, where there are almost two complete orbits of Jupiter and two-thirds of an orbit of Saturn, shows a strong optimal value for $[\tau[[\tau^2]]]$. When the "nine planets" IVP is solved for only 20 days, with only less than a third of an completed for any of the planets, at high accuracy there is a somewhat distinct optimal value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$. However, a much stronger optimal value of the PEC corresponding to $[\tau[[\tau^2]]]$ appears when the "nine planets" IVP is solved for 400 days, with full orbits being completed by the three inner planets. Based on these observations (as well as other similar ones by the author), it is reasonable to hypothesize that the optimal value of the $\overline{\text{PEC}}$ (2.48) corresponding to



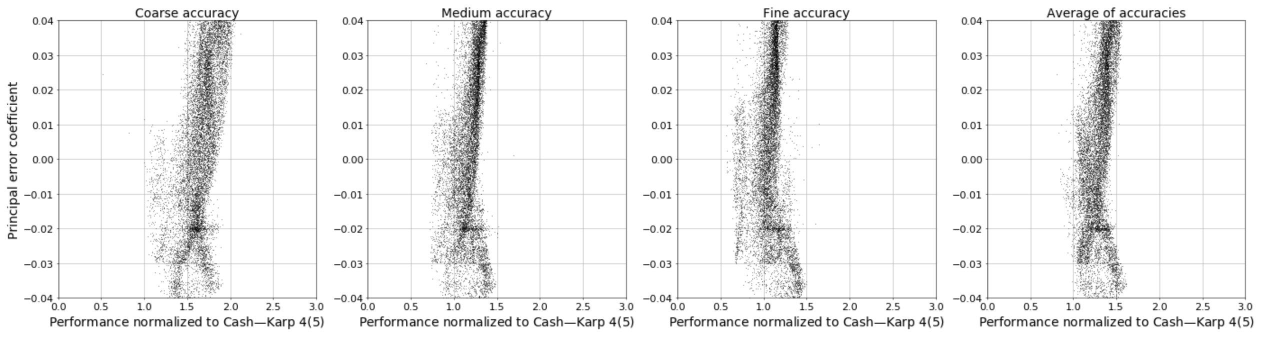
(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.

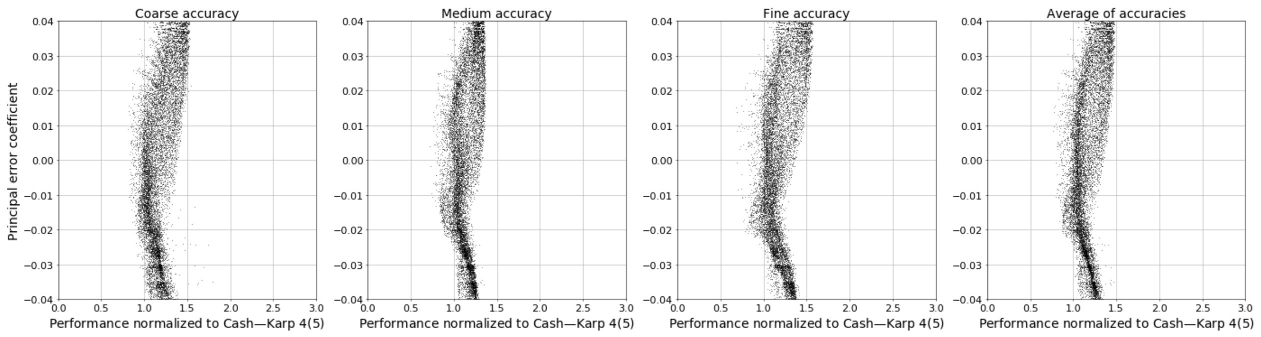


(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.

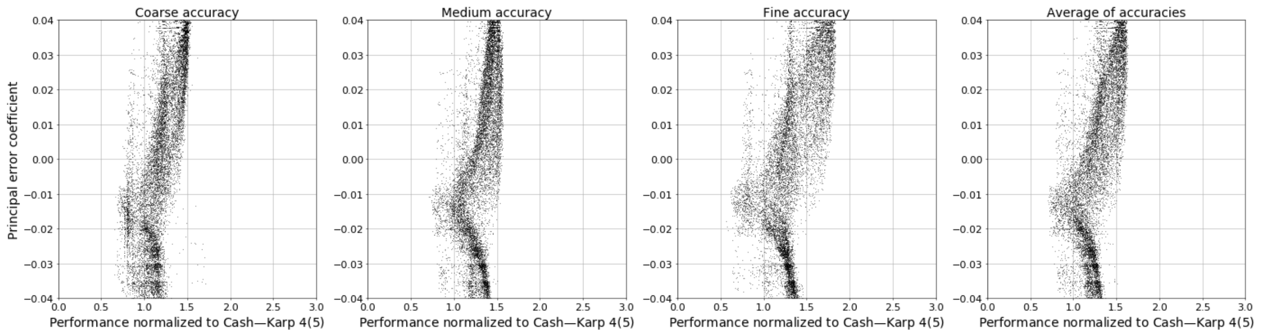


(d) The performance solving the "Arenstorf orbit" IVP (2.15).

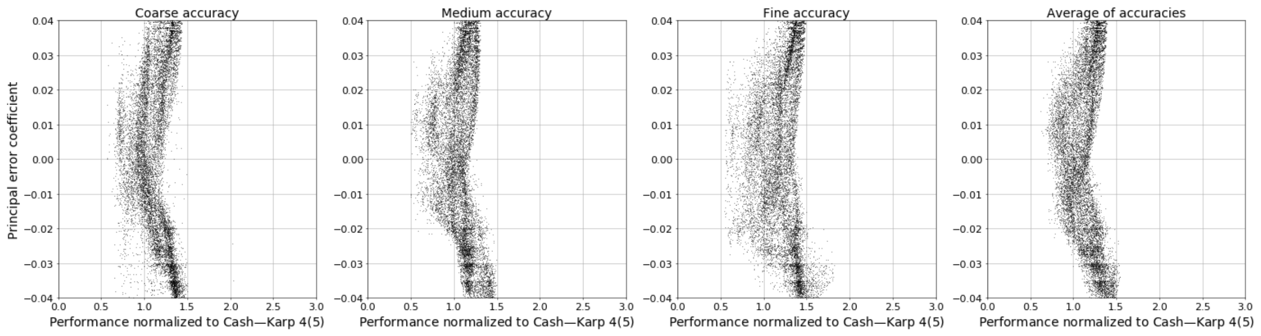
Figure 5.25: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ for the family using Case VIa from Section 3.5.1.



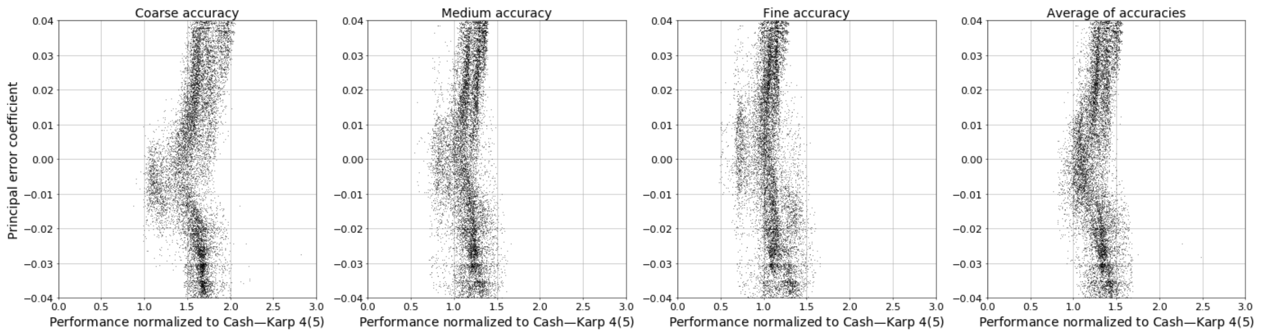
(a) The performance solving the “non-stiff D” IVPs (2.13).



(b) The performance solving the “non-stiff D2” IVP (2.13) with $\epsilon = 0.3$.

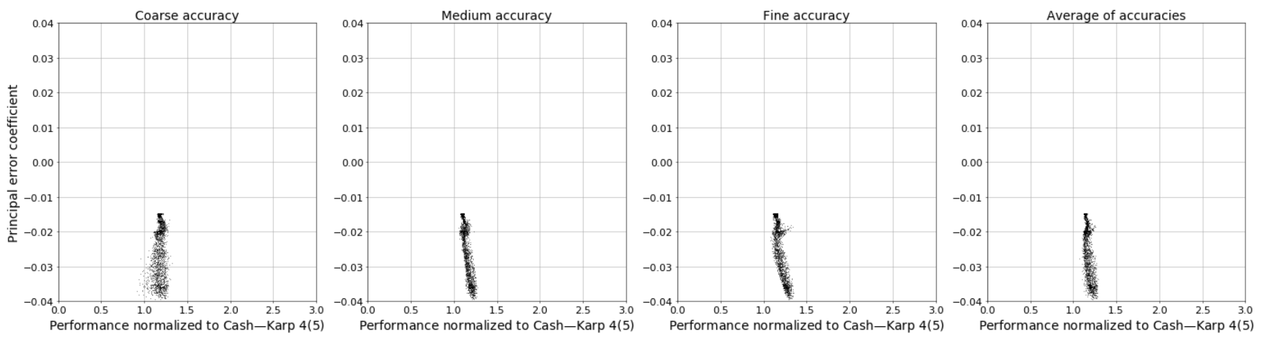


(c) The performance solving the “non-stiff D5” IVP (2.13) with $\epsilon = 0.9$.

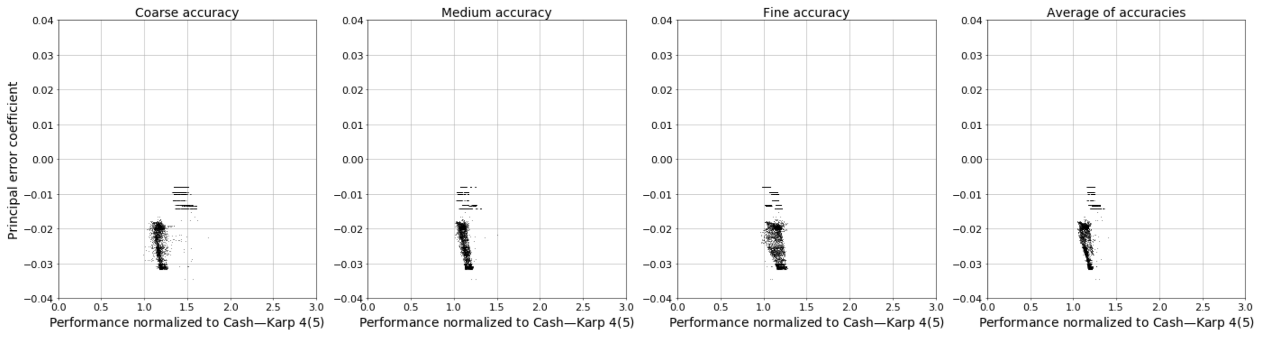


(d) The performance solving the “Arenstorf orbit” IVP (2.15).

Figure 5.26: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ for the family using Case VIb from Section 3.5.1.

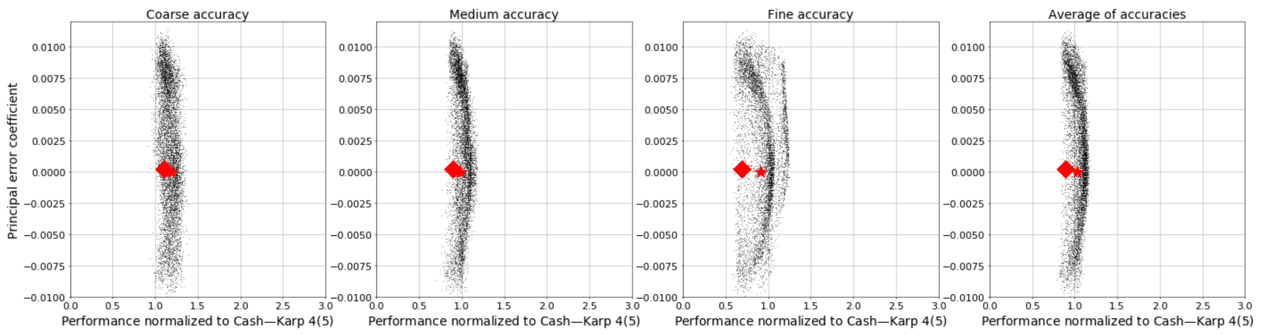


(a) The performance of the family using Case Va solving the "non-stiff D" IVPs (2.13).

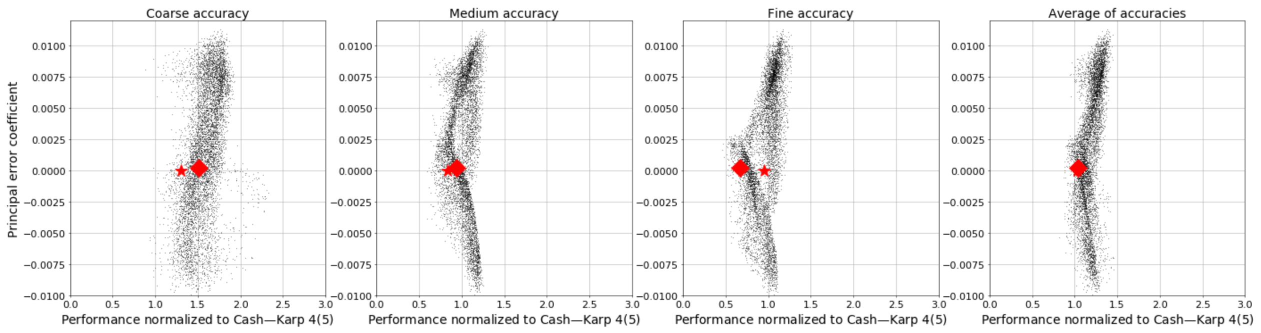


(b) The performance of the family using Case Vb solving the "non-stiff D" IVPs (2.13).

Figure 5.27: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ for the families using Case V from Section 3.5.1.



(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "Arenstorf orbit" IVP (2.15).

Figure 5.28: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau^2[[\tau]]]$ for the $5(4)_{6(7)\text{C}(2)}$ family. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

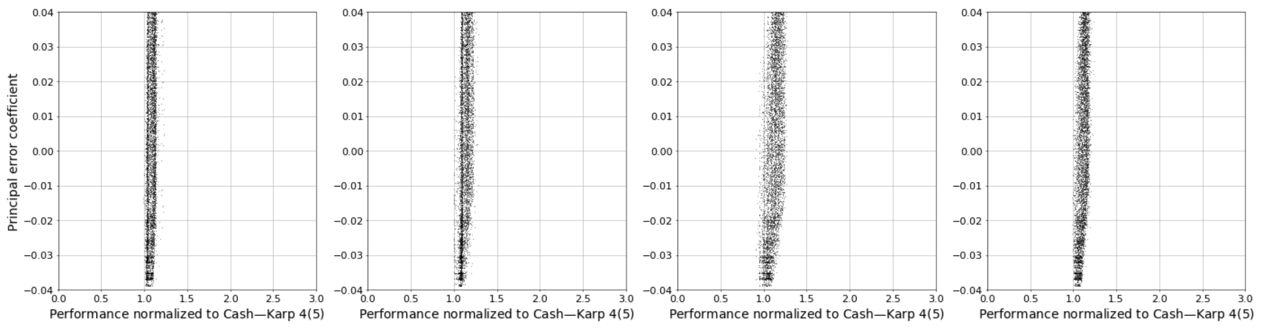
$[\tau[[\tau^2]]]$ only becomes defined with longer integrations that include full orbits. A further hypothesis is that this is possibly due to cancellation of local errors from different parts of the periodic solution.

During the course of this study, other possibilities that were examined (not shown) to try and explain the optimal values of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ included energy conservation (the Hamiltonian), conservation of angular momentum, and linear dispersion or dissipation. Both energy conservation (the Hamiltonian) and angular momentum showed similar behaviour to the variable-stepsize method tested on the "Arenstorf orbit" IVP (2.15) in the paper on `odeToJava` [106], where the error increased to certain magnitude and appeared to remain near that magnitude even during long integrations, although the error may have been growing slowly. This is likely because, in the tests by Sharp et al. [160], they only observed the expected quadratic error growth after 100,000 periods of the Kepler IVP (2.13), something that to examine using the methodologies in this study, must be left to future work. Because there do not seem to be specific values of the PECs corresponding to the tall trees that are more optimal for celestial mechanics IVPs, despite the numerical behaviour solving the scalar test equation (2.10) with appropriate λ sometimes being discussed in the literature with respect to non-linear IVPs that can include celestial mechanics IVPs [4, 82, 160, 179], linear dispersion or dissipation do not seem to be why optimal values occur either. Therefore, the optimal values of the $\overline{\text{PEC}}$s (2.48) observed in this study are likely a non-linear effect. However, now that it is known from this study, determining the ideal values of the $\overline{\text{PEC}}$s (2.48) must now be considered if the best possible ERK pairs are to be constructed for a particular application area, regardless of whether the specific reasons for this behaviour are known.

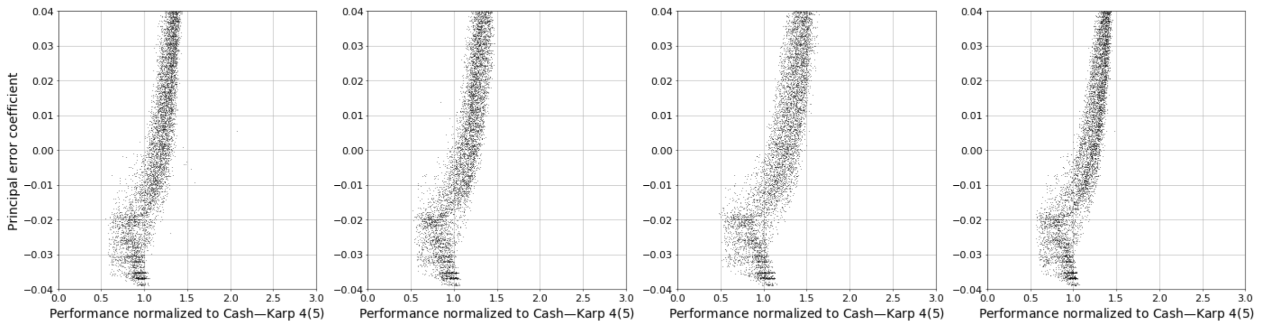
5.11.2 Examining the effect of the $\overline{\text{PEC}}$ corresponding to $[[\tau^4]]$ for the families with the C(2) simplifying assumptions, while restricting the $\overline{\text{PEC}}$ corresponding to $[\tau[[\tau^2]]]$

The $5(4)_{6(6)\text{C}(2)}$ family

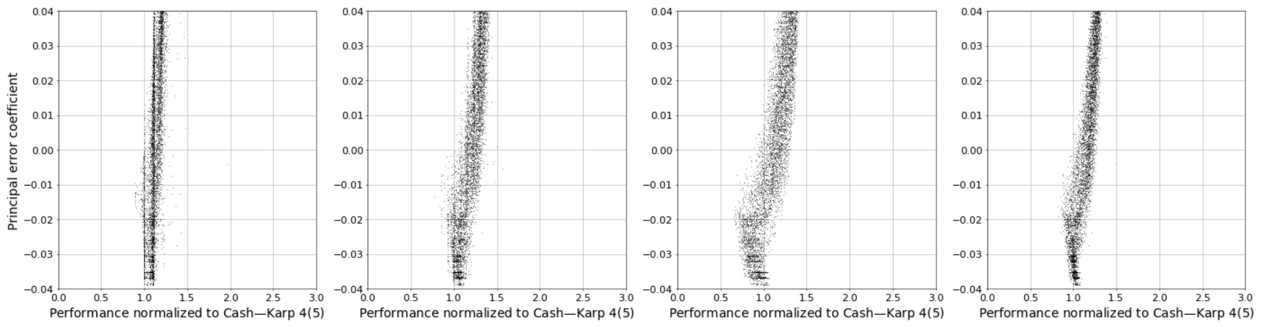
Based on Figure 5.23, the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ is restricted to the relatively narrow range of $[-0.0175, -0.0225]$, which is where the best average performance solving the "non-stiff D" IVPs (2.13) occurs. Experimentation by the author (not shown) indicates that when the best values of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ occupies a fairly broad range, it is sometimes necessary make an arbitrary choice of a narrow interval for examining subsequent PECs. This is because a narrower range often leads to more obvious optimal values for subsequently examined $\overline{\text{PEC}}$s (2.48). Experimentation with the different families and individual IVPs shows that each different but arbitrary and narrow range of values leads to slightly different optimal values of the subsequently examined $\overline{\text{PEC}}$s (2.48). This is probably because the values of the elementary differentials (2.36), order conditions (2.43), and PECs are often dependent on each other. For the $\overline{\text{PEC}}$s (2.48) examined subsequently to the one corresponding to $[\tau[[\tau^2]]]$, it was found to be sufficient to select random pairs uniformly over the entire range of interest, rather than sampling in smaller intervals as



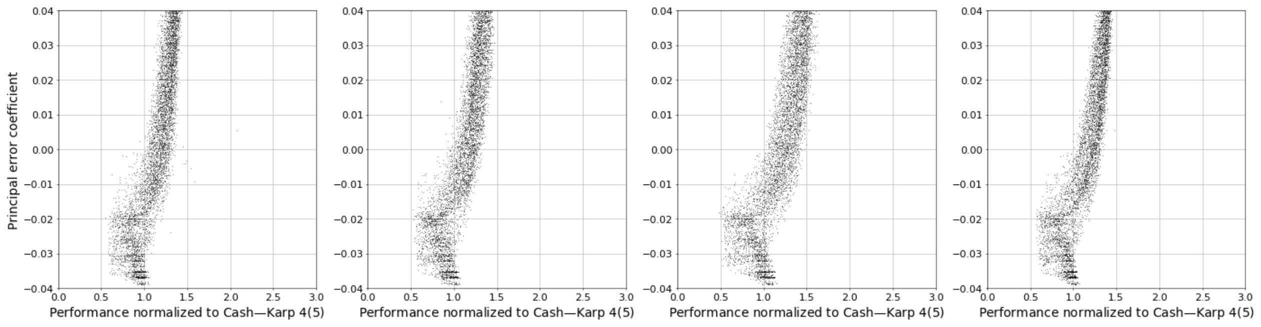
(a) The “Jovian” IVP described in Section 5.3.4 solved for 400 days.



(b) The “Jovian” IVP described in Section 5.3.4 solved for 7200 days.



(c) The “nine planets” IVP described in Section 5.3.4 solved for 20 days.



(d) The “nine planets” IVP described in Section 5.3.4 solved for 400 days.

Figure 5.29: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to the $[\tau[[\tau^2]]]$ in comparison to the performance using the $5(4)_{6(6)\text{C}(2)}$ family to solve different IVPs from celestial mechanics for different amounts of time.

described above for the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$.

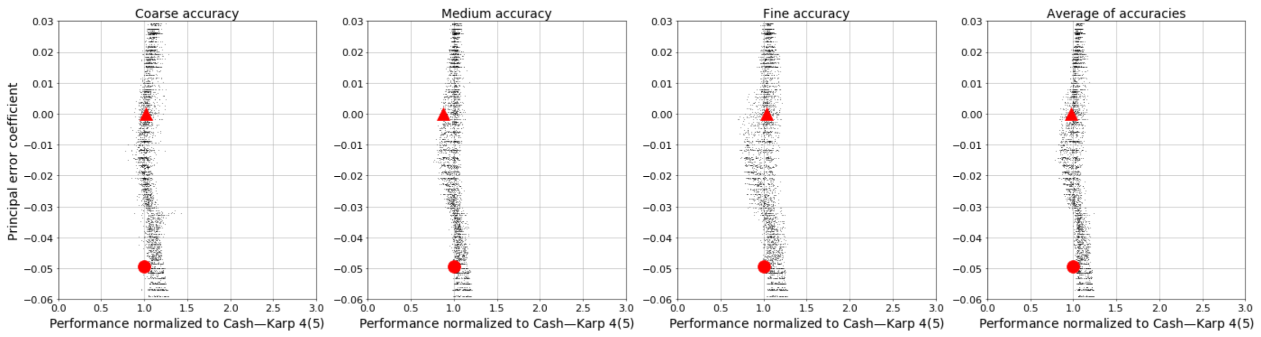
Examination of the performance using the $5(4)_{6(6)\mathrm{C}(2)}$ family to solve the "non-stiff D" IVPs (2.13) with the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ restricted to the interval $[-0.0225, -0.0175]$ shows that the next most important $\overline{\mathrm{PEC}}$ (2.48) is the one corresponding to $[[\tau^4]]$. In Figure 5.30a, the value of the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ for 3200 pairs from the $5(4)_{6(6)\mathrm{C}(2)}$ family, with the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ restricted to the interval $[-0.0225, -0.0175]$, is shown in relation to performance solving the "non-stiff D" IVPs (2.13). It can be seen that when the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ restricted to the interval $[-0.0225, -0.0175]$ that the best performance occurs when the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ has values in the interval $[-0.02, 0.00]$.

The $5(4)_{6(7)\mathrm{C}(2)}$ family

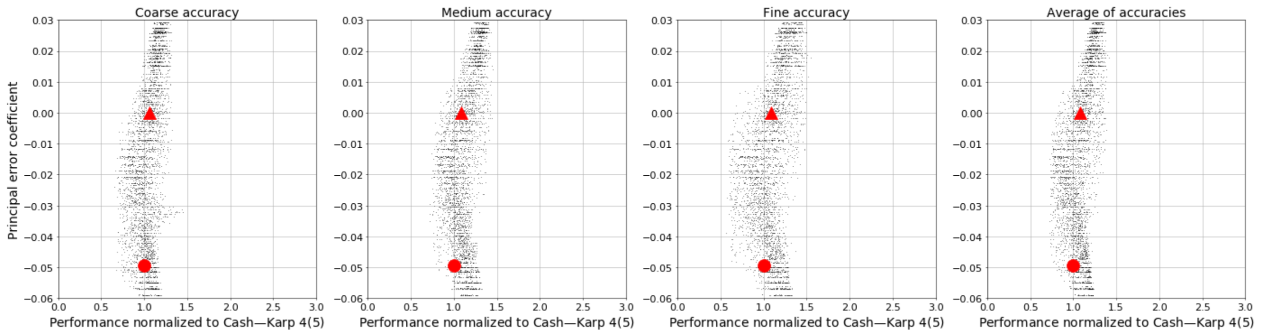
In Figure 5.31, examination of the performance of the $5(4)_{6(7)\mathrm{C}(2)}$ family solving the "non-stiff D" IVPs (2.13), with the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ restricted to $[0.0025, 0.0050]$, shows that the next most dominant $\overline{\mathrm{PEC}}$ (2.48) is also $[[\tau^4]]$. It can be seen that this is mostly because if the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ is greater than zero better performance is had at coarse accuracy.

5.11.3 Examining the effect of the $\overline{\mathrm{PEC}}$ corresponding to $[\tau^5]$ for the $5(4)_{6(6)\mathrm{C}(2)}$ family, while restricting the $\overline{\mathrm{PEC}}$s corresponding to $[\tau[[\tau^2]]]$ and $[[\tau^4]]$

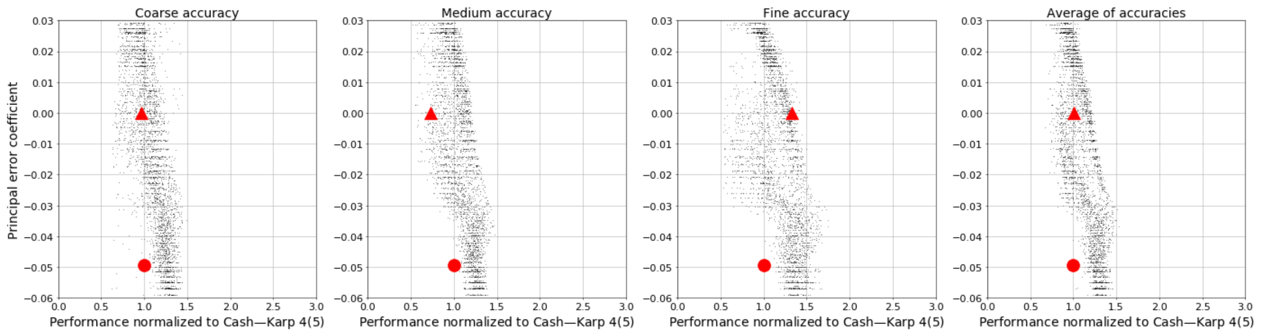
Based on Figure 5.30a, the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ for the $5(4)_{6(6)\mathrm{C}(2)}$ family can be restricted to the range $[-0.015, 0.005]$. For subsequently restricted $\overline{\mathrm{PEC}}$s (2.48) the range chosen can generally be a bit wider than for the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$. As shown in Figure 5.32a, examination shows that after applying the above restriction based on $[\tau[[\tau^2]]]$ and $[[\tau^4]]$ that keeping the $\overline{\mathrm{PEC}}$ (2.48) corresponding to the bushy tree $[\tau^5]$, i.e., the sixth-order quadrature condition, as small as possible gives the best performance solving the "non-stiff D" IVPs (2.13). Additionally, observe that pairs that perform better solving the "Arenstorf orbit" IVP (2.15) occur when the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau^5]$ is at the larger end of its range. This indicates a tradeoff between the most efficient pairs for solving the "Arenstorf orbit" IVP (2.15) IVP and IVPs based more directly on the Kepler IVP (2.13). In fact, with the restrictions on the $\overline{\mathrm{PEC}}$s (2.48) corresponding to $[\tau[[\tau^2]]]$ and $[[\tau^4]]$, the best performance solving the "Arenstorf orbit" IVP (2.15) is in fact when the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau^5]$ is at the larger end of the range seen. What can also be observed from Figure D.3 is that using smaller values of the $\overline{\mathrm{PEC}}$ (2.48) corresponding to the bushy tree $[\tau^5]$ is not at all obvious as a well-defined way of finding the best pairs for the "non-stiff D" IVPs (2.13), unless the $\overline{\mathrm{PEC}}$s (2.48) corresponding to $[\tau[[\tau^2]]]$ and $[[\tau^4]]$ are restricted first.



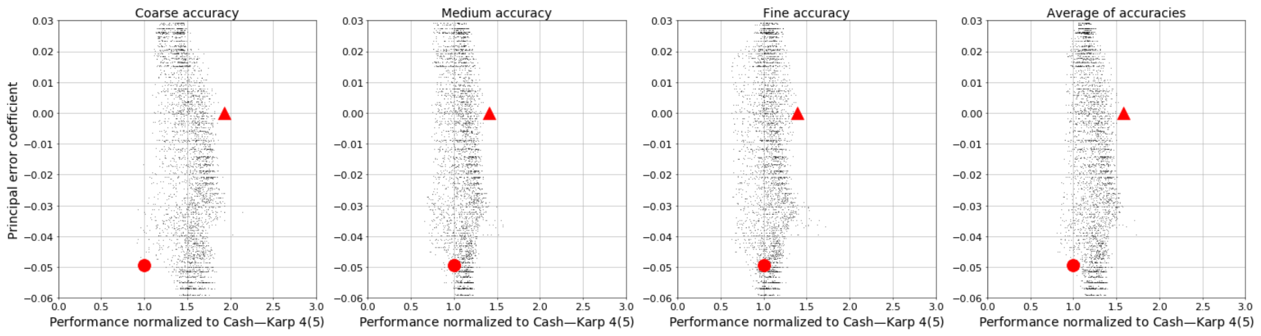
(a) The performance solving the “non-stiff D” IVPs (2.13).



(b) The performance solving the “non-stiff D2” IVP (2.13) with $\epsilon = 0.3$.

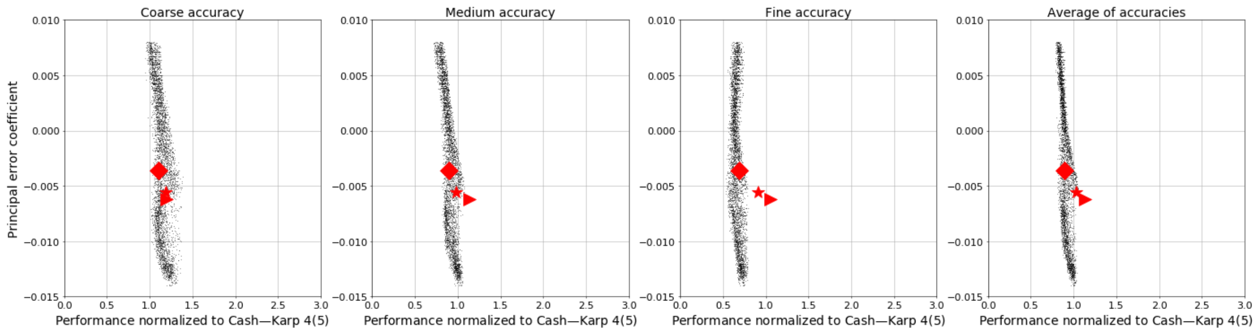


(c) The performance solving the “non-stiff D5” IVP (2.13) with $\epsilon = 0.9$.

