Adaptive Exponential Runge–Kutta Pairs for Stiff Differential Equations

by

Thoma Zoto

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Applied Mathematics

Department of Mathematical and Statistical Sciences
University of Alberta

© Thoma Zoto, 2022

Abstract

Numerical methods are one of the most important aspects in the computer modelling of physical phenomena governed by differential equations. In order to study these phenomena precisely and in a timely manner, we need to use robust and efficient numerical methods for solving the underlying differential equations. The aim of exponential integrator methods is to provide robustness when solving stiff differential equations and the aim of this work is not only to construct new exponential integrators, but also to introduce ideas that allow for efficient implementation of such methods.

An extensive discussion on stiffness, concluding with a definition of numerical stiffness, is given in Chapter 2. Exponential integrator methods as well as the order conditions that need to be satisfied by them are presented in Chapter 3. Embedded methods and step-size adjustment is discussed in Chapter 4, together with a new optimized embedded exponential integrator method denoted ERK43ZB. In Chapter 5 we present a novel idea involving the Schur decomposition that allows for a substantial increase in the efficiency and memory usage of exponential integrators, in particular embedded exponential integrators. The thesis ends with two Mathematica scripts given in Appendix A and B that are used to check the order of an exponential method and to systematically construct new optimized embedded exponential methods, respectively.

To my dear parents.

Acknowledgements

I would like to show my deep appreciation for my supervisor, Professor John C. Bowman, for his support and guidance during my studies for this thesis. His energy, kindness, and deep knowledge of numerical algorithms, were undoubtedly a driving force to push me every time I would get stuck.

All the figures were generated using Asymptote: The Vector Graphics Language.

Table of Contents

1 Introduction **1**

2 Stiffness Definition **4**

2.1 Classical Runge–Kutta methods 4

2.2 First taste of stiffness 7

2.3 Stiffness in higher dimensions. 11

2.4 Attempts at defining stiffness 17

2.5 Definition of numerical stiffness 29

3 Exponential Integrator Methods **36**

3.1 Exponential integrators 36

3.2 High-order exponential integrators 41

4 Embedded Methods **48**

4.1 Classical embedded methods 48

4.2 Exponential embedded methods 50

5 Examples and Implementation **59**

5.1 Implementation details 59

5.2 Examples and applications 65

6 Conclusion and Future Work 76

Bibliography 79

A Mathematica script to check order conditions 83

B Mathematica script to construct stiff embedded ERK methods 88

Index 111

List of Figures

2.1 Error when solving $\frac{dy}{dt} = -Ly$ with explicit and implicit Euler methods. 8

2.2 Error when solving $\frac{dy}{dt} = -20y$ with explicit and implicit Euler methods. 10

2.3 Graphical interpretation of explicit and implicit Euler methods. . . . 11

2.4 Slope field of $\frac{dy}{dt} = -20y$. 12

2.5 Component-wise plot and phase plot of solution 2.11. 13

2.6 Numerical solutions of System 1 and System 2 computed with the Runge–Kutta–Fehlberg method. 15

2.7 Plots of nearby solution curves for System 1 and System 2. 25

2.8 Plots of nearby solution curves for System 1 and System 2 zoomed in. 26

2.9 Slope field plots for System 1 and System 2. 27

2.10 Slope field plot of $\frac{dy}{dt} = 5(y - t^2)$. 29

2.11 Slope field plots of Equation 2.24. 32

2.12 Slope field plots of Equation 2.27. 33

3.1 Error when solving $\frac{dy}{dt} = y^2 - Ly$ with explicit Euler, implicit Euler, integrating factor Euler and exponential Euler. 40

5.1 Error at $t = 1$ when solving Equation 5.18 with ERK4HO5M, ERK4HO5V, and RK4. 66
5.2 Error when solving Equation 5.19 with ERK4KM, ERK4CMM, ERK4HO5M, and ERK43ZBM. 68
5.3 Error when solving Equation 5.22 with ERK4KM, ERK4CMM, ERK4HO5M, and ERK43ZBM. 69
5.4 Error comparison when solving Equation 5.22 with and without the Schur decomposition. 71
5.5 Error comparison when solving Equation 5.22 with and without the Schur decomposition. 72
5.6 Error comparison of the third-order estimates for Equation 5.24 from $t = 0$ to $t = 3$. 73
5.7 Evolution of the error for Equation 5.24 from $t = 0$ to $t = 3$. 74
5.8 Evolution of the error for Equation 5.26 from $t = 0$ to $t = 30$. 75

Chapter 1

Introduction

Many scientific phenomena are modelled by time-dependent differential equations and since we cannot solve many of these equations algebraically, we turn to numerical methods for approximating their solutions. Obviously, we want to get the approximation of the solution as fast as possible and as close to the exact solution as possible. To achieve this we could either use a faster computer for running the simulations or use a method that is much better than its competitors at providing an approximation to the solution. Exponential integrators are a family of numerical methods for the time integration of an equation like:

$$\frac{dy}{dt} = F(t, y) - Ly, \quad y(0) = y_0. \tag{1.1}$$

where L is a number in the case of a scalar ordinary differential equation (ODE), or a constant matrix in the case of a system of ODEs. The term $F(t, y)$ is usually an arbitrary nonlinearity that can also be a scalar or a vector.

The reason why we cannot use simpler explicit numerical methods, but have to turn to these seemingly more complicated ones, is that Equation 1.1 is what we call

a stiff equation. Not only that, but it is a special type of stiff equation, because most of its stiffness comes from the linear term. We discuss in great detail what stiffness is in Chapter 2 as it is utterly important to understand how and why exponential integrators do not suffer from stiffness coming from the linear term, while other explicit methods do.

Note that if $F(t, y) = 0$, then the solution of Equation 1.1 is an exponential function. In this case, although a numerical method, an exponential integrator would compute the exact solution of the equation. In the more general case where $F(t, y) \neq 0$, an exponential integrator will deal exactly with the linear part of Equation 1.1 and approximate the nonlinearity. When $L = 0$, exponential integrator methods reduce to classical Runge–Kutta methods so from now on we will refer to this class of methods as exponential Runge–Kutta methods. We will go over their history in Chapter 3 as well as provide a discussion on the *stiff-order conditions* used to formally construct exponential Runge–Kutta methods. Unlike their classical counterparts, these conditions are much more restrictive and this hints that it will be harder to construct ERK methods than to construct classical RK methods. The conditions were derived by Hochbruck & Ostermann [2005] for methods up to stiff-order four and by Luan & Ostermann [2014] for methods up to stiff-order five. Some delicacies about how to properly solve these conditions are discussed in Chapter 3 and a Mathematica script to check if the conditions hold for a given ERK method is given in Appendix A. At the end of Chapter 3, we provide a discussion on embedded ERK methods. Like their classical counterparts, they allow for automatically changing the step size during the integration in order to keep the error within a range specified by the user. A Mathematica script to derive stiff $(4, 3)$ embedded ERK methods is given in Appendix B.

As their name suggests, exponential Runge–Kutta methods rely heavily on the

computation of exponential functions in order to approximate the solution of Equation 1.1. This is not a big issue if we are attempting to solve a scalar equation, but it needs to be properly taken care of in the case of systems of ODEs. The implementation is the most tricky when the matrix L is a general nondiagonal matrix as calculating the exponential of a general matrix is both tricky to perform robustly and computationally expensive [Moler & Van Loan 2003]. In Chapter 5, we mention common ways of calculating the exponential of a general matrix and then proceed to describe a new way of implementing exponential Runge–Kutta methods that only requires exponential of diagonal matrices.

Although this work deals primarily with the case when Equation 1.1 is a given system of ODEs, in practice such an equation is what we get after we discretize the spatial operator of a given partial differential equation. For example, Cox & Matthews [2002] were attempting to numerically solve partial differential equations (PDEs) with high-order spatial derivatives when they introduced their exponential Runge–Kutta method (also called exponential time differencing method). Further examples of PDEs whose spatially discretized versions are equivalent to Equation 1.1 are reaction-diffusion equations and the incompressible Navier-Stokes equations in two and three spatial dimensions. These equations satisfy the theoretical assumptions that Hochbruck & Ostermann [2005] used to derive their stiff-order conditions [Henry 2006] [Lunardi 1996]. Exponential Runge–Kutta methods are thus applicable in a wide range of practical problems. In Chapter 5, we look at an example of the heat equation with a source term. We solve that problem with a variety of exponential Runge–Kutta methods, including the one derived in this work, and provide error plots to compare the methods.

Chapter 2

Stiffness Definition

2.1 Classical Runge–Kutta methods

Classical Runge–Kutta (RK) methods are a family of single step methods designed to solve systems of differential equations, first introduced around the year 1900 by Runge [1895] and Kutta [1901]. To see how they work, we consider the following system of ODEs:

$$\frac{dy}{dt} = f(t, y), \quad y(0) = y_0. \tag{2.1}$$

The RK methods can be of two types, explicit or implicit, and we will focus in this thesis on explicit methods. Implicit methods, while they have better stability properties, are much more computationally expensive (especially when the system of ODEs is large), because they need to use an iterative method (like Newton's method) at each step in order to solve a nonlinear equation. Since we are focusing on explicit methods, we can write a general s-stage Runge–Kutta method for solving

Equation 2.1 as

$$y_n^{i+1} = y_n^0 + h \sum_{j=0}^{i} a_{ij} f(t_n + c_j h, y_n^j) \quad i = 0, \ldots, s-1, \tag{2.2}$$

where

$n = 0, 1, \ldots;$

$y_0^0 = y_0;$

$y_n^s = y_{n+1};$

h is called the time step;

$t_n = nh;$

a_{ij} are called the Runge–Kutta weights;

c_j are called the step fractions for stage j.

For $n \geq 1$, $y_n^0 = y_{n-1}^s$ is the approximation of the solution at time nh, also denoted by y_n. The weights of the method can be nicely organized in what is called a Butcher tableau (Table 2.1).

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_1 & a_{00} & & & & \\
c_2 & a_{10} & a_{11} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_{s-1} & a_{(s-2)0} & \cdots & \cdots & a_{(s-2)(s-2)} \\
\hline
1 & a_{(s-1)0} & \cdots & \cdots & a_{(s-1)(s-1)}
\end{array}$$

Table 2.1: General Runge–Kutta tableau

The most simple and well known Runge–Kutta method is the explicit Euler

method. It was first derived by Leonhard Euler in the work titled *Institutiones Calculi Integralis* in 1770 with the motivation coming from a particle moving with a given velocity for a specified amount of time. However, the simplest way to derive the explicit Euler method is by approximating the derivative in Equation 2.1 by the forward difference formula

$$\frac{y(t_n + h) - y(t_n)}{h} \approx f(t_n, y(t_n)) \tag{2.3}$$

and so the explicit Euler written in the form 2.2 becomes

$$y_{n+1} = y_n + hf(t_n, y_n). \tag{2.4}$$

Derived this way, it was completely arbitrary to evaluate the right-hand side at $y(t_n)$ and if we choose to evaluate it at $y(t_n + h)$, then we get what is called the implicit Euler method:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}). \tag{2.5}$$

Comparing 2.4 and 2.5, we see that the implicit method is much more computationally expensive since in order to solve for y_{n+1} we need to find the root of a nonlinear equation.

A Runge–Kutta method is called of order p if the error satisfies the following:

$$\left| y_n - y(t_n) \right| = Ch^p \tag{2.6}$$

for some constant C, where $y(t_n)$ is the exact solution at time t_n. In this sense both the explicit and implicit Euler method are first-order methods. However, in practice we want to use methods that are much more accurate than that. To get those

methods, there is a well established theory with constraints on the weights of the method called *order conditions*. This thesis is not primarily focused on classical Runge–Kutta methods and we are not including the classical order conditions here, but a detailed derivation of them can be found in the book by Butcher [2016]. The most widely used classical RK method is the "original" Runge–Kutta method [Kutta 1901], a fourth-order method we denote by RK4 whose Butcher tableau is in Table 2.2.

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

Table 2.2: RK4 tableau

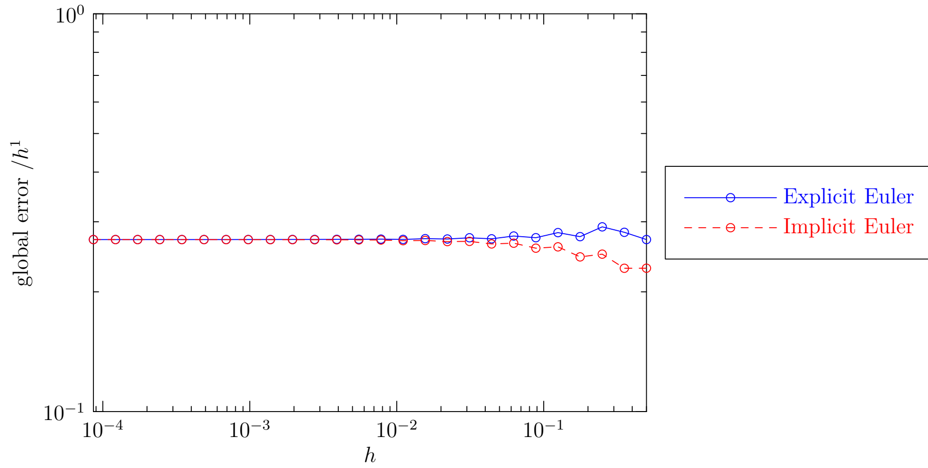
2.2 First taste of stiffness

Although both implicit and explicit RK methods were designed to solve all problems of the type 2.1, there is a particular $f(t, y)$ where the results differ significantly:

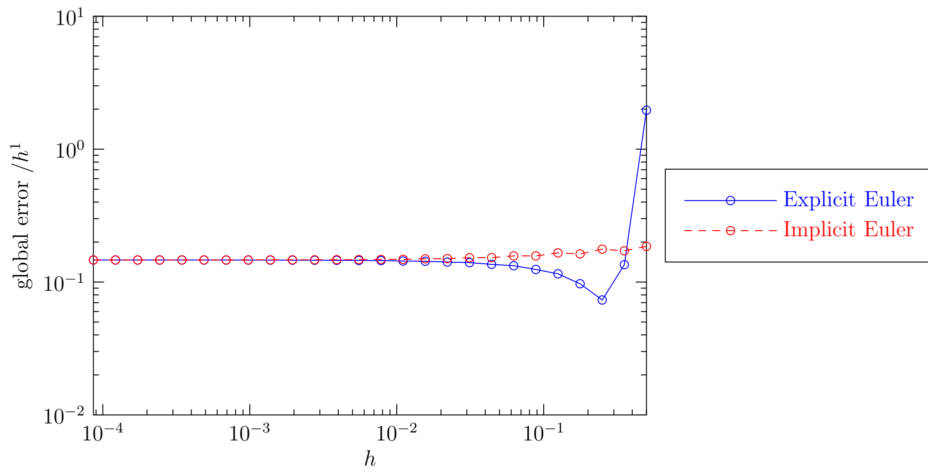
$$\frac{dy}{dt} = -Ly, \quad y(0) = y_0 \neq 0. \tag{2.7}$$