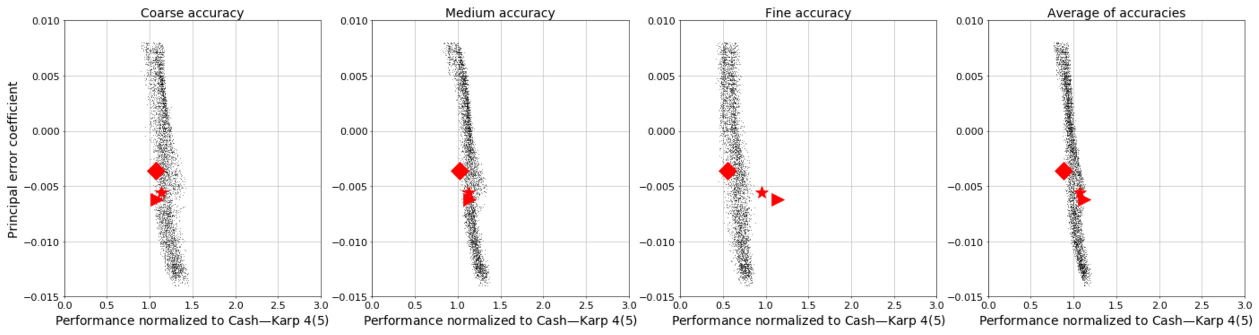


(d) The performance solving the “Arenstorf orbit” IVP (2.15).

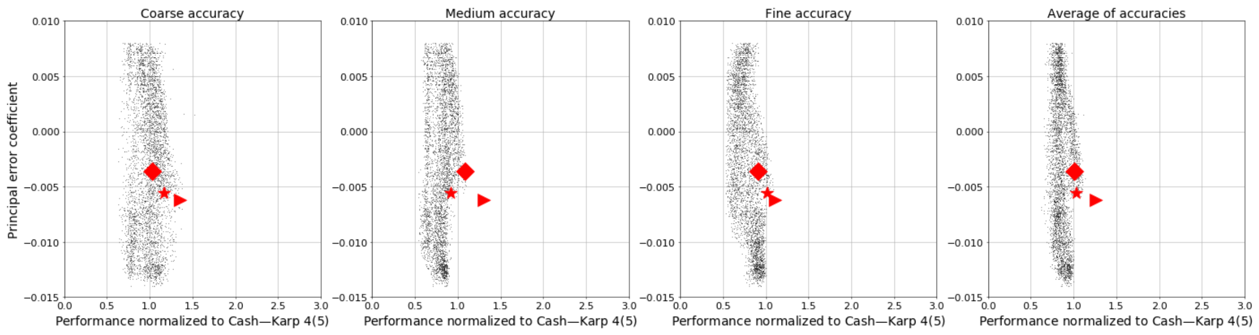
Figure 5.30: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ for $5(4)_{6(6)\text{C}(2)}$ family with $[\tau[[\tau^2]]]$ restricted to $[-0.0175, 0.0225]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



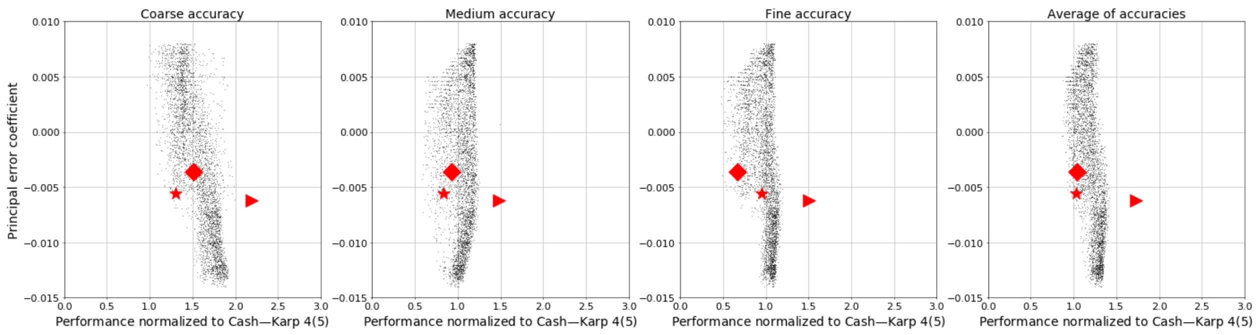
(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.

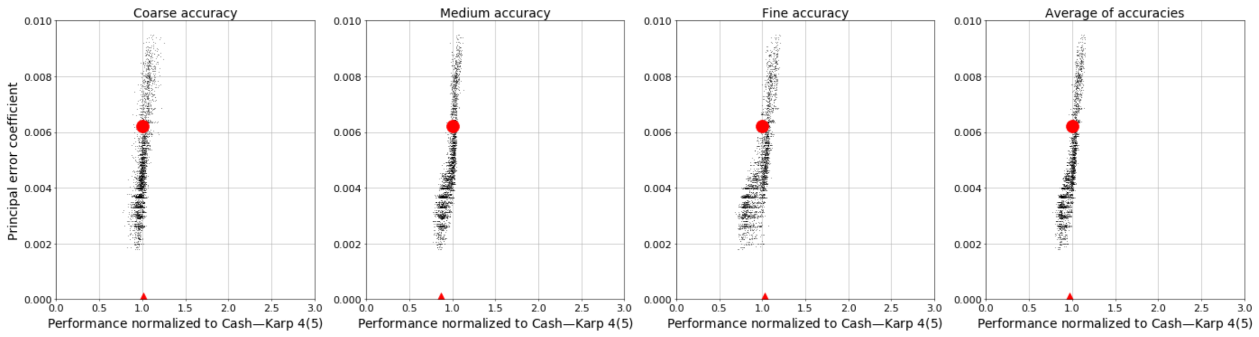


(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.

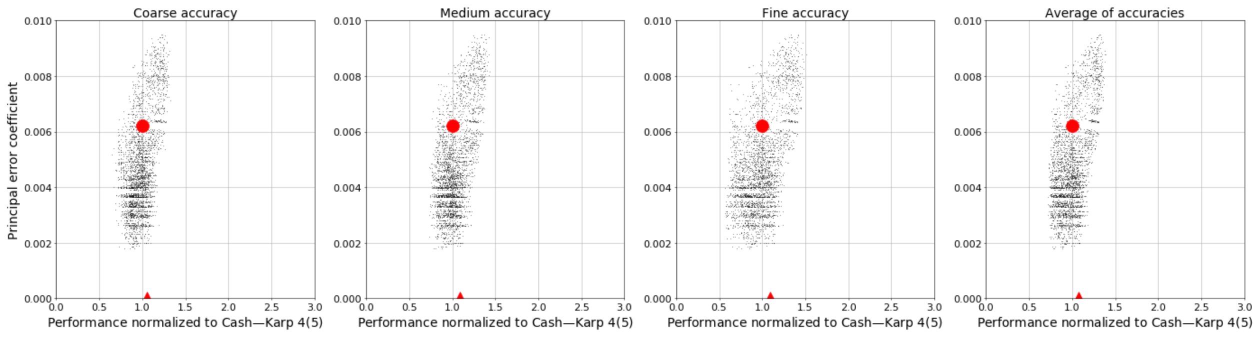


(d) The performance solving the "Arenstorf orbit" IVP (2.15).

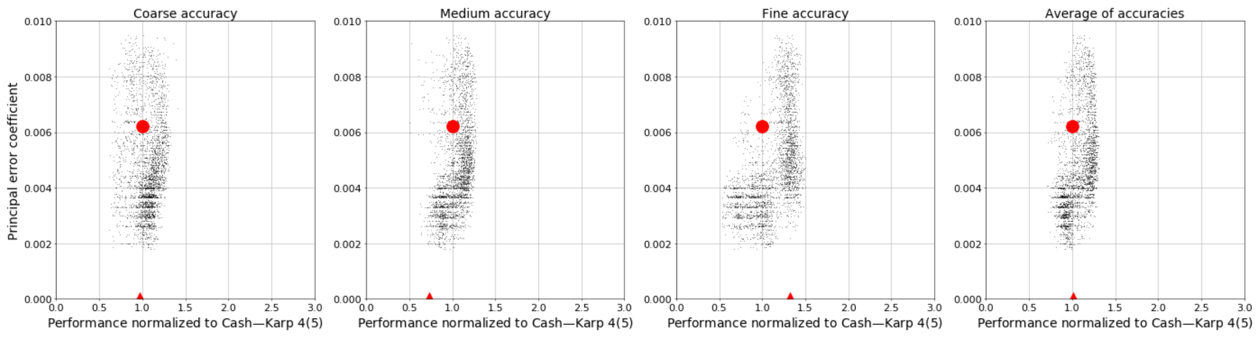
Figure 5.31: The value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ for the $5(4)_{6(7)\text{C}(2)}$ family with $[\tau[[\tau^2]]]$ restricted to $[0.0025, 0.0050]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



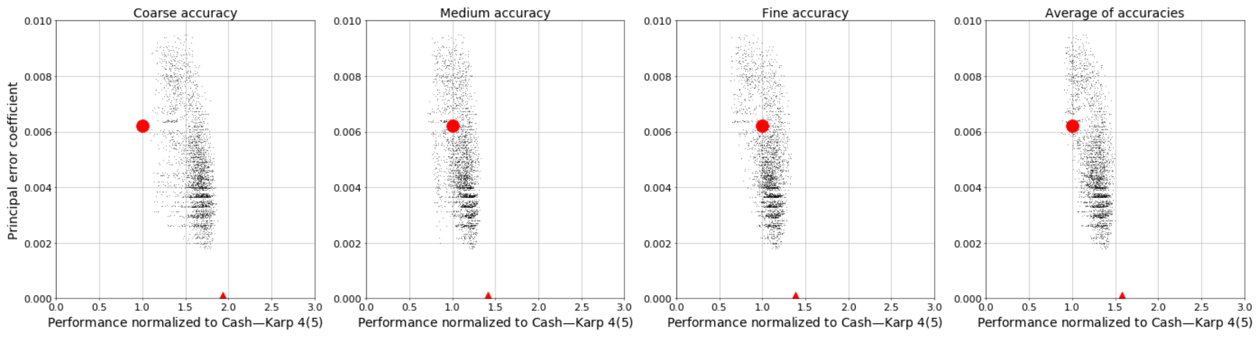
(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.



(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.



(d) The performance solving the "Arenstorf orbit" IVP (2.15).

Figure 5.32: The performance in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau^5]$ for the $5(4)_{6(6)\text{C}(2)}$ family with $[\tau[[\tau^2]]]$ restricted to $[-0.0175, 0.0225]$ and $[[\tau^4]]$ restricted to $[-0.015, 0.005]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

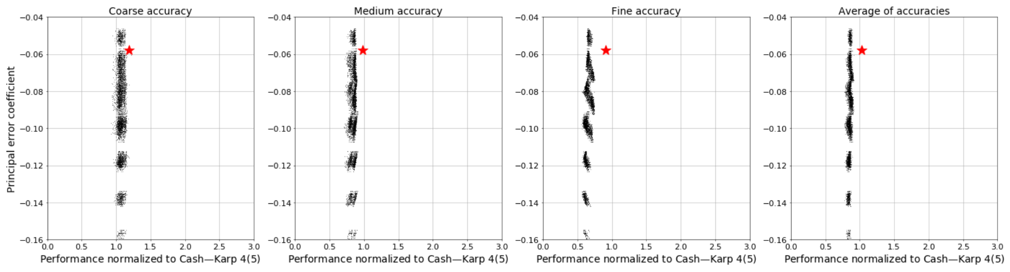
5.11.4 Examining the effect of the seventh-order $\overline{\text{PEC}}$s for the $5(4)_{6(7)\text{C}(2)}$ family, while restricting the $\overline{\text{PEC}}$s corresponding to $[\tau[[\tau^2]]]$ and $[[\tau^4]]$

As seen in Section 5.9, the seventh-order error coefficient assumes greater importance when the leading error coefficient is aggressively minimized while constructing ERK pairs from the $5(4)_{6(7)\text{C}(2)}$ family. In particular, after the $\overline{\text{PEC}}$s (2.48) corresponding to $[\tau[[\tau^2]]]$ and $[[\tau^4]]$ are constrained to the intervals $[0.0025, 0.0050]$ and $[0.0040, 0.0060]$, respectively, the next most obvious determinants of performance are the seventh-order $\overline{\text{PEC}}$s (2.48). There are several seventh-order $\overline{\text{PEC}}$s (2.48) that show the relatively similar behaviour, i.e., those corresponding to $[\tau^6]$, $[[\tau^2][[\tau]]]$, $[[\tau][[\tau^2]]]$, $[\tau[[[\tau^2]]]]$, and $[[\tau[[\tau^2]]]]$. Observe that except for the $\overline{\text{PEC}}$ (2.48) corresponding to $[[\tau^2][[\tau]]]$, all of these rooted trees correspond to adding a new branch to $[\tau[[\tau^2]]]$, which corresponds to the $\overline{\text{PEC}}$ (2.48) shown to have the most dominant effect on performance in Section 5.11.1. For the exception of $[[\tau^2][[\tau]]]$, it can easily be seen that the tree $[[\tau^2][[\tau]]]$ can be constructed from $[\tau[[\tau^2]]]$ by moving a subtree and adding a new branch.

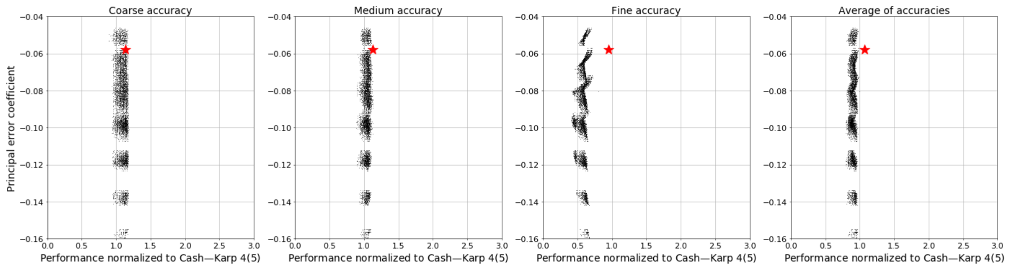
In Figure 5.33, the performance in comparison to the $\overline{\text{PEC}}$ (2.48) corresponding to $[[\tau[[\tau^2]]]]$ is shown because this is the $\overline{\text{PEC}}$ mentioned above that gives the most distinct optimal behaviour for the IVPs under consideration in this section. Additionally, the “non-stiff D6” IVP (2.13) with $\epsilon = 0.99$ is included in Figure 5.33d because this shows that the observed behaviour much more distinct as $\epsilon \to 1$. The behaviour seen in Figure 5.33 clearly indicates that one of the consequences of an aggressively optimized leading error coefficient is a greater sensitivity of the performance on specific IVPs to the specific Butcher tableau coefficients selected, especially when significant stepsize changes are required, such as for the “non-stiff D6” IVP (2.13). Without a clear theoretical understanding of why there are optimal values for certain $\overline{\text{PEC}}$s (2.48), it would be difficult to know whether the performance of pairs chosen on the basis of the more optimal values of the seventh-order $\overline{\text{PEC}}$s (2.48) seen in Figure 5.33 is actually generalizable. This sensitivity to the specific coefficients selected is most noticeable at high accuracies. However, the author's best hypothesis is that it is related to the aggressively minimized leading error coefficient in conjunction with the stability of the step controller and asymptotic behaviour of the solution, rather than issues with floating-point arithmetic. This hypothesis could be readily checked by re-doing the calculations in higher-precision arithmetic. Unfortunately, it is not currently feasible to test this hypothesis with `pythODE` because neither Python nor SciPy currently have the required support for quadruple-precision (or higher) floating-point arithmetic.

5.11.5 Examining restrictions on the $\overline{\text{PEC}}$s for Case VI from Section 3.5

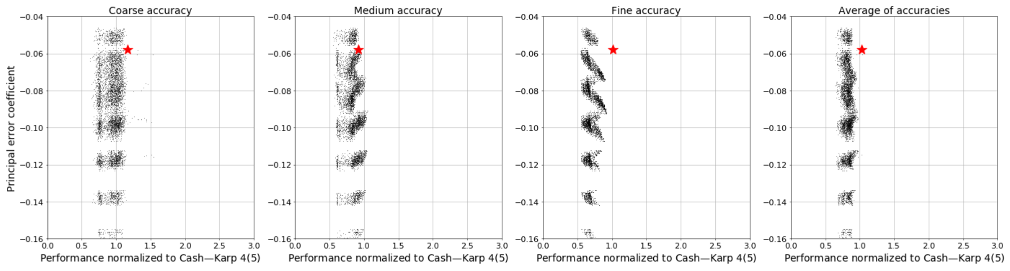
The more complex space of free parameters of the family based on Case VI from Section 3.5 makes the process of finding the properties of the most efficient pairs more involved. Although pairs from Case VI that can be recommended for use are found by this study, hopefully once there is a better theoretical understanding of why there are specific values of $\overline{\text{PEC}}$s (2.48) that are more optimal, further improvements and a narrower range of best performances may be found.



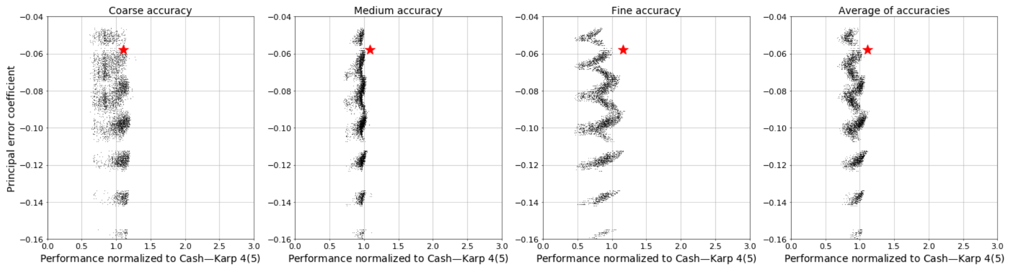
(a) The performance solving the "non-stiff D" IVPs (2.13).



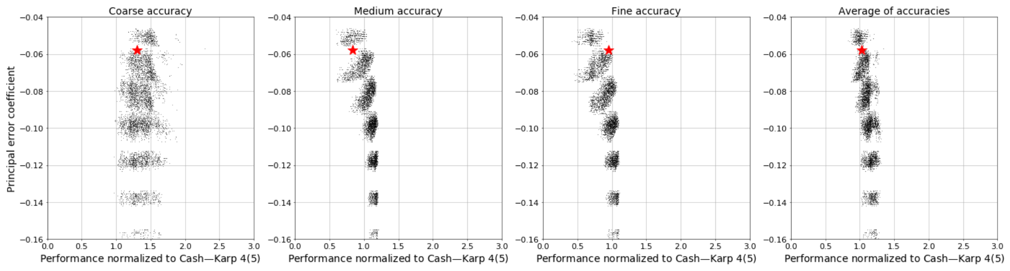
(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.



(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.



(d) The performance solving the "non-stiff D6" IVP (2.13) with $\epsilon = 0.99$.



(e) The performance solving the "Arenstorf orbit" IVP (2.15).

Figure 5.33: The performance of the $5(4)_{6(7)\mathrm{C}(2)}$ family in comparison to $\overline{\text{PEC}}$s corresponding to $\overline{\text{PEC}}$s (2.48) corresponding to $[[\tau[[\tau^2]]]]$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

As mentioned in Chapter 4, each subcase of Case VI is searched with both $\lambda = 1, \mu \in [-1, 1]$ and $\mu = 1, \lambda \in [-1, 1]$. Each of these subcases searched, i.e., VIa and VIb, requires a different strategy for selecting the most efficient sets of free parameters. Because much of the discussion of selecting the best pairs from Case VI is similar to the $5(4)_{6(6)\mathrm{C}(2)}$ and $5(4)_{6(7)\mathrm{C}(2)}$ families, only the overall performance solving the "non-stiff D" IVPs (2.13) is shown. Case VI is particularly interesting among the families of $5(4)_{6(6)}$ ERK pairs constructed in Chapter 3 because there is no equivalence (equality) among the $\overline{\text{PEC}}$s (2.48) of sixth-order. For example, for $5(4)_6$ ERK pairs with the C(2) simplifying assumptions (2.44b) the equivalence $[\tau^5] \equiv [\tau^3[\tau]] \equiv [\tau[\tau]^2]$ always holds. In constrast, there are no such equivalent sixth-order $\overline{\text{PEC}}$s (2.48) for $5(4)_6$ ERK pairs from Case VI. As an aside, Case V does has multiple equivalences because $c_6 = 1$ means it always requires the D(1) simplifying assumptions (2.44c). Unfortunately, to fully exploit this property a theoretical understanding of why there are specific optimal values of $\overline{\text{PEC}}$s (2.48) is required. If future work finds that equivalence (equality) among the $\overline{\text{PEC}}$s (2.48) can cause issues with performance or other numerical behaviour, for example, by introducing problematic corner cases, then there may be a specific advantage to finding ERK pairs without the C(2) simplifying assumptions (2.44b).

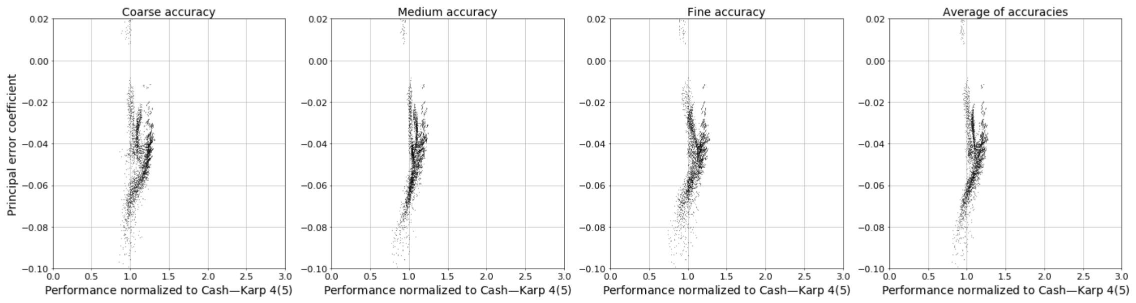
Although the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ is the most dominant indicator of performance for IVPs from celestial mechanics, as already seen in Section 5.11.1, the two subcases of Case VI and different areas of each space of free parameters have different subsequent $\overline{\text{PEC}}$s (2.48) that are important.

Case VIa with $\lambda = 1, \mu \in [-1, 1]$ did not achieve the performance required to be competitive with other families and is not mentioned further. Case VIa with $\lambda = 1, \mu \in [-1, 1]$ has the second most dominant $\overline{\text{PEC}}$ (2.48) as the one corresponding to $[[[\tau]^2]]$ that is shown in Figure 5.34a. The next most dominant $\overline{\text{PEC}}$ (2.48) is the seventh-order one corresponding to $[[\tau^5]]$ that is shown in Figure 5.34b. In Figure 5.34, it is obvious that the selection of the best pairs for celestial mechanics from Case VI can be much more sensitive to the particular coefficients than the $5(4)_{6(6)\mathrm{C}(2)}$ family.

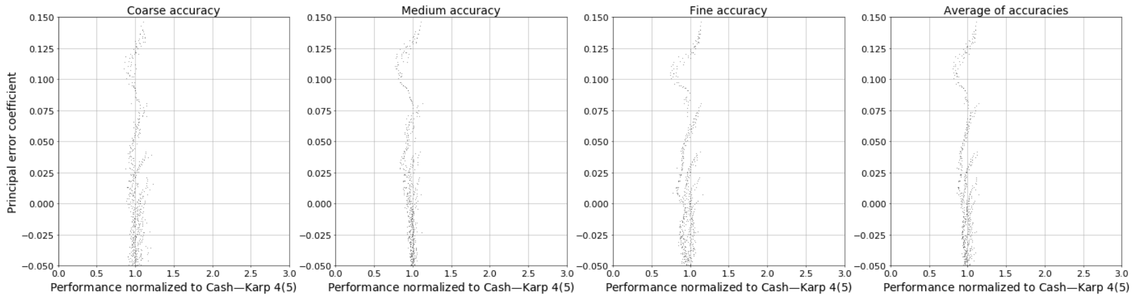
Case VIb with $\lambda = 1, \mu \in [-1, 1]$ has the second most dominant $\overline{\text{PEC}}$ (2.48) as the one corresponding to $[\tau^3[\tau]]$ that is shown in Figure 5.35a. The next most dominant $\overline{\text{PEC}}$ (2.48) is the seventh-order one corresponding to $[\tau[[\tau[\tau]]]]$ that is shown in Figure 5.35b. Case VIb with $\mu = 1, \lambda \in [-1, 1]$ has the second most dominant $\overline{\text{PEC}}$ (2.48) as the one corresponding to $[\tau^2[\tau^2]]$ that is shown in Figure 5.36a. Then the next most dominant $\overline{\text{PEC}}$ (2.48) is $[\tau^5]$ that is shown in Figure 5.36b.

5.11.6 An appraisal of selecting coefficients using the values of the $\overline{\text{PEC}}$s

After the restrictions given in Section 5.11.3 for the $5(4)_{6(6)\mathrm{C}(2)}$ family, a further restriction for the $5(4)_{6(6)\mathrm{C}(2)}$ family can be given on the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$. This is shown specifically in Figure 5.37, where the best performance can clearly be seen to occur if the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ is further restricted to < -0.020 while carefully avoiding values that are too close to -0.020. Detailed investigation of the performance data (not shown) shows that all of the individual IVPs from the "non-stiff D" IVPs (2.13) contribute to this need for a further restriction. The behaviour when the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$

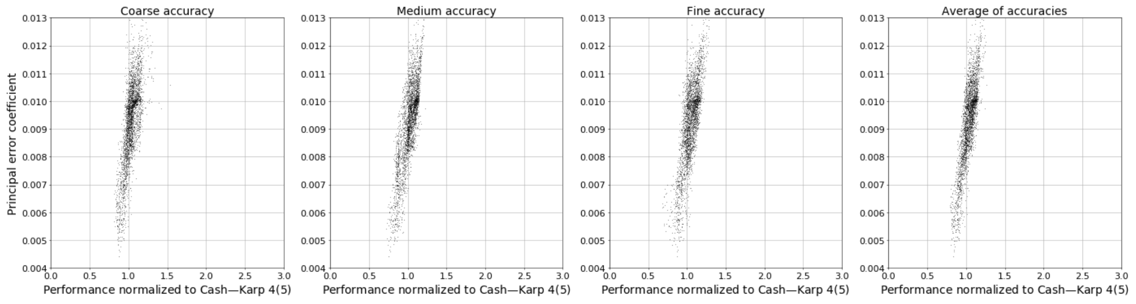


(a) The performance in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[[[\tau]^2]]$ with $[\tau[[\tau^2]]]$ restricted to $[-0.0125, -0.0075]$.

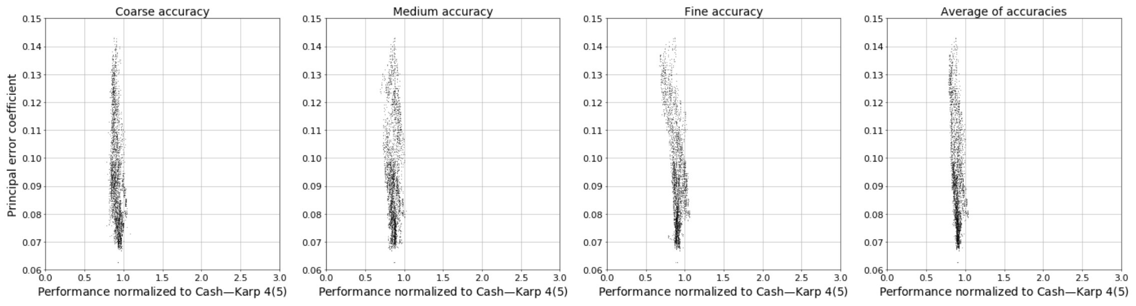


(b) The performance in comparison to value of the of the $\overline{\text{PEC}}$ (2.48) corresponding to $[[\tau^5]]$ $[[[\tau]^2]] \leq -0.06$ and $[\tau[[\tau^2]]]$ restricted to $[-0.0125, -0.0075]$.

Figure 5.34: The average performance of the "non-stiff D" IVPs (2.13) using Case VIa with $\lambda = 1, \mu \in [-1, 1]$.

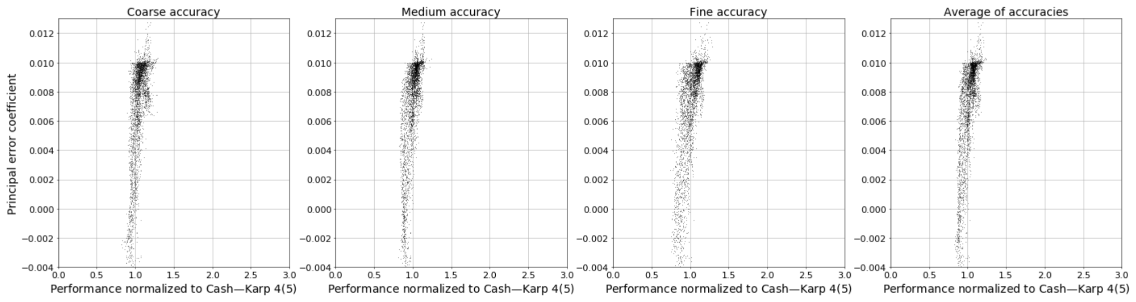


(a) The performance in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau^3[\tau]]$ with $[\tau[[\tau^2]]]$ restricted to $[-0.0125, -0.0075]$.

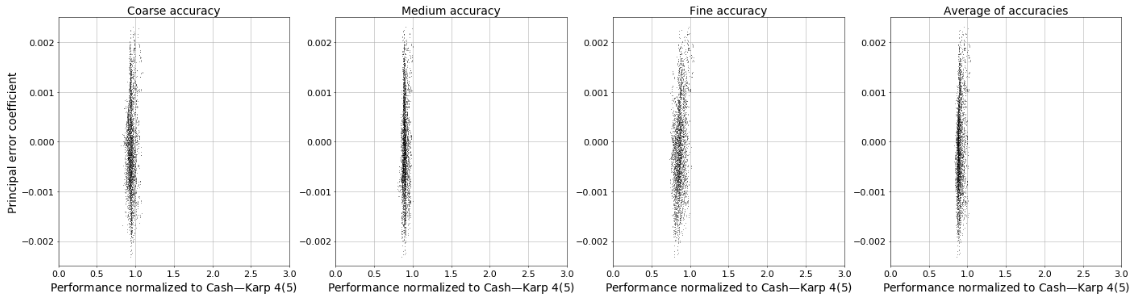


(b) The performance in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau[\tau]]]]$ with $[\tau^3[\tau]] < 0.0070$ and $[\tau[[\tau^2]]]$ restricted to $[-0.0125, -0.0075]$.

Figure 5.35: The average performance of the "non-stiff D" IVPs (2.13) using Case VIb with $\lambda = 1, \mu \in [-1, 1]$.



(a) The performance in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau^2[\tau^2]]$ with $[\tau[[\tau^2]]]$ restricted to $[-0.020, -0.015]$.



(b) The performance in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau^5]$ with $[\tau^2[\tau^2]] \leq 0.000$ and $[\tau[[\tau^2]]]$ restricted to $[-0.020, -0.015]$.

Figure 5.36: The average performance of the "non-stiff D" IVPs (2.13) using Case VIb with $\mu = 1, \lambda \in [-1, 1]$.

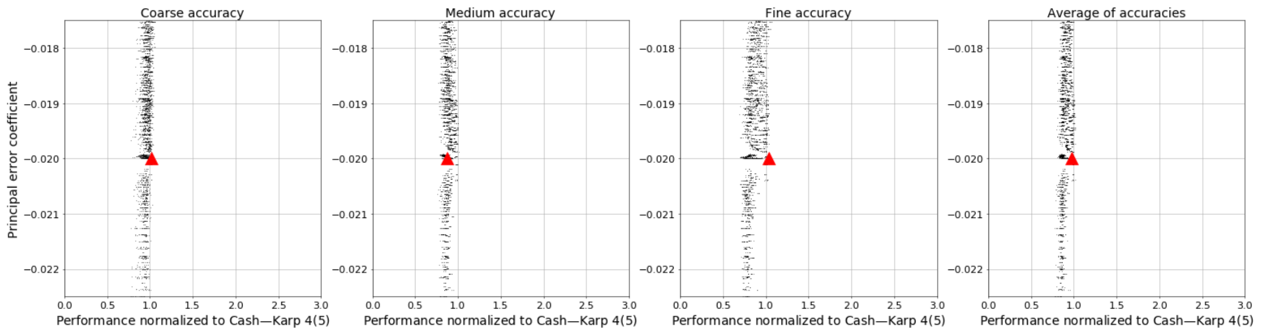


Figure 5.37: The average performance solving the "non-stiff D" IVPs (2.13) in comparison to the value of the $\overline{\text{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$ with the $\overline{\text{PEC}}$ (2.48) corresponding to $[[\tau^4]]$ restricted to $[-0.015, 0.0005]$ and $[\tau^5] \leq 0.0035$. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

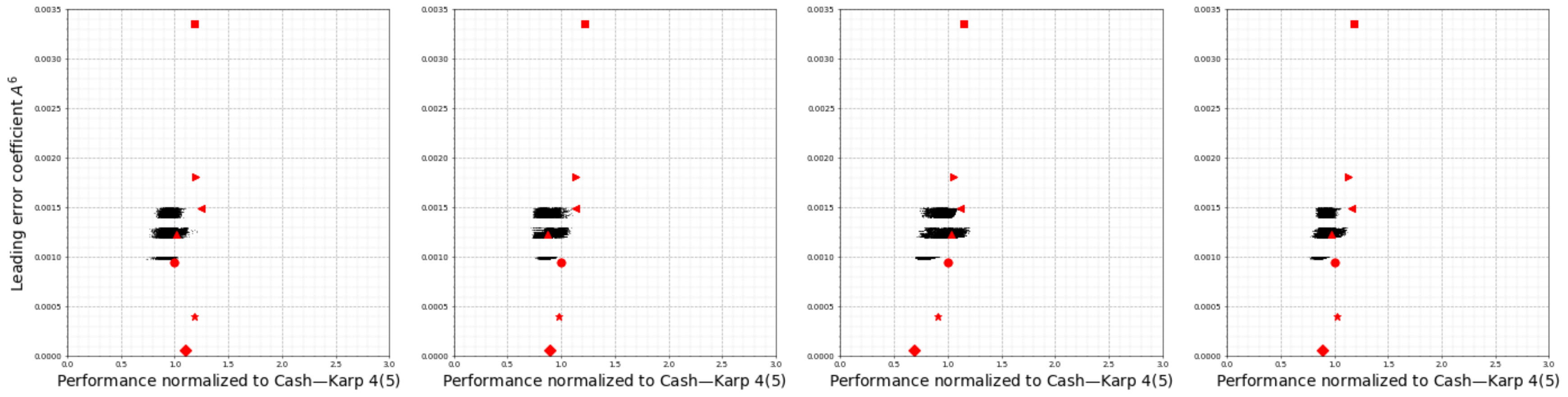
is extremely close or exactly -0.020 may give clues for future work that will help assist in determining exactly why there are specific values of $\overline{\text{PEC}}$ (2.48) that consistently give better performance.

The overall effect of restricting the values of $\overline{\text{PEC}}$s (2.48) on the $5(4)_{6(6)\text{C}(2)}$ and $5(4)_{6(7)\text{C}(2)}$ is shown in Figures 5.38 and 5.39, respectively. The overall effect of restricting the values of $\overline{\text{PEC}}$s (2.48) on the families of VIa from Section 3.5.1 with $\lambda = 1, \mu \in [-1, 1]$ and VIb with $\mu = 1, \lambda \in [-1, 1]$ from Section 3.5.1 is shown in Figures 5.40 and 5.41, respectively.

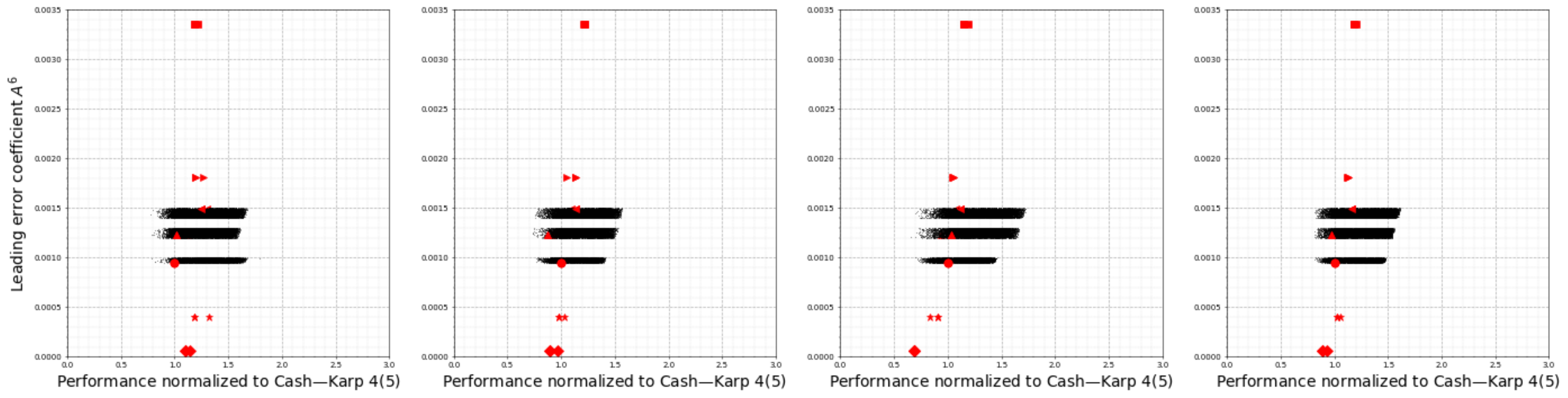
Examining performance of $5(4)_6$ ERK pairs solving the "non-stiff D" IVPs (2.13) and restricting the values of the $\overline{\text{PEC}}$ (2.48) to certain ranges that have been empirically determined is an excellent methodology for eliminating the worst performing pairs from consideration, as well as ensuring the best performing pairs are included. Figure 5.38 shows the effect of restrictions discussed of the $\overline{\text{PEC}}$s (2.48) on the $5(4)_{6(6)\text{C}(2)}$ family is bringing the performance variation between individual pairs, most of which is not explained by other properties that have been examined in the literature, from nearly a factor of two down to as low as 15%. The author considers 15% to be an amount that, barring results to the contrary in future studies, is probably due to the expected variation inherent between individual ERK pairs. However, these restrictions on the $\overline{\text{PEC}}$s (2.48) are subject to at least some judgment and arbitrary choices. Therefore, different restrictions on the $\overline{\text{PEC}}$s (2.48) than presented in this study are often equally useful. As expected based on the correlation in performance between these simpler IVPs and the more complex IVPs from celestial mechanics in Section 5.10, it is shown below that pairs constructed in a systematic manner do give at least equivalent, and often surprisingly better performance, on some more complex IVPs from celestial mechanics in comparison to published $5(4)_6$ ERK pairs. Finally, even though there is eventually a clear tradeoff in performance for the Kepler IVP (2.13) in comparison to the "Arenstorf orbit" IVP (2.15), pairs constructed to perform best solving the "non-stiff D" IVPs (2.13) perform consistently better solving the "Arenstorf orbit" IVP (2.15) than randomly selected pairs with otherwise similar properties.

5.12 Selecting the lower-order component of an ERK embedded pair

With the narrowed regions of the space of free parameters by selecting appropriate range of the $\overline{\text{PEC}}$s (2.48) discussed in Section 5.11 shown in Figure 5.38a and 5.39a, it is now easier to examine the selection of the lower-order component of $5(4)_6$ ERK pairs, i.e., using either $\hat{b}_6$ or $\hat{b}_7$ depending on the family. The exception is Case VI from Section 3.5 with $\hat{b}_6 = 0$, where the lower-order component of a pair for this family must be selected in conjunction with choosing the higher-order component of a pair. In Figure 5.42, the effect of the choice of $\hat{b}_6$ is shown for the $5(4)_{6(6)\text{C}(2)}$ family when solving selected "non-stiff D" IVPs (2.13). In Figure 5.43, the effect of the choice of $\hat{b}_7$ is shown when solving the $5(4)_{6(7)\text{C}(2)}$ family for a selection of "non-stiff D" IVPs (2.13).

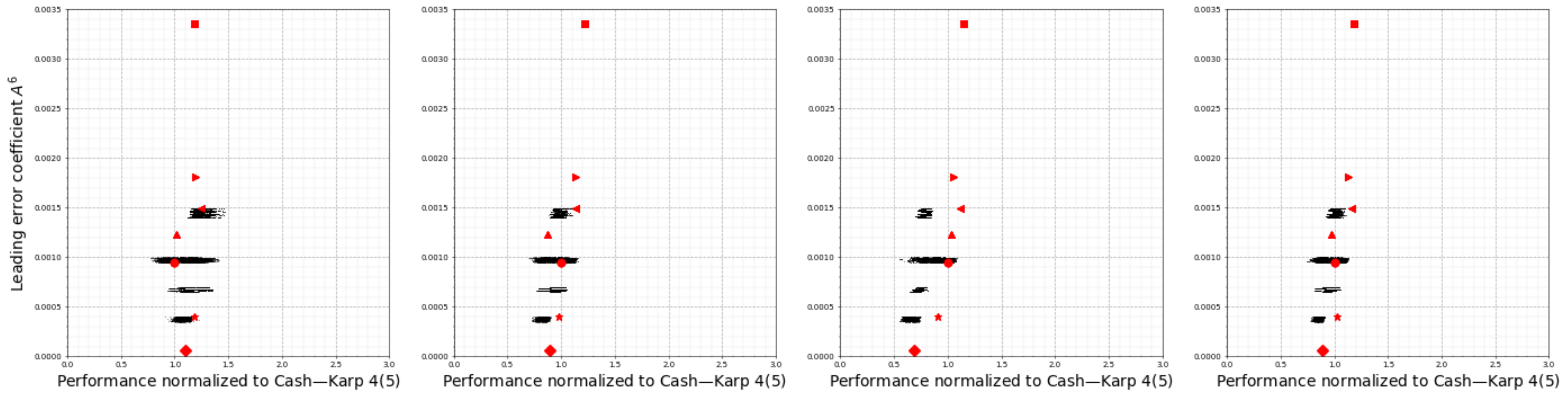


(a) Restricted $\overline{\text{PEC}}$s (2.48). The procedure for each interval of A^6 is slightly different, but follows closely the procedure given in this chapter.

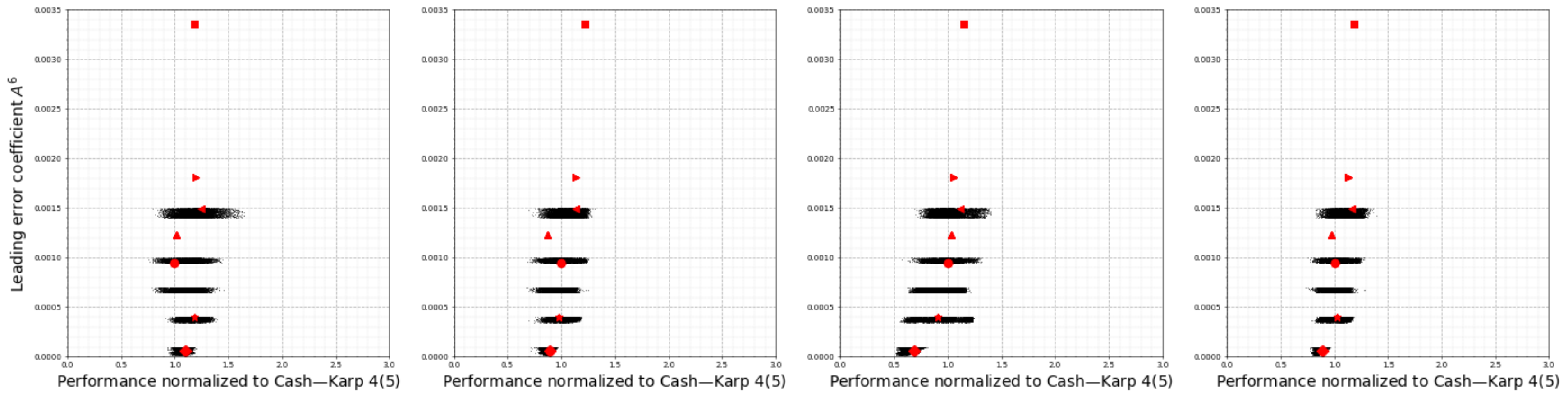


(b) Unrestricted $\overline{\text{PEC}}$s (2.48).

Figure 5.38: Comparing the average performance solving the "non-stiff D" IVPs (2.13) for pairs from the $5(4)_{6(6)\mathrm{C}(2)}$ family that have the restrictions on the $\overline{\text{PEC}}$s (2.48) discussed in Section 5.11 to pairs chosen without restrictions on the $\overline{\text{PEC}}$s (2.48) with pairs. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

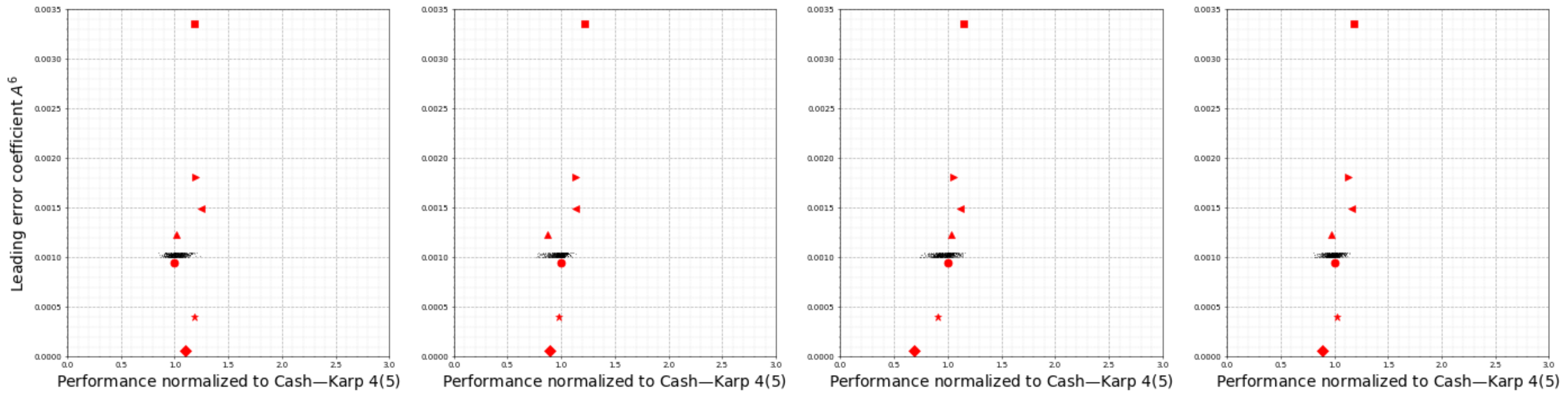


(a) Restricted $\overline{\text{PECs}}$ (2.48). The procedure for each interval of A^6 is slightly different, but follows closely the procedure given in this chapter.

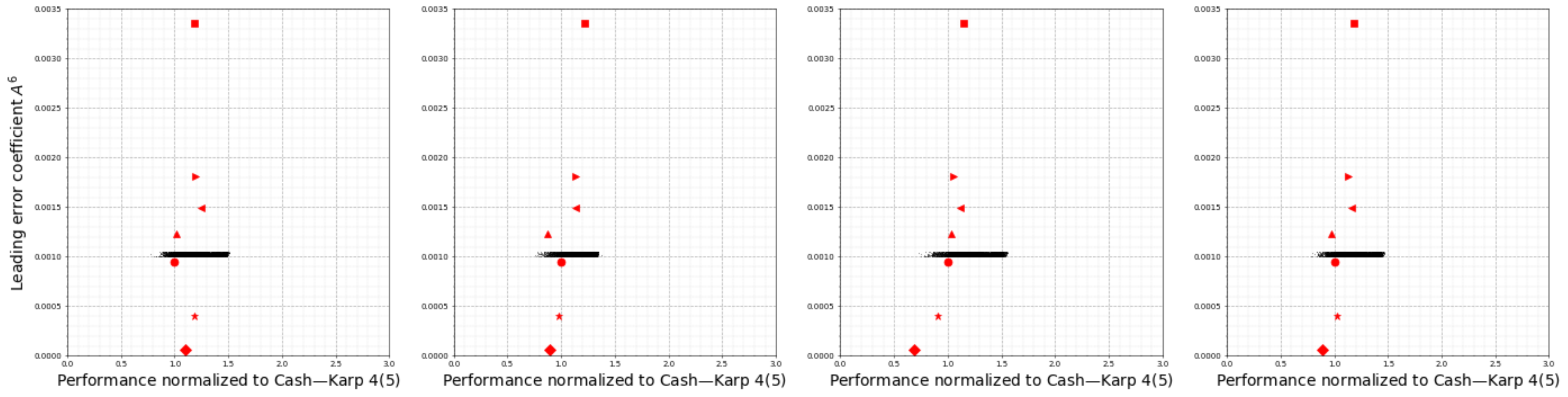


(b) Unrestricted $\overline{\text{PECs}}$ (2.48).

Figure 5.39: Comparing the average performance solving the "non-stiff D" IVPs (2.13) for pairs from the $5(4)_{6(7)\mathrm{C}(2)}$ family that have the restrictions on the $\overline{\text{PECs}}$ (2.48) discussed in Section 5.11 to pairs chosen without restrictions on the $\overline{\text{PECs}}$ (2.48) with pairs. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

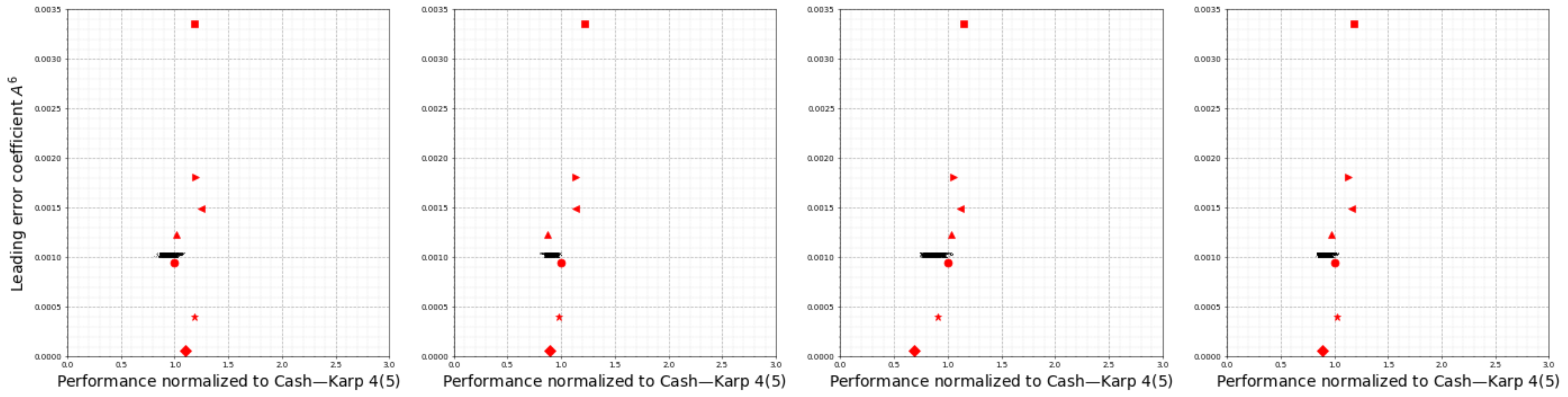


(a) Restricted $\overline{\text{PEC}}$s (2.48). The procedure for each interval of A^6 is slightly different, but follows closely the procedure given in this chapter.

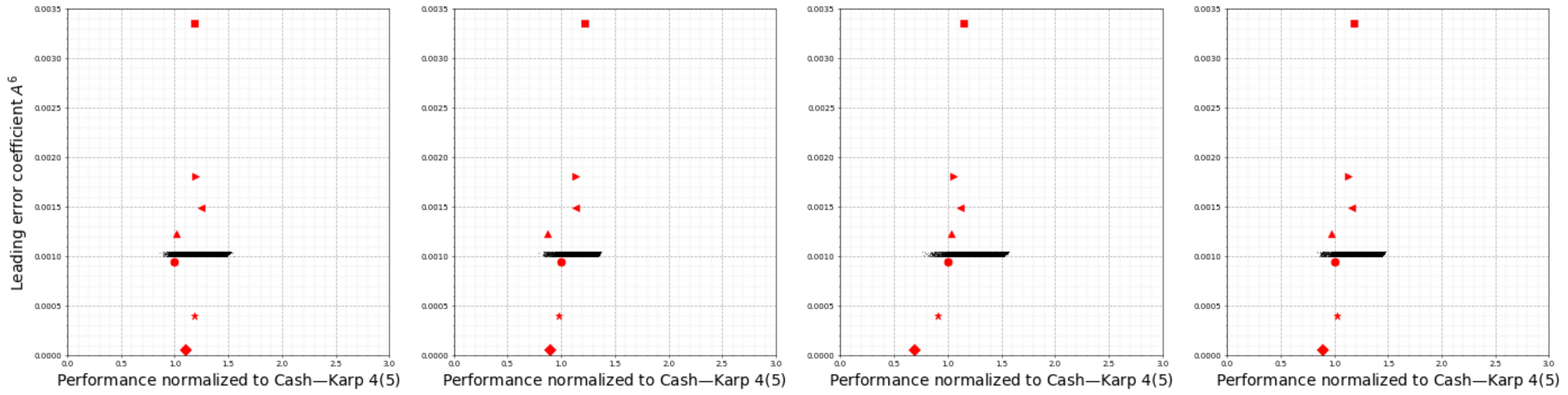


(b) Unrestricted $\overline{\text{PEC}}$s (2.48).

Figure 5.40: Comparing the average performance solving the "non-stiff D" IVPs for pairs from the family of VIa from Section 3.5.1 with $\lambda = 1, \mu \in [-1, 1]$ that have the restrictions on the $\overline{\text{PEC}}$s (2.48) discussed in Section 5.11 to pairs chosen without restrictions on the $\overline{\text{PEC}}$s (2.48) with pairs. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



(a) Restricted $\overline{\text{PEC}}$s (2.48). The procedure for each interval of A^6 is slightly different, but follows closely the procedure given in this chapter.



(b) Unrestricted $\overline{\text{PEC}}$s (2.48).

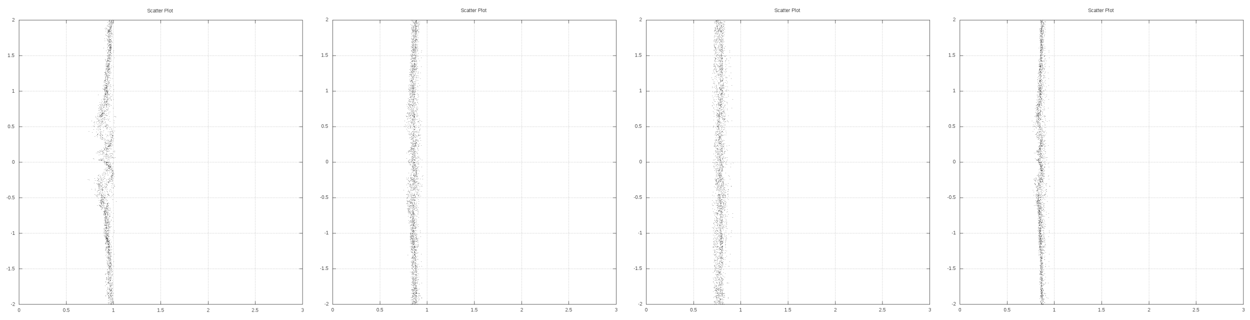
Figure 5.41: Comparing the average performance solving the "non-stiff D" IVPs for pairs from the family of VIb with $\mu = 1, \lambda \in [-1, 1]$ from Section 3.5.1 that have the restrictions on the $\overline{\text{PEC}}$s (2.48) discussed in Section 5.11 to pairs chosen without restrictions on the $\overline{\text{PEC}}$s (2.48) with pairs. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

It is clearly be seen from Figures 5.42 and 5.43 that there is at best little advantage to any choice of $\hat{b}_6$ and $\hat{b}_7$ in terms of the average performance solving the "non-stiff D" IVPs (2.13). This is because in selecting $\hat{b}_6$ and $\hat{b}_7$, for any value selected there are unavoidable tradeoffs between the coarse, medium, and fine accuracies as well as between individual IVPs. Better methodologies for selecting the lower-order component of an ERK pair are probably where some of the largest future improvements can be made to the coefficient selection process beyond what is already shown by this study. However, these improvements would likely have to be made in conjunction with detailed study of the error estimation and improved step-control schemes. For example, the control theory briefly discussed in Section 2.6.2 would have to be applied in a more in-depth way than has previously been done for IVP methods. Although advanced PID control is often an option in IVP solvers, a data-intensive empirical assessment of behaviour and tradeoffs between I, PI, and PID step control using `pythODE` would be of great benefit to understanding Figures 5.42 and 5.43. An important benefit of knowing the effect of individual PECs on performance is that if their effect is not accounted for, the effects examined in further studies into aspects of ERK method construction such as error estimation and step control may be hard to detect. In fact, it is extremely difficult to see even the behaviour seen in Figures 5.42 and 5.43, or any well-defined behaviour at all, if the $\overline{\text{PEC}}$s (2.48) are not restricted.

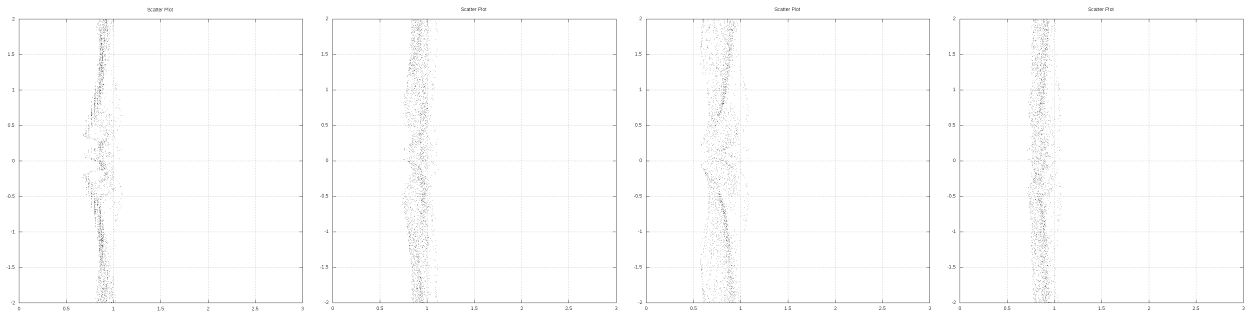
5.13 Revisiting the classic characteristic numbers

With the data provided by `pythODE` and `OCSage`, it is possible to evaluate the effectiveness of the classic characteristic numbers (2.56) in the coefficient selection process. The most complete and most recent expositions of the classic characteristic numbers (2.56) for selecting coefficients for RK methods are those by Kennedy and Carpenter from 1999 [96] and 2001 [94]. Although the latter addresses IMEX methods (already mentioned briefly in Section 4.4.1), both publications include much of the theory discussed in Chapter 2 that is directly applicable to just ERK pairs. Previous expositions of the classic characteristic numbers (2.56) are those by Verner from 1990 [189] and 1991 [190], which also addresses different norms of the PECs. The classic characteristic numbers (2.56) were originally introduced by Dormand and Prince in 1981 for the construction of eighth- and seventh-order embedded ERK pairs [136], but they also mention using them in the construction of the DP5(4)$_{6(7)}$ pair (2.79) the year prior [47]. Although the ideal values of the classic characteristic numbers (2.56) (close to unity) have some intuitive reasoning behind making the error estimate resemble the actual local error in some sense, this reasoning is only applicable in asymptotic limit. However, there are generally many assumptions and simplifications between the mathematical description of numerical methods and actual real-world behaviour [78].

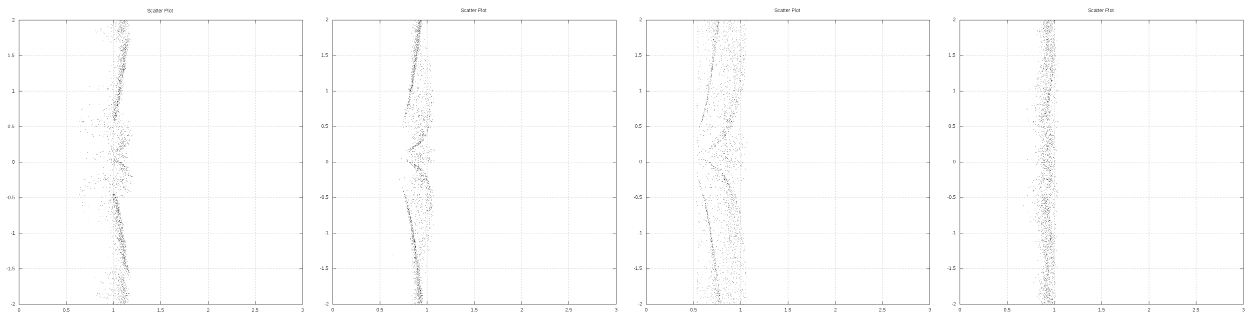
The guidelines published in the literature for coefficient selection [16, 47, 94, 136, 189, 190] mentioning the classic characteristic numbers (2.56) are generally qualitative and there is actually little published data or rigorous analysis to support any specific quantitative guidelines. The standard recommendations are that the B and C characteristic numbers be "of order unity" [94, 96, 190], "small" [136], or that the B and C



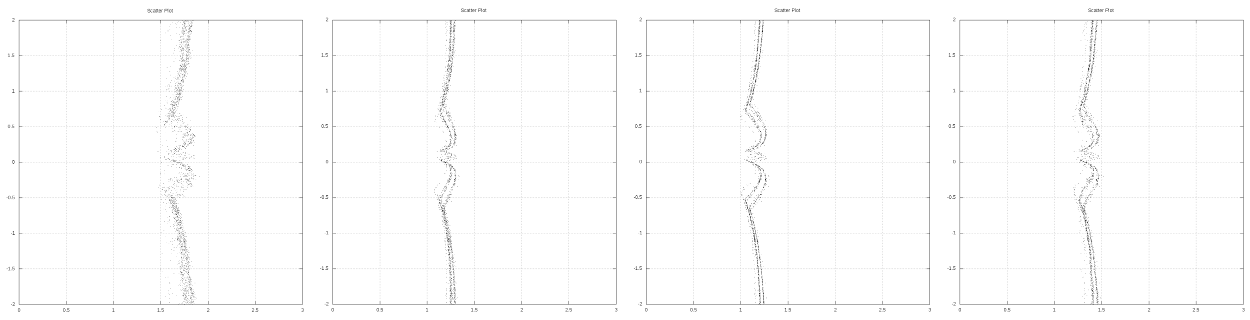
(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.

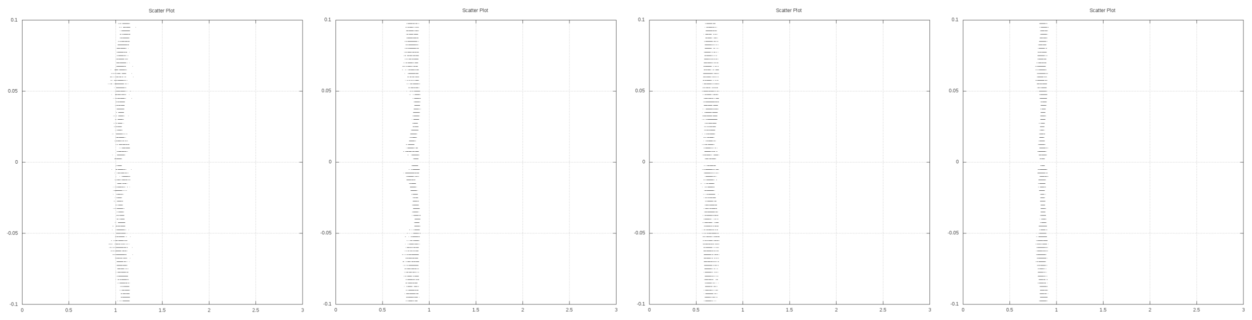


(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.

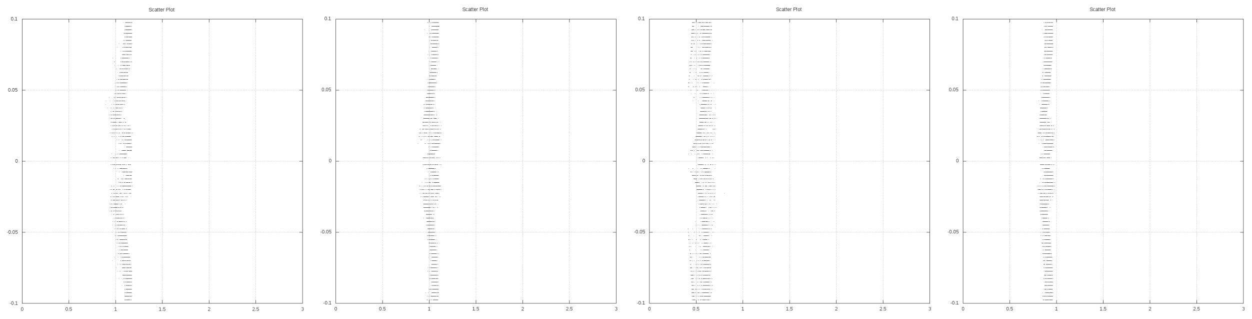


(d) The performance solving the "Arenstorf orbit" IVP (2.15).

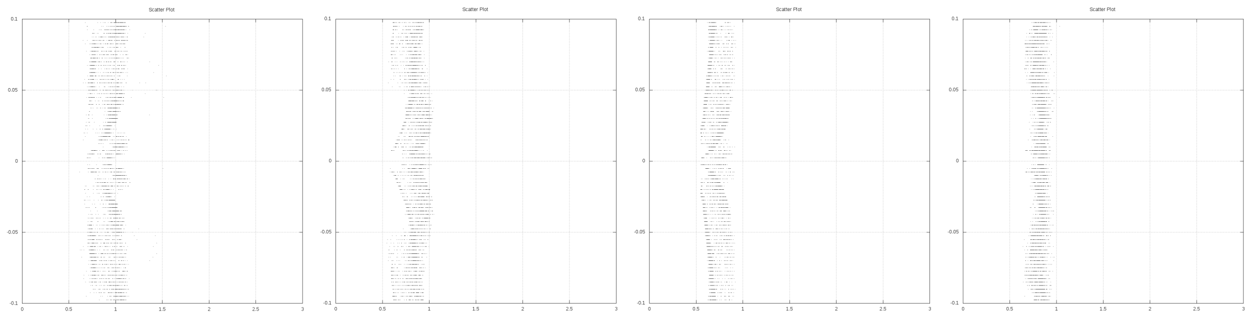
Figure 5.42: The performance of different values of $\hat{b}_6$ for the $5(4)_{6(6)\mathrm{C}(2)}$ family with restrictions on the $\overline{\mathrm{PEC}}$s (2.48) discussed in Section 5.11.