The Equation 2.7 is a simple equation that brings out fundamental differences between explicit and implicit Runge–Kutta methods. Its exact solution is given by

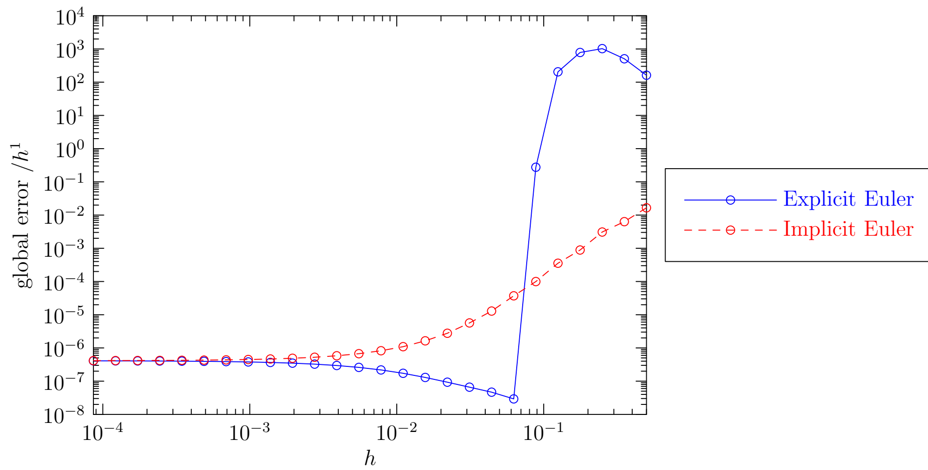
$$y = y_0 e^{-tL}. \tag{2.8}$$



(a) $L = 2, \quad y_0 = 1.$



(b) $L = 4, \quad y_0 = 1.$



(c) $L = 20, \quad y_0 = 1.$

Figure 2.1: Error when solving $\frac{dy}{dt} = -Ly$ with explicit and implicit Euler methods.

We solve Equation 2.7 over a sequence of time steps for three different values of L, using the implicit Euler 2.5 and explicit Euler 2.4 methods. We calculate the error for each method and then plot it in the three subfigures of Figure 2.1. Note that we have not plotted the actual global error of the method, but the global error divided by the time step (remember that $e_n = Ch^p$, where h is the time step and C does not depend on h). This is a much more informative way of plotting the error, because we are essentially plotting the constant C and can very easily tell the order of the method in discussion by simply looking if the plotted error line is horizontal or not.

When $L = 2$ in Equation 2.7, we see from Figure 2.1a that both methods retain their theoretical order of one. However, when we increase L to 4, the explicit Euler starts showing signs of stability issues if the time step is greater than 0.125 even though the implicit Euler fully retains order one. When we take it to the extreme and set $L = 20$, we see in Figure 2.1c that the stability issue for the explicit Euler starts at $h = 0.0625$. That is, if we were to solve Equation 2.7 with the explicit Euler method, we would need to use a time step smaller than 0.0625. On the other hand, implicit Euler appears to have order higher than one for about one third of the time steps considered. Indeed, while explicit Euler fails badly for large L, implicit Euler seems to behave better at large time steps than would be expected from the theoretical order. We might be tempted to say, by Figure 2.2, that the order of the method for time steps ranging from $h = 0.03125$ to $h = 0.5$ is four instead of one. However, upon closer inspection of how much error the implicit Euler method has for large time steps, we can conclude that it suffers a drop in accuracy despite remaining stable. The discrepancy between implicit and explicit methods in dealing with certain equations is a manifestation of numerical stiffness. For many years, stiffness has even been defined this way: a differential equation was considered

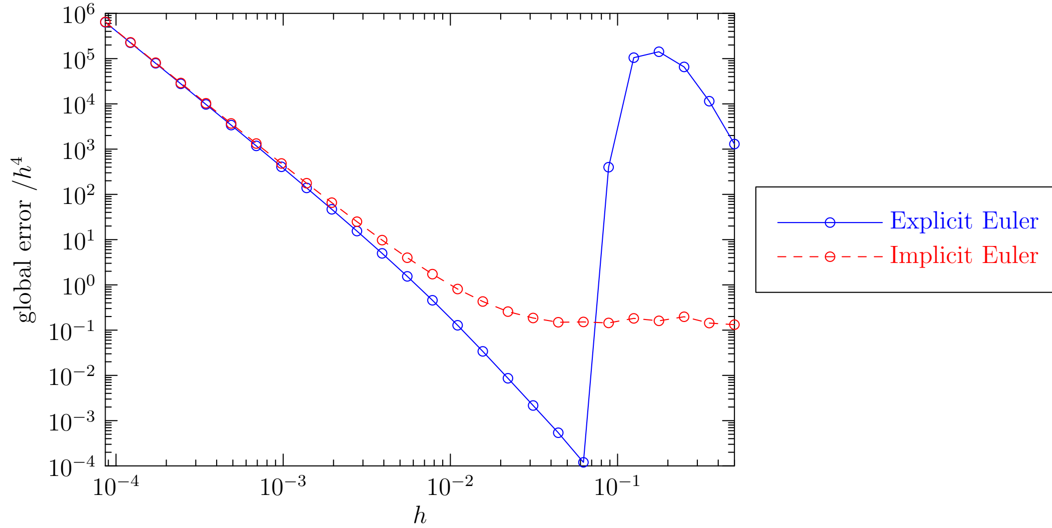


Figure 2.2: Error when solving $\frac{dy}{dt} = -20y$ with explicit and implicit Euler methods.

to be stiff when explicit methods performed much more poorly on it than implicit methods.

To conclude this section, we would like to present a graphical description of why the explicit Euler method performs so poorly compared to the implicit Euler method. Looking at 2.4 and 2.5 and remembering that $f(t_n, y_n)$ is the derivative of the solution at the point (t_n, y_n), we can understand these methods as "marching" forward in time in the direction of the tangent line to the solution curve passing through the point (t_n, y_n). At each time t_n, these methods compute an approximation to the exact solution that lies on a nearby solution curve. Combining these two understandings and looking at Figure 2.3, we can see why the implicit Euler does not require a small step size for an accurate computation while the explicit Euler method does. The implicit Euler "marches forward" in the direction of the tangent line to the nearby solution curve at the point (t_{n+1}, y_{n+1}) (red segment), while the explicit Euler "marches forward" in the direction of the tangent line to the nearby solution curve at the point (t_n, y_n) (green segment). We can see from the picture

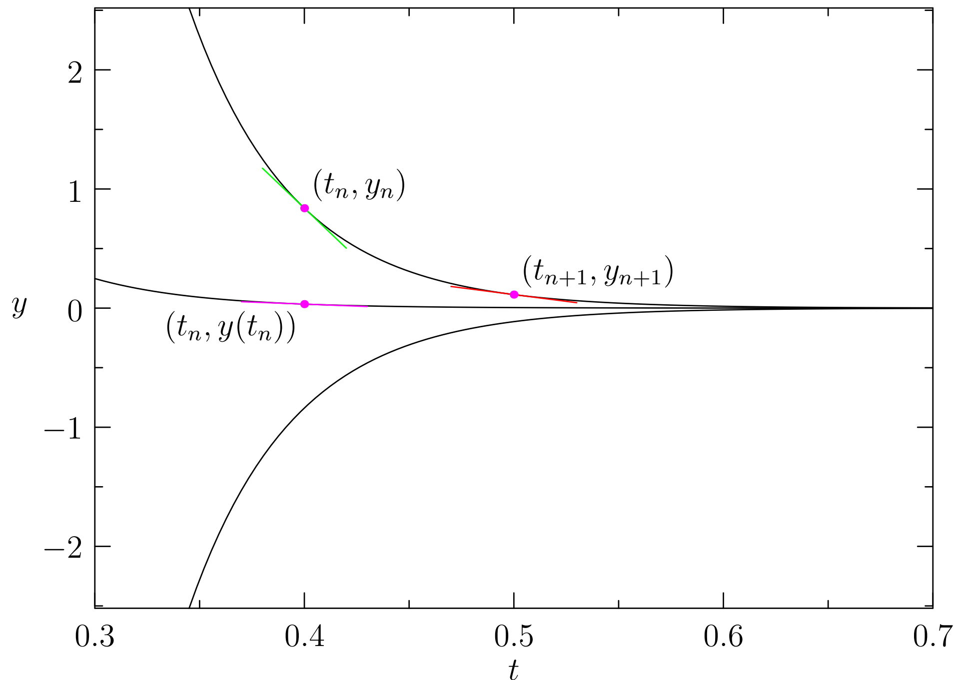


Figure 2.3: Graphical interpretation of explicit and implicit Euler methods.

that the red segment is much closer to the direction of the exact solution at the point $(t_n, y(t_n))$, unlike the green segment. One can conclude from this discussion that the bigger the step size, the more aligned the red segment will be with the direction of the exact solution at the point $(t_n, y(t_n))$. This explains why the implicit Euler method actually has improved accuracy for large step sizes when solving stiff equations. This phenomenon causes the explicit Euler method to only work for sufficiently small step sizes. For completeness, we also show the corresponding slope field in Figure 2.4.

2.3 Stiffness in higher dimensions.

Equation 2.7 is the simplest example problem that exhibits numerical stiffness as it is just one equation. However, there exist more complicated examples involv-

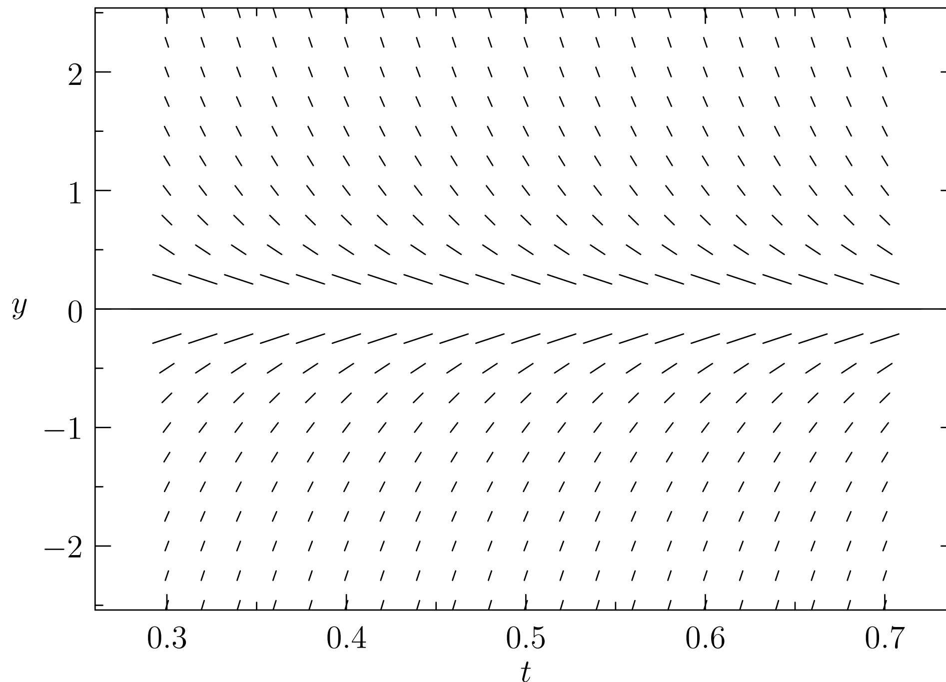


Figure 2.4: Slope field of $\frac{dy}{dt} = -20y$.

ing systems of ODEs that exhibit stiffness. Defining stiffness for a general system of ODEs is not an easy task. Perhaps the earliest such attempt comes from Curtiss & Hirschfelder [1952] and the most exhaustive discussion on stiffness is in Chapter 6 of Lambert [1991]. We present the problem of stiffness by following some example ODE systems from this book, while adding some more visuals to the ones presented there, in order to explain the stiffness phenomenon better. Consider the following systems.

System 1:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 2\sin(t) \\ 2(\cos(t) - \sin(t)) \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}, \tag{2.9}$$

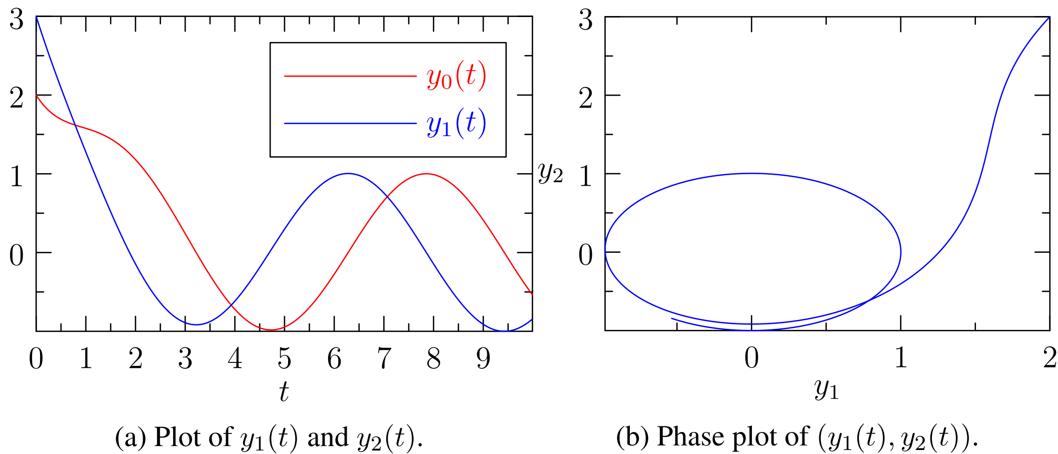
System 2:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 2\sin(t) \\ 999(\cos(t) - \sin(t)) \end{bmatrix}, \quad \begin{bmatrix} y_1(0) \\ y_2(0) \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}. \tag{2.10}$$

The particular solution for the given initial conditions is the same for both System 1 and System 2:

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = 2\exp(-t) \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}. \tag{2.11}$$

In Figure 2.5 we display a plot of both components of the exact solution against time as well as the plot of this particular solution in the phase space.