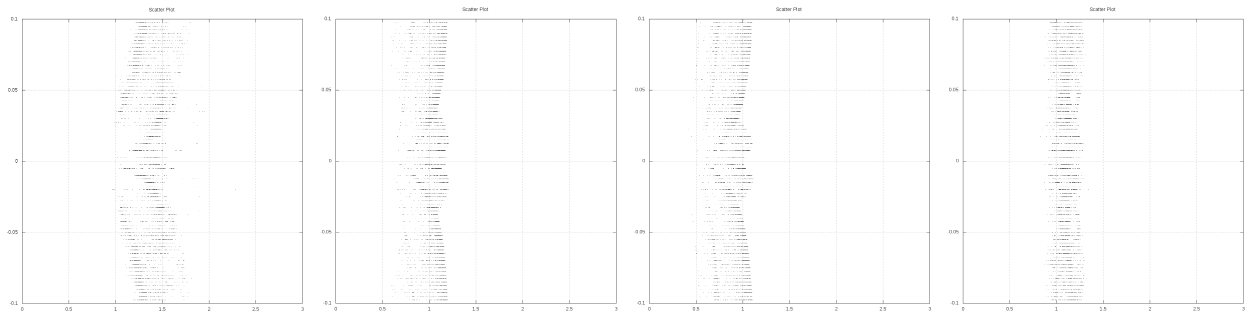
(a) The performance solving the "non-stiff D" IVPs (2.13).



(b) The performance solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$.



(c) The performance solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$.



(d) The performance solving the "Arenstorf orbit" IVP (2.15).

Figure 5.43: The performance of different values of $\hat{b}_7$ for the $5(4)_{6(7)\text{C}(2)}$ family with restrictions on the $\overline{\text{PEC}}$s (2.48) discussed in Section 5.11.

characteristic numbers should be "similar in size" [189, 190]. In fact, Bogacki and Shampine appear to have used the extra degrees of freedom of families of seven-stage fifth-order ERK pair to find ERK pairs with the B and C classic characteristic numbers (2.56) much closer to one than is practical when constructing efficient $5(4)_6$ ERK pairs [16]. Despite the lack of rigorous analysis to support the classic characteristic numbers, there are simplistic arguments that particular values of the classic characteristic numbers imply that the error estimate better reflects the leading error coefficient and has appropriate asymptotic behaviour.

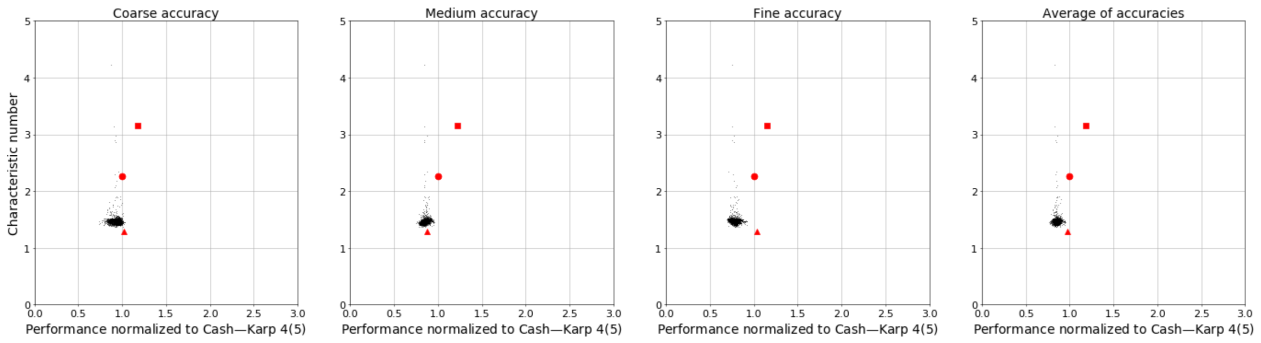
Shampine also introduced the E characteristic number (2.56d) in 1986 [146] and mentioned the best values were less than 0.5. However, Shampine did not call the E characteristic number (2.56d) by the notation "E" in his 1986 paper [146], and other than use by Kennedy and Carpenter [94, 96] (who appear to be the first to use the notation "E"), the E characteristic number (2.56d) has not been mentioned by others since. The behaviour of the E characteristic number (2.56d) (not shown), which gets smaller as $\hat{A}^5$ gets bigger, often looks like the inverse of Figures 5.42 and 5.43.

However, it is important to examine the classic characteristic numbers empirically. In Figures 5.44 and 5.45 the B and C numbers are shown for the performance solving the "non-stiff D" IVPs (2.13) with the restrictions discussed in Section 5.11. At best, the classic characteristic numbers (2.56) should be checked to make sure that they are not extremely large, indicating that an error estimate may not even remotely resemble the actual local error. In addition, overly minimizing the C characteristic number (2.56b) within the relatively narrow range that it occurs for most families often reduces performance, such as seen from Figure 5.44d.

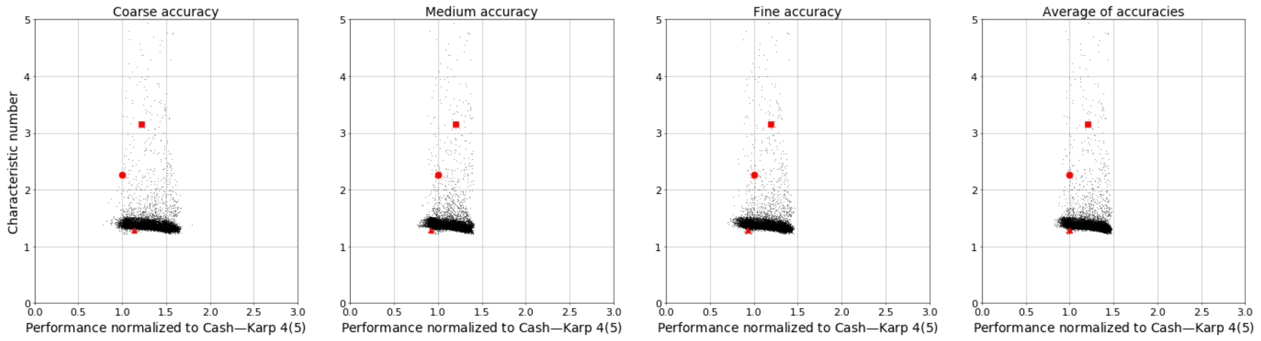
The behaviour seen in Figures 5.44 and 5.45 is similar to what was seen on all IVPs examined during this study. Therefore, without compelling evidence in future studies, the B (2.56a) and C (2.56b) characteristic numbers are not likely to be useful. The E characteristic number (2.56d) is probably still useful because, without performance data to the contrary, good practice dictates that the error coefficients of the error estimate should be relatively close in magnitude to that of the method used to advance the solution.

5.14 Final selection of efficient $5(4)_6$ ERK pairs and examples of their overall performance

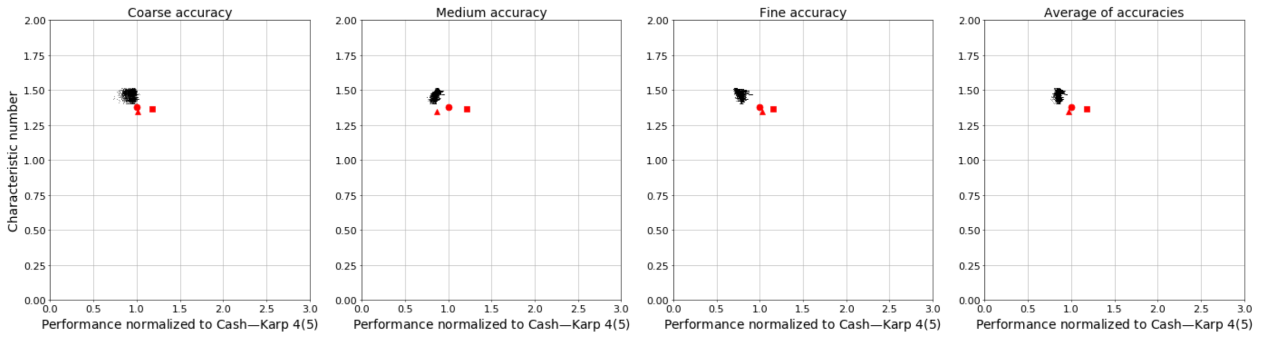
To present ERK pairs for practical use, no more than a few pairs must be selected out of possibly hundreds of candidate ERK pairs still remaining after performance testing and applying the appropriate restrictions, such as those described in Section 5.11. Irregardless of the what the performance data on IVP test sets indicates, final selection of ERK pairs must be based on good practice from the literature, experience, and common sense. Each ERK pair that is a final candidate and its properties should be manually inspected before its selection for use. An advantage to software such as `OCSage` is that nearly all conceivable properties can be checked for anything out of place in comparison to other successful ERK pairs. Sage notebooks, such as shown in Figure 4.1, helped a great deal with inspecting the properties of individual ERK pairs. Although there are probably many other undesirable properties that can be discovered through examination of data



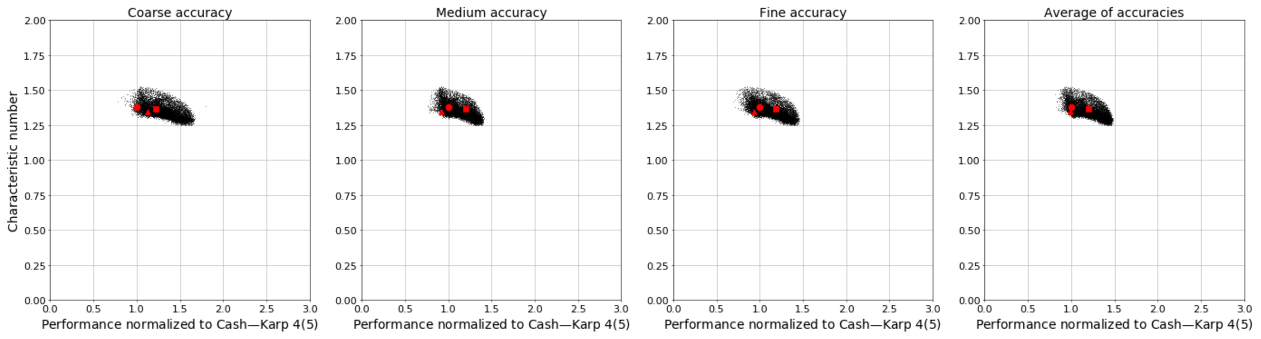
(a) The B characteristic number (2.56a) with the restrictions described in Section 5.11.3.



(b) The B characteristic number (2.56a) without the restrictions described in Section 5.11.3.

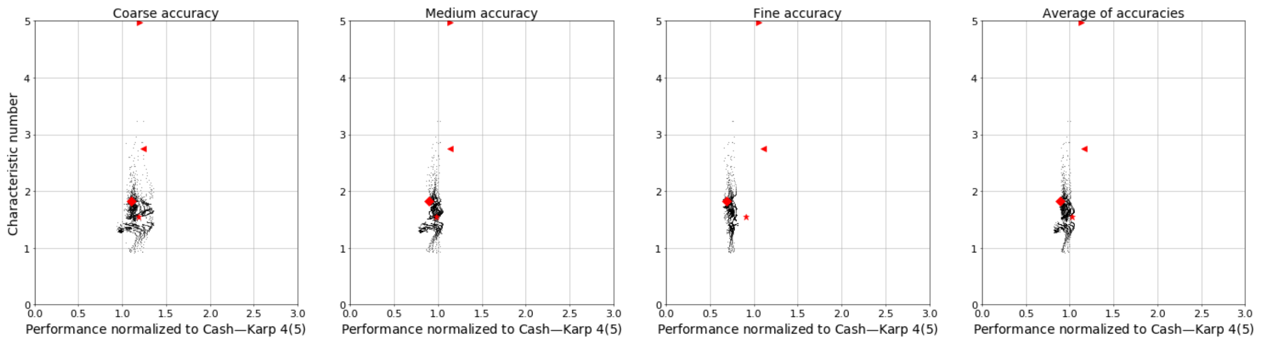


(c) The C characteristic number (2.56b) with the restrictions described in Section 5.11.3

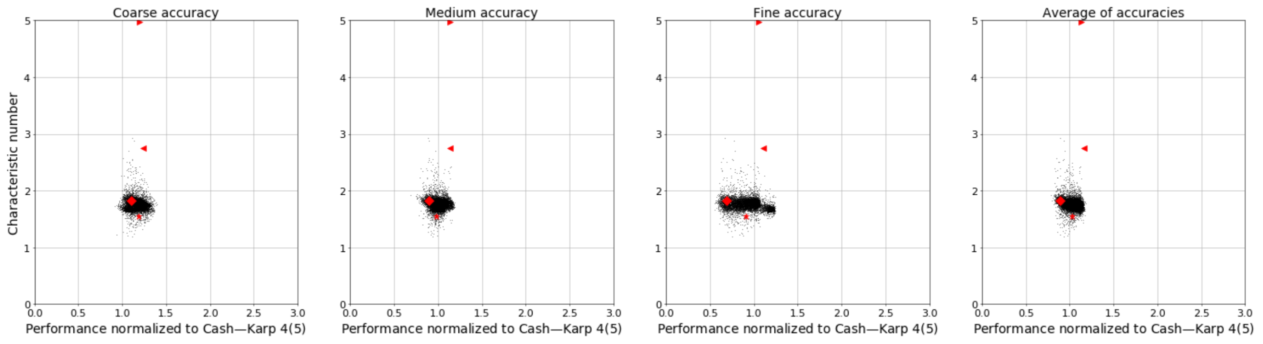


(d) The C characteristic number (2.56b) without the restrictions described in Section 5.11.3.

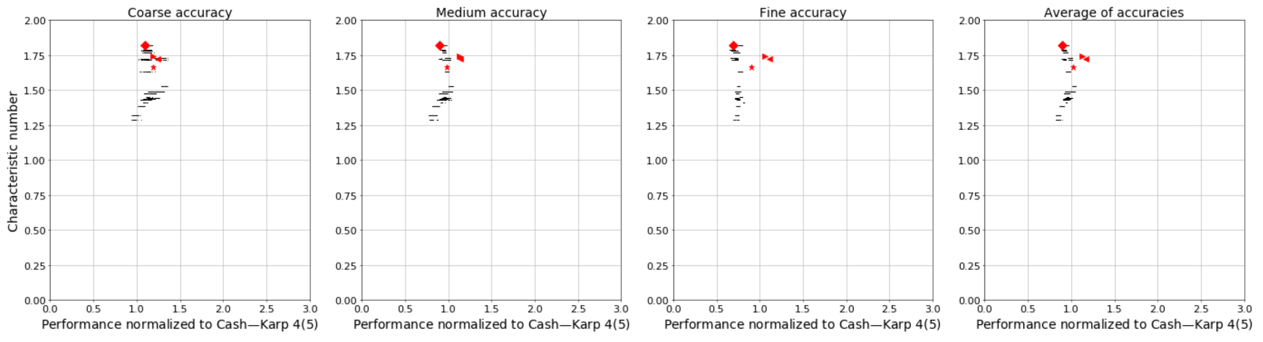
Figure 5.44: The values of the classic characteristic numbers (2.56) in comparison to performance solving the "non-stiff D" IVPs (2.13) for the $5(4)_{6(6)C(2)}$ family. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.



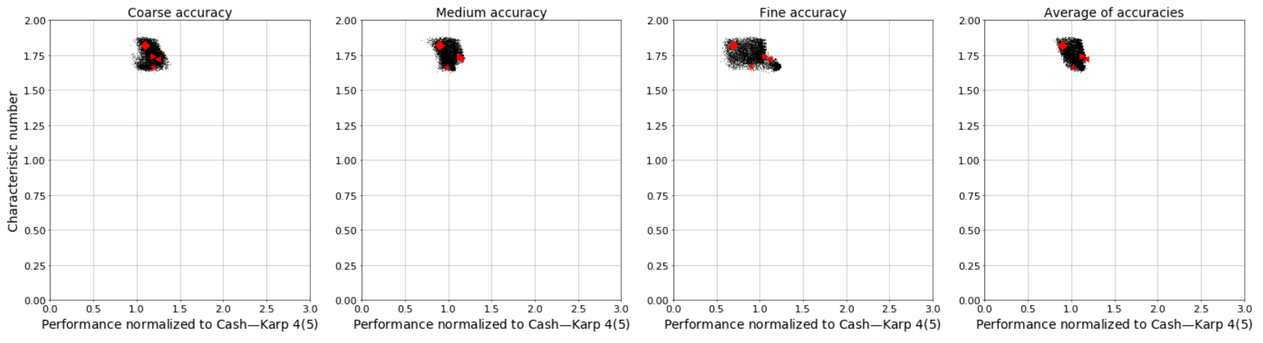
(a) The B characteristic number (2.56a) with the restrictions described in Section 5.11.4.



(b) The B characteristic number (2.56a) without the restrictions described in Section 5.11.4.



(c) The C characteristic number (2.56b) with the restrictions described in Section 5.11.4.



(d) The C characteristic number 2.56b without the restrictions described in Section 5.11.4.

Figure 5.45: The values of the classic characteristic numbers (2.56) in comparison to performance solving the "non-stiff D" IVPs (2.13) for the $5(4)_{6(7)\mathrm{C}(2)}$ family. See Table 5.2 for the symbols indicating published $5(4)_6$ ERK pairs.

from further performance testing, some specific undesirable properties to avoid that might not be found by only performance testing using standard IVP test sets are:

- A large magnitude of the coefficients of the Butcher tableau (2.34), i.e., the D characteristic number (2.56c), when equivalent pairs with coefficients of a smaller magnitude exist. However, the best performing pairs from some families do require that the magnitude of the coefficients of the Butcher tableau (2.34) are at least somewhat large. For instance, pairs from the $5(4)_{6(7)\mathrm{C}(2)}$ family with $A^6 \in [0.00035, 0.00040]$ suffer from severe performance loss on many IVPs tested if the D characteristic number (2.56c) is forced to be less than approximately 10, as opposed to a D characteristic number (2.56c) of 12 that the DP$5(4)_{6(7)}$ pair (2.79) has. Investigation (not shown) indicates that this is probably because the overly minimized the D characteristic number (2.56c) means the values of the $\overline{\mathrm{PEC}}$s (2.48) do not end up in the ideal ranges that are discussed in Section 5.11. Although published guidelines for the D characteristic number (2.56c) include arbitrary upper limits, such as 20 [94], any upper limit is probably both overly conservative and not conservative enough depending on the specific application and family under study.

- An extremely small magnitude of the coefficients of the Butcher tableau (2.34) may give issues with roundoff error because the values in the summations given by (2.33) might unnecessarily be significantly different in magnitude from each other. Although not observed during this study, the author expects the effect may become noticeable when the magnitude of the coefficients are less than approximately 10^{-3}. Future work may actually identify specific IVPs and circumstances where extremely small coefficients cause an issue, but even if no empirical evidence is immediately forthcoming overly small coefficients should still be avoided.

- A relatively small distance between the components of the $\mathbf{c}$ vector of the coefficients of the Butcher tableau (2.34) may be disadvantageous if alternatives exist that are further apart. This is because relatively close components of the $\mathbf{c}$ vector may result in roundoff error due to subtractive cancellation while applying the ERK formula (2.33). However, because numerical methods are based on mathematical analysis, the intuitive idea of it being better to "evenly sample" the timestep is not necessarily a valid one. In fact, many of the better pairs constructed in this study have some components of the $\mathbf{c}$ vector closer together than many of the published $5(4)_6$ ERK pairs. Future performance testing may identify specific instances where the components of the $\mathbf{c}$ vector must be relatively far apart.

- A magnitude of the $\overline{\mathrm{PEC}}$s (2.48), including the higher-order ones (greater than the sixth-order $\overline{\mathrm{PEC}}$s (2.48) for $5(4)_6$ ERK pairs), that is too large. However, large $\overline{\mathrm{PEC}}$s (2.48) sometimes necessary, for instance, for the $5(4)_{6(7)\mathrm{C}(2)}$ family with leading error coefficients less than approximately 0.0007. Based on searches in `OCSage`, to have a leading error coefficient of $A^6 \in [0.00035, 0.00040]$ the author found the magnitude of at least one sixth-order $\overline{\mathrm{PEC}}$ (2.48) needs to be greater than 0.5 and the magnitude of at least one seventh-order $\overline{\mathrm{PEC}}$ (2.48) needs to be greater than 1.5. The author's hypothesis is that these large $\overline{\text{PEC}}$s (2.48) (greater than the 1.0 that corresponds to a simple truncation of the Taylor series) are responsible for the poorer performance often seen at coarse accuracies from the $5(4)_{6(7)\text{C}(2)}$ family, such as that clearly seen in Figures 5.12 and 5.14. However, convincing empirical evidence based on performance data is difficult to find because of the difficulty in doing controlled experimentation on the magnitude of the PECs within the same family of $5(4)_6$ ERK pairs.

Finally, as mentioned during their construction in Section 3.5.2, $5(4)_{6(7)}$ ERK pairs based on Cases II–IV have at least one PEC of the error estimate that vanishes. Experimentation by the author (not shown) indicated that the best pairs from Cases II–IV in Section 3.5.2 do not perform as well as pairs from the $5(4)_{6(7)\text{C}(2)}$ family, even when the leading error coefficients are held within a similar range. It is difficult to demonstrate conclusively that it is vanishing PECs in the error estimate that account for poor performance. However, this disadvantage and these obvious issues with performance based on the author's experimentation unfortunately preclude constructing competitive pairs from Cases II–IV.

5.14.1 Specific systematically constructed $5(4)_6$ ERK pairs for IVPs from celestial mechanics

A $5(4)_{6(6)}$ ERK pair from the $5(4)_{6(6)\text{C}(2)}$ family with $\hat{b}_6 = \frac{16}{100}$, $c_2 = \frac{17}{100}$, $c_3 = \frac{111}{400}$, $c_5 = \frac{37}{50}$, $c_6 = \frac{97}{100}$, and $A_6 = 0.00099349206670410308$ is given by

$$\begin{array}{r|rrrrrr}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{17}{100} & \frac{17}{100} & 0 & 0 & 0 & 0 & 0 \\
\frac{111}{400} & \frac{111}{2176} & \frac{12321}{54400} & 0 & 0 & 0 & 0 \\
\frac{4440}{8801} & \frac{2669262363480}{11588973888817} & -\frac{5516850960000}{11588973888817} & \frac{511416960000}{681704346401} & 0 & 0 & 0 \\
\frac{37}{50} & \frac{14132565029}{43090920000} & -\frac{553076}{997475} & \frac{240226496}{456193431} & \frac{8027473394837}{18247737240000} & 0 & 0 \\
\frac{97}{100} & -\frac{5120038304072421907}{7923144265858800000} & \frac{2759941}{1994950} & \frac{219024991005196 84}{41940232036704045} & -\frac{121532418488282516049 05771}{93979671947846424036 00000} & \frac{12406183577011}{12360503097360} & 0 \\
\hline
\mathbf{b} & \frac{929924623}{10612816560} & 0 & \frac{1062260224000}{2727224432613} & \frac{69974267288050869263}{90387211757385223944 0} & \frac{1124213000}{3263218219} & \frac{25558830000}{253187419939} \\
\hat{\mathbf{b}} & \frac{2220456409}{17095387500} & 0 & \frac{220277882624}{1230696946125} & \frac{14330038895260780 9}{34471821460961250 0} & \frac{2047038807}{17734881625} & \frac{4}{25}
\end{array} \tag{5.2}$$

A $5(4)_{6(7)}$ ERK pair from the $5(4)_{6(7)\text{C}(2)}$ family with $\hat{b}_7 = \frac{19}{100}$, $c_2 = \frac{21}{100}$, $c_3 = \frac{8}{25}$, $c_4 = \frac{189}{200}$, $c_5 = \frac{121}{125}$, and $A_6 = 0.00037118380356134293$, and $A^7 = 0.0021952940095218451$ is given by

$$\begin{array}{r|rrrrrrr}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{21}{100} & \frac{21}{100} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{8}{25} & \frac{8}{105} & \frac{128}{525} & 0 & 0 & 0 & 0 & 0 \\
\frac{189}{200} & \frac{274239}{163840} & -\frac{158193}{25600} & \frac{178605}{32768} & 0 & 0 & 0 & 0 \\
\frac{121}{125} & \frac{1237893283}{632100000} & -\frac{142865063}{19753125} & \frac{941885901}{150500000} & -\frac{534336}{32921875} & 0 & 0 & 0 \\
1 & \frac{5108222489}{2176330464} & -\frac{5939975}{686973} & \frac{3106763821}{423960480} & \frac{47548864}{711017055} & -\frac{63962500}{670387509} & 0 & 0 \\
1 & \frac{209687}{2195424} & 0 & \frac{64565}{132192} & \frac{1095680}{143451} & -\frac{117578125}{10820304} & \frac{32713}{8976} & 0 \\
\hline
\mathbf{b} & \frac{209687}{2195424} & 0 & \frac{64565}{132192} & \frac{1095680}{143451} & -\frac{117578125}{10820304} & \frac{32713}{8976} & 0 \\
\hat{\mathbf{b}} & \frac{1682001271}{27095191200} & 0 & \frac{4890084329}{8157348000} & -\frac{48340794112}{8852122125} & \frac{55191322375}{5341623408} & -\frac{1569602453}{332336400} & \frac{19}{100}
\end{array} \tag{5.3}$$

In comparison to the $\text{DP}5(4)_{6(7)}$ pair, the slightly smaller $A_6 = 0.00037118380356134293$ and $A^7 = 0.0021952940095218451$ that is half the size, gives a noticeable performance advantage. In fact, even though (5.3) is designed for specifically for IVPs from celestial mechanics it outperforms the $\text{DP}5(4)_{6(7)}$ pair (2.79) on the DE test set. For instance, at coarse accuracies the relative performance of (5.3) is 1.120, as compared to 1.212 for the $\text{DP}5(4)_{6(7)}$ pair (2.79). At medium accuracies the relative performance of (5.3) is 0.953, as compared to 1.139 for the DP5(4)$_{6(7)}$ pair (2.79). At fine accuracies the relative performance of (5.3) is 0.996, as compared to 1.110 for the DP5(4)$_{6(7)}$ pair (2.79). The average relative performance of (5.3) at these three accuracies is 1.02, as compared to 1.15 for the DP5(4)$_{6(7)}$ pair (2.79). From Figure 5.16a, where larger A^7 obviously limits performance to poorer than the performance of (5.3), a large part of the 13% improvement of (5.3) over the DP5(4)$_{6(7)}$ pair (2.79) can be explained because A^7 is half the magnitude.

The only disadvantage is that the coefficients c_4 and c_5 are quite close to each other, i.e., only 0.023 apart, but no performance issues were observed at all. In addition, the pair (5.3) has $c_3 \neq \frac{3}{2}c_2$ indicating the C(3) simplifying assumptions (2.44b) do not hold.

A 5(4)$_6$ ERK pair from the family of Case VIa from Section 3.5.1 with $c_3 = \frac{1}{20}$, $c_5 = \frac{13}{25}$, $c_6 = \frac{21}{25}$, $\lambda = 1$, $\mu = \frac{1}{20}$, and $A_6 = 0.0010015924255361622$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{130423}{1686340} & \frac{130423}{1686340} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{20} & \frac{11604217947619}{185317498140320} & -\frac{2338343040603}{185317498140320} & 0 & 0 & 0 & 0 \\
\frac{560598}{1305545} & \frac{96461081294844039998 5238334}{23132982819621694788 2562545} & \frac{5486804627322276940803615 9262248}{5332846529407389299536714 349885} & -\frac{573635841781318297580743872}{408888503516050796219739 95} & 0 & 0 & 0 \\
\frac{13}{25} & \frac{957605246820829021868 57}{188952440810258150575 00} & \frac{25852185784292621161042125036}{21181936169927714550947393 75} & -\frac{70272518143127078688}{41729095127480131 25} & a_{5,4} & 0 & 0 \\
\frac{21}{25} & \frac{174776610371632763751745233 6879}{230084898732443633332458842500} & \frac{8825147013035688056820441550988297822 76}{3684227398393412185406675287784306 3125} & -\frac{55418408457853208833329176 64}{18370842847473399357255062 5} & a_{6,4} & a_{6,5} & 0 \\
\hline
\mathbf{b} & \frac{2348485259067}{8871271251752} & \frac{39221328957644651925896 00000}{58722463348025037640989 22647} & -\frac{380663679119360}{508768444851861} & b_4 & b_5 & b_6 \\
\hat{\mathbf{b}} & \frac{152247684684077}{5702960090412} & \frac{72682052843815326250000}{91318473888514085 3319} & -\frac{220099607407680}{2146702298953} & \hat{b}_4 & \hat{b}_5 & 0
\end{array} \tag{5.4}$$

where

$$a_{5,4} = \frac{9116255266251006479145738991}{10433278800833050573097771150 0}$$
$$a_{6,4} = -\frac{185910751560849132730575289703029242479337}{577981871312260027788223502858979892 11500}$$
$$a_{6,5} = \frac{2531846898618232850 2392}{94718842612427426211 13}$$
$$b_4 = -\frac{715807631822203703771311 55}{7163024525369848435546548}$$
$$\hat{b}_4 = \frac{922871452352464474891008261 9845}{1279936498154951959463131803 8568}$$
$$b_5 = -\frac{584186192486875}{1932517170833184}$$
$$\hat{b}_5 = \frac{436879786949375}{60391161588537}$$
$$b_6 = \frac{363103380659006875}{915008712245089056}$$

A 5(4)$_6$ ERK pair from the family of Case VIb from Section 3.5.1 with $c_3 = \frac{19}{100}$, $c_5 = \frac{59}{100}$, $c_6 = \frac{93}{100}$, $\lambda = \frac{87}{200}$, $\mu = 1$, and $A_6 = 0.0010421361266417699$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{87}{548} & \frac{87}{548} & 0 & 0 & 0 & 0 & 0 \\
\frac{19}{100} & \frac{16423799}{117302100} & \frac{58636}{1173021} & 0 & 0 & 0 & 0 \\
\frac{3080625}{7462541} & \frac{1105759974055014599 8887}{6025987033624044493 1045} & -\frac{3630206683153565872309 3011}{2579122450391091043048 7260} & \frac{29114714492237605971 4875}{17787051382007524434 8188} & 0 & 0 & 0 \\
\frac{59}{100} & \frac{57505606497}{211339283500} & -\frac{21679249788672944884 36}{11746948885604554169 25} & \frac{36290924455352}{19232440320407} & a_{5,4} & 0 & 0 \\
\frac{93}{100} & -\frac{146220051201}{1056696417500} & -\frac{1026958049518772362845753321}{2132978348234503822290833 75} & \frac{523743884283861768 6}{744776251407761075} & a_{6,4} & a_{6,5} & 0 \\
\hline
\mathbf{b} & \frac{1}{9} & -\frac{468465191797407209 59}{416367849446972650 08} & \frac{32768063548125}{21064948510048} & b_4 & b_5 & b_6 \\
\hat{\mathbf{b}} & \frac{1}{9} & -\frac{4656020795272057}{562948337588184} & \frac{9146437429375}{853984399056} & \hat{b}_4 & \hat{b}_5 & 0
\end{array} \tag{5.5}$$

$$
\begin{aligned}
a_{5,4} &= \frac{870092576955638503066661624}{3147300636610885205517798 75}\\
a_{6,4} &= -\frac{1079600444663726404797686853092285334 03}{358128318085544992049927156770728756 25}\\
a_{6,5} &= \frac{2015399920718943}{1080426934963525}\\
b_4 &= -\frac{72664604910700150364553024515382 20}{26448076934779075783419666789805113}\\
\hat{b}_4 &= -\frac{26800681228371103814883869 0}{68526533600483112168190101}\\
b_5 &= \frac{57927004473125}{106243144437472}\\
\hat{b}_5 &= \frac{3160979800625}{1339199299632}\\
b_6 &= \frac{2892660895646875}{15390326667308292}
\end{aligned}
$$

A $5(4)_6$ ERK pair from the family of Case VIb from Section 3.5.1 with $c_3 = \frac{3}{25}$, $c_5 = \frac{63}{100}$, $c_6 = \frac{9}{10}$, $\lambda = 1$, $\mu = \frac{189}{200}$, and $A_6 = 0.0010209372714612293$ is given by

$$
\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0\\
\frac{200}{989} & \frac{200}{989} & 0 & 0 & 0 & 0 & 0\\
\frac{3}{25} & \frac{62402538537}{240190750000} & -\frac{33579648537}{240190750000} & 0 & 0 & 0 & 0\\
\frac{25302015}{66137108} & \frac{3456039335310370585393 63}{2314331616359715385309696} & \frac{2737482275093524131724921461}{4705036176059301378334611968} & -\frac{2050116921547877128567731 25}{5881295220074126722918264 96} & 0 & 0 & 0\\
\frac{63}{100} & \frac{78503299596631}{545713384000000} & \frac{23275983809682628586979699 9}{26174399661154261769840000 00} & -\frac{1038800899496279}{6684259524105920} & b_{5,4} & 0 & 0\\
\frac{9}{10} & \frac{28000415251}{545713384000} & \frac{7354836905688747124058466284657 97}{2665226472601116742223161323152 00} & -\frac{634911910987274180}{507006395490979667} & b_{6,4} & b_{6,5} & 0\\
\hline
\mathbf{b} & \frac{1}{9} & \frac{6294806203400478990763}{14003493110096036396570} & -\frac{673143584028125}{3511001459536974} & b_4 & b_5 & b_6\\
\hat{\mathbf{b}} & \frac{1}{9} & \frac{8967142302929690493}{20291976684677635 70} & -\frac{604308517939375}{270077035348998} & \hat{b}_4 & \hat{b}_6 & 0
\end{array}
\tag{5.6}
$$

$$
\begin{aligned}
a_{5,4} &= \frac{402198724324169272233434482 38}{727791507149496380641544593 75}\\
a_{6,4} &= -\frac{73795350978538140406695425240249084 67}{44985803327809287405012483252140517 50}\\
a_{6,5} &= \frac{450665515849591575}{459007423158614054}\\
b_4 &= \frac{1009738797889196251874653850418886 4}{8962316494072233164736957611479111 5}\\
\hat{b}_4 &= -\frac{4377006790513940937881634158}{15713538000938313645444088 95}\\
b_5 &= \frac{1712175742300000}{5958340165765791}\\
\hat{b}_5 &= \frac{988338099053750}{662037796196199}\\
b_6 &= \frac{287314685967350}{1243391658829983}
\end{aligned}
$$

5.14.2 Overall performance of the pairs presented in this section

Table 5.3 gives method abbreviations for the specific pairs tested in Figures 5.46–5.49.

It is immediately seen from Figures 5.46–5.50 that although the eighth- and seventh-order Dormand–Prince ERK pair [136] is often the most efficient at high accuracies, even when using small values for the tolerances (such as 10^{-14}) it is extremely difficult to reach the highest possible accuracies for many IVPs. This is also seen to a lesser extent for the sixth- and fifth-order ERK pair from Verner [72, pg.181][187], which is commonly seen in software including `FATODE` [203].

What can also clearly be seen is that the optimized methods from this study, i.e., (5.2)–(5.6), do tend

Table 5.3: Abbreviations used for the work-precision diagrams in Figures 5.46–5.49.

'BS32'	The Bogacki–Shampine third- and second-order pair [15].
'RKF45'	The RKF4(5)$_{6(6)}$ pair (2.78) using the fourth-order component to advance the step.
'RKF54'	The RKF4(5)$_{6(6)}$ pair (2.78) using the fifth-order component to advance the step.
'CK54'	The CK4(5)$_{6(6)}$ pair (2.83).
'DP54'	The DP5(4)$_{6(7)}$ pair (2.79).
'PP54'	The PP5(4)$_{6(7)}$ pair (2.84).
'V65',	The Verner sixth- and fifth-order pair [72, pg.181].
'DP78'	The Dormand–Prince seventh- and eighth-order pair [136].
'VVI'	The pair given by (5.2).
'VVIFSAL'	The pair given by (5.3).
'VIAMU'	The pair given by (5.4).
'VIBLA'	The pair given by (5.5).
'VIBMU'	The pair given by (5.6).

towards the lower end and sometimes consistently outperform classic pairs. In particular, observe on "non-stiff D5" and "non-stiff D6" IVPs (2.13) in Figures 5.47 and 5.48, respectively, that the pair (5.3) significantly outperforms most other methods and especially DP5(4)$_{6(7)}$ pair (2.79). This is likely because of the much smaller $A^7 = 0.0021$ and more consistent asymptotic behaviour in the former. Even for the "nine planets" and "Jovian asteroid" IVPs in Figures 5.49 and 5.50 respectively, the pair (5.3) converges more consistently.

The limits of trying to correlate performance of between the simple Kepler IVP (2.13) and more complex IVPs can also be seen because although the pairs constructed, i.e., (5.2)–(5.6), do quite well solving the "nine planets" and "Jovian asteroid" IVPs in Figures 5.49 and 5.50, they are not always the best performing. However, the pairs constructed, i.e., (5.2)–(5.6), have much more consistent convergence behaviour compared to pairs such as the DP5(4)$_{6(7)}$ pair (2.79) and PP5(4)$_{6(7)}$ pair (2.84). Even the pair (5.3) with a leading error coefficient of $A^6 = 0.00037118380356134293$ has more consistent convergence.

However, choosing random pairs according to just the error coefficients gives a much larger range of performance in 5(4)$_6$ ERK pairs than seen in Figures 5.46–5.50 (the author plotted several thousand pairs to make an extremely crowded work-precision diagram in order to observe this). Therefore, ensuring that the PECs fall within appropriate ranges, which is not always reflected by the classic error coefficients A^q (2.54), individual PECs with large magnitude tend to be obscured by the 2-norm. This systematic way of constructing embedded pairs is far more well-defined than trial and error. It may even lead to better ways of selecting pairs once a theoretical understanding of why some PECs are dominant is found.

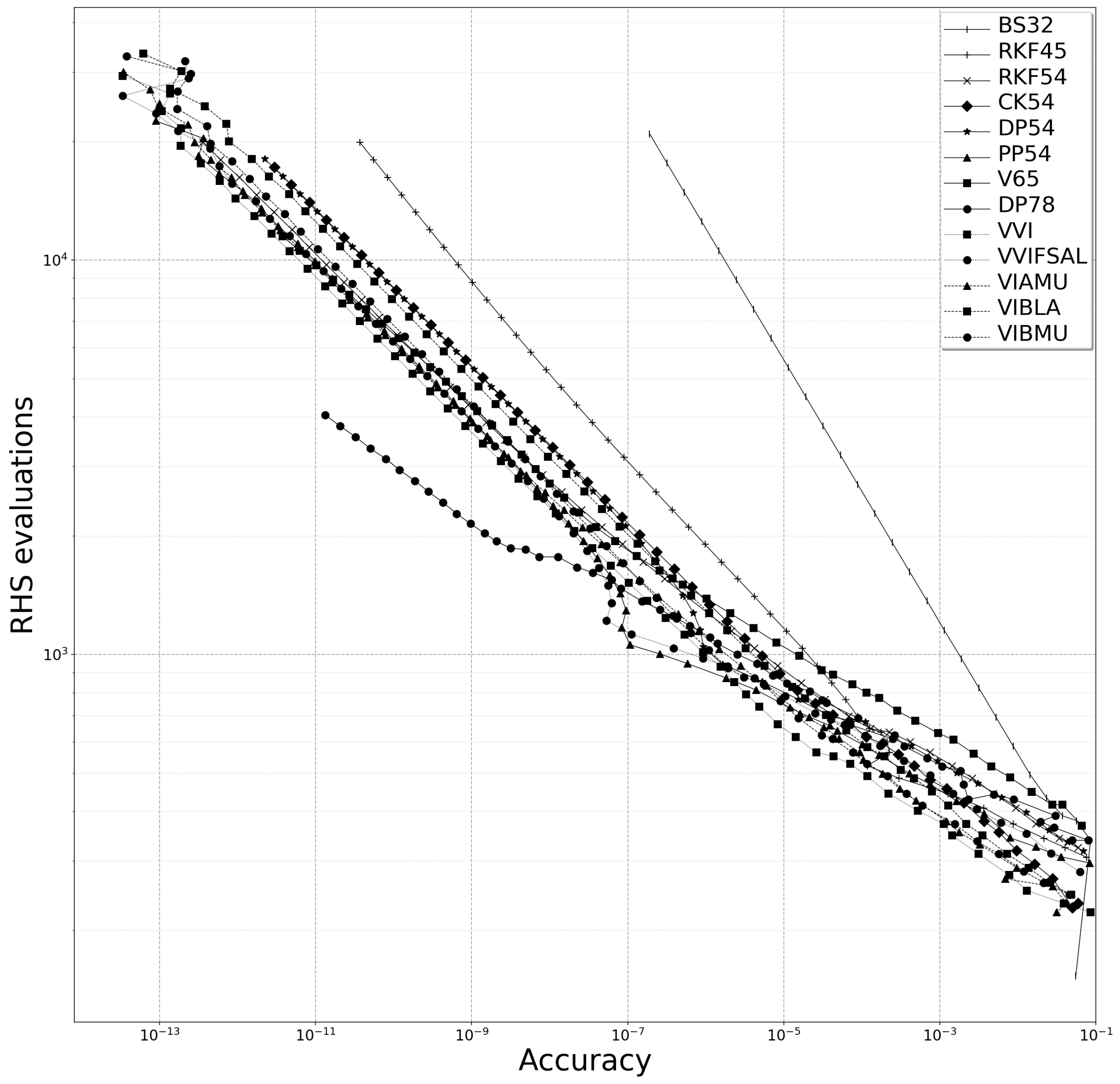


Figure 5.46: A work-precision diagram from solving the "non-stiff D2" IVP (2.13) with $\epsilon = 0.3$. See Table 5.3 for method abbreviations.

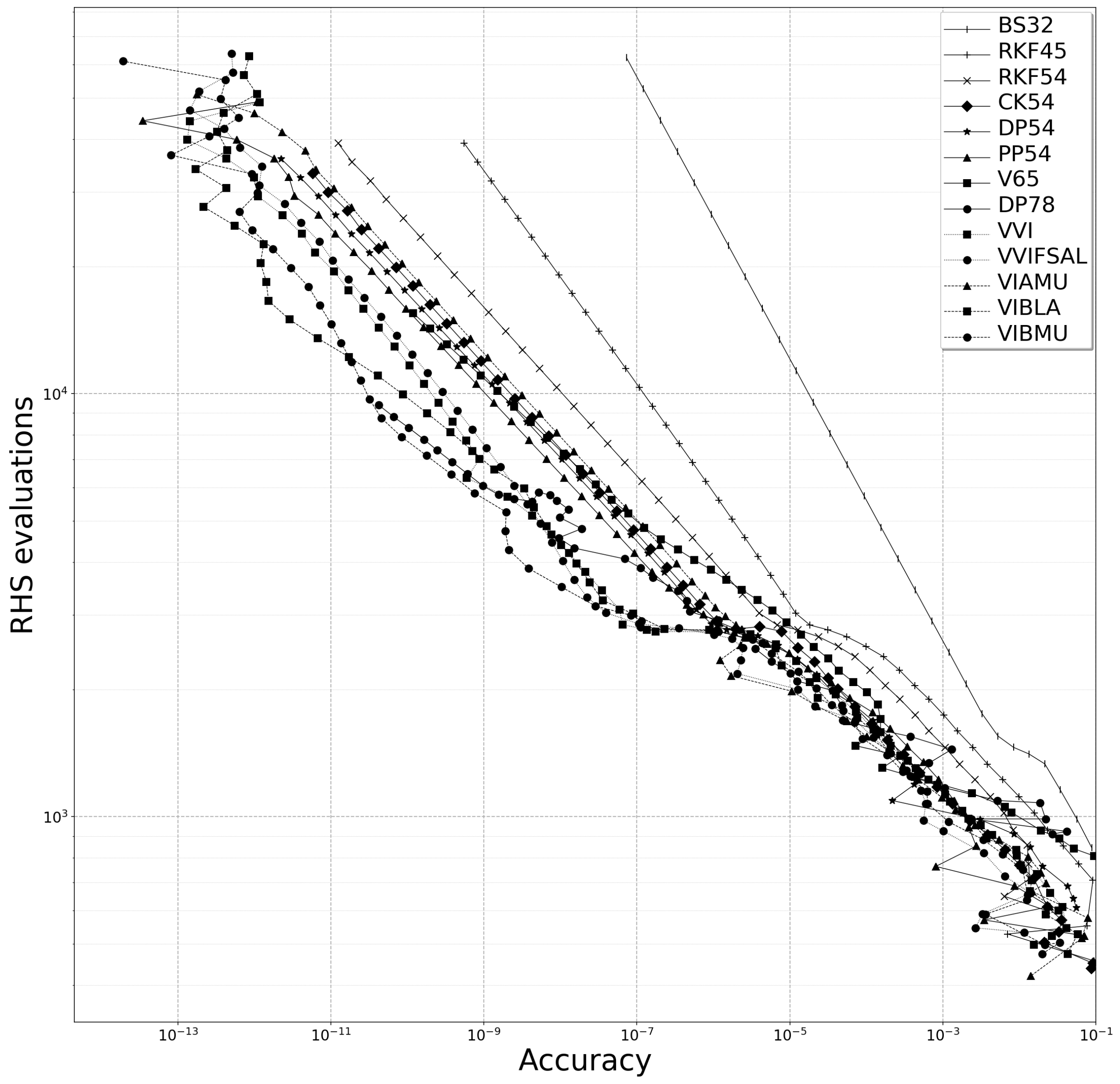


Figure 5.47: A work-precision diagram from solving the "non-stiff D5" IVP (2.13) with $\epsilon = 0.9$. See Table 5.3 for method abbreviations.

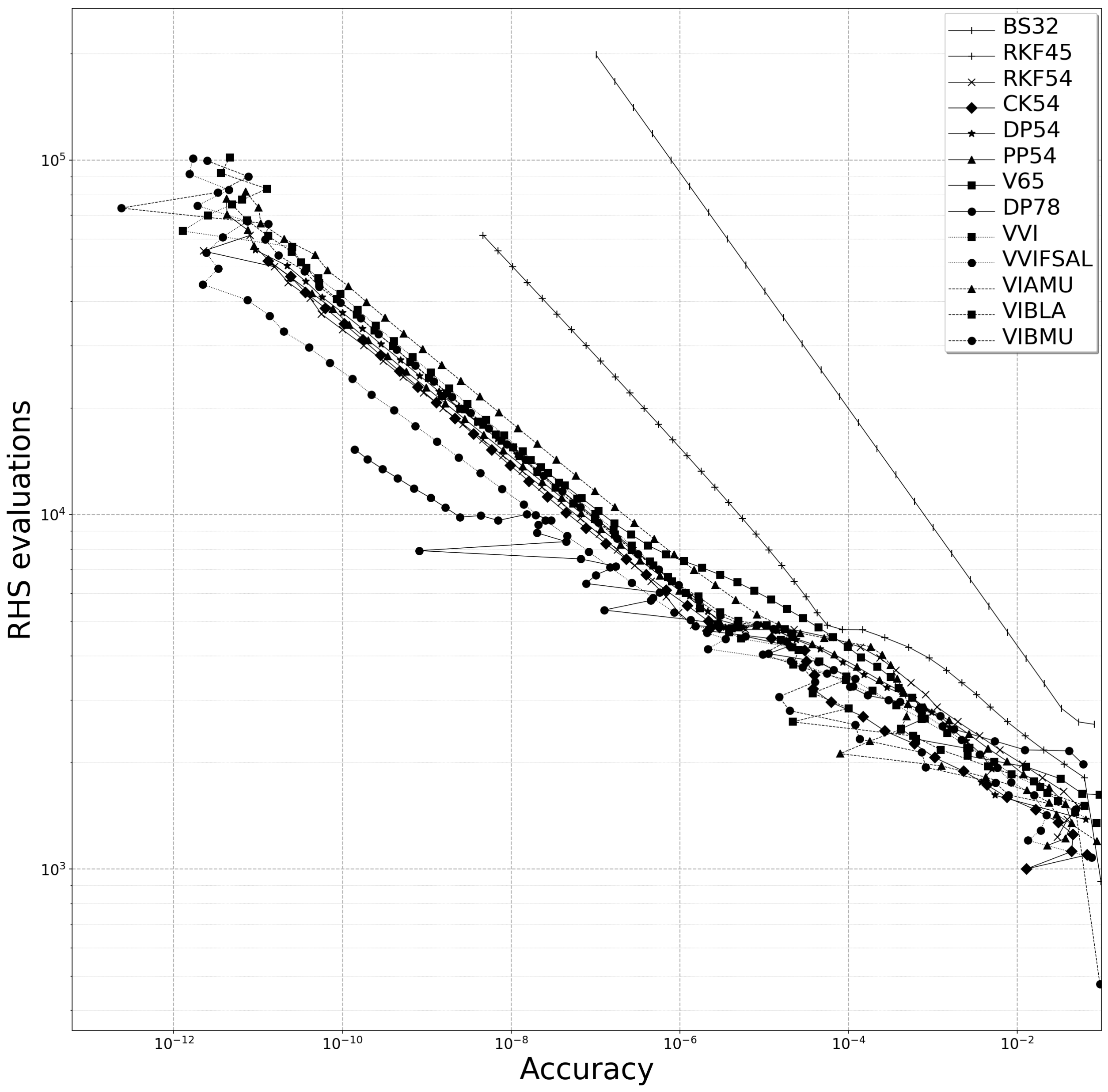


Figure 5.48: A work-precision diagram from solving the "non-stiff D6" IVP (2.13) with $\epsilon = 0.99$. See Table 5.3 for method abbreviations.

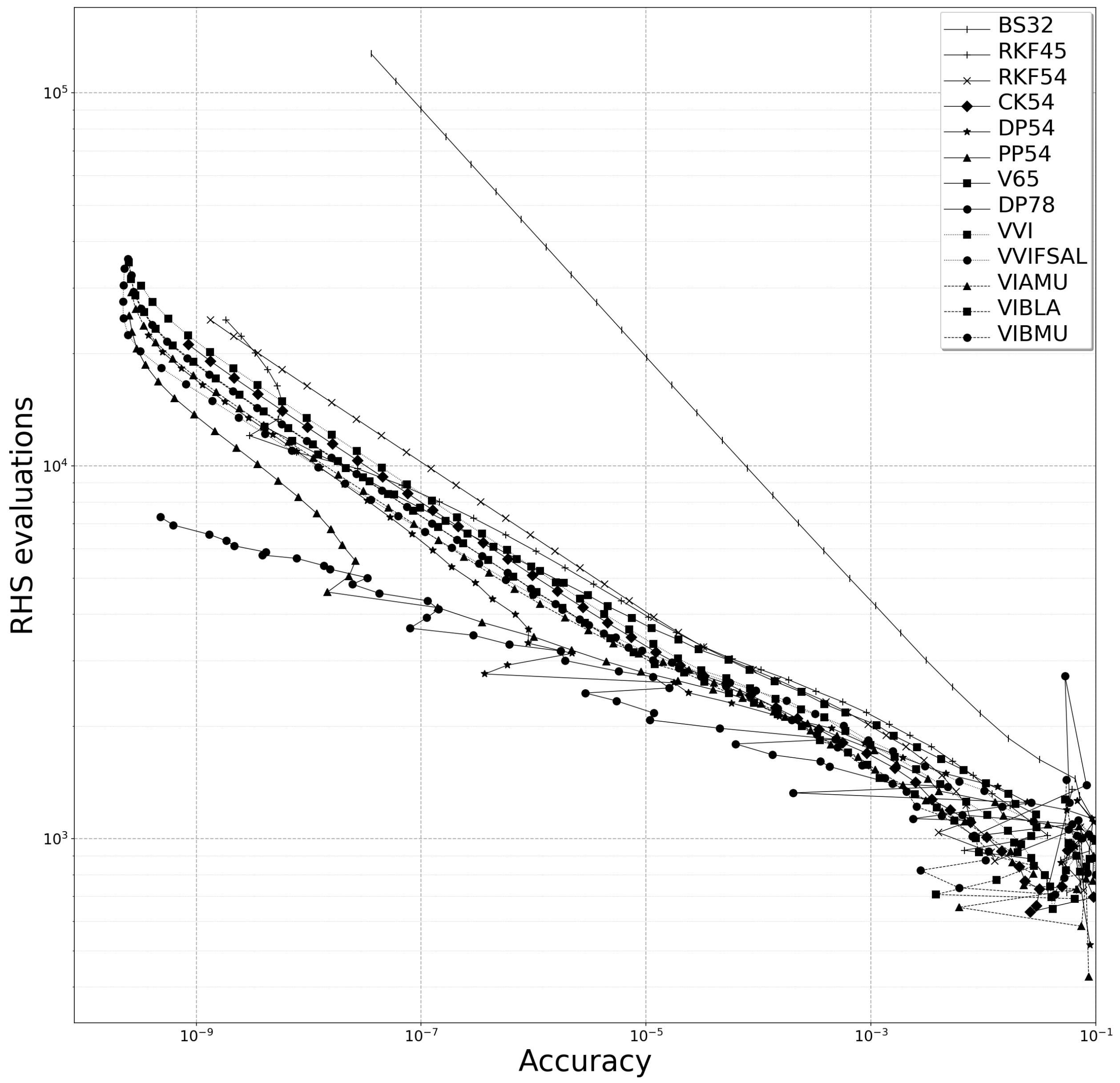


Figure 5.49: A work-precision diagram from solving the "Jovian asteroid" IVP described in Section 5.3.4. See Table 5.3 for method abbreviations.

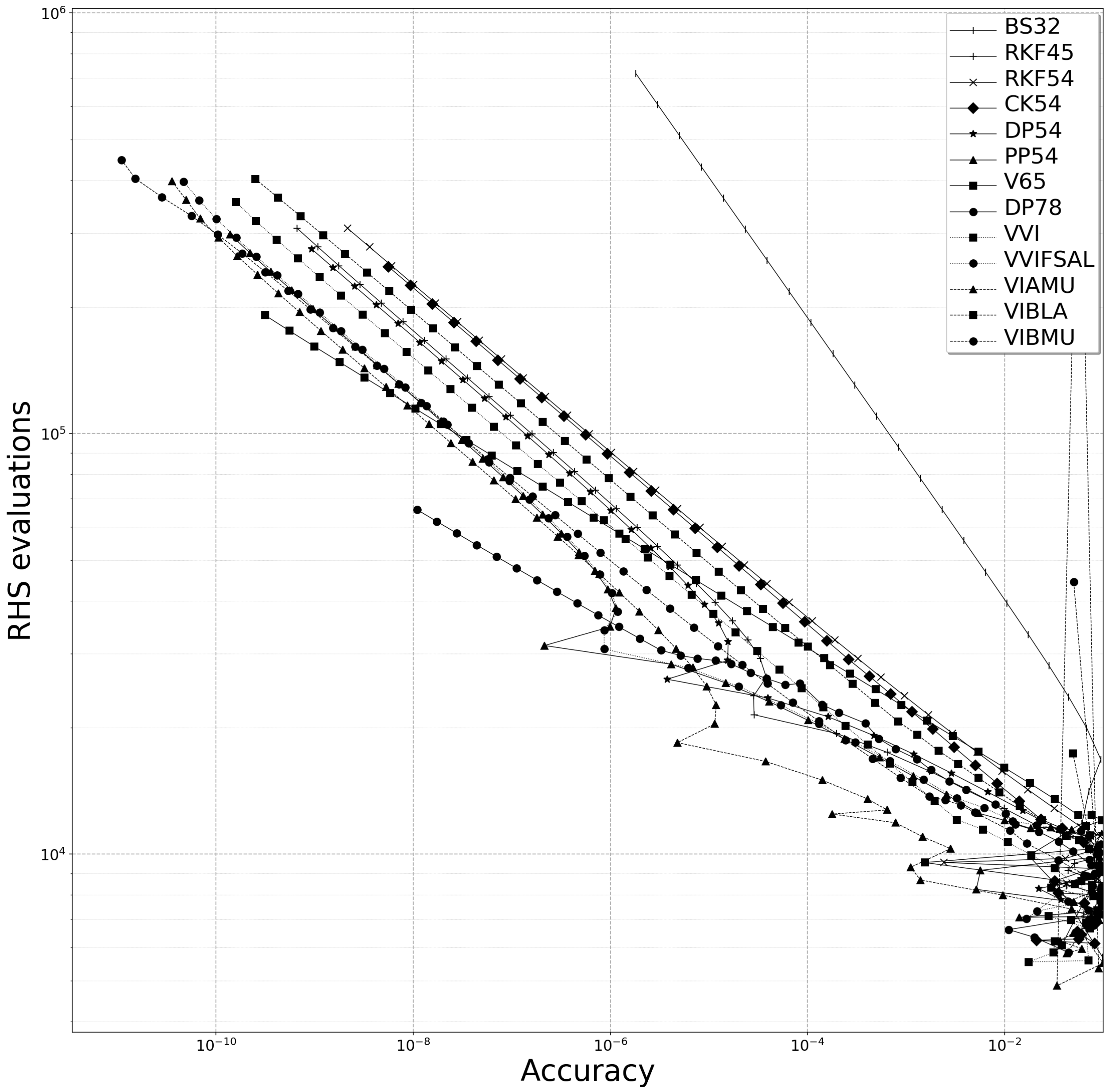


Figure 5.50: A work-precision diagram from solving the "nine planets" IVP described in Section 5.3.4. See Table 5.3 for method abbreviations.

Chapter 6

Conclusions and recommendations for future work

6.1 Conclusions

This study shows that efficient and competitive $5(4)_6$ ERK pairs exist that do not satisfy even the ubiquitously used C(2) simplifying assumptions (2.44b) and D(1) simplifying assumptions (2.44c). Even with the new work presented on the values of specific PECs being a significant factor in the performance of ERK pairs, it is still not clear whether forgoing the standard simplifying assumptions may actually lead to better performing ERK pairs. At this time, it can only be concluded that $5(4)_6$ ERK pairs without the simplifying assumptions can perform at least as well as otherwise similar pairs that satisfy the standard simplifying assumptions. However, close examination of some of the figures from Chapter 5 indicates that forgoing the standard simplifying assumptions may lead to more favourable performance tradeoffs among some IVPs. Therefore, possible research directions to explore this is discussed as future work in Section 6.2.5 below.

An additional benefit to finding the complete solution is that it gave confidence that any observations about the different families of $5(4)_6$ ERK pairs were not an artifact of a restricted solution, especially having fewer free parameters (degrees of freedom) than necessary. Despite sometimes being portrayed as "cheap" in the literature, finding the complete solution for ERK pairs based on the six-stage fifth-order order conditions in Chapter 3 showed that constructing embedded ERK pairs, in comparison to just a single ERK method, does require giving up potentially important free parameters. However, given that ERK pairs have the enormous advantage of error estimation and step control, finding the complete solution ensures as few free parameters are given up as possible and allows the practitioner to be confident that they are using the best ERK pairs that can be constructed for a given order and number of stages.

During the preliminary investigations into numerical behaviour of RK pairs for this study, the author observed that aggressively minimizing the leading error coefficient (or attempting to find more optimal RK pairs based on other properties) often resulted in RK pairs with unexpectedly poor performance. At best, attempting to systematically construct RK pairs using information available in the literature often did not seem to be any more effective than choosing coefficients randomly. This experience highlights the importance of showing, at least for some classes of IVPs, the importance of specific values of individual PECs (2.47) to the performance of ERK pairs. Not only does this study show that it is in fact possible to minimize trial and error by systematically selecting the coefficients for ERK pairs in a well-defined way. It finally addresses an observation by Verner in the early 1990s [189, 190] that the classic characteristic numbers (2.56) seemed to be insufficient for distinguishing between ERK pairs from the same family. In fact, based on performance data in this study, the classic characteristic numbers (2.56) that have featured in many studies do not appear to be an important factor for selecting ERK pairs. This also means that, for at least some classes of IVPs, the implicit or explicit [16] assumption common in the literature that the best that can be done is treating all PECs (2.47) of equal importance can no longer be assumed.

The result of this new insight into the performance differences between ERK pairs, which otherwise seem similar, is that it will allow determining with confidence the best ERK pairs for many classes of IVP as well as confidently determining where existing pairs are inadequate. Appropriately selecting properties such as the magnitude of the leading error coefficient still remains important. However, being able to reduce the possibility of inappropriate choices and increase the confidence that the best choice has been made when selecting ERK pairs by examining the individual PECs (2.47) will prove invaluable for many applications. The performance data in Chapter 5 shows that more systematic recommendations can now be made, instead of relying on the trial and error that has commonly been used when selecting specific IVP methods.

In addition, restrictions on the full parameter space of a family, using properties such as the $\overline{\text{PEC}}$s (2.48) even if they are not under study, may be required to better study specific aspects of the construction of ERK pairs. This was seen by the effect of the value of $\hat{b}_6$ or $\hat{b}_7$ shown in Section 5.12, which was not at all obvious if the $\overline{\text{PEC}}$s (2.48) were not restricted. For any empirical study of numerical methods, restrictions on the dominant factors in performance should become commonplace if they are not the properties under study. Otherwise, it may be impossible to tell if the properties under study are the actual cause of the performance differences observed.

IVP methods, including ERK methods, are well-studied in the literature and have been the subject of many software packages and other pieces of published code. However, this study shows that a complete software system that incorporates a proper database is probably required for a complete, in-depth, and rigorous empirical study into the construction and performance of IVP methods. Because of the larger amount of performance data than is typically presented by studies into numerical methods that was required to have confidence in any observed behaviour, the author feels strongly that this study would not have been possible without building a complete software system that uses a proper database. There are many other possible empirical studies of IVP methods that would be significantly more complicated than this one and this would certainly require software similar to, and often even more complex, than what is described in thesis.

The software developed is additionally important because, based on experience with the families of $5(4)_6$ ERK pairs, every family of IVP methods studied required at least some unique consideration. The observation made during this study that the values of specific $\overline{\text{PEC}}$s (2.48) explained most of the performance variation within a specific family was surprising to the author. However, the specific observation on the importance of individual PECs may not apply to studies of other IVP methods, and it will be essential to empirically to determine the specific properties that are in fact most important. Therefore, because there may be more surprises in additional studies similar to this one, for future studies, it is necessary to have software, such as `OCSage` and `pythODE`, that can examine help examine the implications of every known property that might prove to be the dominant factor in the performance differences between IVP methods.

6.2 Future work and other research directions based on the work in this thesis

6.2.1 The requirement for more extensive non-stiff IVP test sets

Something that became obvious during this study is that for most classes of non-stiff IVPs, the current non-stiff IVP test sets are inadequate for selecting ERK pairs in a systematic and well-defined way. Other than IVPs from celestial mechanics, there are not enough representatives of any other class of IVPs in the currently published non-stiff IVP test sets. In addition, many IVPs in published test sets are relatively simple, and it is not clear that their numerical properties correspond well enough to IVPs from real-world applications for studies similar to this one. If the observation based on Figure 5.29 holds for more classes of IVPs, i.e., that more optimal pairs only emerge after the IVP is solved for a sufficient amount of problem time, then test sets incorporating IVPs that are solved for longer amounts of problem time are necessary as well. The short integration times for many IVPs in published test sets (for instance, setting the final time of all IVPs in the "non-stiff DE" test set to $t_f = 20$ [83]) are understandable because these IVP test sets originated in the 1970s and 1980s, when computer resources were more limited than they are today. However, relatively simple IVPs will remain important for connecting the empirically observed numerical behaviour of IVP methods to theory and because they have a lower computational cost for generating performance data. Therefore, for other classes of IVPs, additional study is required to determine appropriate test sets that also have a correlation in numerical performance between relatively simple IVPs and IVPs from real-world applications, analogous to how a performance correlation was seen between simple Kepler IVPs (2.13) and more complex IVPs from celestial mechanics. In order to have confidence in any conclusions reached, these new IVP test sets should contain many IVPs (the author feels this should at least a dozen) from the same class, including several IVPs that are relatively simple, and several more IVPs that directly correspond to real-world applications.

6.2.2 Future work involving the solution of ERK order conditions

Future work on the six-stage fifth-order order conditions can include finding families with singularities that were deliberately not addressed in Chapter 3, such as ERK pairs that have components of the $\mathbf{c}$ vector equal to each other. In addition, the techniques in Chapter 3 and the `OCSage` package described in Chapter 4 can be readily applied to find better solutions to systems of order conditions higher than fifth-order, as well as more complex IVP methods based on RK methods.

6.2.3 Future work on better pairs for solving IVPs from celestial mechanics and other classes of IVPs

The most obvious class of IVPs with well-defined optimal ERK pairs are the IVPs from celestial mechanics based on the Kepler IVP (2.13). This is, in part, because there are many IVPs in test sets that are based on the relatively simple Kepler IVP (2.13). The optimal value, different from the Kepler IVP (2.13), for the $\overline{\text{PEC}}$ (2.48) corresponding to the rooted tree $[\tau[[\tau^2]]]$ for the two instances of the Euler rigid body problem (see the "non-stiff B5" IVP in Figure D.1 and the subfigure labelled "Euler" in Figure D.2 that was mentioned in Section 5.3.2) is probably apparent because of the well-defined and well-studied mathematical structure of IVPs from mechanics in general. The fact that there may be different ERK pairs that are the best for rigid body problems, in comparison to celestial mechanics, may have an important application in fields such as computer graphics. However, there are many important classes of IVPs that have well-defined and well-studied mathematical structure that are all candidates for finding better application-specific ERK pairs.

6.2.4 Future development of `OCSage`, `pythODE`, and software tools in general

`OCSage` and `pythODE` are currently single-purpose software prototypes that demonstrate the possibilities of a software system for substantially enhancing the study and construction of IVP methods. Even in cases where the methods constructed are not meant for practical use, it is shown when studying the effects of $\hat{b}_6$ and $\hat{b}_7$ in Section 5.12, that using these software packages can narrow down the full spaces of free parameters and make the study of particular properties much clearer.

Careful decisions will need to be made about the future development of `OCSage` and `pythODE` so that they have adequate performance for their expected usage, but that the overall software system does not become too complex for the available development resources. In particular, to effectively study the best IVP methods for large IVPs, such as semi-discretized PDEs, the capability to handle at least several orders of magnitude more computational cost is required in comparison to what is used for the celestial mechanics IVPs studied in this thesis. However, this is not as daunting as it might seem at first because a combination of using more than just two computers, further optimization of the code, and incorporating numerical libraries written using low-level languages will easily amount to the ability to handle several orders of magnitude additional computational cost.

Software packages similar to `OCSage` and `pythODE` could form the basis for building a collection of universal reference implementations for IVP method construction and IVPs would strongly benefit the study of numerical methods for IVPs. This is because, in many publications, many of the details of both IVP method construction and the IVPs used for performance testing remain unpublished. On the one hand, this is understandable because of the difficulties and limited cost/benefit tradeoff that comes from preparing research code for publication. However, having a software platform of reference implementations for IVPs and IVP methods would greatly enhance reproducibility, and the ability of other researchers to build on each other's work.

6.2.5 Theoretical studies of the numerical behaviour observed in this thesis

Ultimately, to fully realize the potential of the systematic construction of ERK pairs based on the values of the $\overline{\text{PEC}}$s (2.48) described in Chapter 5, the specific mathematical reasons for the numerical behaviour observed should be found. Other than the cancellation of error that can occur in periodic IVPs, another hypothesis for the observed optimal values of PECs might be a non-linear analogue to dispersion or dissipation. Given the well-studied mathematical structure of the Kepler IVP (2.13), with the clues given by the performance data from Chapter 5, the specific reasons why more optimal pairs exist can probably be found. Because there are many classes of IVPs with well-studied mathematical structure, the results of a theoretical study on the numerical behaviour observed for the Kepler IVP (2.13) could be immediately be extended to other classes of IVPs.