(a) Plot of $y_1(t)$ and $y_2(t)$.

(b) Phase plot of $(y_1(t), y_2(t))$.

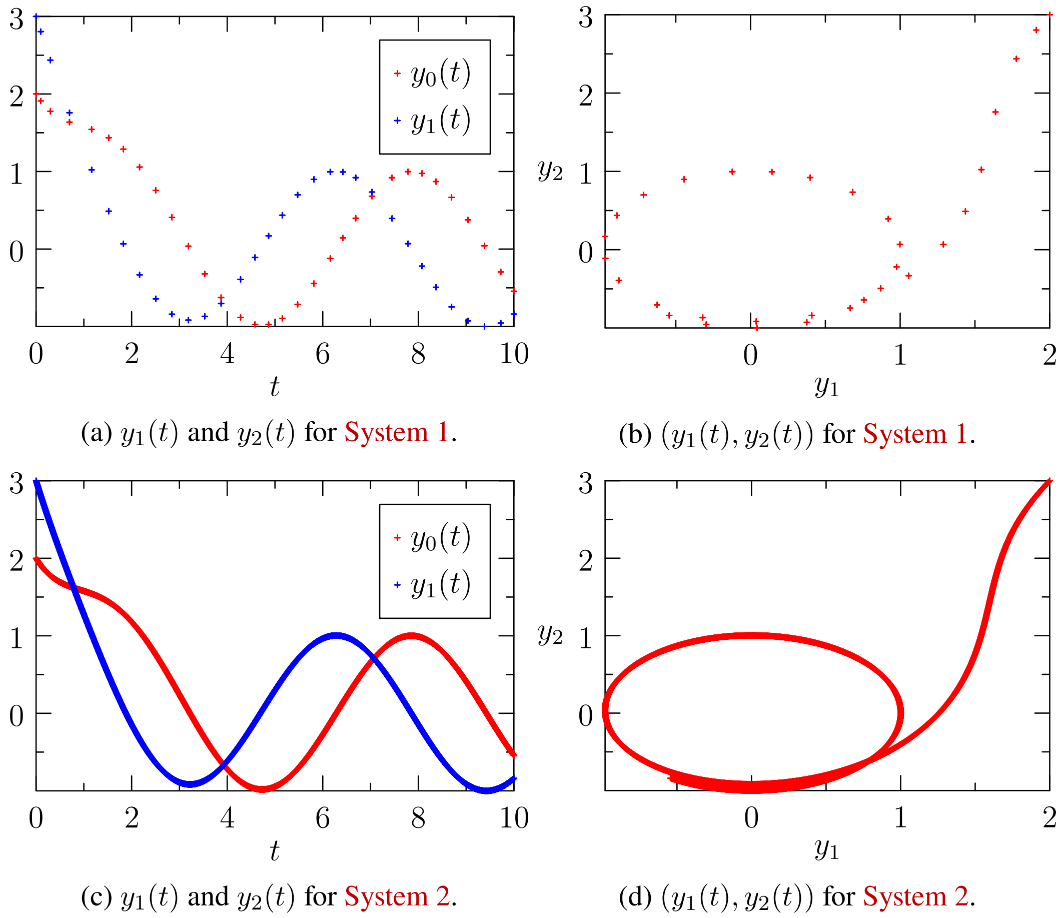
Figure 2.5: Component-wise plot and phase plot of solution 2.11.

We saw when solving Equation 2.7 with an explicit method that the more stiff the problem is, the smaller the time step for the explicit RK method will have to be in order for the method not to fail. Instead of adjusting the time step manually, in our numerical test stiffness test we will use *embedded Runge–Kutta* pairs, which provide high-order and low-order estimates of the solution that can be used to automate time

step adjustment. We could pick any embedded Runge–Kutta method, but in order to have some of the results from Lambert [1991] reproduced, we adopt the method used there. Therefore, in order to observe the first anomalies related to stiffness, we attempt to solve System 1 and System 2 with the Runge–Kutta–Fehlberg (RKF45) method. As in Lambert [1991], we select the initial step size 0.1 and error tolerance 0.01. We apply RKF45 with these parameter values to System 1 to obtain Figures 2.6a and 2.6b. The simulation uses step size 0.1 until $t = 2$ and step size 0.2 from $t = 2$ to $t = 10$, requiring 60 steps to create an accurate approximation of the analytical solution.

To reproduce the results of Lambert [1991], we also apply the RKF45 with the same starting parameter values to System 2. From the start, the method refuses the initial step size and reduces it to 0.0125 and after that the step size switches between 0.003125 and 0.0015625. For this system, it takes RKF45 3373 steps to create an accurate approximation of the analytical solution. This number is much larger than the number of steps needed to obtain the solution of System 1. If we consider the fact that RKF45 does 6 function evaluations at every step, then the algorithm made over 19878 extra function evaluations when approximating the solution of System 2. This heavy computation can also be seen graphically in Figure 2.6c and Figure 2.6d.

We have observed one ODE system that is solved easily by explicit methods and one ODE system that requires a huge computational effort when solved with an explicit method, despite both of them having the same analytical solution. We say that System 2 is more *stiff* than System 1. Lambert [1991] chooses the systems above on purpose to have the same exact solution, in order to demonstrate that the phenomenon of stiffness does not depend on the particular solution, but more broadly on the general solution of the system. This observation will become very



(a) $y_1(t)$ and $y_2(t)$ for System 1.

(b) $(y_1(t), y_2(t))$ for System 1.

(c) $y_1(t)$ and $y_2(t)$ for System 2.

(d) $(y_1(t), y_2(t))$ for System 2.

Figure 2.6: Numerical solutions of System 1 and System 2 computed with the Runge–Kutta–Fehlberg method.

useful later in this chapter.

Whenever the differential equation has a linear part, the general solution depends on the eigenvalues of that linear operator, so in our case it seems fitting to begin exploring the stiffness phenomenon by looking at the eigenvalues of this operator. In this spirit, we give below the solutions of System 1 and System 2 for general initial conditions:

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \kappa_1 \exp(-t) \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \kappa_2 \exp(-3t) \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}, \tag{2.12}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \kappa_1 \exp(-t) \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \kappa_2 \exp(-1000t) \begin{bmatrix} 1 \\ -998 \end{bmatrix} + \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}, \tag{2.13}$$

where κ_1 and κ_2 are constants that depend on the chosen initial conditions. The eigenvalues of the linear operator in System 1 are -1 and -3 while the eigenvalues of the linear operator in System 2 are -1 and -1000, giving an analytical hint of how different these two systems are. As in Lambert [1991], having a connection between stiffness and the eigenvalues of the system allows us to attempt an explanation of the numerical results above through the linear stability theory of the RKF45 method. For now, the reader not familiar with the stability of a numerical method can understand it as the restriction forced on the time step, given the eigenvalue with the largest norm of the problem. For real eigenvalues, the stability region of RKF45 is approximately $(-3, 0)$. This means that in order for the scheme to not *blow up* (generate unbounded approximations) when solving System 1, we must have $-3h \in (-3, 0)$. Therefore, we must pick a step size $h < 1.0$. On the other hand, in order for the scheme to remain stable when solving System 2, we must have $-1000h \in (-3, 0)$. Hence stability requires that $h < 0.003$ for the explicit RK scheme.

2.4 Attempts at defining stiffness

There are quiet a few different attempts at defining the phenomenon of stiffness in the literature. Each one of them encapsulates a great deal of stiff problems in practice, but as we will show, there are some examples that do not fit to most of the definitions used. In this section, we aim to exhibit both stiff systems that are not included by the commonly used definitions and systems that are included by the definitions, but are not stiff at all.

First, let us borrow the following example equation from Lambert [1991] to generalize what we examined in the previous section. Consider the following system

$$y' = Ay + \phi(t), \tag{2.14}$$

where $y \in \mathbb{R}^m$, $\phi \in \mathbb{R}^m$ and A is a constant $m \times m$ matrix whose eigenvalues are $\lambda_i \in \mathbb{C}, i = 1, 2, \ldots, m$, with corresponding eigenvectors $c_i \in \mathbb{C}^m, i = 1, 2, \ldots, m$. Equation 2.14 has the general solution

$$y(t) = \sum_{i=1}^{m} \kappa_i e^{\lambda_i t} c_i + \psi(t), \tag{2.15}$$

where κ_i are constants that depend on the imposed initial conditions and the integral $\psi(t)$ depends on $\phi(t)$. If

$$\operatorname{Re} \lambda_i < 0, \quad i = 1, 2, \ldots, m, \tag{2.16}$$

then each of the terms $e^{\lambda_i t} c_i \to 0$ as $t \to \infty$, so that the solution $y(t)$ asymptotically approaches $\psi(t)$ as $t \to \infty$. Note that we do not constrain the eigenvalues to be purely real: if λ_i is real then $e^{\lambda_i t}$ decays monotonically and if λ_i is complex then $e^{\lambda_i t}$

decays while oscillating. Because the exponential terms decay to zero very quickly, their sum is called the *transient part* of the solution. Conversely, $\psi(t)$ remains part of the solution for all time and hence is called the *persistent part* of the solution. This is where an idea describing stiffness that is commonly found in the literature has its roots: one encounters numerical stiffness when the solution is evolving in two different time scales.

Lambert [1991] denotes a *fast transient* to be the the exponential term corresponding to the eigenvalue with large $|\mathrm{Re}\,\lambda_i|$ (this term decays quickly), while denoting a *slow transient* to be the exponential term corresponding to the eigenvalue with small $|\mathrm{Re}\,\lambda_i|$ (this term decays slowly). Since matrices have a finite number of eigenvalues, then every system of ODEs will have a $\lambda_{\max}$ defining the fastest transient and a $\lambda_{\min}$ defining the slowest transient. In order to be able to reach the persistent solution, we must integrate numerically until even the slowest transient decays, but use a small enough time step to avoid numerical issues arising from the fastest transient. Hence, Lambert [1991] hints that the smaller $|\mathrm{Re}\,\lambda_{\min}|$ is, the longer we need to integrate with the small step size dictated by $|\mathrm{Re}\,\lambda_{\max}|$. Therefore, certain problems present the issue of having to use a very small step size and spend an unreasonable amount of time integrating the system. As defined above, the whole transient part of the solution needs to decay before we get to the part of the simulation mainly governed by the persistent state. In order to achieve this when solving System 2 with the RKF45 method, we had to integrate up to $t = 10$. Even though this is not such a long simulation, it took a lot longer to complete than expected, due to the small step size enforced by the eigenvalue of large modulus. Moreover, as Lambert [1991] notes, the term corresponding to $\lambda_{\max}$ becomes a negligible part of the solution shortly after the simulation has started, yet it dictates the use of the small step size until even the slowest transient has decayed.

From the discussion above, it seems that it is appropriate to use the ratio $\frac{|\mathrm{Re}\,\lambda_{\max}|}{|\mathrm{Re}\,\lambda_{\min}|}$ to quantify how stiff a system is. We will call this ratio the *stiffness ratio*. One of the first definitions given for stiffness is the following:

Statement 2.1. *A linear constant coefficient system is stiff if all of its eigenvalues have negative real part and the stiffness ratio is large.*

This statement is widely used in the literature as a definition of stiffness. Although it works well for many examples, we will be borrowing two examples from [Lambert 1991] and show that this is not a very satisfactory definition. To System 1 and System 2, we add the following:

System 3:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ -1.999 & 0.999 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 2\sin(t) \\ 0.999(\cos(t) - \sin(t)) \end{bmatrix}, \tag{2.17}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \kappa_1 \exp(-t) \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \kappa_2 \exp(-0.001t) \begin{bmatrix} 1 \\ 1.999 \end{bmatrix} + \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}, \tag{2.18}$$

System 4:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -0.002 & 0.001 \\ 0.998 & -0.999 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} 0.002\sin(0.001t) \\ 0.999(\cos(0.001t) - \sin(0.001t)) \end{bmatrix}, \tag{2.19}$$

$$\begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix} = \kappa_1 \exp(-t) \begin{bmatrix} 1 \\ -998 \end{bmatrix} + \kappa_2 \exp(-0.001t) \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \begin{bmatrix} \sin(0.001t) \\ \cos(0.001t) \end{bmatrix}. \tag{2.20}$$

It is straightforward to calculate the eigenvalues of the linear term in the systems above and find that $\lambda_{\max} = -1$ while $\lambda_{\min} = -0.001$. Therefore, the stiffness ratio is equal to 1000 for System 2, System 3 and System 4. When the solution we are trying to approximate numerically does have a sharp decay, we would evidently be forced to use a small step size until we have integrated past that decay in order to capture its effect. However, as Lambert [1991] demonstrates with System 2, the issue seems to be that the numerical scheme is forced to use the small step size even when the fast transient causing that sharp decay is no longer a prominent part of the solution. Even more peculiar is the fact that when choosing initial conditions such that both the fast and the slow transient are not present in the exact solution, we still have to use the small step size dictated by the fast transient.

Following Lambert [1991], let us see what happens when we try to solve numerically System 3 with initial conditions $y(0) = [2, 3.999]^T$ (which leads to $\kappa_1 = \kappa_2 = 1$). Using RKF45 with the same parameters as used before, we see that the step size needed eventually becomes 0.4. At first, this seems like a problem unrelated to stiffness since the stability constraints for the explicit method will be satisfied without having to use an extremely small step size (this because none of the eigenvalues is of large modulus). Yet, as Lambert [1991] also found, if we want to integrate until we reach the persistent solution (say, to within 0.005$\%$ accuracy), we will have to integrate from $t = 0$ up to $t = 1010$ for a total of about 2500 steps. The number of steps taken to reach the persistent solution is on par with System 2, despite System 3 having no eigenvalues of large modulus. While this may at first appear to support the claim that stiffness can be measured by the stiffness ratio, Lambert [1991] disproves this by picking initial conditions such that the slow transient is not part of the solution. In this case, we will not have to integrate until a very slow transient has died and we will arrive at the persistent solution much quicker.

Lambert [1991] recognizes that the numerical behaviour of System 3 thus depends dramatically on the initial data of the problem. However, Statement 2.1 predicts stiffness whenever the stiffness ratio is large, independent of the initial conditions. In light of this, he calls such systems *pseudostiff*.