Determining if there are any practical advantages or disadvantages to ERK pairs not satisfying the classic C(2) simplifying assumptions (2.44b) will have wait until this theoretical understanding of why certain PECs (2.47) have the largest effect on performance. In particular, without the C(2) simplifying assumptions (2.44b) the values of certain PECs (2.47) are not multiples of each other, i.e., the analogous $\overline{\text{PEC}}$s (2.48) are not equal, and it should be investigated whether this property can provide an advantage for some classes of IVPs. Some of the performance data presented in Chapter 5 shows that there may in fact sometimes be an advantage to not satisfying the C(2) simplifying assumptions (2.44b). For instance, comparing the tradeoffs between the "non-stiff D" IVPs (2.13) and more complex IVPs from celestial mechanics in Figures 5.18 and 5.20, it can be seen that pairs from Case VIb often have a more favourable tradeoff in comparison to pairs from the $5(4)_{6(6)\mathrm{C}(2)}$ family. A theoretical understanding of these observed tradeoffs that seem to favour not satisfying the C(2) simplifying assumptions (2.44b) may lead to ERK pairs with a well-defined advantage precisely because they do not satisfy the C(2) simplifying assumptions (2.44b).

6.2.6 Computational studies using specialized `pythODE` modules

Specialized `pythODE` modules could be written to study virtually any numerical behaviour on a step-by-step basis. For instance, if used in combination with the generated code from `OCSage`, it is feasible to write specialized `pythODE` modules to track the contribution of each PEC (2.47) to the error of the IVP solution at each step. This would have an extremely high computational cost in comparison to just solving the IVPs under consideration. However, it would provide valuable information to help determine whether certain values of the $\overline{\text{PEC}}$s (2.48) promote a cancellation of error, under what circumstances, or if another phenomenon is responsible.

6.2.7 Futher study of error estimation and step control

From Section 5.10, there appear to be unavoidable tradeoffs in the performance between individual IVPs based on the specific value selected for $\hat{b}_6$ or $\hat{b}_7$. This was also observed when solving many different classes of IVPs, where "oversized" error estimates often favoured IVPs that require large stepsize changes and more "accurate" error estimates favoured those that do not. Therefore, study of better step-control schemes may lead to better overall performance for embedded ERK pairs. Considering that control-theoretic studies of IVP methods do not yet draw on much of the enormous literature on control theory, the improvement could be quite substantial. Once the effects of different values of the $\overline{\text{PEC}}$s of the fifth-order component of a $5(4)_6$ ERK pair are removed, in Figures 5.42–5.43 the behaviour $\hat{b}_6$ and $\hat{b}_7$ look remarkably like those observed when solving linear IVPs. This also means that if step-control can be better analyzed for linear IVPs, then it can probably be analyzed for simple non-linear IVPs such as the Kepler IVP (2.13).

6.2.8 Controlling roundoff error

Although performance issues that appeared to be due to roundoff error were occasionally observed during this study, they are not significant overall in comparison to the local and global errors of the IVP method. However, because roundoff error can become substantial for many applications when scaling beyond the relatively simple IVPs in many test sets, it is important to choose pairs that reduce roundoff. Although spacing the components of the $\mathbf{c}$ vector sufficiently far apart and reducing the magnitude of D characteristic number (2.56c) have been considered in the literature as methodologies of reducing roundoff error, no significant effects were seen from applying these particular methodologies in this study. A more detailed study of the effects of coefficient choice on roundoff error needs to be conducted. In particular, different strategies for summation have been described for IVP methods in the literature, and consideration of these different implementation strategies for RK formulae should be incorporated into the construction of ERK pairs [54].

6.2.9 Are "cheap" error estimates actually cheap?

One conclusion from the method construction in this study is that embedded ERK pairs are not necessarily as "cheap" as they have been portrayed. When constructing a six-stage fifth-order ERK method, incorporating an embedded pair for giving an error estimate generally requires giving up several free parameters. This either leads to having less flexibility in terms of choosing the error coefficients for $5(4)_{6(6)}$ ERK pairs or giving up fewer free parameters but having an extra RHS evaluation when a step is rejected for $5(4)_{6(7)}$ ERK pairs. The step-doubling error estimate defined in Section 2.6.1 should be considered, although the author thinks that it is not likely that step-doubling will actually provide enough of an advantage over embedded pairs to compensate for the 50% increase in RHS evaluations. Following a similar pattern to the eighth-order ERK method with fifth- and third-order ERK methods as error estimators from Dormand and Prince [72, pgs.254–255], a second- and third-order error estimate used with a six-stage fifth-order ERK method might provide an advantage by allowing more free parameters for the fifth-order ERK component. Another alternative is to consider seven-stage fifth-order pairs, where the many free parameters the complete solution will provide might give an advantage that offsets the computational cost of an extra stage.

6.2.10 Are there specific advantages to lower-order IVP methods?

Lower-order IVP methods remain in common use, especially for semi-discretized PDEs, despite some authors calling for more common usage of high-order numerical methods, with the expectation that this will achieve better accuracy and efficiency [36, 74, 169, 198, 199]. It is clearly seen from the limited experimentation with the Brusselator IVP in Section 5.6 that there may in fact be some justification for continued use of low-order IVP methods. Large amounts of performance data may be crucial to either identifying the advantages that might be inherent to lower-order IVP methods or finding specific improvements that must be made to higher-order IVP methods for them to be competitive at coarse accuracies. The narrower range of stability for step-control when using higher-order methods, which is briefly described in Section 2.6.2, is one possibility. However, lower-order IVP methods remain in use for applications where there is no step-control as well. Therefore, detailed investigation in conjunction with performance data will be required to test any hypotheses.

6.2.11 A study of other properties of IVP methods

Properties and characteristic numbers for which performance testing did not show a well-defined effect are generally not discussed within this thesis. These include many properties that have been studied in the literature, such as the linear stability region, the differences in stability regions between components of an embedded RK pair, dissipation, dispersion, contractivity, etc. [4, 82, 94, 160, 179] All of these properties should be studied in detail using the large amounts of performance data now available from `OCSage` and `pythODE`. In particular, the author hypothesizes that for stiff IVP methods, classic properties like linear stability are necessary but not sufficient for constructing the most efficient IVP methods and that properties such as the values of individual PECs may become important. However, future studies of more complex numerical methods, such as those for stiff IVPs, will have to account for effects such as the ones that might occur from coupling between the integration methods and the numerical methods used to solve the systems of linear or non-linear equations.

6.2.12 Fitness functions for constructing IVPs using meta-heuristic search and optimization methods

The families constructed from the six-stage fifth-order order conditions are complex enough to support new work and insights into IVP methods, but simple enough to be practical for simple techniques involving grid searches and performance testing a smaller random set of the pairs found. Many families of IVP methods, especially combinations of more basic numerical methods, have many more free parameters. Therefore, techniques must be found to handle method construction involving the spaces of free parameters with more than four or five free parameters. Many of the contemporary optimization methodologies based on meta-heuristics, such as particle swarm [52, 144], may be ideal for this.

However, because efficient ERK method construction does not appear to be a straightforward and well-defined minimization problem, some modifications to how meta-heuristics are often used may be necessary. The author's experience early in this study showed that published information on selecting coefficients for RK methods is insufficient to construct a well-defined *fitness function*. In many cases, the author found RK methods that had seemingly similar properties but significantly different performance solving many IVPs.

The fitness function implicitly used in Chapter 5 is an average of the performances of the Kepler IVP (2.13) with different eccentricities and accuracies. This is justified because most of the more complex IVPs tested from celestial mechanics usually had the performance solving them strongly correlated with the performance solving at least one instance of the tested Kepler IVPs (2.13). Any practical and useful fitness function will likely require a combination of characteristic numbers and performance testing. This makes the software system consisting of `OCSage` and `pythODE` ideal for the evaluation of the fitness functions that meta-heuristic optimization methods would require. In order to find well-defined fitness functions for constructing RK methods, developing additional test sets for other important classes of IVPs will be essential as well.

References

[1] R. Alexander. Diagonally implicit Runge–Kutta methods for stiff ODEs. *SIAM J Numer Anal*, 14(6):1006–1021, Dec 1977.

[2] M. Alexe and A. Sandu. Forward and adjoint sensitivity analysis with continuous explicit Runge–Kutta schemes. *Appl Math Comput*, 208(2):328–346, 2009.

[3] Y.M. Altman. *Undocumented Secrets of MATLAB–Java Programming.* Chapman & Hall/CRC, 2011.

[4] Z.A. Anastassi and T.E. Simos. An optimized Runge–Kutta method for the solution of orbital problems. *J Comput Appl Math*, 175(1):1–9, 2005. Selected Papers of the International Conference on Computational Methods in Sciences and Engineering.

[5] D.P. Anderson. BOINC: A system for public-resource computing and storage. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, GRID '04, pages 4–10, Washington, DC, USA, 2004. IEEE Computer Society.

[6] H. Anton. *Elementary linear algebra.* John Wiley & Sons, 8th edition, 2000.

[7] R.F. Arenstorf. Existence of periodic solutions passing near both masses of the restricted three-body problem. *AIAA Journal*, 1:238–240, 1963.

[8] E. Arge, A.M. Bruaset, and H.P. Langtangen. *Object-oriented Numerics*, pages 7–26. Birkhäuser, 1997.

[9] H. Arndt, M. Bundschus, and A. Naegele. Towards a next-generation matrix library for Java. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 1, pages 460–467, Seattle, July 2009. IEEE.

[10] U.M. Ascher and L.R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations.* SIAM, 1998.

[11] U.M. Ascher, S.J. Ruuth, and R.J. Spiteri. Implicit-explicit Runge–Kutta methods for time-dependent partial differential equations. *Appl Numer Math*, 25(2-3):151–167, 1997. Special issue on time integration (Amsterdam, 1996).

[12] K.J. Astrom and R.M. Murray. *Feedback Systems: An Introduction for Scientists and Engineers.* Princeton University Press, 2008.

[13] J.O. Attia. *Electronics and Circuit Analysis Using MATLAB.* CRC Press, Inc., 1999.

[14] C. Bendtsen. A Maple V package for the analysis and construction of Runge–Kutta methods. Technical report, The Danish Computer Centre for Research and Education, 1996.

[15] P. Bogacki and L.F. Shampine. A 3(2) pair of Runge–Kutta formulas. *Appl Math Lett*, 2(4):321–325, 1989.

[16] P. Bogacki and L.F. Shampine. An efficient Runge–Kutta $(4, 5)$ pair. *Comput Math Appl*, 32(6):15–28, 1996.

[17] R.F. Boisvert, J.R. Rice, and E.N. Houstis. A system for performance evaluation of partial differential equations software. *IEEE T Software Eng*, 5(4):418 – 425, 1978.

[18] F. Bornemann. Runge–Kutta methods, trees, and Maple. *Selçuk J Appl Math*, 2(1):3–15, 2001.

[19] F. Bornemann. Runge–Kutta methods, trees, and Mathematica. *ArXiv Mathematics e-prints*, 2002.

[20] S. Boscarino. Error analysis of IMEX Runge–Kutta methods derived from differential-algebraic systems. *SIAM J Numer Anal*, 45(4):1600–1621, 2007.

[21] S. Boscarino. On an accurate third-order implicit-explicit Runge–Kutta method for stiff problems. *Appl Numer Math*, 59(7):1515–1528, 2009.

[22] A. Boulesteix. Over-optimism in bioinformatics research. *Bioinformatics*, 26(3):437–439, February 2010.

[23] P.N. Brown, G.D. Byrne, and A.C. Hindmarsh. VODE: a variable-coefficient ODE solver. *SIAM J Sci Stat Comput*, 10(5):1038–1051, 1989.

[24] J.C. Butcher. Coefficients for the study of Runge–Kutta integration processes. *J Austral Math Soc*, 3:185–201, 1963.

[25] J.C. Butcher. On Runge–Kutta processes of high order. *J Austral Math Soc*, 4:179–194, 1964.

[26] J.C. Butcher. On the attainable order of Runge–Kutta methods. *Math Comp*, 19:408–417, 1965.

[27] J.C. Butcher. *The numerical analysis of ordinary differential equations*. John Wiley & Sons Ltd., 2003.

[28] F. Cameron. A MATLAB package for automatically generating Runge–Kutta trees, order conditions, and truncation error coefficients. *ACM Trans Math Softw*, 32(2):274–298, 2006.

[29] M.H. Carpenter, C.A. Kennedy, H. Bijl, S.A. Viken, and V.N. Vatsa. Fourth-order Runge–Kutta schemes for fluid mechanics applications. *J Sci Comput*, 25(1):157–194, 2005.

[30] J.R. Cash and A.H. Karp. A variable order Runge–Kutta method for initial value problems with rapidly varying right-hand sides. *ACM Trans Math Softw*, 16(3):201–222, 1990.

[31] C.R. Cassity. Solutions of the fifth-order Runge–Kutta equations. *SIAM J Numer Anal*, 3:598–606, 1966.

[32] C.R. Cassity. The complete solution of the fifth-order Runge–Kutta equations. *SIAM J Numer Anal*, 6:432–436, 1969.

[33] C.R. Cassity and C. Steinkopff. Optimization of parameters for the complete solution of the fifth-order Runge–Kutta equations. *SIAM Rev*, 11(1):109–110, 1969. SIAM 1968 Fall Meeting.

[34] D. Cavaglieri and T. Bewley. Low-storage implicit/explicit Runge–Kutta schemes for the simulation of stiff high-dimensional ODE systems. *J Comput Phys*, 286:172–193, 2015.

[35] S.V. Chekanov. *Numeric Computation and Statistical Data Analysis on the Java Platform*. Springer Publishing Company, Incorporated, 1st edition, 2016.

[36] J. Chen. High-order time discretizations in seismic modeling. *Geophysics*, 72(5):SM115–SM122, 2007.

[37] A.J. Christlieb, C.B. Macdonald, B.W. Ong, and R.J. Spiteri. Revisionist integral deferred correction with adaptive step-size control. *Comm App Math Comp Sc*, 2015.

[38] N.S. Clerman and W. Spector. *Modern Fortran: Style and Usage*. Cambridge University Press, 2011.

[39] S.D. Cohen and A.C. Hindmarsh. CVODE, a stiff/nonstiff ODE solver in C. *Comput Phys*, 10(2):138–143, March 1996.

[40] G. Corliss and Y.F. Chang. Solving ordinary differential equations using Taylor series. *ACM Trans Math Software*, 8(2):114–144, 1982.

[41] D.A Cox, J. Little, and D. O'Shea. *Using algebraic geometry.* Springer, second edition, 2006.

[42] T. Crick, B.A. Hall, and S. Ishtiaq. "Can I implement your algorithm?": A model for reproducible research software. *CoRR*, abs/1407.5981, 2014.

[43] C.J. Date. *Relational Theory for Computer Professionals: What Relational Databases Are Really All About.* O'Reilly Media, 2015.

[44] C.J. Date. *SQL and relational theory: how to write accurate SQL code.* O'Reilly Media, third edition, 2015.

[45] D.L. Donoho, A. Maleki, I.U. Rahman, M. Shahram, and V. Stodden. Reproducible research in computational harmonic analysis. *Comput Sci Eng*, 11(1):8–18, Jan 2009.

[46] J.R. Dormand and P.J. Prince. New Runge–Kutta algorithms for numerical simulation in dynamical astronomy. *Celestial Mech*, 18(3):223–232, 1978.

[47] J.R. Dormand and P.J. Prince. A family of embedded Runge–Kutta formulae. *J Comput Appl Math*, 6(1):19–26, 1980.

[48] J.R. Dormand and P.J. Prince. A reconsideration of some embedded Runge–Kutta formulae. *J Comput Appl Math*, 15(2):203–211, 1986.

[49] J.R. Dormand and P.J. Prince. Runge–Kutta triples. *Comput Math Appl*, 12(9):1007–1017, 1986.

[50] J.R. Dormand and P.J. Prince. Practical Runge–Kutta processes. *SIAM J Sci Statist Comput*, 10(5):977–989, 1989.

[51] S. Dupal and M. Yoshizawa. REU numerical analysis project: design and optimziation of explicit Runge–Kutta formulas, 2007.

[52] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, Oct 1995.

[53] S. Elnashaie, F. Uhlig, and C. Affane. *Numerical techniques for chemical and biological engineers using MATLAB A simple bifurication approach.* Springer, 2007.

[54] W. H. Enright and W. B. Hayes. Robust and reliable defect control for Runge–Kutta methods. *ACM Trans Math Softw*, 33(1), 2007.

[55] W.H. Enright, T.E. Hull, and B. Lindberg. Comparing numerical methods for stiff systems of ODEs. *BIT*, 15(2):10–48, 1975.

[56] W.H. Enright and J.D. Pryce. Two FORTRAN packages for assessing initial value methods. *ACM Trans Math Softw*, 13(1):1–27, 1987.

[57] B. Erocal and W. Stein. The Sage project: Unifying free mathematical software to create a viable alternative to Magma, Maple, Mathematica and MATLAB. In *Mathematical Software - ICMS 2010*, volume 6327 of *Lecture Notes in Computer Science*, pages 12–27. Springer Berlin/Heidelberg, 2010.

[58] I.Th. Famelis, S.N. Papakostas, and Ch. Tsitouras. Symbolic derivation of Runge–Kutta order conditions. *J Symb Comput*, 37(3):311–327, 2004.

[59] T. Feagin. High-order explicit Runge–Kutta methods, 2017. http://sce.uhcl.edu/rungekutta/.

[60] E. Fehlberg. New high-order Runge–Kutta formulas with an arbitrarily small truncation error. *Z Angew Math Mech*, 46:1–16, 1966.

[61] E. Fehlberg. Low-order classical Runge–Kutta formulas with stepsize control and their application to some heat transfer problems. Technical report, NASA, 1969.

[62] M.T. Fletcher. Discovery and optimization of low-storage Runge–Kutta methods. Master's thesis, Naval Postgraduate school, Monterey California, 2015.

[63] E. Gabriel, G.E. Fagg, G. Bosilca, et al. Open MPI: Goals, concept, and design of a next generation MPI implementation. In D. Kranzlmüller, P. Kacsuk, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19 - 22, 2004. Proceedings*, pages 97–104, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[64] J. Gray, D.T. Liu, M. Nieto-Santisteban, A. Szalay, D.J. DeWitt, and G. Heber. Scientific data management in the coming decade. *SIGMOD Rec*, 34(4):34–41, December 2005.

[65] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 2014.

[66] K. Gustafsson. Control-theoretic techniques for stepsize selection in explicit Runge–Kutta methods. *ACM Trans Math Software*, 17(4):533–554, 1991.

[67] K. Gustafsson, M. Lundh, and G. Söderlind. A PI stepsize control for the numerical solution of ordinary differential equations. *BIT*, 28(2):270–287, 1988.

[68] M.A. Haidekker. *Linear Feedback Controls: The Essentials*. Elsevier, 2013.

[69] E. Hairer. A Runge–Kutta method of order 10. *J Inst Math Appl*, 21(1):47–59, 1978.

[70] E. Hairer. Origin of the term elementary differentials. Personal communciation, August 2009.

[71] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: Nonstiff problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, first edition, 1987.

[72] E. Hairer, S.P. Nørsett, and G. Wanner. *Solving ordinary differential equations I: Nonstiff problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer-Verlag, second edition, 1993.

[73] E. Hairer and G. Wanner. *Solving ordinary differential equations II: Stiff and differential-algebraic problems*, volume 14 of *Springer Series in Computational Mathematics*. Springer-Verlag, 1996.

[74] J.S. Hesthaven. High-order accurate methods in time-domain computational electromagnetics: A review. In P.W. Hawkes, editor, *Advances in Imaging and Electron Physics*, volume 127, pages 59–123. Elsevier, 2003.

[75] T. Hey and A.E. Trefethen. *The Data Deluge: An e-Science Perspective*, pages 809–824. John Wiley & Sons, Ltd, 2003.

[76] T. Hey and A.E. Trefethen. Cyberinfrastructure for e-science. *Science*, 308(5723):817–821, 2005.

[77] D.J. Higham. Global error versus tolerance for explicit Runge–Kutta methods. *IMA J Numer Anal*, 11(4):457–480, 1991.

[78] D.J. Higham and L.N. Trefethen. Stiffness of ODEs. *BIT*, 33(2):285–303, 1993.

[79] A.C. Hindmarsh. ODEPACK, a systematized collection of ODE solvers. *Sci Comput*, 1:55–64, 1983.

[80] A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, and C.S. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans Math Softw*, 31(3):363–396, 2005.

[81] M.E. Hosea and L.F. Shampine. Efficiency comparisons of methods for integrating ODEs. *Comput Math Appl*, 28(6):45 – 55, 1994.

[82] F.Q. Hu, M.Y. Hussaini, and J.L. Manthey. Low-dissipation and low-dispersion Runge–Kutta schemes for computational acoustics. *J Comput Phys*, 124(1):177–191, 1996.

[83] T.E. Hull, W.H. Enright, B.M. Fellen, and A.E. Sedgwick. Comparing numerical methods for ordinary differential equations. *SIAM J Num Anal*, 9(4):603–607, 1972.

[84] T.E. Hull, W.H. Enright, and K.R. Jackson. Runge–Kutta research at Toronto. *Appl Num Math*, 22(1-3):225–236, 1996.

[85] T.E. Hull and R.L. Johnston. Optimum Runge–Kutta methods. *Math Comp*, 18:306–310, 1964.

[86] W. Hundsdorfer and J.G. Verwer. *Numerical solution of time-dependent advection-diffusion-reaction equations*. Springer-Verlag, 2003.

[87] J.D. Hunter. Matplotlib: A 2D graphics environment. *Comput Sci Eng*, 9(3):90–95, 2007.

[88] M. Hutson. Artificial intelligence faces reproducibility crisis. *Science*, 359(6377):725–726, 2018.

[89] Z. Ivezic, A.J. Connolly, J.T. VanderPlas, and A. Gray. *Statistics, Data Mining, and Machine Learning in Astronomy: A Practical Python Guide for the Analysis of Survey Data*. Princeton University Press, 2014.

[90] D.S. Johnson. A theoretician's guide to the experimental analysis of algorithms. In *Data structures, near neighbor searches, and methodology: Fifth and Sixth DIMACS Implementation Challenges Piscataway, NJ, (1996/Baltimore, MD, 1999)*, volume 59 of *DIMACS Ser Discrete Math Theoret Comput Sci*, pages 215–250. Amer Math Soc, Providence, RI, 2002.

[91] E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. http://www.scipy.org/.

[92] S. Juba, A. Vannahme, and A. Volkov. *Learning PostgreSQL*. Packt Publishing, 2015.

[93] N. Kajler and N. Soiffer. A survey of user interfaces for computer algebra systems. *J Symb Comput*, 25(2):127–159, Feb 1998.

[94] C.A. Kennedy and M.H. Carpenter. Additive Runge–Kutta schemes for convection-diffusion-reaction equations. Technical report, NASA, 2001.

[95] C.A. Kennedy and M.H. Carpenter. Diagonally implicit Runge–Kutta method for ordinary differential equations. a review. Technical report, NASA, 2016.

[96] C.A. Kennedy, M.H. Carpenter, and M.R. Lewis. Low-storage, explicit Runge–Kutta schemes for the compressible Navier–Stokes equations. Technical report, NASA, 1999.

[97] D.I. Ketcheson and A.J. Ahmadia. Optimal stability polynomials for numerical integration of initial value problems. *Commun Appl Math Comput Sci*, 7(2):247–271, 2012.

[98] D.I. Ketcheson, K. Mandli, A.J. Ahmadia, A. Alghamdi, M. Quezada de Luna, M. Parsani, M.G. Knepley, and M. Emmett. PyClaw: Accessible, extensible, scalable tools for wave propagation problems. *SIAM J Sci Comp*, 34(4):C210–C231, 2012.

[99] S. Khashin. A symbolic-numeric approach to the solution of the Butcher equations. *Canad Appl Math Quart*, 17(3):555–569, 2010.

[100] R.D. King, J. Rowland, S.G. Oliver, et al. The automation of science. *Science*, 324(5923):85–89, 2009.

[101] P. Knoll and S. Mirzaei. Scientific computing with Java. *Comput Appl Eng Educ*, 18(3):495–501, 2010.

[102] H.P. Konen and H.A. Luther. Some singular explicit fifth-order Runge–Kutta solutions. *SIAM J Numer Anal*, 4:607–619, 1967.

[103] A. Kroshko. Integrating-factor-based 2-additive Runge–Kutta methods for advection-reaction-diffusion equations. Master's thesis, University of Saskatchewan, 2011.

[104] A. Kroshko, O. Sharomi, A.B. Gumel, and R.J. Spiteri. Design and analysis of NSFD methods for the diffusion-free Brusselator. In Abba Gumel, editor, *Mathematics of Continuous and Discrete Dynamical Systems*, volume 618 of *Contemporary Mathematics*. AMS, 2014.

[105] A. Kroshko and R.J. Spiteri. Efficient SIMD solution of multiple systems of stiff IVPs. *J Comput Sci*, 4(5):377–385, 2012.

[106] A. Kroshko and R.J. Spiteri. odeToJava: A PSE for the numerical solution of IVPs. *ACM Trans Math Softw*, 41(3):17:1–17:33, 2015.

[107] E.J. Kubatko, B.A. Yeager, and D.I. Ketcheson. Optimal strong-stability-preserving Runge–Kutta time discretizations for discontinuous galerkin methods. *J Sci Comput*, 60(2):313–344, Aug 2014.

[108] J.D. Lambert. *Numerical methods for ordinary differential systems: The initial value problem.* John Wiley & Sons Ltd., 1991.

[109] V.J. Law. *Numerical methods for chemical engineers Using Excel, VBA, and MATLAB.* CRC Press, 2013.

[110] B. Leimkuhler and S. Reich. *Simulating Hamiltonian dynamics.* Cambridge University Press, 2004.

[111] R.J. LeVeque. Python tools for reproducible research on hyperbolic problems. *Comput Sci Eng*, 11(1):19–27, January 2009.

[112] R.J. LeVeque and D.G. Crighton. *Finite volume methods for hyperbolic problems.* Cambridge texts in applied mathematics. Cambridge Univ. Press, 2002.

[113] H.A. Luther and H.P. Konen. Some fifth-order classical Runge–Kutta formulas. *SIAM Rev*, 7:551–558, 1965.

[114] F. Mazzia, J.R. Cash, and K.E.R. Soetaert. A test set for stiff initial value problem solvers in the open source software R: Package deTestSet. *J Comp Appl Math*, 236(16):4119–4131, 2012.

[115] F. Mazzia and C. Magherini. Test set for initial value problem solvers, release 2.4. Technical Report 4, Department of Mathematics, University of Bari, Italy, 2008.

[116] M.M. McKerns, L. Strand, T. Sullivan, A. Fang, and M.A.G. Aivazis. Building a framework for predictive science, 2011. http://arxiv.org/abs/1202.1056.

[117] R.E. Mickens and A.B. Gumel. Numerical study of a non-standard finite-difference scheme for the van der Pol equation. *J Sound Vibration*, 250(5):955–963, 2002.

[118] G.R. Mirams, C.J. Arthurs, M.O. Bernabeu, et al. Chaste: An open source C++ library for computational physiology and biology. *PLOS Comput Biol*, 9(3):1–8, 03 2013.

[119] A.P. Mishina and I.V. Proskuryakov. *Higher algebra: linear algebra, polynomials, general algebra.* Pergamon Press, 1965.

[120] K.W. Morton and D.F. Mayers. *Numerical solution of partial differential equations.* Cambridge University Press, 1995.

[121] J. Moses. Macsyma: A personal history. *J Symb Comput*, 47(2):123–130, Feb 2012.

[122] X.X. Newhall, E.M. Standish, and J.G. Williams. DE 102 - a numerically integrated ephemeris of the moon and planets spanning forty-four centuries. *Astron Astrophys*, 125:150–167, 1983.

[123] J. Niegemann, R. Diehl, and K. Busch. Efficient low-storage Runge–Kutta schemes with optimized stability regions. *J Comp Phys*, 231(2):364–372, 2012.

[124] U. Nowak and S. Gebauer. A new test frame for ordinary differential equation solvers. Technical report, Konrad-Zuse-Zentrum für Informationstechnik, 1988.

[125] T.E. Oliphant. Python for scientific computing. *Comput Sci Eng*, 9(3):10 –20, May–June 2007.

[126] T.E. Oliphant. *Guide to NumPy*. CreateSpace Independent Publishing Platform, 2nd edition, 2015.

[127] H. Owhadi, C. Scovel, T.J. Sullivan, M. McKerns, and M. Ortiz. Optimal uncertainty quantification. *SIAM Review*, 55(2):271–345, 2013.

[128] S.N. Papakostas and G. Papageorgiou. A family of fifth-order Runge–Kutta pairs. *Math Comput*, 65(215):1165–1181, 1996.

[129] M. Parsani, D.I. Ketcheson, and W. Deconinck. Optimized low-order explicit Runge–Kutta schemes for high-order spectral difference method. In *Proceedings of the 11th Finnish Mechanics Days*. University of Oulu, 2012.

[130] M. Parsani, D.I. Ketcheson, and W. Deconinck. Optimized explicit Runge–Kutta schemes for the spectral difference method applied to wave propagation problems. *SIAM J Sci Comput*, 35(2):A957–A986, 2013.

[131] M. Patterson. Implementing Runge–Kutta solvers in Java. Bachelor's thesis, Acadia University, April 2003.

[132] F. Pérez and B.E. Granger. iPython: A system for interactive scientific computing. *Comput Sci Eng*, 9(3):21–29, 2007.

[133] D. Petcu and M. Drăgan. Designing an ODE solving environment. In *Advances in Software Tools for Scientific Computing*, pages 89–131. Springer, Berlin, 2000.

[134] J.D. Pintér. *Computational Global Optimization in Nonlinear Systems An Interactive Tutorial*. Lionheart publishing, 2009.

[135] A. Preuss. A study of additive splitting methods for advection-reaction-diffusion equations. Master's thesis, University of Saskatchewan, January 2014.

[136] P.J. Prince and J.R. Dormand. High order embedded Runge–Kutta formulae. *J Comput Appl Math*, 7(1):67–75, 1981.

[137] K. Radhakrishnan and A.C. Hindmarsh. Description and use of LSODE, the Livemore solver for ordinary differential equations. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA, 1993.

[138] L.B. Rall and G.F. Corliss. An introduction to automatic differentiation. In C.H. Bischof M. Berz and G.F. Corliss, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 1–18. SIAM, 1996.

[139] A. Ralston. Runge–Kutta methods with minimum error bounds. *Math Comp*, 16(80):431–437, 1962.

[140] L. Ramalho. *Fluent Python*. O'Reilly Media, Inc., 1st edition, 2015.

[141] A.H. Register. *A Guide to MATLAB Object-Oriented Programming*. Scitech Pub Inc, 2007.

[142] B. Sanderse and B. Koren. Accuracy analysis of explicit Runge–Kutta methods applied to the incompressible Navier–Stokes equations. *J Comp Phys*, 231(8):3041–3063, Apr 2012.

[143] W.E. Schiesser and G.W. Griffiths. *A Compendium of Partial Differential Equation Models: Method of Lines Analysis with Matlab*. Cambridge University Press, 1 edition, 2009.

[144] T. Schöene. Step-optimized particle swarm optimization. Master's thesis, University of Saskatchewan, 2011.

[145] L.F. Shampine. Local error estimation by doubling. *Computing*, 34(2):179–190, 1985.

[146] L.F. Shampine. Some practical Runge–Kutta formulas. *Math Comp*, 46(173):135–150, 1986.

[147] L.F. Shampine. *Numerical solution of ordinary differential equations*. Chapman & Hall, 1994.

[148] L.F. Shampine. Error estimation and control for ODEs. *J Sci Comput*, 25(1-2):3–16, 2005.

[149] L.F. Shampine. Design of software for ODEs. *J Comput Appl Math.*, 205(2):901–911, 2007.

[150] L.F. Shampine, R.C. Allen, and S. Pruess. *Fundamentals of Numerical Computing*. John Wiley & Sons, Inc., 1997.

[151] L.F. Shampine and R.M. Corless. Initial value problems for ODEs in problem solving environments. *J Comput Appl Math*, 125(1–2):31–40, 2000.

[152] L.F. Shampine, I. Gladwell, and S. Thompson. *Solving ODEs with MATLAB*. Cambridge University Press, 2003.

[153] L.F. Shampine and M.W. Reichelt. The MATLAB ODE suite. *SIAM J Sci Comput*, 18(1):1–22, 1997. Dedicated to C. William Gear on the occasion of his 60th birthday.

[154] L.F. Shampine and H.A. Watts. Global error estimation for ordinary differential equations. *ACM Trans Math Software*, 2(2):172–186, 1976.

[155] P.W. Sharp. Numerical comparisons of some explicit Runge–Kutta pairs of orders 4 through 8. *ACM Trans Math Softw*, 17(3):387–409, 1991.

[156] P.W. Sharp. High order explicit Runge–Kutta pairs for ephemerides of the solar system and the moon. *J Appl Math and Decision Sci*, 4(2):183–192, 2000.

[157] P.W. Sharp. N-body simulations: the performance of eleven integrators. Technical Report 506, Department of Mathematics- University of Auckland, 2003. https://researchspace.auckland.ac.nz/handle/2292/5119.

[158] P.W. Sharp. Comparisons of integrators on a diverse collection of restricted three-body test problems. *IMA J Numer Anal*, 24(4):557–575, 2004.

[159] P.W. Sharp. N-body simulations: The performance of some integrators. *ACM Trans Math Softw*, 32(3):375–395, 2006.

[160] P.W. Sharp, J.C. Castillo-Rogez, and K.R. Grazier. The performance of phase-lag enhanced explicit Runge–Kutta Nyström pairs on N-body problems. *J Comput Appl Math*, 236(9):2378 – 2386, 2012.

[161] P.W. Sharp and E. Smart. Explicit Runge–Kutta pairs with one more derivative evaluation than the minimum. *SIAM J Sci Comput*, 14(2):338–348, 1993.

[162] P.W. Sharp and J.H. Verner. Completely imbedded Runge–Kutta pairs. *SIAM J Numer Anal*, 31(4):1169–1190, 1994.

[163] G.E. Shilov. *Linear algebra*. Dover Publications Inc., English edition, 1977. Translated from the Russian and edited by Richard A. Silverman.

[164] G. Söderlind. Automatic control and adaptive time-stepping. *Numer Algorithms*, 31(1):281–310, 2002.

[165] G. Söderlind, L. Jay, and M. Calvo. Stiffness 1952–2012: Sixty years in search of a definition. *BIT Numerical Mathematics*, 55(2):531–558, Jun 2015.

[166] G. Söderlind and L. Wang. Evaluating numerical ODE/DAE methods, algorithms and software. *J Comput Appl Math*, 185(2):244–260, 2006.

[167] K. Soetaert, J. Cash, and F. Mazzia. *Solving Differential Equations in R*. Springer, 2012.

[168] M. Sofroniou. Symbolic derivation of Runge–Kutta methods. *J Symb Comput*, 18(3):265–296, 1994.

[169] H. Spachmann, R. Schuhmann, and T. Weiland. Higher order explicit time integration schemes for maxwell's equations. *Int J Numer Model El*, 15(5–6):419–437, 2002.

[170] C. Spiegel. *PEM Fuel Cell Modeling and Simulation Using MATLAB*. Academic Press, 2008.

[171] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*, volume 12 of *Texts in Applied Mathematics*. Springer-Verlag, third edition, 2002. Translated from the German by R. Bartels, W. Gautschi and C. Witzgall.

[172] A. Stuart and A.R. Humphries. *Dynamical systems and numerical analysis*. Cambridge University Press, 1998.

[173] V. Szebehely. *Theory of orbits: The restricted problem of three bodies*. Academic press, 1967.

[174] V.G Szebehely and H. Mark. *Adventures in celestial mechanics*. Wiley-VCH, second edition, 2004.

[175] G.L. Taboada, J. Touriño, and R. Doallo. Java for high performance computing: assessment of current research and practice. In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*, PPPJ '09, pages 30–39, New York, NY, USA, 2009. ACM.

[176] The Astropy Collaboration, T.P. Robitaille, E.J. Tollerud, et al. Astropy: A community Python package for astronomy. *A&A*, 558:A33, 2013.

[177] M. Torrilhon and R. Jeltsch. Essentially optimal explicit Runge–Kutta methods with application to hyperbolic–parabolic equations. *Numer Math*, 106(2):303–334, 2007.

[178] L.N. Trefethen. *Spectral Methods in MATLAB*. SIAM, 2000.

[179] K. Tselios and T.E. Simos. Runge–Kutta methods with minimal dispersion and dissipation for problems arising from computational acoustics. *J Comput Appl Math*, 175(1):173–181, 2005. Selected Papers of the International Conference on Computational Methods in Sciences and Engineering.

[180] Ch. Tsitouras. Optimized explicit Runge–Kutta pair of orders 9(8). *Appl Numer Math*, 38(1–2):123–134, 2001.

[181] Ch. Tsitouras. Runge–Kutta interpolants for high precision computations. *Numer Algorithms*, 44(3):291–307, Jun 2007.

[182] Ch. Tsitouras. Runge–Kutta pairs of orders 5(4) using the minimal set of simplifying assumptions. In T.E. Simos, G. Psihoyios, and Ch. Tsitouras, editors, *American Institute of Physics Conference Series*, volume 1168 of *American Institute of Physics Conference Series*, pages 69–72, 2009.

[183] Ch. Tsitouras. Runge–Kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Comput Math Appl*, 62(2):770–775, 2011.

[184] Ch. Tsitouras, I.Th. Famelis, and T.E. Simos. On modified Runge–Kutta trees and methods. *Comput Math Appl*, 62(4):2101–2111, 2011.

[185] S. van der Walt, S.C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Comput Sci Eng*, 13(2):22–30, 2011.

[186] P. Vandewalle, J. Kovacevic, and M. Vetterli. Reproducible research in signal processing. *IEEE Signal Proc Mag*, 26(3):37–47, May 2009.

[187] J.H. Verner. Explicit Runge–Kutta methods with estimates of the local truncation error. *SIAM J Numer Anal*, 15(4):772–790, 1978.

[188] J.H. Verner. Families of imbedded Runge–Kutta methods. *SIAM J Numer Anal*, 16(5):857–875, 1979.

[189] J.H. Verner. A contrast of some Runge–Kutta formula pairs. *SIAM J Numer Anal*, 27(5):1332–1344, 1990.

[190] J.H. Verner. Some Runge–Kutta formula pairs. *SIAM J Numer Anal*, 28(2):496–511, 1991.

[191] J.H. Verner. A classification scheme for studying explicit Runge–Kutta pairs. Technical report, Simon Fraser University, 1992.

[192] J.H. Verner. Strategies for deriving new explicit Runge–Kutta pairs. *Ann Num Math*, 1:225–244, 1994.

[193] J.H. Verner. High-order explicit Runge–Kutta pairs with low stage order. *Appl Numer Math*, 22(1–3):345–357, 1996.

[194] J.H. Verner. Numerically optimal Runge–Kutta pairs with interpolants. *Numer Algor*, 53(2-3):383–396, 2010.

[195] J.H. Verner. Explicit Runge–Kutta pairs with lower stage-order. *Numer Algor*, 65(3):555–577, 2014.

[196] S. Voightmann. *General linear methods for integrated circuit design*. PhD thesis, Humboldt-Universitat zu Berlin, 2006.

[197] R. Wang and R. Spiteri. Linear instability of the fifth-order WENO method. *SIAM J Numer Anal*, 45(5):1871–1901, 2007.

[198] Z.J. Wang. *Adaptive high-order methods in computational fluid dynamics*. Advances in computational fluid dynamics. World Scientific, 2011.

[199] Z.J. Wang, K. Fidkowski, R. Abgrall, F. Bassi, et al. High–order CFD methods: current status and perspective. *Int J Numer Meth Fl*, 72(8):811–845, 2012.

[200] R.C. Whaley and A. Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software Pract Exper*, 35(2):101–121, February 2005.

[201] I.M. Wilbers, H.P. Langtangen, and Å. Ødegård. Using Cython to speed up numerical Python programs. In *Proceedings of MekIT'09*, pages 495–512, 2009.

[202] H. Zhang. *Efficient time stepping methods and sensitivity analysis for large scale systems of differential equations*. PhD thesis, Virginia Polytechnic Institute and State University, 2014.

[203] H. Zhang and A. Sandu. FATODE: A library for forward, adjoint, and tangent linear integration of ODEs. *SIAM J Sci Comput*, 36(5), 2014.

Appendix A

Some examples of tableaux of six-stage fifth-order ERK methods with rational coefficients

The methods here are provided to give the reader specific examples of Butcher tableaux for all families constructed in this thesis without having to install and run `OCSage`. In order to fit these tableaux in the thesis document, the free parameters are rational numbers composed of small integers.

A.1 Examples of six-stage fifth-order ERK methods from the restricted case with $b_2 = 0$

A fifth-order six-stage ERK method of the family constructed in Section 3.3 using the C(2) simplifying assumptions (2.44b) with $c_6 = 1$ and free parameters $a_{6,3} = \frac{7}{5}$, $c_2 = \frac{1}{5}$, $c_3 = \frac{2}{7}$, $c_4 = \frac{4}{5}$, and $c_5 = \frac{8}{9}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\
\frac{2}{7} & \frac{4}{49} & \frac{10}{49} & 0 & 0 & 0 & 0 \\
\frac{4}{5} & -\frac{2269}{16625} & -\frac{2034}{3325} & \frac{3677}{2375} & 0 & 0 & 0 \\
\frac{8}{9} & -\frac{2686}{3645} & -\frac{1268}{2187} & \frac{79954}{32805} & -\frac{1520}{6561} & 0 & 0 \\
1 & -\frac{13049}{12160} & \frac{313}{456} & \frac{7}{5} & \frac{55}{192} & -\frac{729}{2432} & 0 \\
\hline
\mathbf{b} & \frac{65}{768} & 0 & \frac{2401}{5472} & \frac{875}{1152} & -\frac{2187}{4864} & \frac{1}{6}
\end{array} \tag{A.1}$$

A fifth-order six-stage ERK method of the family constructed in Section 3.3 using the C(2) simplifying assumptions (2.44b) with $c_6 \neq 1$ and free parameters $c_2 = \frac{5}{100}$, $c_3 = \frac{20}{100}$, $c_5 = \frac{29}{100}$, and $c_6 = \frac{7}{8}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{20} & \frac{1}{20} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{5} & -\frac{1}{5} & \frac{2}{5} & 0 & 0 & 0 & 0 \\
\frac{1}{4} & -\frac{9}{64} & \frac{5}{16} & \frac{5}{64} & 0 & 0 & 0 \\
\frac{29}{100} & -\frac{68933}{1000000} & \frac{10469}{50000} & \frac{23229}{200000} & \frac{522}{15625} & 0 & 0 \\
\frac{7}{8} & -\frac{6920375}{2420224} & \frac{35}{128} & \frac{5064255}{83456} & -\frac{2517669}{20864} & \frac{38390625}{605056} & 0 \\
\hline
\mathbf{b} & -\frac{41}{406} & 0 & \frac{9125}{1458} & 1 & \frac{781250}{91611} & \frac{20864}{66339}
\end{array} \tag{A.2}$$

A fifth-order six-stage ERK method of the family constructed in Section 3.3 with $b_2 = 0$ and without using the C(2) simplifying assumptions (2.44b) with $c_6 = 1$ and free parameters $a_{6,3} = \frac{1}{2}$, $c_2 = \frac{1}{5}$, $c_3 = \frac{4}{9}$, $c_4 = \frac{12}{17}$, and $c_5 = \frac{15}{17}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{9} & -\frac{4}{81} & \frac{40}{81} & 0 & 0 & 0 & 0 \\
\frac{12}{17} & -\frac{1600969}{1316684} & \frac{8961605}{3620881} & -\frac{8012097}{14483524} & 0 & 0 & 0 \\
\frac{15}{17} & -\frac{8966873}{9275744} & \frac{3839165}{2318936} & \frac{1391103}{4637872} & -\frac{201}{1888} & 0 & 0 \\
1 & -\frac{19383031}{11056608} & \frac{2232125}{921384} & \frac{1}{2} & \frac{53465}{165024} & -\frac{170510}{345519} & 0 \\
\hline
\mathbf{b} & \frac{709}{5400} & 0 & \frac{203391}{268000} & -\frac{54043}{108000} & \frac{289867}{361800} & -\frac{191}{1000}
\end{array} \qquad (A.3)$$

A fifth-order six-stage ERK method of the family constructed in Section 3.3 with $b_2 = 0$ and without using the C(2) simplifying assumptions (2.44b) with $c_6 \neq 1$ and free parameters $c_2 = \frac{1}{2}$, $c_3 = \frac{1}{5}$, $c_5 = \frac{4}{9}$, and $c_6 = \frac{12}{17}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{5} & \frac{4}{25} & \frac{1}{25} & 0 & 0 & 0 & 0 \\
\frac{1}{4} & \frac{9}{64} & \frac{1}{32} & \frac{5}{64} & 0 & 0 & 0 \\
\frac{4}{9} & \frac{262}{3645} & -\frac{116}{3645} & -\frac{22}{81} & \frac{2464}{3645} & 0 & 0 \\
\frac{12}{17} & -\frac{472314}{2923235} & -\frac{1692}{24565} & \frac{15228966}{6431117} & -\frac{46964256}{20462645} & \frac{38870280}{45017819} & 0 \\
\hline
\mathbf{b} & \frac{41}{64} & 0 & -\frac{51125}{5676} & \frac{7232}{651} & -\frac{28431}{9856} & \frac{584647}{511872}
\end{array} \qquad (A.4)$$

A.2 Examples of $5(4)_{6(6)}$ ERK pairs from the restricted case with $b_2 = 0$

A $5(4)_{6(6)}$ ERK pair from the $5(4)_{6(6)\mathrm{C}(2)}$ family using the C(2) simplifying assumptions (2.44b) with $c_6 = 1$ and free parameters $\hat{b}_6 = \frac{1}{40}$, $c_2 = \frac{2}{7}$, $c_3 = \frac{2}{9}$, and $c_5 = \frac{9}{11}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{2}{7} & \frac{2}{7} & 0 & 0 & 0 & 0 & 0 \\
\frac{2}{9} & \frac{11}{81} & \frac{7}{81} & 0 & 0 & 0 & 0 \\
\frac{9}{29} & \frac{10107}{97556} & \frac{1701}{48778} & \frac{16767}{97556} & 0 & 0 & 0 \\
\frac{9}{11} & \frac{304083}{356708} & -\frac{8505}{178354} & -\frac{32043195}{8204284} & \frac{8038278}{2051071} & 0 & 0 \\
1 & -\frac{678857}{65124} & \frac{63}{134} & \frac{18573597}{363676} & -\frac{144937940}{3370167} & \frac{372680}{129033} & 0 \\
\hline
\mathbf{b} & \frac{1147}{9720} & 0 & -\frac{72171}{379960} & \frac{13438339}{20120400} & \frac{980947}{2580660} & \frac{67}{2800} \\
\hat{\mathbf{b}} & \frac{233}{1944} & 0 & -\frac{2187}{10856} & \frac{170723}{251505} & \frac{389983}{1032264} & \frac{1}{40}
\end{array} \qquad (A.5)$$

A $5(4)_{6(6)}$ ERK pair from the $5(4)_{6(6)\mathrm{C}(2)}$ family using the C(2) simplifying assumptions (2.44b) with $c_6 \neq 1$ and free parameters $\hat{b}_6 = \frac{1}{4}$, $c_2 = \frac{1}{7}$,

$c_3 = \frac{4}{7}$, $c_5 = \frac{8}{11}$, and $c_6 = \frac{7}{8}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{7} & \frac{1}{7} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{7} & -\frac{4}{7} & \frac{8}{7} & 0 & 0 & 0 & 0 \\
\frac{14}{17} & \frac{203}{9826} & \frac{1372}{4913} & \frac{5145}{9826} & 0 & 0 & 0 \\
\frac{8}{11} & -\frac{143020}{270193} & \frac{41552}{38599} & \frac{28812}{192995} & \frac{41616}{1350965} & 0 & 0 \\
\frac{7}{8} & \frac{24876341}{45613056} & -\frac{1715}{3712} & \frac{693906493}{1026293760} & -\frac{172333301}{769720320} & \frac{14413399}{42467328} & 0 \\
\hline
\mathbf{b} & \frac{9073}{62720} & 0 & \frac{3560683}{1468800} & \frac{24471653}{3175200} & -\frac{8067191}{1347840} & -\frac{178176}{54145} \\
\hat{\mathbf{b}} & \frac{1532427}{9175040} & 0 & \frac{127544207}{88473600} & \frac{555753647}{464486400} & -\frac{194300711}{106168320} & \frac{1}{40}
\end{array} \tag{A.6}$$

A.3 An example of a $5(4)_{6(7)}$ ERK pair from the restricted case with $b_2 = 0$

A $5(4)_{6(7)}$ ERK pair from the $5(4)_{6(7)\mathrm{C}(2)}$ family using the C(2) simplifying assumptions (2.44b) and free parameters $\hat{b}_7 = \frac{1}{5}$, $c_2 = \frac{1}{7}$, $c_3 = \frac{3}{7}$, $c_4 = \frac{5}{11}$, and $c_5 = \frac{7}{8}$ is given by

$$\begin{array}{c|ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{7} & \frac{1}{7} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{3}{7} & -\frac{3}{14} & \frac{9}{14} & 0 & 0 & 0 & 0 & 0 \\
\frac{5}{11} & -\frac{5035}{23958} & \frac{5075}{7986} & \frac{350}{11979} & 0 & 0 & 0 & 0 \\
\frac{7}{8} & \frac{953603}{377856} & -\frac{501613}{125952} & \frac{3173975}{755712} & -\frac{156695}{83968} & 0 & 0 & 0 \\
1 & \frac{155423}{10458} & -\frac{11977}{498} & \frac{216608}{18675} & -\frac{1452}{3071} & -\frac{503808}{537425} & 0 & 0 \\
1 & \frac{757}{6300} & 0 & \frac{69629}{36000} & -\frac{190333}{133200} & \frac{41984}{97125} & -\frac{83}{1440} & 0 \\
\hline
\mathbf{b} & \frac{757}{6300} & 0 & \frac{69629}{36000} & -\frac{190333}{133200} & \frac{41984}{97125} & -\frac{83}{1440} & 0 \\
\hat{\mathbf{b}} & \frac{41579}{327500} & 0 & \frac{59970463}{39300000} & -\frac{48510957}{48470000} & \frac{17076992}{45440625} & -\frac{75281}{1572000} & \frac{1}{50}
\end{array} \tag{A.7}$$

A.4 Examples of six-stage fifth-order ERK methods with $b_2 \neq 0$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Cassity's restricted Case I described in Section 3.4.1 with free parameters $a_{6,3} = \frac{5}{2}$, $c_3 = \frac{1}{11}$, $c_4 = \frac{5}{6}$, and $\frac{\lambda}{\omega} = \frac{1}{3}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{3}{5} & \frac{3}{5} & 0 & 0 & 0 & 0 & 0 \\
\frac{1}{11} & \frac{193}{2178} & \frac{5}{2178} & 0 & 0 & 0 & 0 \\
\frac{5}{6} & \frac{5222297747156 5}{8185519538856} & \frac{139557725579275}{114597273543984} & -\frac{23490354006905}{3472644652848} & 0 & 0 & 0 \\
\frac{14733}{25430} & -\frac{1388980098655288191}{3564578677252285000} & -\frac{839575146724 21197}{399232811852255920} & \frac{1738005009652647}{1611629306686000} & \frac{583868380894056}{578665369683 8125} & 0 & 0 \\
1 & -\frac{6900774559898 5573}{196348881233562 75} & -\frac{13812251854127}{2238960975174} & \frac{5}{2} & -\frac{6143516581}{3340785700} & \frac{879571361919395}{877593578644 44} & 0 \\
\hline
\mathbf{b} & -\frac{52717}{662985} & -\frac{125}{108} & \frac{2547534}{6695017} & \frac{529497}{2373560} & \frac{5227516998600125}{3334855598848872} & \frac{5}{76}
\end{array} \tag{A.8}$$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Cassity's restricted Case II described in Section 3.4.1 with free parameters $a_{6,3} = \frac{5}{2}$, $c_2 = \frac{2}{11}$, $c_3 = \frac{5}{11}$, and $c_4 = \frac{4}{7}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{2}{11} & \frac{2}{11} & 0 & 0 & 0 & 0 & 0 \\
\frac{5}{11} & -\frac{113}{308} & \frac{23}{28} & 0 & 0 & 0 & 0 \\
\frac{4}{7} & -\frac{450074252371109}{1176875694925\,0} & \frac{1562373735064\,3}{2353751389850} & -\frac{13193128772553}{5884378474625} & 0 & 0 & 0 \\
\frac{20599}{61855} & \frac{4510447248555745\,31}{9939710239257750\,0} & -\frac{21980728147895677}{2981913071777325} & \frac{5099545489735920\,9}{16566183732096250} & \frac{752124345816767}{851975163364950\,0} & 0 & 0 \\
1 & \frac{19600825620466619}{1513789724300124} & -\frac{1182101149849}{55116379107} & \frac{5}{2} & \frac{2119360135879}{550467887292} & \frac{707473190881800}{224660518880333} & 0 \\
\hline
\mathbf{b} & \frac{653}{20599} & \frac{1331}{4968} & -\frac{1267112}{1116261} & \frac{25990825}{22297032} & \frac{2049402589497627087\,5}{37074377467708072992} & \frac{19321}{165024}
\end{array} \tag{A.9}$$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Case III described in Section 3.4.1 with free parameters $a_{6,3} = \frac{5}{2}$, $c_2 = \frac{2}{11}$, $c_3 = \frac{5}{6}$, $c_4 = \frac{5}{7}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{2}{11} & \frac{2}{11} & 0 & 0 & 0 & 0 & 0 \\
\frac{5}{6} & -\frac{113}{168} & \frac{253}{168} & 0 & 0 & 0 & 0 \\
\frac{5}{7} & -\frac{738089838618683977}{225007707411704700} & \frac{213717237961025249}{45001541482340940} & -\frac{14148046681506814}{18750642284308725} & 0 & 0 & 0 \\
\frac{199029}{645205} & \frac{16161028022292391722706\,9}{30881372294186314621875} & -\frac{27026086509185701042\,3}{40834872455122399500} & \frac{2250204078271371818\,57}{282021664787089631250} & \frac{175624548280233319199\,1}{196072205042452791250\,0} & 0 & 0 \\
1 & \frac{28737851105521296804\,52}{327965773635464504025} & -\frac{45090018533722151}{3954789788046540} & \frac{5}{2} & \frac{907858129392963}{2972833526714900} & \frac{457568357483798193864\,00}{548952406544382943558\,09} & 0 \\
\hline
\mathbf{b} & \frac{210139}{4776696} & \frac{1331}{4968} & -\frac{279720}{2031851} & \frac{4160933}{7331288} & \frac{22055974330782323094625}{156289912422718373579232} & \frac{19321}{165024}
\end{array} \tag{A.10}$$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Case Va described in Section 3.4.1 with free parameters $a_{6,3} = \frac{5}{2}$, $c_3 = \frac{4}{11}$, $c_4 = \frac{2}{3}$, $c_5 = \frac{5}{7}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{9716}{27959} & \frac{9716}{27959} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{495465244}{8341431329} & \frac{2537782512}{8341431329} & 0 & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{291374115779992947\,7}{986242579971571792\,8} & \frac{30767793327214517897\,23}{24458815983294980461\,44} & -\frac{2949442558227539}{9969802177196736} & 0 & 0 & 0 \\
\frac{5}{7} & -\frac{773166309491733195}{3102593456961423152} & \frac{44248129202257826351765\,33}{41195987714057219698403\,84} & -\frac{845258143833169}{6272745393544448} & \frac{57618151119}{23860116847\,84} & 0 & 0 \\
1 & \frac{566049231315377}{622397828125920} & -\frac{12163416111265811886451}{52000821949723271486\,40} & \frac{5}{2} & -\frac{7311351040371}{1131921478160} & \frac{1580833012486}{247436047815} & 0 \\
\hline
\mathbf{b} & \frac{2161811}{19987200} & \frac{42774464019537163\,27}{30220526641564882944} & \frac{1156068001}{2699827200} & -\frac{4372281}{4283200} & \frac{389944009}{303361200} & \frac{5285401}{91944720}
\end{array} \tag{A.11}$$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Case Vb described in Section 3.4.1

with free parameters $a_{6,3} = \frac{5}{2}$, $c_3 = \frac{4}{11}$, $c_4 = \frac{2}{3}$, $c_5 = \frac{5}{7}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{7}{24} & \frac{7}{24} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{10246}{85547} & \frac{20862}{85547} & 0 & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{1014414325}{3166597854} & \frac{508266562}{527766309} & \frac{107327}{4478922} & 0 & 0 & 0 \\
\frac{5}{7} & -\frac{62934371767}{232592409140} & \frac{824876926338}{1104813943415} & \frac{25609771}{120206065} & \frac{69651}{2764580} & 0 & 0 \\
1 & \frac{887575057}{877712220} & -\frac{10916872534}{5193130635} & \frac{5}{2} & -\frac{1086113}{137940} & \frac{32898502}{4407183} & 0 \\
\hline
\mathbf{b} & \frac{1}{9} & -\frac{14336}{343995} & \frac{4114121}{6463800} & -\frac{697}{600} & \frac{967603}{690120} & \frac{2299}{42840}
\end{array} \tag{A.12}$$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Case VIa described in Section 3.4.1 with free parameters $c_3 = \frac{4}{11}$, $c_5 = \frac{2}{3}$, $c_6 = \frac{5}{7}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{39003}{159797} & \frac{39003}{159797} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{21958605538}{265237210293} & \frac{6771935374}{24112473663} & 0 & 0 & 0 & 0 \\
\frac{46848}{54805} & \frac{616003138001113163352}{894568468719054082225} & -\frac{42925108966667986796449152}{17090730594877528224138625} & \frac{352017071411108020992}{131457046341647013625} & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{13246159096454397}{32243754499054848} & \frac{25307953897291390434 01}{14392926291749681475 9} & -\frac{9291493208050 57}{6589327419565 2} & \frac{4327356771931368799525}{3349309299446417257728} & 0 & 0 \\
\frac{5}{7} & -\frac{3436285950072898805}{8618238054483125376} & \frac{231055056201235427717484136}{134951364087710290403219 37} & -\frac{3952342013587237 35}{2881998749988761 2} & \frac{537224875995443145125018975}{419585316172831405066184064} & \frac{4802386019918409}{3511823310508425 22} & 0 \\
\hline
\mathbf{b} & \frac{194572881211}{1754124042240} & \frac{290454347683282001657 9}{181371594153250848873888} & \frac{5036678016271}{9775543691520} & \frac{866953853625195737941375}{2762426517250755947544576} & -\frac{38063321469}{200536460480} & \frac{28299624199}{121042153440}
\end{array} \tag{A.13}$$

A fifth-order six-stage ERK method without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ that belongs to Case VIb described in Section 3.4.1 with free parameters $c_3 = \frac{4}{11}$, $c_5 = \frac{2}{3}$, $c_6 = \frac{5}{7}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|cccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{7}{24} & \frac{7}{24} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{10246}{85547} & \frac{20862}{85547} & 0 & 0 & 0 & 0 \\
\frac{46848}{54805} & \frac{296249601676398}{1152281491970875} & -\frac{432036769177744 38}{218933483474466 25} & \frac{8041377527812068}{3127621192492375} & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{878}{707} & \frac{150849721314}{9950051461} & -\frac{41211511}{2813026} & \frac{46456495982675}{33228832759554} & 0 & 0 \\
\frac{5}{7} & -\frac{1408126}{1212505} & \frac{2540427596732820 6}{1757626840328345} & -\frac{19406073053961}{13913395377 56} & \frac{5746090870778048341258 5}{42367154987361948975518} & \frac{1024078653}{76500710020} & 0 \\
\hline
\mathbf{b} & \frac{1}{9} & \frac{31816704}{999227233} & \frac{9051373661}{18228408480} & \frac{377100414799998146125}{1197087063534510149664} & -\frac{217857}{1237360} & \frac{24977603}{112742604}
\end{array} \tag{A.14}$$

A.5 Examples of $5(4)_{6(6)}$ ERK pairs with $b_2 \neq 0$

A $5(4)_{6(6)}$ ERK pair without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ where the fifth-order component belongs to Case Va described in Section 3.4.1 and the $5(4)_{6(6)}$ ERK pair construction is described in 3.5.1 with free parameters $\hat{b}_6 = \frac{1}{4}$, $c_3 = \frac{4}{11}$, $c_5 = \frac{2}{3}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{r|rrrrrr}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{19641}{42509} & \frac{19641}{42509} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{83461099918}{360769089483} & \frac{47727659894}{360769089483} & 0 & 0 & 0 & 0 \\
\frac{46848}{54805} & -\frac{4615397352628046 7072}{8686362091306078 3025} & -\frac{6174062374465254892735104}{3997029516314491930895375} & \frac{1789300187576788246272}{610513138279286991125} & 0 & 0 & 0 \\
\frac{2}{3} & \frac{8728786054630157}{8205169202910 72} & \frac{4069103983347452 1019}{1509179619340823121} & -\frac{35918840276803}{891101789148} & \frac{9770995037103307 2475}{28955369987509761792} & 0 & 0 \\
1 & -\frac{4622870689816386245}{357896209613158656} & -\frac{537759443239693402323189736}{171778098290612943783173 85} & \frac{3518454982602231}{722759853292 90} & -\frac{656360818729932547702929185}{1953461074965157684552158 72} & -\frac{102148583901021}{1036222939050970} & 0 \\
\hline
\mathbf{b} & \frac{19928832725}{176667181056} & -\frac{15999987164511039 1489}{48476891446692043129 92} & \frac{3340489227669}{6104183509760} & \frac{293035891230875114421125}{767278272787793401135104} & \frac{97198461}{25831127360} & -\frac{129652589}{10189797856} \\
\hat{\mathbf{b}} & \frac{30705014940869}{250204895170560} & \frac{9594034573526316 0481}{1200899029410138290760} & \frac{48957625262403}{112273375268800} & \frac{8027793349705101047725}{27313255092012376633344} & \frac{96984984609}{2286458382725} & \frac{1}{40}
\end{array} \tag{A.15}$$

A $5(4)_{6(6)}$ ERK pair without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ where the fifth-order component belongs to Case Vb described in Section 3.4.1 and the $5(4)_{6(6)}$ ERK pair construction is described in 3.5.1 with free parameters $\hat{b}_6 = \frac{1}{4}$, $c_3 = \frac{4}{11}$, $c_5 = \frac{2}{3}$, $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{r|rrrrrr}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{7}{24} & \frac{7}{24} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{10246}{85547} & \frac{20862}{85547} & 0 & 0 & 0 & 0 \\
\frac{46848}{54805} & \frac{296249601676398}{1152281491970875} & -\frac{432036769177774438}{218933483474466625} & \frac{8041377527812068}{3127621192492375} & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{878}{707} & \frac{150849721314}{9950051461} & -\frac{41211511}{2813026} & \frac{46456495982675}{332288327595554} & 0 & 0 \\
1 & \frac{45982}{10605} & -\frac{291630773053567 4}{56367041526565} & \frac{76571372647}{1448708390} & -\frac{484304306749963576 0895}{1164611507357721212988} & -\frac{103210247}{350482220} & 0 \\
\hline
\mathbf{b} & \frac{1}{9} & \frac{10918912}{239251591} & \frac{740849241}{1575294560} & \frac{72713620652822609 0375}{1943526171096530761248} & \frac{11009}{1237360} & -\frac{114433}{11362596} \\
\hat{\mathbf{b}} & \frac{1}{9} & \frac{24636608}{211104345} & \frac{2417352883}{6751262400} & \frac{1333025074568592 25}{488507269346872128} & \frac{4315249}{37120800} & \frac{1}{40}
\end{array} \tag{A.16}$$

A $5(4)_{6(6)}$ ERK pair without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ where the fifth-order component belongs to Case VIa described in Section 3.4.1 and the $5(4)_{6(6)}$ ERK pair construction is described in 3.5.1 with free parameters $c_3 = \frac{4}{11}$, $c_5 = \frac{2}{3}$, $c_6 = \frac{7}{8}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{r|rrrrrr}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{68883}{111392} & \frac{68883}{111392} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{67465155386}{258438476901} & \frac{26512472578}{258438476901} & 0 & 0 & 0 & 0 \\
\frac{46848}{54805} & -\frac{5044210803731105 8176}{6274215522808259 0075} & -\frac{2046153189424287138415 2576}{1958465004366984007896 0875} & \frac{230599659021910659 9936}{85295283496667567 0875} & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{3312016735696 03}{2772035578609 92} & -\frac{2385472200936947 8888}{1091126766369930 1983} & \frac{11665083034594}{2620351707561} & -\frac{1777564905785093525}{4399749867601130112} & 0 & 0 \\
\frac{7}{8} & -\frac{1217188272749624988 19}{476038139367309967 36} & -\frac{1564666943346487155615 6395}{506139670092991055799 3876} & \frac{3422808638875321 29}{5020545808798412 8} & -\frac{121607678404532946190070455 25}{350588992309543045028142120 96} & \frac{985100736091815}{1874548938494566 4} & 0 \\
\hline
\mathbf{b} & \frac{7647763157}{67767646464} & -\frac{33679378455928733696}{50215743175970571 6213} & \frac{557479442036}{1039822106175} & \frac{12429879222761661381125}{21035458189555259747302 4} & \frac{25254747}{713028700} & -\frac{16595531392}{79696838025} \\
\hat{\mathbf{b}} & \frac{8949631}{75633534} & \frac{14858346058244096}{87835828539392289} & \frac{2269867435}{4621431583} & \frac{136627662619403 75}{371346779134289 16} & -\frac{4180491}{28521148} & 0
\end{array} \tag{A.17}$$

A $5(4)_{6(6)}$ ERK pair without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ where the fifth-order component belongs to Case VIb described in Section 3.4.1 and the $5(4)_{6(6)}$ ERK pair construction is described in 3.5.1 with free parameters $c_3 = \frac{4}{11}$, $c_5 = \frac{4}{6}$, $c_6 = \frac{7}{8}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{7}{24} & \frac{7}{24} & 0 & 0 & 0 & 0 & 0 \\
\frac{4}{11} & \frac{10246}{85547} & \frac{20862}{85547} & 0 & 0 & 0 & 0 \\
\frac{46848}{54805} & \frac{296249601676398}{1152281491970875} & -\frac{432036769177744438}{218933483474466625} & \frac{8041377527812068}{3127621192492375} & 0 & 0 & 0 \\
\frac{2}{3} & -\frac{878}{707} & \frac{150849721314}{99500051461} & -\frac{41211511}{2813026} & \frac{46456495982675}{33228832759554} & 0 & 0 \\
\frac{7}{8} & \frac{1039}{1616} & -\frac{84963072376071}{12883895206072} & \frac{2145204685701}{296695478272} & -\frac{76883999572949592015825}{198760363922384420349952} & -\frac{303899085}{17944689664} & 0 \\
\hline
\mathbf{b} & \frac{1}{9} & \frac{4065216}{98515361} & \frac{453973487}{949396275} & \frac{149757580997128450375}{270236115061822825308} & -\frac{14847}{1546700} & -\frac{7323712}{41820975} \\
\hat{\mathbf{b}} & \frac{1}{9} & \frac{2788480}{42220869} & \frac{7393705}{16878156} & \frac{2633786267362000}{7632926083544877} & \frac{7373}{185604} & 0
\end{array} \qquad (A.18)$$