It might seem tempting to try and fix Statement 2.1 by requiring in addition that $|\mathrm{Re}\,\lambda_{\mathrm{max}}|$ must be very large. This has also been examined by Lambert [1991]: consider System 4, where $|\mathrm{Re}\,\lambda_{\mathrm{max}}| = 1$. The period of $\sin(0.001t)$ is 2000π (implying that the persistent solution for this system varies very slowly). According to Lambert [1991], we would expect to use a step size that is a multiple of 100. However, the transient part of the solution corresponding to the eigenvalue -1, forces the adaptive RKF45 algorithm to settle for a step size of 0.4, while the transient part of the solution due to the eigenvalue -0.001 takes too long to become negligible. Hence, the simulation takes many steps to achieve the persistent solution, just like with System 2. System 4 has been very cleverly picked by Lambert [1991], as it is a linear transformation of System 2. This confirms that one cannot escape stiffness with a change of variables. We therefore seek to define stiffness to include System 2 and System 4 but not System 1 and System 3.

Before we proceed to the next candidate statement for the definition of stiffness, let us observe the system $y' = Ay$, where A is the matrix from System 3. As Lambert [1991] states, this homogeneous system is indeed stiff. However, note that the persistent solution in this case is zero and it does not vary at all. This shows that the nature of the persistent solution affects stiffness and it is "the presence of a persistent solution which varies at a rate comparable with that of the fastest transient in System 3" that leads Lambert [1991] to make the distinction between pseudostiff systems and genuinely stiff systems. Therefore, we lose all hope of defining stiffness as something based only on the eigenvalues of the linear operator and must therefore

consider the system as a whole. This observation will come in handy in the later sections as well.

Another attempt to define stiffness, also discredited by Lambert [1991], is

Statement 2.2. *Stiffness occurs when stability requirements, rather than those of accuracy, constrain the step size.*

We recall that stability issues are encountered when there is an unbounded accumulation of error where as accuracy deals with the minimization of error. As in Lambert [1991], when solving System 2, the RFK45 method refused the initial step size of 0.1 from the begining. The estimated error was still too high even with the new step size, forcing the scheme to keep refusing and halving the step size until it was 0.0125. This contradicts Statement 2.2, since the RFK45 scheme ran into problems when solving the stiff system before we could speak about any possible accumulation. On the other hand, the same initial step size was changed by the algorithm much later for the nonstiff System 1.

Another definition that has been proposed is

Statement 2.3. *Stiffness occurs when some components of the solution decay much more rapidly than others.*

Although qualitative, this statement seems to be an improved version of Statement 2.1, as it looks at rates of change not only of the transient solutions, but also of the persistent solution. According to Lambert [1991], this is also the main issue with this statement. If we assume the persistent solution is zero (i.e. we are dealing with a homogeneous equation), then any component of the solution fits the hypothesis of Statement 2.3, but the conclusion of the statement will not necessarily follow as that would mean that all homogeneous equations are stiff!

We now give the statement adopted as definition by Lambert [1991].

Statement 2.4. *If a numerical method with a finite region of absolute stability, applied to a system with any initial conditions, is forced to use in a certain interval of integration a step size which is excessively small in relation to the smoothness of the exact solution in that interval, then the system is said to be stiff in that interval.*

This is a good definition if we want to just know by testing whether a system is stiff or not (i.e. we have to actually apply a method and pay attention when it fails). For a long time, stiff systems were solved numerically by first trying an explicit method and, if that failed, switching to an an implicit method. Although Statement 2.4 differentiates between pseudostiff and genuinely stiff systems, it is not very helpful when we want to define stiffness in a way that is helpful for using exponential integrators: we seek a definition that distinguishes between stiffness coming from a linear term and stiffness coming from a nonlinear term.

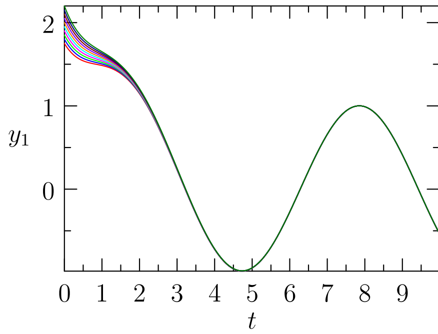
Lastly we consider a point of view on stiffness that was first introduced by Curtiss & Hirschfelder [1952] and described by Lambert [1991]. We will expand on these ideas to make them applicable to the stiff differential equations that we aim to solve with exponential integrators. Lambert selected initial conditions $y(0) = [2, 3]^T$ so that System 1 and System 2 have the same exact solution, in order to show that stiffness does not depend on some particular solution of the problem. Indeed it does not, so in order to see something more illuminating, we include the nearby solutions in our focus as well. In Figure 2.7 we have plotted the components as well as the phase curves corresponding to various initial conditions for both System 1 and System 2. As seen in Figure 2.7a and 2.7b, there is no anomaly in the plots of the first component of the solutions for both problems. However, we see in the plots of the second component of the solutions in Figure 2.7c and 2.7d that the nearby curves of the second component of the solutions approach each other at a much

faster rate in the case of System 2 than their counterparts in System 1. This anomaly can also be observed with the phase curves of both systems in Figure 2.7e and 2.7f.

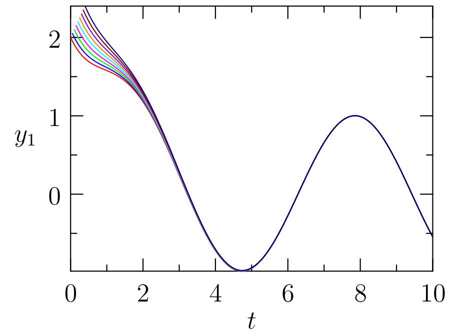
Lambert suggests to zoom in some interval, say $[0, 0.001]$, in Figure 2.7c and 2.7d, so we can better observe the rate at which the nearby curves of the second component of the solutions for these systems approach each other. In order to better generalize the discussion at the end of Section 2.2, we propose to zoom in Figure 2.7e and 2.7f, where the solution curves are to be seen as curves in the 2D phase space parametrized by t.

We see that the nearby phase curves are almost parallel to each other for the nonstiff System 1 and approach each other at a high angle for the stiff System 2. For example, for some t_n, the exact value of the solution for System 2 is $y(t_n) = [1.998, 2.99]$ and the numerical algorithm estimates it with some error to $y_n = [1.998, 3.01]$. The difference between $y(t_n)$ and y_n is negligible, but the difference between the slope field vector at $y(t_n)$ and the slope field vector at y_n is big (see Figure 2.7f). We can deduce from Figure 2.7e that a similar situation will yield the corresponding vectors to be almost colinear in the same direction. Recalling that the vectors mentioned above are given by evaluating the right hand side of the ODE at the points $y(t_n)$ and y_n, we can see that System 2 would pose problems for explicit Runge–Kutta methods. To get a more general idea of the slope field vectors for System 1 and System 2 see Figure 2.9.

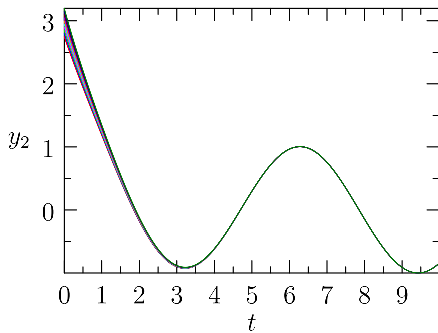
As described in Section 2.2 for the case of a scalar ODE, one remedy to this problem observed in System 2 is obviously to take very small step sizes when using an explicit method. In that case, the difference between $y(t_n)$ and y_n would decrease further and hence the vectors $f(t_n, y_n)$ and $f(t_n, y(t_n))$ will be almost colinear in the same direction. Another remedy is to use an implicit Runge–Kutta method. The discussion in Section 2.2 about why implicit Runge–Kutta methods can solve



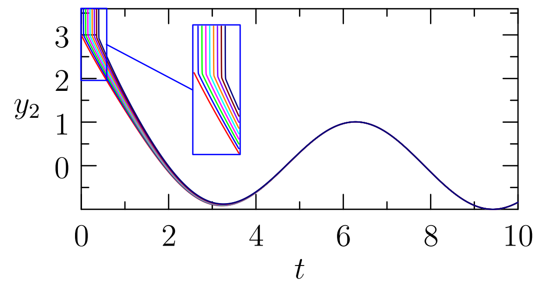
(a) $y_1(t)$ solution curves for System 1.



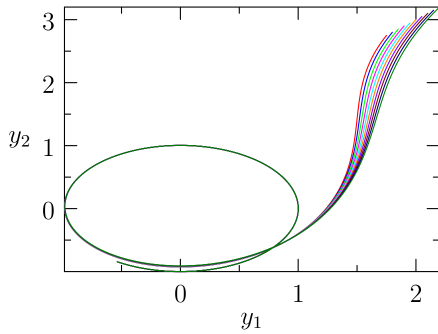
(b) $y_1(t)$ solution curves for System 2.



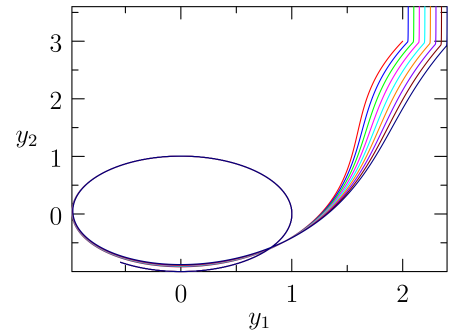
(c) $y_2(t)$ solution curves for System 1.



(d) $y_2(t)$ solution curves for System 2.

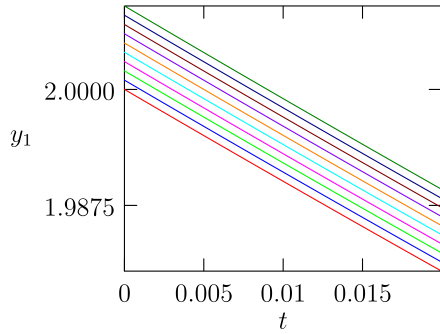


(e) Phase curves for System 1.

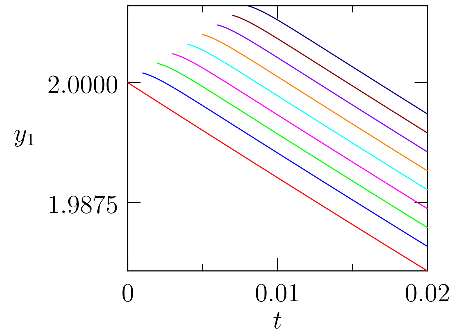


(f) Phase curves for System 2.

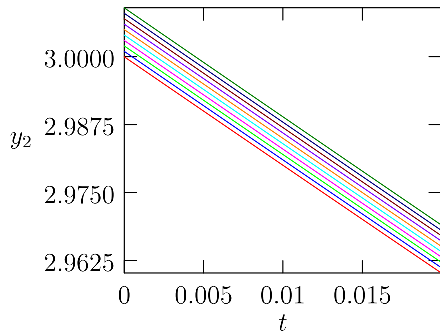
Figure 2.7: Plots of nearby solution curves for System 1 and System 2.



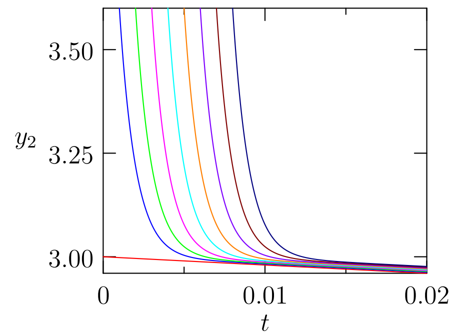
(a) $y_1(t)$ curves for System 1 (zoomed).



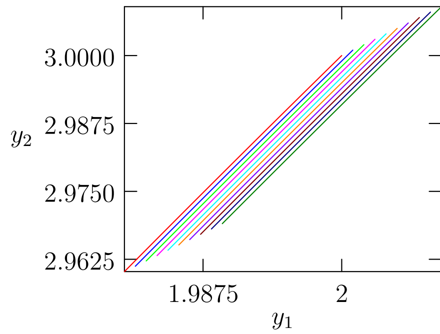
(b) $y_1(t)$ curves for System 2 (zoomed).



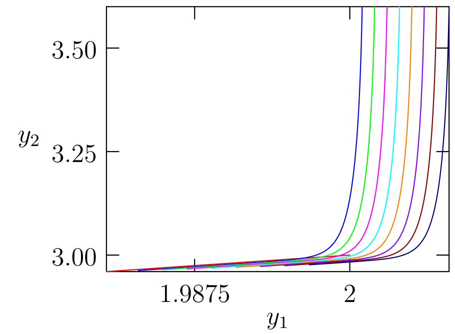
(c) $y_2(t)$ curves for System 1 (zoomed).



(d) $y_2(t)$ curves for System 2 (zoomed).

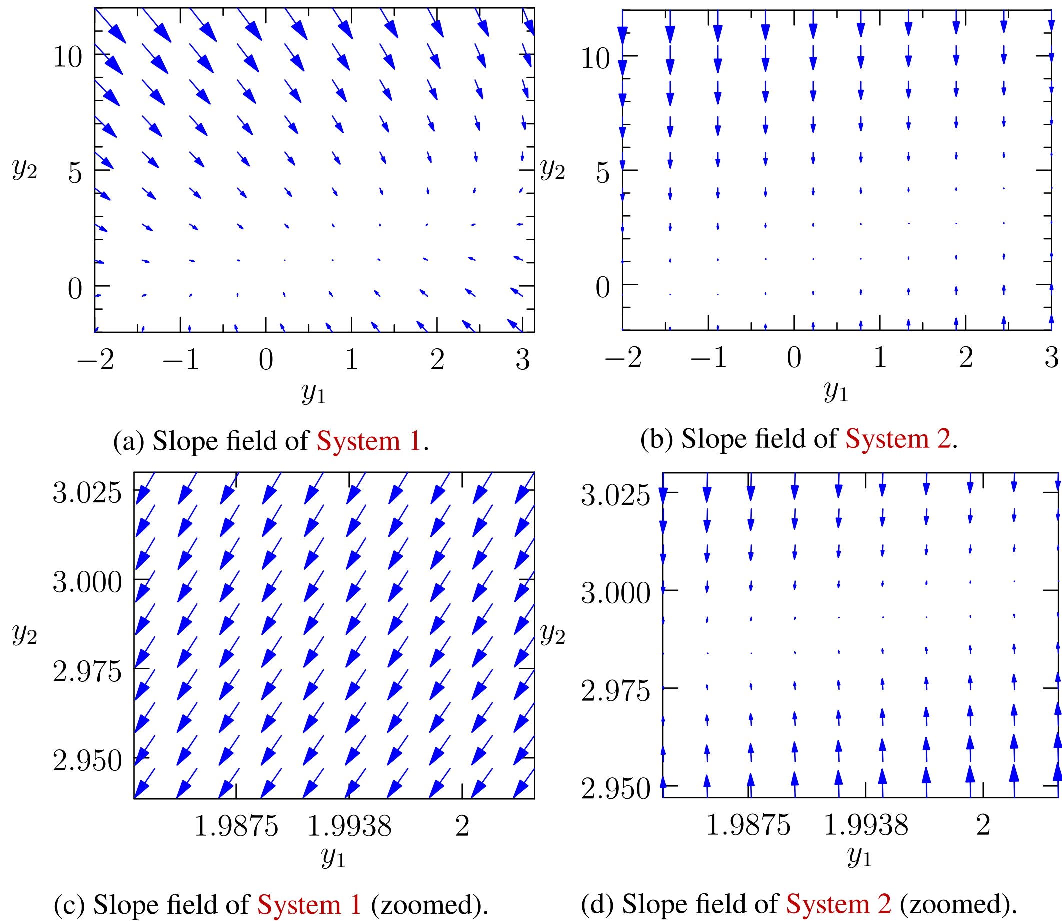


(e) Phase curves for System 1 (zoomed).



(f) Phase curves for System 2 (zoomed).

Figure 2.8: Plots of nearby solution curves for System 1 and System 2 zoomed in.



(a) Slope field of System 1.

(b) Slope field of System 2.

(c) Slope field of System 1 (zoomed).

(d) Slope field of System 2 (zoomed).

Figure 2.9: Slope field plots for System 1 and System 2.

stiff ODEs with large step sizes, while explicit methods cannot, can generalize to the case of systems of ODEs. Just like in the scalar ODE case, these families of methods can be understood as "marching forward" in some direction, given in this case by a vector in the slope field (instead of the slope of a tangent line like for scalar ODEs). The direction that implicit methods follow is closer to the direction that the exact solution evolves than the direction that explicit methods use. This allows implicit methods to take larger step sizes than explicit methods can take, circumventing numerical stiffness.

We can now give the last statement considered by Lambert [1991], first briefly

mentioned by Curtiss & Hirschfelder [1952], which will eventually lead us to our ultimate definition of stiffness.

Statement 2.5. *A system is said to be stiff in a given interval of time if, in that interval, the neighbouring solution curves approach the solution curve at a rate which is very large in comparison with the rate at which the solution varies.*

As Lambert [1991] points out for Statement 2.4, Statement 2.5 also includes the idea that the stiffness of a system will depend on where in the phase space is the numerical integration taking place. Moreover, using this statement we can indeed distinguish between pseudostiff and genuinely stiff systems. The reason why he does not prefer this definition is that it requires knowledge of at least two solutions of the system in order to decide whether nearby curves approach at a fast or slow rate to the particular solution of interest. As Curtiss & Hirschfelder [1952] describe and based on the connection we made to the slope field, however, we can translate the geometric phenomenon of nearby curves approaching at a fast rate to the more analytic interpretation that the function $f(t, y)$ has a large Lipschitz constant

$$\sup_{\substack{t_1 \neq t_2 \\ y_1 \neq y_2}} \frac{\left|f(t_1, y_1) - f(t_2, y_2)\right|}{\left|(t_1, y_1) - (t_2, y_2)\right|}.$$

Essentially, stiff systems are those for which “a small change in y leads to a large change in $f(t, y)$” [Lambert 1991]. He argues against this statement as well, with the reason being that it is not apparent how large (and relative to what) the Lipschitz constant should be. His reasoning to use Statement 2.4 as the definition for stiffness is justified in the setting when he was writing his book, because he was only interested in determining which systems are stiff with the least possible effort, in order to not waste limited resources trying to solve them with explicit methods. We are

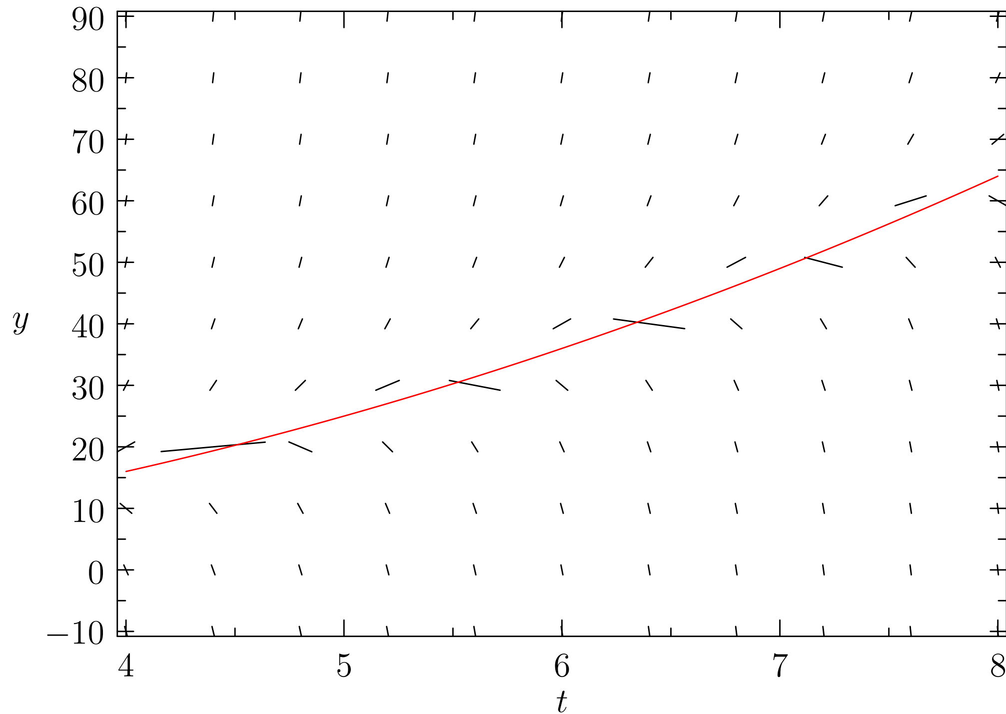


Figure 2.10: Slope field plot of $\frac{dy}{dt} = 5(y - t^2)$.

interested in exploring the definition of stiffness to specifically detect *linear stiffness* (numerical stiffness coming from the linear source term of the ODE system) so that we can apply an exponential integrator appropriately.

2.5 Definition of numerical stiffness

Before we present the statement that we will adopt as a definition for stiffness, let us examine in detail the example equation given by Curtiss & Hirschfelder [1952]:

$$\frac{dy}{dt} = 5(y - t^2). \tag{2.21}$$

The slope field for this equation is plotted in Figure 2.10. We can see both analytically and geometrically that "for a particular t and for large values of y the slope is very

large in the positive sense, while for small values of y the slope is very large in the negative sense; in the vicinity of $y = t^2$ the slope changes rapidly from large positive to large negative"[Curtiss & Hirschfelder 1952]. In order for the slope to change from large in the negative sense to large in the positive sense, it has to be equal to zero at the given t and for some y. Obviously this happens when $dy/dt = 0$. Therefore, one way to test for stiffness is to find the fixed point of the equation. Then, we compare dy/dt for a point $(t, \tilde{y})$ in the vicinity of a fixed point (sometimes the ODE system will have more than one fixed point for some values of t). Of course, this is a one-dimensional equation and the right-hand side is of the form $F(t) - Ly$, so it is easy to imagine what happens. However, the concept of a fixed point also makes sense for a system of ODEs and it can even be adapted for the more general equation (F no longer depends only on t):

$$\frac{dy}{dt} = F(t, y) - Ly = f(t, y). \tag{2.22}$$

Assume first that the equation 2.22 is one dimensional and that $F(y) = y^2 + t$ and $L = 6$. Then, $-6y + y^2 + t = 0$, so

$$y = 3 \pm \sqrt{9 - t} \tag{2.23}$$

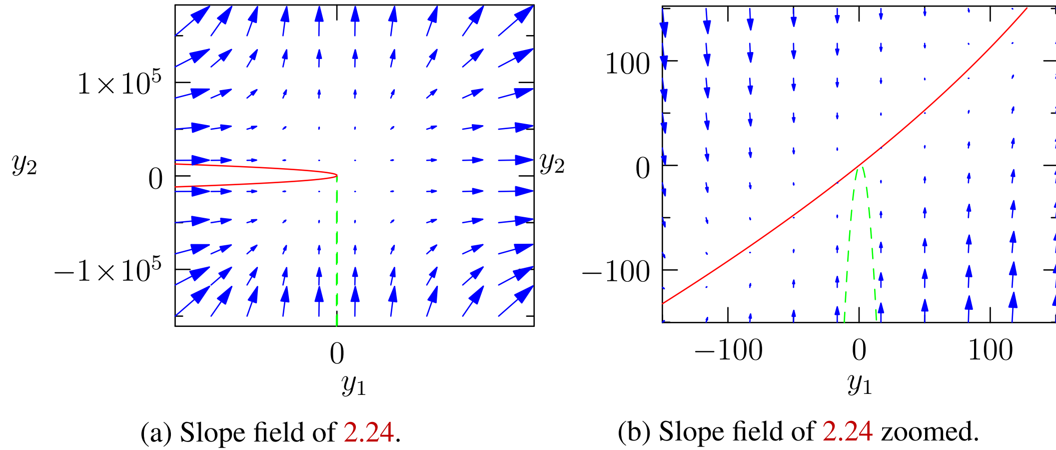
will be the fixed point at each t. To test how the slope changes, we fix t and calculate the corresponding y from equation 2.23 and then consider test values $\tilde{y}_1 \in (y - \varepsilon, y)$ and $\tilde{y}_2 \in (y, y + \varepsilon)$ for some sufficiently small ε. If either $f(t, \tilde{y}_1)$ or $f(t, \tilde{y}_2)$ differs drastically from $f(t, \tilde{y})$, we are dealing with a stiff problem.

Now let us assume that equation 2.22 is multidimensional. The methods we are mostly interested in (considered in Hochbruck & Ostermann [2005]) are invariant

under the transformation of 2.22 to the autonomous form of the equation (where f does not have an explicit time dependence), so we can restrict ourselves to such problems. Let us look at the 2 dimensional case first and consider the equation:

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} + \begin{bmatrix} y_1^2 \\ y_2^2 \end{bmatrix}. \tag{2.24}$$

Here things are a little harder to imagine as we no longer have just one ty plot to visualize the solution. The solution lives in a three dimensional space with the axes being t, y_1 and y_2. Hence, we can understand it as there being one ty_1 plot for each y_2 and one ty_2 plot for each y_1. Let us look at the slope field of the problem. We want to find the points of Equation 2.24 where dy_1/dt and dy_2/dt become zero (but not necessarily at the same time). Recall that $dy_1/dt = 0$ at a point (y_1, y_2) means that the corresponding vector in the slope field is parallel to the y_2 axis and that $dy_2/dt = 0$ at a point (y_1, y_2) means that the corresponding vector in the slope field is parallel to the y_1 axis. To observe the behaviour of stiffness in this scenario, we first find the equations for fixed points of the components of the system of ODEs 2.24; denote by C_1 the curve corresponding to $dy_1/dt = 0$ and by C_2 the curve corresponding to $dy_2/dt = 0$. Pick one of them, say C_2. Fix y_1 and calculate the corresponding y_2 from the equation for C_2. Consider some test values $\tilde{y}_{2a} \in (y_2 - \varepsilon, y_2)$ and $\tilde{y}_{2b} \in (y_2, y_2 + \varepsilon)$ for some sufficiently small ε. The system is stiff if the field vector at $(y_1, \tilde{y}_{2a})$ or $(y_1, \tilde{y}_{2b})$ points towards the direction of the field vector at (y_1, y_2) at an angle much greater than zero. This can be measured by the dot product of the vectors being considered. We repeat the process for the curve C_1 as well and see if we can observe any anomalies. Below are the equations for C_1



(a) Slope field of 2.24.

(b) Slope field of 2.24 zoomed.

Figure 2.11: Slope field plots of Equation 2.24.

and C_2 for the differential equation 2.24:

$$C_1: \quad 0 = -2y_1 + y_2 + y_1^2, \tag{2.25}$$

$$C_2: \quad 0 = 998y_1 - 999y_2 + y_2^2. \tag{2.26}$$

We plot these curves and the slope field in Figure 2.11. We note that stiffness manifests itself only in some parts of the phase. That is, stiffness can be a local phenomenon!

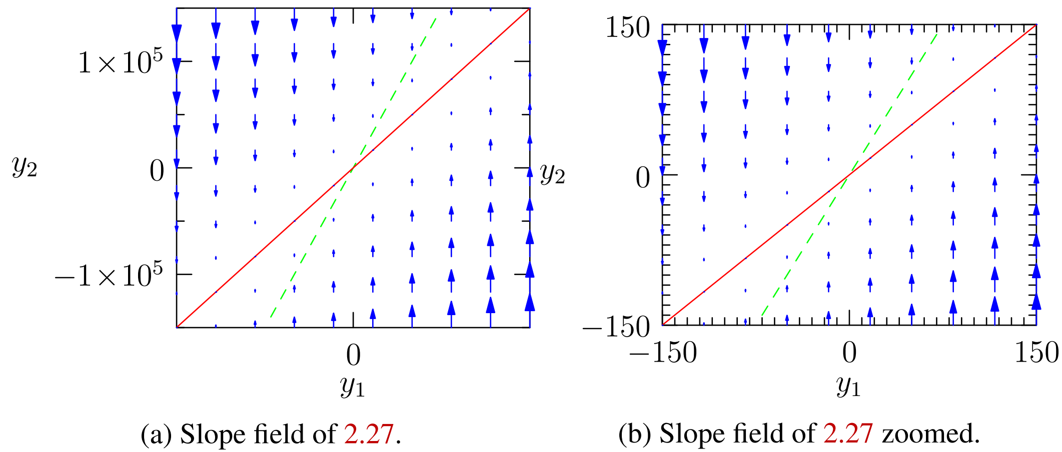
In contrast, the system obtained by removing the nonlinear terms from Equation 2.24,

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}, \tag{2.27}$$

is stiff everywhere. Here, the equations for the curves C_1 and C_2 are

$$C_1: \quad 0 = -2y_1 + y_2 \tag{2.28}$$

$$C_2: \quad 0 = 998y_1 - 999y_2; \tag{2.29}$$



(a) Slope field of 2.27.

(b) Slope field of 2.27 zoomed.

Figure 2.12: Slope field plots of Equation 2.27.

these are plotted together with the corresponding slope field in Figure 2.12. This is another confirmation that stiffness cannot be measured by properties of the linear operator only. This makes it crucial to have a definition that can recognize if the equation is stiff or not in the region of the phase space of interest.