A.6 Examples of $5(4)_{6(7)}$ ERK pairs with $b_2 \neq 0$

A $5(4)_{6(7)}$ ERK pair without the C(2) simplifying assumptions (2.44b) and $b_2 \neq 0$ where the fifth-order component belongs to Case III described in Section 3.4.1 and the $5(4)_{6(7)}$ ERK pair construction is described in 3.5.2 with free parameters $\hat{b}_7 = \frac{1}{40}$, $c_2 = \frac{2}{7}$, $c_3 = \frac{5}{7}$, and $\frac{\lambda}{\mu} = -\frac{7}{4}$ is given by

$$\begin{array}{c|ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{2}{7} & \frac{2}{7} & 0 & 0 & 0 & 0 & 0 & 0 \\
\frac{5}{7} & -\frac{607}{1708} & \frac{261}{244} & 0 & 0 & 0 & 0 & 0 \\
\frac{185136}{236881} & -\frac{4229171860606623}{26584021467571682} & \frac{21168284512037635}{26584021467571682} & \frac{1918907498779390}{13292010733785841} & 0 & 0 & 0 & 0 \\
\frac{1639}{3735} & -\frac{657584713618457}{8062837044261300} & \frac{1191191579277853387759}{2367565710507570802500} & \frac{744049258295258188}{2632860900984236625} & -\frac{245272295388552795917512283}{924331983084958001072636250} & 0 & 0 & 0 \\
1 & \frac{29}{252} & -\frac{381667932653}{59242078260} & -\frac{100614134684}{3615126723} & \frac{2650372230739821290097}{13374534248482303890} & \frac{511926161540400}{33366967955113} & 0 & 0 \\
1 & \frac{55}{504} & \frac{23105737781}{118484156520} & -\frac{9334105991}{14460506892} & \frac{3148624794629923801921}{3504127973102363619180} & \frac{1195027633334587125}{279748659335667392} & \frac{1}{64} & 0 \\
\hline
\mathbf{b} & \frac{55}{504} & \frac{23105737781}{118484156520} & -\frac{9334105991}{14460506892} & \frac{3148624794629923801921}{3504127973102363619180} & \frac{1195027633334587125}{279748659335667392} & \frac{1}{64} & 0 \\
\hat{\mathbf{b}} & \frac{55}{504} & \frac{23105737781}{118484156520} & -\frac{9334105991}{14460506892} & \frac{3148624794629923801921}{3504127973102363619180} & \frac{1195027633334587125}{279748659335667392} & -\frac{3}{320} & \frac{1}{40}
\end{array} \qquad (A.19)$$

Appendix B

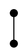
Expressions for elementary differentials, rooted trees, scalar sums, and order conditions

B.1 Order conditions using variable substitutions

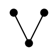
B.1.1 Standard six-stage fifth-order order conditions using variable substitutions

The fifth-order quadrature conditions can be written without requiring any standard variable substitutions as

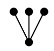
$$b_1 + b_2 + b_3 + b_4 + b_5 + b_6 = 1, \tag{B.1a}$$



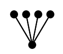
$$b_2c_2 + b_3c_3 + b_4c_4 + b_5c_5 + b_6c_6 = \frac{1}{2}, \tag{B.1b}$$



$$b_2c_2^2 + b_3c_3^2 + b_4c_4^2 + b_5c_5^2 + b_6c_6^2 = \frac{1}{3}, \tag{B.1c}$$

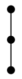


$$b_2c_2^3 + b_3c_3^3 + b_4c_4^3 + b_5c_5^3 + b_6c_6^3 = \frac{1}{4}, \tag{B.1d}$$

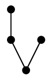


$$b_2c_2^4 + b_3c_3^4 + b_4c_4^4 + b_5c_5^4 + b_6c_6^4 = \frac{1}{5}. \tag{B.1e}$$

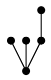
The fifth-order non-branching conditions of height two that involve just $\mathbf{b}, \mathbf{c}, \mathbf{p}$, along with alternate forms involving $\mathbf{s}$, can be written as



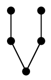
$$b_3p_3 + b_4p_4 + b_5p_5 + b_6p_6 = \frac{1}{6},$$
$$r_3c_3 + r_4c_4 + r_5c_5 = \frac{1}{6}, \tag{B.2a}$$



$$b_3c_3p_3 + b_4c_4p_4 + b_5c_5p_5 + b_6c_6p_6 = \frac{1}{8},$$
$$s_3c_3 + s_4c_4 + s_5c_5 = \frac{1}{8}, \tag{B.2b}$$

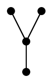


$$b_3c_3^2p_3 + b_4c_4^2p_4 + b_5c_5^2p_5 + b_6c_6^2p_6 = \frac{1}{10}, \tag{B.2c}$$

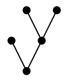


$$b_3p_3^2 + b_4p_4^2 + b_5p_5^2 + b_6p_6^2 = \frac{1}{20}. \tag{B.2d}$$

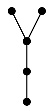
The fifth-order conditions that involve $\mathbf{q}$ can be written as



$$b_3q_3 + b_4q_4 + b_5q_5 + b_6q_6 = \frac{1}{12},$$
$$r_2c_2^2 + r_3c_3^2 + r_4c_4^2 + r_5c_5^2 = \frac{1}{12}, \tag{B.3a}$$

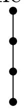


$$b_3c_3q_3 + b_4c_4q_4 + b_5c_5q_5 + b_6c_6q_6 = \frac{1}{15}, \tag{B.3b}$$

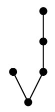


$$b_4a_{4,3}q_3 + b_5(a_{5,4}q_4 + a_{5,3}q_3) + b_6(a_{6,5}q_5 + a_{6,4}q_4 + a_{6,3}q_3) = \frac{1}{60},$$
$$r_4(a_{4,3}c_3 + a_{4,2}c_2) + r_5(a_{5,2}c_2 + a_{5,3}c_3 + r_5a_{5,4}c_4) = \frac{1}{60}. \tag{B.3c}$$

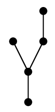
The remaining fifth-order conditions, except the tall tree of fifth-order, can be written as



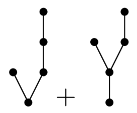
$$b_4a_{4,3}p_3 + b_5(a_{5,4}p_4 + a_{5,3}p_3) + b_6(a_{6,5}p_5 + a_{6,4}p_4 + a_{6,3}p_3) = \frac{1}{24},$$
$$r_3p_3 + r_4p_4 + r_5p_5 = \frac{1}{24}, \tag{B.4a}$$



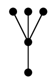
$$b_4c_4a_{4,3}p_3 + b_5(c_5a_{5,4}p_4 + c_5a_{5,3}p_3) + b_6(c_6a_{6,5}p_5 + c_6a_{6,4}p_4 + c_6a_{6,3}p_3) = \frac{1}{30}, \tag{B.4b}$$



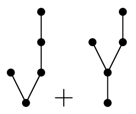
$$b_4a_{4,3}c_3p_3 + b_5(a_{5,4}c_4p_4 + a_{5,3}c_3p_3) + b_6(a_{6,5}c_5p_5 + a_{6,4}c_4p_4 + a_{6,3}c_3p_3) = \frac{1}{40},$$
$$r_3c_3p_3 + r_4c_4p_4 + r_5p_5c_5 = \frac{1}{40},$$

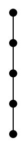


$$b_4a_{4,3}p_3(c_3 + c_4) + b_5(a_{5,4}p_4(c_5 + c_4) + a_{5,3}p_3(c_5 + c_3)) + b_6(a_{6,5}p_5(c_6 + c_5) + a_{6,4}p_4(c_6 + c_4)+$$
$$a_{6,3}p_3(c_6 + c_3)) = \frac{7}{120}, \tag{B.4c}$$



$$b_3a_{3,2}c_2^3 + b_4(a_{4,3}c_3 + a_{4,2}c_2) + b_5(a_{5,4}c_4 + a_{5,3}c_3 + a_{5,2}c_2) + b_6(a_{6,5}c_5 + a_{6,4}c_4 + a_{6,3}c_3 + a_{6,2}c_2) = \frac{1}{20},$$
$$r_2c_2^3 + r_3c_3^3 + r_4c_4^3 + r_5c_5^3 = \frac{1}{20}, \tag{B.4d}$$

where in [113] and [31] the order conditions for the trees  are combined additively because these authors only found the order conditions for single ODEs rather than systems. Finally, the tall tree of fifth-order can be written as

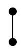


$$b_5a_{5,4}a_{4,3}p_3 + b_6(a_{6,4}a_{4,3}p_3 + a_{6,5}(a_{5,4}p_4 + a_{5,3}p_3)) = \frac{1}{120},$$
$$r_5a_{5,4}p_4 = \frac{1}{120}. \tag{B.5}$$

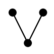
B.1.2 Standard fourth-order seven-stage order conditions using variable substitutions

The fourth-order quadrature conditions can be written as

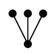
$$b_1 + b_2 + b_3 + b_4 + b_5 + b_6 + b_7 = 1, \tag{B.6a}$$



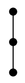
$$b_2c_2 + b_3c_3 + b_4c_4 + b_5c_5 + b_6c_6 + b_7c_7 = \frac{1}{2}, \tag{B.6b}$$



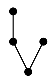
$$b_2c_2^2 + b_3c_3^2 + b_4c_4^2 + b_5c_5^2 + b_6c_6^2 + b_7c_7^2 = \frac{1}{3}, \tag{B.6c}$$



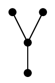
$$b_2c_2^3 + b_3c_3^3 + b_4c_4^3 + b_5c_5^3 + b_6c_6^3 + b_7c_7^3 = \frac{1}{4}, \tag{B.6d}$$



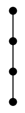
$$b_3p_3 + b_4p_4 + b_5p_5 + b_6p_6 + b_7p_7 = \frac{1}{6},$$
$$r_3c_3 + r_4c_4 + r_5c_5 + r_5c_5 + r_6c_6 = \frac{1}{6}, \tag{B.7a}$$



$$b_3c_3p_3 + b_4c_4p_4 + b_5c_5p_5 + b_6c_6p_6 + b_7c_7p_7 = \frac{1}{8},$$
$$s_3c_3 + s_4c_4 + s_5c_5 + s_6c_6 + s_7c_7 = \frac{1}{8}, \tag{B.7b}$$



$$b_3q_3 + b_4q_4 + b_5q_5 + b_6q_6 + b_7q_7 = \frac{1}{12},$$
$$r_2c_2^2 + r_3c_3^2 + r_4c_4^2 + r_5c_5^2 + r_6c_6^2 = \frac{1}{12}, \tag{B.8a}$$



$$b_4a_{4,3}p_3 + b_5(a_{5,4}p_4 + a_{5,3}p_3) + b_6(a_{6,5}p_5 + a_{6,4}p_4 + a_{6,3}p_3) + b_7(a_{7,6}p_6 + a_{7,5}p_5 + a_{7,4}p_4 + a_{7,3}p_3) = \frac{1}{24},$$
$$r_3p_3 + r_4p_4 + r_5p_5 + r_6p_6 = \frac{1}{24}, \tag{B.9a}$$

1

Table B.1: Rooted trees and scalar sums up to fifth-order.

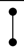
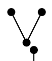
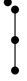
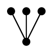
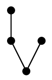
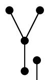
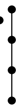
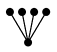
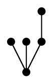
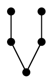
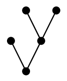
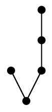
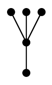
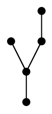
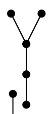
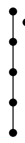
p		tree	Φ
1	τ		$\sum b_i$
2	$[\tau]$		$\sum b_i c_i$
3	$[\tau^2]$		$\sum b_i c_i^2$
	$[[\tau]]$		$\sum b_i a_{i,j} c_j$
4	$[\tau^3]$		$\sum b_i c_i^3$
	$[\tau[\tau]]$		$\sum b_i c_i a_{i,j} c_j$
	$[[\tau^2]]$		$\sum b_i a_{i,j} c_j^2$
	$[[[\tau]]]$		$\sum b_i a_{i,j} a_{j,k} c_k$
5	$[\tau^4]$		$\sum b_i c_i^4$
	$[\tau^2[\tau]]$		$\sum b_i c_i^2 a_{i,j} c_j$
	$[[\tau]^2]$		$\sum b_i a_{i,j} c_j a_{i,k} c_k$
	$[\tau[\tau^2]]$		$\sum b_i c_i a_{i,j} c_j^2$
	$[\tau[[\tau]]]$		$\sum b_i c_i a_{i,j} a_{j,k} c_k$
	$[[\tau^3]]$		$\sum b_i a_{i,j} c_j^3$
	$[[\tau[\tau^2]]]$		$\sum b_i a_{i,j} c_j a_{j,k} c_k$
	$[[[\tau^2]]]$		$\sum b_i a_{i,j} a_{j,k} c_k^2$
	$[[[[\tau]]]]$		$\sum b_i a_{i,j} a_{j,k} a_{k,l} c_l$

Table B.2: Rooted trees and scalar sums of sixth-order.

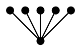
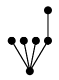
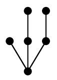
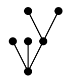
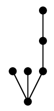
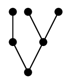
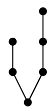
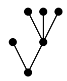
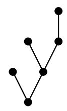
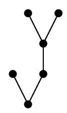
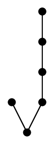
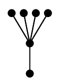
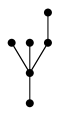
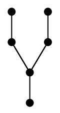
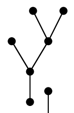
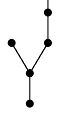
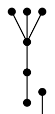
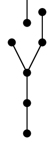
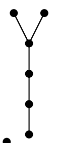
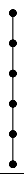
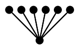
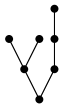
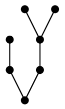
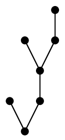
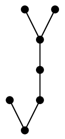
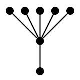
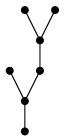
p		graph	Φ
6	$[\tau^5]$		$\sum b_i c_i^5$
	$[\tau^3[\tau]]$		$\sum b_i c_i^3 a_{i,j} c_j$
	$[\tau[\tau]^2]$		$\sum b_i c_i a_{i,j} c_j a_{i,k} c_k$
	$[\tau^2[\tau^2]]$		$\sum b_i c_i^2 a_{i,j} c_j^2$
	$[\tau^2[[\tau]]]$		$\sum b_i c_i^2 a_{i,j} a_{j,k} c_k$
	$[[\tau][\tau^2]]$		$\sum b_i a_{i,j} c_j a_{i,k} c_k^2$
	$[[\tau][[\tau]]]$		$\sum b_i a_{i,j} c_j a_{i,k} a_{k,l} c_l$
	$[\tau[\tau^3]]$		$\sum b_i c_i a_{i,j} c_j^3$
	$[\tau[\tau[\tau]]]$		$\sum b_i c_i a_{i,j} c_j a_{j,k} c_k$
	$[\tau[[\tau^2]]]$		$\sum b_i c_i a_{i,j} a_{j,k} c_k^2$
6	$[\tau[[[\tau]]]]$		$\sum b_i a_{i,j} c_j a_{i,k} a_{k,l} c_l$
	$[[\tau^4]]$		$\sum b_i a_{i,j} c_j^4$
	$[[\tau^2[\tau]]]$		$\sum b_i a_{i,j} c_j^2 a_{j,k} c_k$
	$[[[\tau]^2]]$		$\sum b_i a_{i,j} a_{j,k} c_k a_{j,k} c_k$
	$[[\tau[\tau]^2]]$		$\sum b_i a_{i,j} c_j a_{j,k} c_k^2$
	$[[\tau[[\tau]]]]$		$\sum b_i a_{i,j} c_j a_{j,k} a_{k,l} c_l$
	$[[[\tau^3]]]$		$\sum b_i a_{i,j} a_{j,k} c_l^3$
	$[[[\tau[\tau]]]]$		$\sum b_i a_{i,j} a_{j,k} c_k a_{k,m} c_m$
	$[[[[\tau^2]]]]$		$\sum b_i a_{i,j} a_{j,k} a_{k,l} c_l^2$
	$[[[[[\tau]]]]]$		$\sum b_i a_{i,j} a_{j,k} a_{k,l} a_{l,m} c_m$

Table B.3: Selected rooted trees and scalar sums of seventh-order. Only seventh-order rooted trees and scalar sums that are specifically mentioned in Chapter 5 are included.

p	tree	graph	Φ
7	$[\tau^6]$		$\sum b_i c_i^5$
	$[[\tau^2][[\tau]]]$		$\sum b_i a_{i,j} c_j^2 a_{i,k} a_{k,l} c_l$
	$[[\tau][[\tau^2]]]$		$\sum b_i a_{i,j} c_j a_{i,k} a_{k,l} c_l^2$
	$[\tau[[\tau[\tau]]]]$		$\sum b_i c_i a_{i,j} a_{j,k} c_k a_{k,l} c_l$
	$[\tau[[[\tau^2]]]]$		$\sum b_i c_i a_{i,j} a_{j,k} a_{k,l} c_l^2$
	$[[\tau^5]]$		$\sum b_i a_{i,j} c_j^5$
	$[[\tau[[\tau^2]]]]$		$\sum b_i a_{i,j} c_j a_{j,k} a_{k,l} c_l^2$

Appendix C

Linear algebra relations

C.1 Vandermonde matrices

A *Vandermonde matrix* [163, pg.15] is an $n \times n$ matrix given by

$$\begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^{(n-1)} & x_2^{(n-1)} & \dots & x_n^{(n-1)} \end{pmatrix} \tag{C.1}$$

A more common and classical definition of the Vandermonde matrix is the transpose of the definition given by (C.1) and Shilov [163, pg.15]. However, many of the properties of Vandermonde matrices (including all that are important in this thesis) are invariant to transposition.

The *Vandermonde determinant* of a 3×3 Vandermonde matrix, Δ_{ijk}, is given by

$$\Delta_{ijk} = \begin{vmatrix} 1 & 1 & 1 \\ c_i & c_j & c_k \\ c_i^2 & c_j^2 & c_k^2 \end{vmatrix} = (c_k - c_j)(c_k - c_i)(c_j - c_i), \tag{C.2}$$

which can be found by using any general-purpose formula for a determinant and simplifying appropriately [163, pg.15].

The Vandermonde determinant of a 4×4 Vandermonde matrix, Δ_{ijkl}, is given by

$$\Delta_{ijkl} = \begin{vmatrix} 1 & 1 & 1 & 1 \\ c_i & c_j & c_k & c_l \\ c_i^2 & c_j^2 & c_k^2 & c_l^2 \\ c_i^3 & c_j^3 & c_k^3 & c_l^3 \end{vmatrix} = (c_i - c_j)(c_i - c_k)(c_i - c_l)(c_j - c_k)(c_j - c_l)(c_k - c_l), \tag{C.3}$$

which can be found by using any popular general-purpose methodology for determinants and simplifying appropriately [163, pg.15].

C.1.1 Specific matrices

There are some specific forms of matrices in this thesis that require consideration. Matrices that can be factored into products including a Vandermonde matrix are ones that appear when considering fifth-order quadrature conditions (B.1), e.g., in Sections 3.3 and 3.4, given by

$$\begin{vmatrix} c_i & c_j & c_k & c_l \\ c_i^2 & c_j^2 & c_k^2 & c_l^2 \\ c_i^3 & c_j^3 & c_k^3 & c_l^3 \\ c_i^3 & c_j^3 & c_k^3 & c_l^3 \end{vmatrix} = c_i\, c_j\, c_k\, c_l \begin{vmatrix} 1 & 1 & 1 & 1 \\ c_i & c_j & c_k & c_l \\ c_i^2 & c_j^2 & c_k^2 & c_l^2 \\ c_i^3 & c_j^3 & c_k^3 & c_l^3 \end{vmatrix} = c_i\, c_j\, c_k\, c_l (c_i - c_j)(c_i - c_k)(c_i - c_l)(c_j - c_k)(c_j - c_l)(c_k - c_l), \tag{C.4}$$

which arises because the c components can be factored out of the columns of (C.4), see [163, pg.11]. This form can be viewed as a member of the *generalized Vandermonde determinants* [119, pg.14]. Note that in this case the determinant is non-zero and the matrix is non-singular if the c components are distinct.

Other similar determinants such as

$$\begin{vmatrix} c_i & c_j & c_k \\ c_i^2 & c_j^2 & c_k^2 \\ c_i^3 & c_j^3 & c_k^3 \end{vmatrix} = c_i c_j c_k \Delta_{ijk},$$

$$\begin{vmatrix} 1 & 1 & 1 \\ c_i^2 & c_j^2 & c_k^2 \\ c_i^3 & c_j^3 & c_k^3 \end{vmatrix} = (c_i c_j + c_i c_k + c_j c_k)\Delta_{ijk},$$

$$\begin{vmatrix} 1 & 1 & 1 \\ c_i & c_j & c_k \\ c_i^3 & c_j^3 & c_k^3 \end{vmatrix} = (c_i + c_j + c_k)\Delta_{ijk}, \tag{C.5}$$

can be factored into a product that includes Vandermonde determinants. Consider Cramers rule applied to

$$\begin{vmatrix} a & 1 & 1 & 1 \\ b & c_i & c_j & c_k \\ c & c_i^2 & c_j^2 & c_k^2 \\ d & c_i^3 & c_j^3 & c_k^3 \end{vmatrix}$$
$$= a\begin{vmatrix} c_i & c_j & c_k \\ c_i^2 & c_j^2 & c_k^2 \\ c_i^3 & c_j^3 & c_k^3 \end{vmatrix} - b\begin{vmatrix} 1 & 1 & 1 \\ c_i^2 & c_j^2 & c_k^2 \\ c_i^3 & c_j^3 & c_k^3 \end{vmatrix} + c\begin{vmatrix} 1 & 1 & 1 \\ c_i & c_j & c_k \\ c_i^3 & c_j^3 & c_k^3 \end{vmatrix} - d\begin{vmatrix} 1 & 1 & 1 \\ c_i & c_j & c_k \\ c_i^2 & c_j^2 & c_k^2 \end{vmatrix} \tag{C.6}$$

where swapping the columns keeps things the same.

The expressions using Vandermonde determinants (C.1), such as (C.5), can give useful relations such as

$$\frac{\Delta_{ijkl}}{\Delta_{ijk}} = (c_i - c_l)(c_j - c_l)(c_k - c_l), \tag{C.7}$$

that appear throughout this thesis.

Appendix D

Performance of some IVPs in comparison to the values of selected $\overline{\text{PEC}}$s

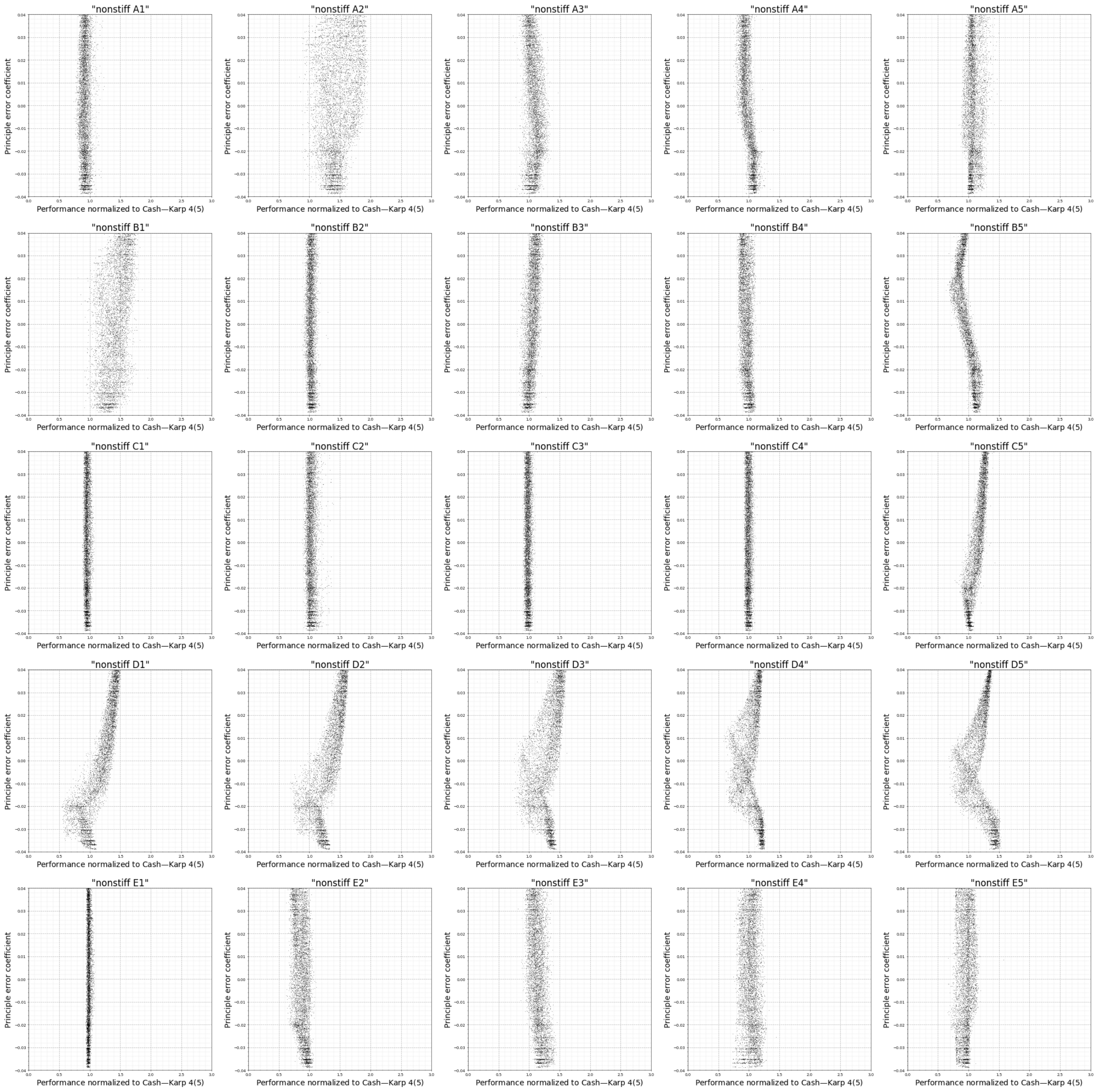


Figure D.1: The performance of the $5(4)_{6(6)\mathrm{C}(2)}$ family when solving the IVPs from the “DE test set” in comparison to different values of the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$.

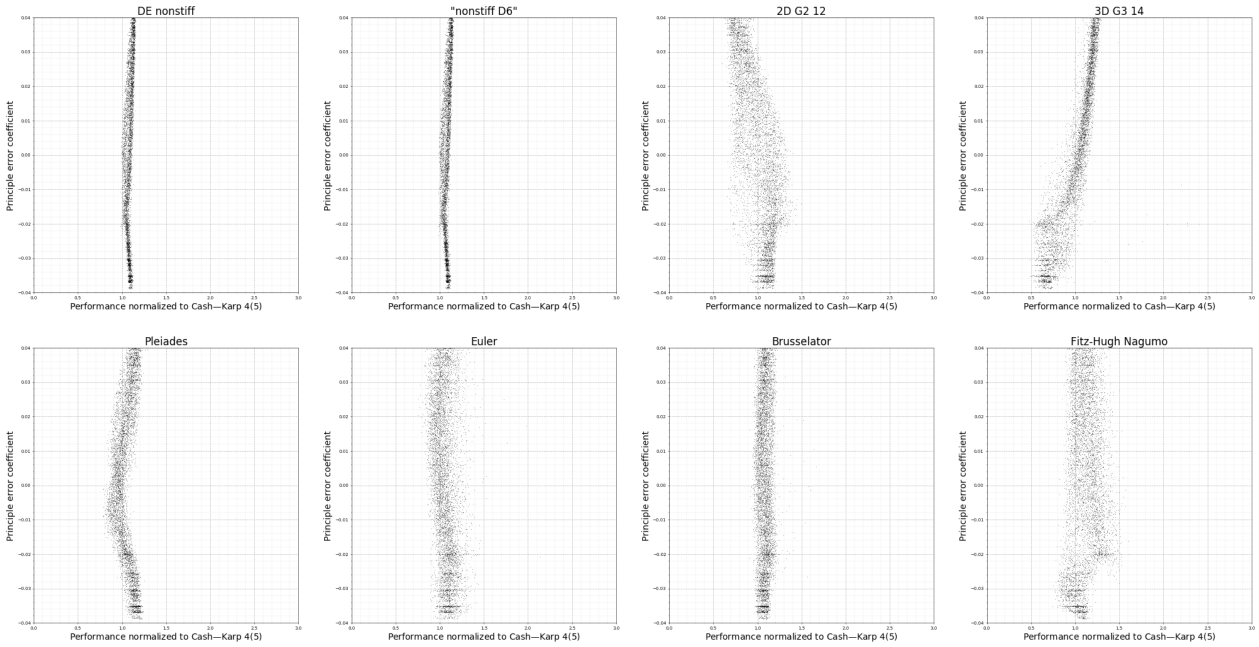


Figure D.2: The performance of the $5(4)_{6(6)\mathrm{C}(2)}$ family when solving other selected IVPs in comparison to different values of the $\overline{\mathrm{PEC}}$ (2.48) corresponding to $[\tau[[\tau^2]]]$.

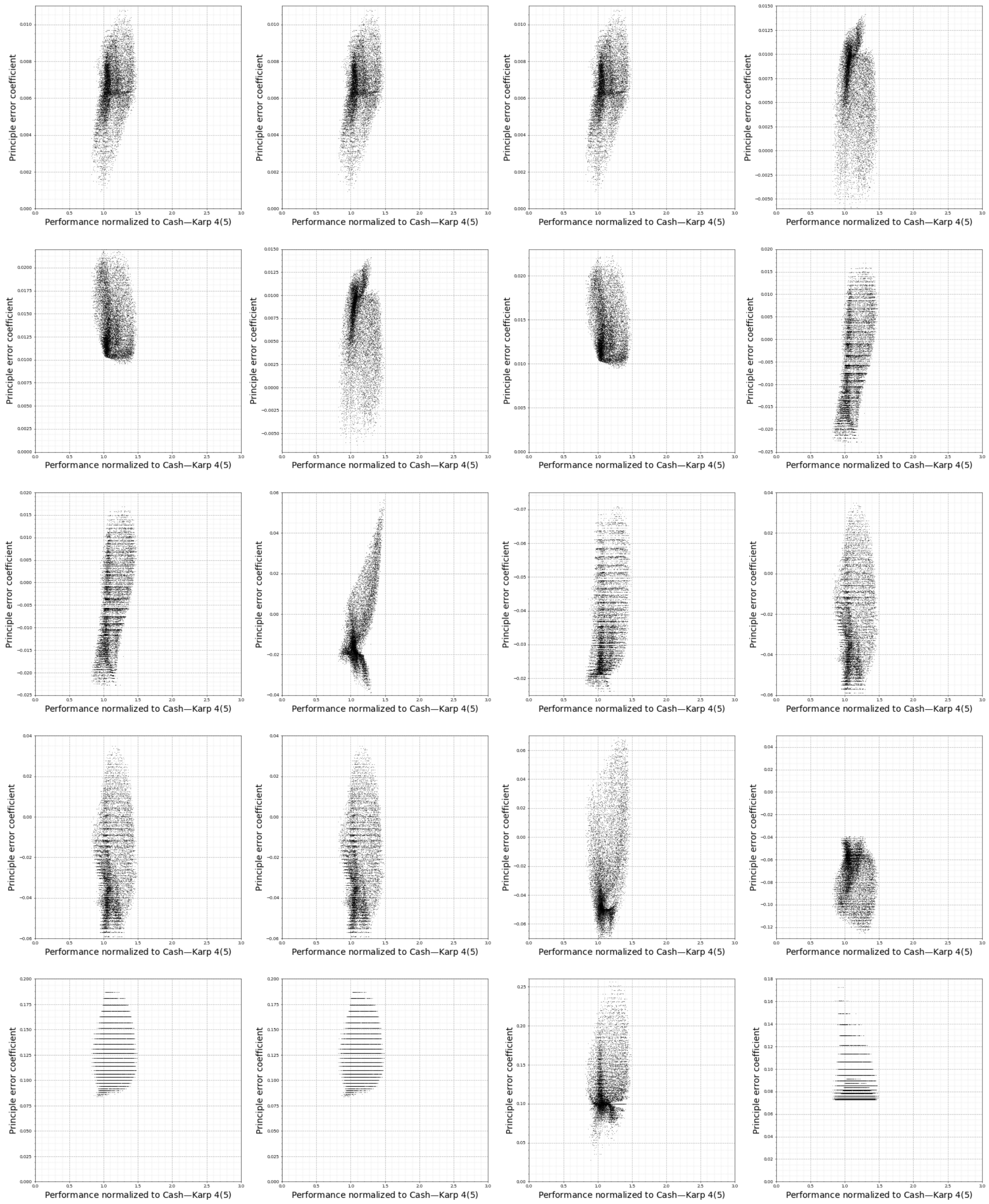


Figure D.3: The performance of the $5(4)_{6(6)\mathrm{C}(2)}$ family when solving the IVPs from the "non-stiff D" class (2.13) in comparison to different values of the sixth-order $\overline{\text{PEC}}$s (2.48). The ordering used for the PECs is left-to-right top-to-bottom in the same order as Table B.2.