The idea of testing the behaviour of the source function near fixed points provides a fast but heuristic way of determining if an equation is stiff in a particular region. However, there exists an alternative, more theoretical, definition of stiffness based on local Lyapunov exponents and curvature [Cartwright 1999]. Although Cartwright is mostly concerned with defining stiffness in chaotic systems, part of his work aligns quite well with the discussion here. In fact, we consider Cartwright's definition to be the preferred definition of stiffness as it is local and provides a way of quantifying the rate at which nearby solution curves approach the exact solution.

Stiff systems can be recognized by how fast the nearby curves approach a steady state and, as mentioned above, what characterizes the rate of separation or contraction of solution curves of a system of ODEs are the Lyapunov exponents. Since Lyapunov exponents give a global picture of the phase space, to investigate what happens locally we will need *local Lyapunov exponents*. Let $\sigma_i(t)$ be the principal

axes of an ellipsoidal ball evolving in time in phase space. The ith local Lyapunov exponent is

$$\gamma_i(\tau, t) = \lim_{\sigma_i(\tau)\to 0} \frac{1}{\tau} \ln \frac{\sigma_i(t+\tau)}{\sigma_i(t)}. \tag{2.30}$$

We hinted earlier that while the nearby solutions play a major role in whether a system of ODEs is stiff or not, the wiggliness of the solution curve at the section of interest in the phase space also plays an important role. Cartwright quantifies the latter by using the curvature

$$\kappa = \frac{y''}{(1+y'^2)^{3/2}}. \tag{2.31}$$

of the solution y. Cartwright [1999] states that "a system is stiff in a given interval if in that interval the most negative local Lyapunov exponent is large, while the curvature of the solution is small." Phrased like this makes the definition look more qualitative than it actually is. He makes an effort to quantify nonlinear stiffness (and hence linear stiffness too) by the ratio

$$R_{nl} = \frac{\left|\min_{1\le i\le n} \gamma_i(\tau, t)\right|}{\kappa(t)}. \tag{2.32}$$

being large, but he does not compare the ratio to anything and hence leaves it open to Lambert's argument against Statement 2.5 (large relative to what?).

Cartwright [1999] also states that the ratio R_{nl} could be averaged over the trajectory to give a measure of stiffness, however, that would bring us back to another problem that arises with using Lipschitz constants: being a global measure instead of a local one. Therefore, we slightly modify the definition given by Cartwright [1999]:

Definition 2.6. *A system is stiff in a given interval if in that interval the most negative*

local Lyapunov exponent is much larger in absolute value than the curvature of the solution curve.

Chapter 3

Exponential Integrator Methods

3.1 Exponential integrators

We saw in the previous chapter that when attempting to solve Equation 2.7 with the explicit Euler method, the numerical solution explodes for large time steps if L is big enough. We also saw that this issue does not occur when attempting to solve the equation with the implicit Euler method. In the literature, this is described as the phenomena of *numerical stiffness* and having now rigorously defined it, we investigate numerical methods that do not suffer from it. First introduced by Certaine [1960], these methods are called exponential integrators and they deal with the stiffness coming from the linear term by treating it exactly.

Let us return to Equation 1.1:

$$\frac{dy}{dt} = F(t, y) - Ly, \quad y(0) = y_0. \tag{3.1}$$

We can rewrite this equation as

$$\frac{d(e^{tL}y)}{dt} = e^{tL}F(t,y). \tag{3.2}$$

Integrating this equation over time step $[t_n, t_n + h]$, where $t_n = nh$, we obtain

$$y(t_n + h) = e^{-hL}y(t_n) + e^{-hL}\int_0^h e^{\tau L}F(t_n + \tau, y(t_n + \tau))\,d\tau. \tag{3.3}$$

If we define the function $G(t) = F(t, y(t))$, we can Taylor expand $G(t_n + \tau)$ around t_n in the above equation to obtain

$$y(t_n + h) = e^{-hL}y(t_n) + e^{-hL}\int_0^h e^{\tau L}\sum_{k=0}^{\infty}\frac{\tau^k}{k!}G^{(k)}(t_n)\,d\tau. \tag{3.4}$$

We can make this representation of the solution more compact by defining $\varphi_0(x) = e^x$ and

$$\varphi_k(-hL) = \frac{1}{h^k}\int_0^h \frac{\tau^{k-1}}{(k-1)!}e^{-(h-\tau)L}\,d\tau, \quad k \in \mathbb{N}. \tag{3.5}$$

Doing an integration by parts in 3.5, we get the following useful inductive relation for the φ_k functions:

$$\varphi_k(0) = \frac{1}{k!}, \tag{3.6}$$

$$\varphi_0(x) = e^x, \tag{3.7}$$

$$\varphi_{k+1}(x) = \frac{\varphi_k(x) - \frac{1}{k!}}{x} \text{ for } k \geq 0. \tag{3.8}$$

Therefore, the exact solution to Equation 3.1 becomes

$$y(t_n + h) = e^{-hL}y(t_n) + \sum_{k=0}^{\infty} h^{k+1}\varphi_{k+1}G^{(k)}(t_n). \tag{3.9}$$

Since this is the exact solution of Equation 3.1, the task of coming up with an exponential Runge–Kutta method becomes equivalent to approximating the infinite sum in 3.9 or approximating the integral in 3.3. The simplest approximation of the integral in 3.3 is by taking F to be a constant:

$$\begin{aligned} y(t_n + h) &= e^{-hL} y(t_n) + e^{-hL} \int_0^h e^{\tau L} F(t_n, y(t_n))\, d\tau \\ &= e^{-hL} y_n + \frac{e^{-hL} - 1}{-hL} F(t_n, y_n) \\ &= \varphi_0(-hL) y_n + h\varphi_1(-hL) F(t_n, y_n). \end{aligned}$$

The above approximation is called the exponential Euler method; it reduces to the explicit Euler method in the *classical limit* $L \to 0$.

Recalling that $F(t, y) = 0$ in Equation 2.7, we can clearly see that the exponential Euler method solves 2.7 exactly. In order to demonstrate how all Euler methods (classical and exponential, explicit and implicit) compare with each other, we will look at a slight modification to Equation 2.7:

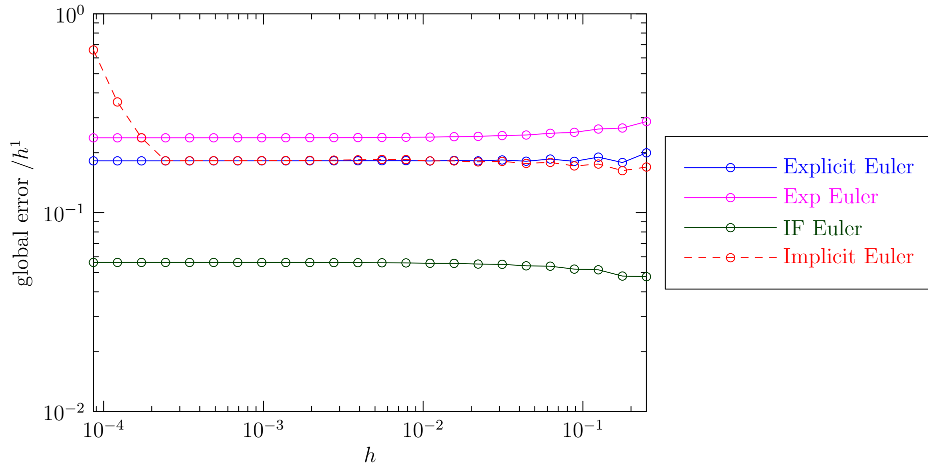
$$\frac{dy}{dt} = y^2 - Ly, \quad y(0) = y_0. \tag{3.10}$$

For completeness, we would like to mention at this point the existence of so called integrating factor methods, which have sometimes been applied to stiff problems. However, they are not as accurate: for example, the integrating factor Euler method (IF Euler)

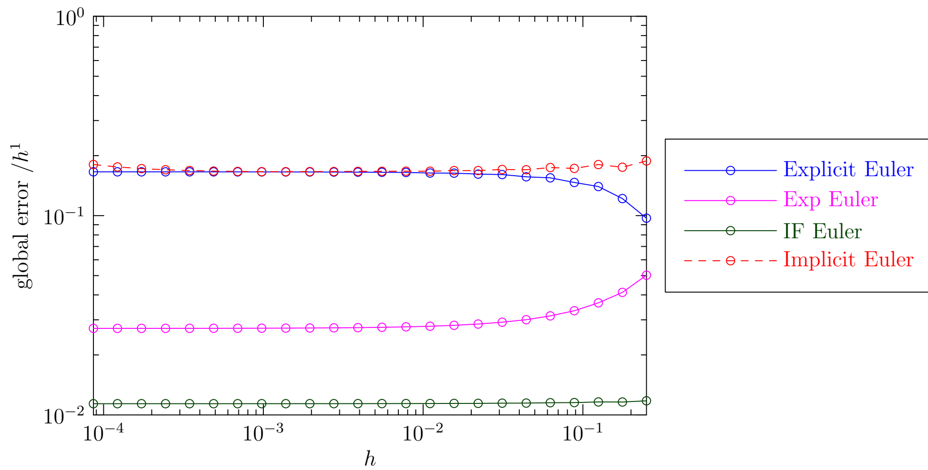
$$y(t_n + h) = e^{-hL}(y_n + hF(t_n, y_n)) \tag{3.11}$$

is only exact when $F = 0$, whereas the exponential Euler method is exact whenever F is constant. Worse, they do not preserve fixed points of the original ODE

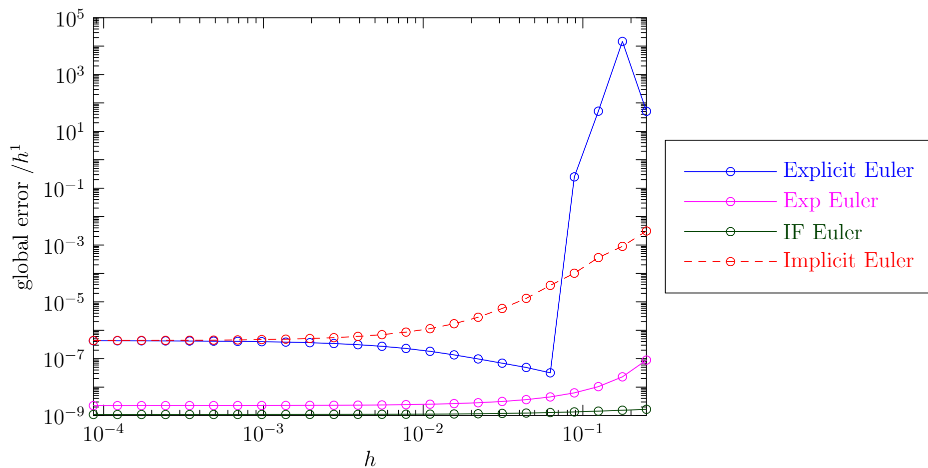
[Boyd 2001] [Cox & Matthews 2002]. We show how the various Euler methods perform on Equation 3.10 in Figure 3.1.



(a) $L = 2, \quad y_0 = 1.$



(b) $L = 4, \quad y_0 = 1.$



(c) $L = 20, \quad y_0 = 1.$

Figure 3.1: Error when solving $\frac{dy}{dt} = y^2 - Ly$ with explicit Euler, implicit Euler, integrating factor Euler and exponential Euler.

3.2 High-order exponential integrators

Just like in the classical case, there also exist high-order exponential integrator methods. Cox & Matthews [2002] attempted to approximate the integral in 3.3 with polynomials of degree higher than one and subsequently claimed to have derived a fourth-order scheme that they called ETD4RK. For the sake of bringing consistency to the names of methods by different authors we denote it ERK4CM and give its Butcher tableau in Table 3.1.

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) & & & \\
\frac{3}{4} & 0 & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) & & \\
1 & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right)\left(\varphi_0\left(-\frac{hL}{2}\right)-1\right) & 0 & \varphi_1\left(-\frac{hL}{2}\right) & \\
\hline
1 & \varphi_1 - 3\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 4\varphi_3 - \varphi_2
\end{array}
$$

Table 3.1: ERK4CM tableau, where $\varphi_i = \varphi_i(-hL)$.

Krogstad [2005] takes a different route than Cox & Matthews [2002]. Instead of approximating the integral in 3.3, he truncates the series in 3.9 in a way that allows for control of the remainder. He discovered an improved version of ERK4CM which he claims is fourth order and denotes by ETDRK4-B. For the sake of consistency, we denote Krogstad's method by ERK4K and give its Butcher tableau in Table 3.2.

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - \varphi_2\left(-\frac{hL}{2}\right) & \varphi_2\left(-\frac{hL}{2}\right) & & \\
1 & \varphi_1 - 2\varphi_2 & 0 & 2\varphi_2 & \\
\hline
1 & \varphi_1 - 3\varphi_2 + 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 2\varphi_2 - 4\varphi_3 & 4\varphi_3 - \varphi_2
\end{array}
$$

$$\varphi_i = \varphi_i(-hL).$$

Table 3.2: ERK4K tableau

Not long after Krogstad introduced ERK4K, Hochbruck & Ostermann [2005] published a rigorous study for deriving exponential integrators. They argued that simply trying to approximate the integral in 3.3 was not enough to guarantee that the methods would retain their order for all stiff problems. To prove this claim they presented a problem (revisited in a later chapter) for which ERK4CM exhibits order two (instead of the claimed order four) and ERK4K exhibits order three (instead of the claimed order four). It became apparent that there is a concrete distinction between the order conditions for classical and exponential Runge–Kutta methods. In their Theorem 4.7, Hochbruck & Ostermann [2005] derive *stiff-order conditions* up to order four, which describe sufficient conditions for an exponential RK method to not suffer a *stiff-order reduction*.

Theorem 3.1 (Hochbruck & Ostermann [2005])**.** *For $2 \leq p \leq 4$, assume that the order conditions of Table 3.3 hold up to order $p-1$ and that $\psi_p(0) = 0$. Further assume that the remaining conditions of order p hold in a weaker form with $a_{(s-1)i}(0)$ instead of $a_{(s-1)i}(-hL)$ for $1 \leq i \leq s-1$. Then, the numerical solution y_n satisfies the error bound*

$$\left\| y_n - y(t_n) \right\| \leq C \cdot h^p,$$

uniformly for the whole time integration interval. The constant C depends on the length of the integration interval, but is independent of the time step h.

We will, however, make an important remark that is crucial when one attempts to solve the stiff-order conditions and construct new exponential Runge–Kutta methods. The operators J and K appearing in the order conditions in Table 3.3 are discretizations of the following bounded operators

$$J_n = \frac{\partial F}{\partial y}(t_n, y(t_n)), \quad K_n = \frac{\partial^2 F}{\partial t \partial y}(t_n, y(t_n)). \tag{3.12}$$

In the case of designing a method to solve a single stiff ODE, the operators J and K as well as the weights of the method are only numbers and hence J and K cancel from the stiff-order equations. If we want our method to retain the stiff order even when solving arbitrary systems of stiff ODEs, we need to be more careful. In this case, the operators J and K are matrices. Moreover, the weights a_{ij} of the method are simply linear combinations of φ_k functions evaluated at scalar multiples of the matrix L. Therefore, the operators J and K do not necessarily commute with each other and also do not necessarily commute with the matrix weights a_{ij}. This has to be taken into consideration when solving the stiff conditions to construct exponential Runge–Kutta methods.

The existence of such a delicacy as described above is not present solely in the case of exponential integrators. Almost the same issue arises when solving order conditions for classical Runge–Kutta methods. Butcher [2009] gives an example of a classical RK method that has order five for nonstiff scalar problems, but order four when solving nonstiff vector problems.

It is worth noting at this point that although Theorem 3.1 is sufficient for an ERK method to retain its order for all stiff problems, it has not been proven that it is necessary. Nevertheless, Hochbruck & Ostermann [2005] show which stiff-order conditions are violated by ERK4CM, ERK4K, and some other previously discovered methods. They proceed to show that there is no method with four stages that satisfies the stiff-order conditions up to and including order four. Hence, they introduce a new method that is stiff-order four, but has five stages. They denoted this method as RK(5.19), but for the sake of consistency we denote this method ERK4HO5 and give its Butcher tableau in Table 3.4.

We have implemented a Mathematica script that checks the stiff order of a method, assuming that we want the method to retain its stiff order for vector problems

No.	order	order condition
1	1	$\psi_1(-hL) = 0$
2	2	$\psi_2(-hL) = 0$
3	2	$\psi_{1,i}(-hL) = 0$
4	3	$\psi_3(-hL) = 0$
5	3	$\sum_{i=1}^{s-1} a_{(s-1)i} J \psi_{2,i}(-hL) = 0$
6	4	$\psi_4(-hL) = 0$
7	4	$\sum_{i=1}^{s-1} a_{(s-1)i} J \psi_{3,i}(-hL) = 0$
8	4	$\sum_{i=1}^{s-1} a_{(s-1)i} J \sum_{j=1}^{i-1} a_{ij}(-hL) J \psi_{2,i}(-hL) = 0$
9	4	$\sum_{i=1}^{s-1} a_{(s-1)i} c_i K \psi_{2,i}(-hL) = 0$

$$\psi_{j,i}(-hL) = \varphi_j(-c_i hL) c_i^j - \sum_{k=0}^{i-1} a_{ik}(-hL) \frac{c_k^{j-1}}{(j-1)!}$$

$$\psi_j(-hL) = \varphi_j(-hL) - \sum_{k=0}^{s-1} a_{(s-1)k}(-hL) \frac{c_k^{j-1}}{(j-1)!}$$

Table 3.3: Stiff-order conditions

$$
\begin{array}{c|ccccc}
0 & & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - \varphi_2\left(-\frac{hL}{2}\right) & \varphi_2\left(-\frac{hL}{2}\right) & & & \\
1 & \varphi_1 - 2\varphi_2 & \varphi_2 & \varphi_2 & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - 2a_{31} - a_{33} & a_{31} & a_{31} & \frac{1}{4}\varphi_2\left(-\frac{hL}{2}\right) - a_{31} & \\
\hline
1 & \varphi_1 - 3\varphi_2 + 4\varphi 3 & 0 & 0 & -\varphi_2 + 4\varphi_3 & 4\varphi_2 - 8\varphi_3
\end{array}
$$

$$
\begin{aligned}
&\varphi_i = \varphi_i(-hL),\\
&a_{31} = \frac{1}{2}\varphi_2\left(-\frac{hL}{2}\right) - \varphi_3 + \frac{1}{4}\varphi_2 - \frac{1}{2}\varphi_3\left(-\frac{hL}{2}\right).
\end{aligned}
$$

Table 3.4: ERK4HO5 tableau

too. Because of this, we are forced to use noncommutative algebra in order to force Mathematica to **not** reduce the operators J and K. In Appendix A we include a version of this script that checks which orders are violated by ERK4K. We find that the orders ERK4K does not satisfy order seven and order eight from Table 3.3 and hence can have an order reduction to order three in the worst case, confirming the findings of Hochbruck & Ostermann [2005]. The script can be modified easily to check the stiff order of any exponential Runge–Kutta method up to order four.

Stiff-order conditions up to order four formed a foundation for the derivation of stiff-order conditions up to order five by Theorem 4.1 in Luan & Ostermann [2014]. Their main convergence result is given in Theorem 3.2.

Theorem 3.2. *Given an exponential Runge–Kutta method, assume that the order conditions of Table 3.3 hold as written there. Further assume that condition* 10 *in Table 3.5 holds in a weak form with $L = 0$ and the remaining conditions in Table 3.5 hold in a weaker form with $a_{(s-1)i}(0)$ instead of $a_{(s-1)i}(-hL)$. Then, the numerical*

solution y_n satisfies the error bound

$$\left\|y_n - y(t_n)\right\| \leq C \cdot h^5,$$

uniformly for the whole time integration interval. The constant C depends on the length of the integration interval, but is independent of the time step h.

No.	order	order condition
10	5	$\psi_5(-hL) = 0$
11	5	$\sum_{i=1}^{s-1} a_{(s-1)i} J\psi_{4,i}(-hL) = 0$
12	5	$\sum_{i=1}^{s-1} a_{(s-1)i} J \sum_{j=1}^{i-1} a_{ij}(-hL) J\psi_{3,j}(-hL) = 0$
13	5	$\sum_{i=1}^{s-1} a_{(s-1)i} J \sum_{j=1}^{i-1} a_{ij}(-hL) J \sum_{k=1}^{j-1} a_{jk}(-hL) J\psi_{2,k}(-hL) = 0$
14	5	$\sum_{i=1}^{s-1} a_{(s-1)i} J \sum_{j=1}^{i-1} a_{ij}(-hL) c_j K\psi_{2,j}(-hL) = 0$
15	5	$\sum_{i=1}^{s-1} a_{(s-1)i} c_i K\psi_{3,i}(-hL) = 0$
16	5	$\sum_{i=1}^{s-1} a_{(s-1)i} c_i K \sum_{j=1}^{i-1} a_{ij}(-hL) J\psi_{2,j}(-hL) = 0$
17	5	$\sum_{i=1}^{s-1} a_{(s-1)i} B(\psi_{2,i}(-hL), \psi_{2,i}(-hL)) = 0$
18	5	$\sum_{i=1}^{s-1} a_{(s-1)i} c_i^2 W\psi_{2,i}(-hL) = 0$

$$\psi_{j,i}(-hL) = \varphi_j(-c_i hL) c_i^j - \sum_{k=0}^{i-1} a_{ik}(-hL) \frac{c_k^{j-1}}{(j-1)!},$$
$$\psi_j(-hL) = \varphi_j(-hL) - \sum_{k=0}^{s-1} a_{(s-1)k}(-hL) \frac{c_k^{j-1}}{(j-1)!}.$$

Table 3.5: Stiff order conditions

Proof. For the proof of this Theorem, see Luan & Ostermann [2014]. □

As in the discussion after Theorem 3.1, we remark about the arbitrary operators J, K, W and the arbitrary bilinear mapping B appearing in the conditions in Table 3.5. The operators J, K are the same as in Theorem 3.1 and W is a discretization of the bounded operator W_n:

$$J_n = \frac{\partial F}{\partial y}(t_n, y(t_n)), \quad K_n = \frac{\partial^2 F}{\partial t \partial y}(t_n, y(t_n)), \quad W_n = \frac{\partial^3 F}{\partial t \partial y^2}(t_n, y(t_n)), \tag{3.13}$$

Unlike in Hochbruck & Ostermann [2005], it is not explicitly stated in the paper by Luan & Ostermann [2014] what these arbitrary operators are and it is not quite clear what exactly the bilinear map B stands for. Nevertheless, in the general vector problem, the discretizations of these operators will be matrices and in order to properly solve the order conditions, we need to take into account the fact that they do not commute with the weights of the method.

Because of the lack of clarity for the bilinear mapping B, in our implementation of these conditions later on, we will choose to satisfy condition 17 in Table 3.5 the same way that Luan & Ostermann [2014] have done: by setting $\psi_{2,i}(-hL) = 0$.

Chapter 4

Embedded Methods

4.1 Classical embedded methods

Arbitrarily choosing a time step h and using it to solve an ODE system with a classical Runge–Kutta method is not the most optimal way of obtaining an approximation to the exact solution. Sometimes, we will be restricting ourselves to a time step that can be bigger without causing any issues (hence wasting computational power) and, sometimes, we will be using a time large enough such that it causes stability problems or problems with stiffness. Therefore, if we stick to using a Runge–Kutta method directly, we end up gambling on what the time step should be.

Embedded Runge–Kutta methods afford a means of adjusting the time step by providing two approximate solutions of different orders:

$$y_n^{s-1} = y_n^0 + h \sum_{j=0}^{s-2} a_{(s-2)j} f(t_n + c_j h, y_n^j) \tag{4.1}$$

and

$$y_n^s = y_n^0 + h \sum_{j=0}^{s-1} a_{(s-1)j} f(t_n + c_j h, y_n^j). \tag{4.2}$$

The error of the lower-order method can be approximated by

$$|y_n^s - y_n^{s-1}| = h \left| \sum_{j=0}^{s-1} (a_{(s-1)j} - a_{(s-2)j}) f(t_n + c_j h, y_n^j) \right|. \tag{4.3}$$

We can specify desired upper and lower tolerances for this error, adjusting the step size accordingly at the end of each time step. If the error is less than the upper bound, the adjusted time step is used to evolve the high-order solution further.

In order to minimize the computational cost, the two Runge–Kutta methods comprising the embedded method are chosen such that their tableaux are almost the same. Specifically, the Butcher tableau of an explicit embedded Runge–Kutta method is given in Table 4.1. Usually, the difference in the order of the two

0						
c_1	a_{00}					
c_2	a_{10}	a_{11}				
$\vdots$	$\vdots$	$\vdots$	$\ddots$			
c_{s-2}	$a_{(s-3)0}$	$a_{(s-3)1}$	$\cdots$	$a_{(s-3)(s-3)}$		
1	$a_{(s-2)0}$	$a_{(s-2)1}$	$\cdots$	$\cdots$	$a_{(s-2)(s-2)}$	
1	$a_{(s-1)0}$	$a_{(s-1)1}$	$\cdots$	$\cdots$	$a_{(s-2)(s-1)}$	$a_{(s-1)(s-1)}$

Table 4.1: General embedded Runge–Kutta tableau

approximations provided by an embedded method is one. Well known classical embedded Runge–Kutta methods are the $(3, 2)$ pair by Bogacki & Shampine [1989] in Table 4.2 and the $(5, 4)$ pair by Dormand & Prince [1980] in Table 4.3, both of which give a higher-order approximation in the second-last stage and a lower-order approximation in the last stage. Note that both RKBS32 and RKDP54 have a nonzero weight for $a_{(s-1)(s-1)}$. This means that to find y_n^s we have to evaluate $f(t_n + h, y_n^{s-1})$, which is needed to calculate the first stage in the next step since

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{3}{4} & 0 & \frac{3}{4} & & \\
\hline
1 & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & \\
1 & \frac{7}{24} & \frac{1}{4} & \frac{1}{3} & \frac{1}{8}
\end{array}
$$

Table 4.2: Bogacki–Shampine $(3, 2)$ tableau

$y_n^{s-1} = y_{n+1}^0$. This FSAL (First Same As Last) property, which only applies when the higher-order approximation appears in a stage earlier than the lower-order one and when $a_{(s-1)(s-1)} \neq 0$, can be exploited to avoid one function evaluation whenever a time step is accepted.

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
\hline
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
1 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array}
$$

Table 4.3: Dormand–Prince $(5, 4)$ tableau

4.2 Exponential embedded methods

Even when y is a vector and L is a matrix, the weights of classical Runge–Kutta methods are just numbers, while the weights of exponential integrator methods are linear combinations of matrix φ_k functions that depend on the step size h and the matrix L. This means that we need to re-evaluate the φ_k functions whenever the step size changes. Since the φ_k functions involve exponentials of matrices, variable time

stepping is often seen an expensive operation when the linear term L is a general matrix [Kassam & Trefethen 2005]. If L is diagonal, however, the exponential matrix can obviously be computed efficiently. Moreover, when L is diagonalizable (e.g. if L is normal) a one-time diagonalization can be used to provide efficient evaluations of the φ_k functions for all subsequent values of h. Alternatively, Krylov subspace methods [Tokman & Loffeld 2010] can be used to efficiently evaluate the matrix-vector products that arise in exponential Runge–Kutta integrators. In any case, one should strive to have as few different step fractions c_i as possible, because the weights of the method also depend on c_i and having many distinct c_i will lead to more φ_k functions to be evaluated.

Two embedded ERK methods were introduced by Whalen *et al.* [2015], one $(4,3)$ method denoted ETD34 and one $(5,3)$ method denoted ETD35. However, the order four approximation in the $(4,3)$ method is the Krogstad method, whose actual stiff order was shown to be 3. The same holds true for the order five approximation in the $(5,3)$ pair; it is of classical order five but only of stiff-order three. Bowman *et al.* [2006] constructed a stiff $(3,2)$ pair ERKBS32, given in Table 4.4, by adding an extra stage to a special case of a stiff third-order method from Hochbruck & Ostermann [2005] to obtain a stiff-order two error estimate, such that the resulting scheme reduces to the classical Bogacki–Shampine pair when $L = 0$. This property is desired when designing exponential integrator methods, because it implies that the method will have similar stability properties to its classical counterpart when numerically treating the nonlinear source term. More recently, Ding & Kang [2017] introduced four embedded ERK methods. The first two are built on the Cox and Matthews method (denoted ERK4(3)3(2)) and the Krogstad method (denoted ERK4(3)3(3)), respectively; hence each of them evidently suffer from order reduction of the higher-order approximation. The third

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) & & & \\
\frac{3}{4} & \frac{3}{4}\varphi_1\left(-\frac{3hL}{4}\right) - a_{11} & \frac{9}{8}\varphi_2\left(-\frac{3hL}{4}\right) + \frac{3}{8}\varphi_2\left(-\frac{hL}{2}\right) & & \\
\hline
1 & \varphi_1 - a_{21} - a_{22} & \frac{1}{3}\varphi_1 & \frac{4}{3}\varphi_2 - \frac{2}{9}\varphi_1 & \\
1 & \varphi_1 - \frac{17}{12}\varphi_2 & \frac{1}{2}\varphi_2 & \frac{2}{3}\varphi_2 & \frac{1}{4}\varphi_2
\end{array}
$$

$$
\varphi_i = \varphi_i(-hL).
$$

Table 4.4: ERKBS32 tableau

embedded ERK method, denoted ERK4(3)4(3) in Ding & Kang [2017], is based on the five-stage method of the stiff-order four ERK4HO5 method introduced by Hochbruck & Ostermann [2005]. A stiff-order three method is appended to ERK4HO5, resulting in a stiff $(4, 3)$ pair. We denote this pair, given in Table 4.5, as ERK43DK. The last embedded pair by Ding & Kang [2017] is a stiff $(5, 4)$ pair

$$
\begin{array}{c|ccccc}
0 & & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - \varphi_2\left(-\frac{hL}{2}\right) & \varphi_2\left(-\frac{hL}{2}\right) & & & \\
1 & \varphi_1 - 2\varphi_2 & \varphi_2 & \varphi_2 & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - 2a_{31} - a_{33} & a_{31} & a_{31} & \frac{1}{4}\varphi_2\left(-\frac{hL}{2}\right) - a_{31} & \\
\hline
1 & \varphi_1 - 3\varphi_2 + 4\varphi 3 & 0 & 0 & -\varphi_2 + 4\varphi_3 & 4\varphi_2 - 8\varphi_3 \\
1 & a_{40} & \frac{1}{2}a_{44} & \frac{1}{2}a_{44} & a_{43} & 0
\end{array}
$$

$$
\begin{aligned}
\varphi_i &= \varphi_i(-hL), \\
a_{31} &= \frac{1}{2}\varphi_2\left(-\frac{hL}{2}\right) - \varphi_3 + \frac{1}{4}\varphi_2 - \frac{1}{2}\varphi_3\left(-\frac{hL}{2}\right).
\end{aligned}
$$

Table 4.5: ERK43DK tableau

based on a stiff fifth-order method introduced by Luan & Ostermann [2014]. This $(5, 4)$ pair is denoted by the original authors as ERK5(4)5(4), but for the sake of

consistency we will denote it ERK54DK. Its tableau is given in Table 4.6.

We saw that exponential integrator methods that converge in the classical limit $L \to 0$ to a well studied Runge–Kutta method are attractive. That is why it is unfortunate that the Krogstad method, which reduces to RK4 in the classical limit, does not have stiff-order four [Hochbruck & Ostermann 2005]. This fact not only implies that there will not be a one-to-one correspondence between RK and ERK methods, but there also will not be a one-to-one correspondence between embedded RK and embedded ERK methods. Hochbruck & Ostermann [2005] showed that, assuming their stiff-order conditions are necessary, at least five stages are needed to achieve stiff-order four. It was later shown by Luan & Ostermann [2014] that at least eight stages are needed to achieve stiff-order five. This means that there will not be an ERK version of the $(5, 4)$ method by Cash & Karp [1990] mentioned earlier.

However, the situation might be better than it at first seems. Usually, to get a classical embedded RK $(4, 3)$ pair, one adds an extra stage to a four-stage fourth-order method to obtain a third-order estimate with five stages, like Balac & Mahé [2013] have done. Since five is also the minimal number of stages required to achieve stiff-order four, we can try to use the extra stage to provide an error estimate at no extra cost. Specifically we seek an embedded ERK method that gives a stiff third-order approximation in the fourth stage and a stiff fourth-order approximation in the fifth stage. For comparison, the $(4, 3)$ stiff pair by Ding & Kang [2017] has a total of six stages. We eliminate the need for a sixth stage by constraining the second-last stage of the high-order method to be third order. That way, we have a $(4, 3)$ stiff pair with only five stages, just like in the classical case. In order to obtain such an embedded method, we need to solve the system of equations coming from the stiff-order conditions from scratch, since we cannot use a previously introduced stiff ERK method like in the case of Ding & Kang [2017] and Whalen *et al.* [2015].

$$
\begin{array}{c|ccccccccc}
0 & & & & & & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1(-\frac{hL}{2}) & & & & & & & & \\
\frac{1}{2} & a_{10} & \varphi_2(-\frac{hL}{2}) & & & & & & & \\
\frac{1}{4} & a_{20} & 0 & \frac{1}{8}\varphi_2(-\frac{hL}{4}) & & & & & & \\
\frac{1}{2} & a_{30} & 0 & a_{32} & a_{33} & & & & & \\
\frac{1}{5} & a_{40} & 0 & 0 & a_{43} & a_{44} & & & & \\
\frac{2}{3} & a_{50} & 0 & 0 & a_{53} & a_{54} & a_{55} & & & \\
1 & a_{60} & 0 & 0 & 0 & a_{64} & a_{65} & a_{66} & & \\
\hline
1 & a_{70} & 0 & 0 & 0 & 0 & a_{75} & a_{76} & a_{77} & \\
1 & a_{70} & 0 & 0 & 0 & 0 & a_{75} & a_{76} & 0 & a_{77}
\end{array}
$$

$$
\begin{aligned}
a_{10} =& \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - \varphi_2\left(-\frac{hL}{2}\right) \quad a_{20} = \frac{1}{4}\varphi_1\left(-\frac{hL}{4}\right) - \frac{1}{8}\varphi_2\left(-\frac{hL}{4}\right) \\
\varphi_i =& \varphi_i(-hL) \quad a_{30} = \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - \frac{3}{2}\varphi_2\left(-\frac{hL}{2}\right) + 2\varphi_3\left(-\frac{hL}{2}\right) \\
a_{32} =& -\frac{1}{2}\varphi_2\left(-\frac{hL}{2}\right) + 2\varphi_3\left(-\frac{hL}{2}\right) \quad a_{33} = 2\varphi_2\left(-\frac{hL}{2}\right) - 4\varphi_3\left(-\frac{hL}{2}\right) \\
a_{40} =& \frac{1}{3}\varphi_1\left(-\frac{hL}{5}\right) - \frac{2}{25}\varphi_2\left(-\frac{hL}{5}\right) - \frac{1}{2}a_{43} \quad a_{43} = \frac{8}{25}\varphi_2\left(-\frac{hL}{5}\right) - \frac{32}{125}\varphi_3\left(-\frac{hL}{5}\right) \\
a_{44} =& \frac{2}{25}\varphi_2\left(-\frac{hL}{5}\right) - \frac{1}{2}a_{43} \quad a_{50} = \frac{2}{3}\varphi_1\left(-\frac{2hL}{3}\right) + \frac{125}{162}a_{43} - a_{54} - a_{55} \\
a_{53} =& -\frac{125}{162}a_{43} \quad a_{54} = \frac{125}{1944}a_{43} - \frac{16}{27}\varphi_2\left(-\frac{2hL}{3}\right) + \frac{320}{81}\varphi_3\left(-\frac{2hL}{3}\right) \\
a_{55} =& \frac{3125}{3888}a_{43} + \frac{100}{27}\varphi_2\left(-\frac{2hL}{3}\right) - \frac{800}{81}\varphi_3\left(-\frac{2hL}{3}\right) \\
\phi =& \frac{5}{32}a_{43} - \frac{1}{28}\varphi_2\left(-\frac{hL}{5}\right) + \frac{36}{175}\varphi_2\left(-\frac{2hL}{3}\right) - \frac{48}{25}\varphi_3\left(-\frac{2hL}{3}\right) \\
&+ \frac{6}{175}\varphi_4\left(-\frac{hL}{5}\right) + \frac{192}{35}\varphi_4\left(-\frac{2hL}{3}\right) + 6\varphi_4 \\
a_{60} =& \varphi_1 - a_{64} - a_{65} - a_{66} \quad a_{64} = \frac{208}{3}\varphi_3 - \frac{16}{3}\varphi_2 - 40\phi \\
a_{65} =& -\frac{250}{3}\varphi_3 + \frac{250}{21}\varphi_2 + \frac{250}{7}\phi \quad a_{66} = -27\varphi_3 + \frac{27}{14}\varphi_2 + \frac{135}{7}\phi \\
a_{70} =& \varphi_1 - a_{75} - a_{76} - a_{77} \quad a_{75} = \frac{125}{3}\varphi_2 - \frac{625}{14}\varphi_3 + \frac{1125}{14}\varphi_4 \\
a_{76} =& -\frac{27}{14}\varphi_2 + \frac{162}{7}\varphi_3 - \frac{405}{7}\varphi_4 \quad a_{77} = \frac{1}{2}\varphi_2 - \frac{13}{2}\varphi_3 + \frac{45}{2}\varphi_4
\end{aligned}
$$

Table 4.6: ERK54DK tableau

To facilitate this process, we will write every weight of the method as a linear combination of all possible $\varphi_k(-c_j hL)$ functions and then insert them into the stiff-order conditions. By requiring the stiff-order conditions to hold, we are imposing restrictions on the coefficients of the φ_k functions in each weight. These restrictions will form a system of equations that we can solve symbolically using Mathematica. The system of restrictions and number of free parameters becomes relatively large (especially as we add stages and if we require all of the c_is to be different). Provided Mathematica can solve the resulting system of restrictions, it is beneficial if the method has as many free parameters as possible. We will focus in two main areas of optimization. First and foremost, we want the third-order method to **never** be fourth order for any problem, because then we would have two fourth-order methods and that would cause the step size adjustment algorithm to fail. To achieve this, we recall that stiff-order conditions reduce to classical order conditions in the case $L = 0$; in our optimization step we therefore need to ensure that our third-order estimate does not satisfy any of the fourth-order conditions evaluated at $L = 0$. Second, since we will be advancing the solution using the fourth-order method, we want to make sure that the error is minimal. To achieve this, we need to implement the stiff-order five conditions given in [Luan & Ostermann 2014] and make sure that our fourth order method is as close to satisfying them weakly as possible (in the sense described in Theorem 3.2). Moreover, we also need to make sure that our fourth-order method is as close as possible to satisfying the stiff-order four conditions strongly. As one can imagine, implementing this idea is not as straightforward as when optimizing classical RK methods. The difficulty lies in the fact that we need to take special care of the arbitrary operators J, K, W, and the bilinear mapping B that appear in some of the stiff-order conditions. This is carefully done in the Mathematica script in Appendix B, which generates the exponential $(4, 3)$ pair ERK43ZB, displayed in

Table 4.7, that satisfies the optimizations described above.

As we have already stated, the method ERK43ZB is a stiff fourth-order method with its second-last stage being a stiff third-order embedded method. The third-order approximation is guaranteed to never be of stiff (and classical) order higher than three. Moreover, the fourth-order estimate in ERK43ZB has minimal error. This has been achieved by minimizing the L^2 norm of a vector whose entries are the coefficients in front of every term involving J, K, W, the bilinear mapping B, $\varphi_k(-c_j hL)$, and any combination of all these operators. If we denote this vector by E_5, then for the method ERK43ZB, $E_5 = 1.14845$. By comparison, ERK4HO5 has $E_5 = 6.67545$. This does not mean that the fourth-order estimate in ERK43ZB will do better than ERK4HO5 in every example, but in general it is guaranteed to be more accurate. As pointed out by Dormand & Prince [1980], there is no need to perform such an optimization for the lower-order estimate since it is only used for step-size control.

Accuracy of the higher-order method is not the only front where the embedded ERK method by Ding & Kang [2017] falls behind. It is not hard to see that the third-order estimate in ERK43DK weakly satisfies stiff-order condition 6 in Table 3.3. Hence, there will exist ODE systems for which both estimates will provide a fourth-order approximation to the solution, as seen in Figure 5.6. This will lead to problems with the step-size adjustment algorithm, as seen in Figure 5.7. The same issue arises with the other stiff pair by Ding & Kang [2017]: the fourth-order estimate in ERK54DK satisfies weakly stiff-order condition 10 in Table 3.5. This is also very easy to see, given how similar the last stage of the fifth-order method and the last stage of the fourth-order method are in Tableau 4.6. There could be further stiff-order five conditions that the fourth-order ERK54DK estimate satisfies, but this is enough to show that this embedded method will also cause step-size adjustment

$$
\begin{array}{c|ccccc}
0 & & & & & \\
\frac{1}{6} & \frac{1}{6}\varphi_1\left(-\frac{hL}{6}\right) & & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - a_{11} & a_{11} & & & \\
\frac{1}{2} & \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) - a_{21} - a_{22} & a_{21} & a_{22} & & \\
\hline
1 & \varphi_1 - a_{31} - a_{32} - a_{33} & a_{31} & a_{32} & a_{33} & \\
1 & \varphi_1 - \frac{67}{9}\varphi_2 + \frac{52}{3}\varphi_3 & 8\varphi_2 - 24\varphi_3 & \frac{26}{3}\varphi_3 - \frac{11}{9}\varphi_2 & a_{43} & a_{44}
\end{array}
$$

$$
\begin{aligned}
\varphi_i &= \varphi_i(-hL) \\
a_{11} &= \frac{3}{2}\varphi_2\left(-\frac{hL}{2}\right) + \frac{1}{2}\varphi_2\left(-\frac{hL}{6}\right) \\
a_{21} &= \frac{19}{60}\varphi_1 + \frac{1}{2}\varphi_1\left(-\frac{hL}{2}\right) + \frac{1}{2}\varphi_1\left(-\frac{hL}{6}\right) \\
&\quad + 2\varphi_2\left(-\frac{hL}{2}\right) + \frac{13}{6}\varphi_2\left(-\frac{hL}{6}\right) + \frac{3}{5}\varphi_3\left(-\frac{hL}{2}\right) \\
a_{22} &= -\frac{19}{180}\varphi_1 - \frac{1}{6}\varphi_1\left(-\frac{hL}{2}\right) - \frac{1}{6}\varphi_1\left(-\frac{hL}{6}\right) \\
&\quad - \frac{1}{6}\varphi_2\left(-\frac{hL}{2}\right) + \frac{1}{9}\varphi_2\left(-\frac{hL}{6}\right) - \frac{1}{5}\varphi_3\left(-\frac{hL}{2}\right) \\
a_{33} &= \varphi_2 + \varphi_2\left(-\frac{hL}{2}\right) - 6\varphi_3 - 3\varphi_3\left(-\frac{hL}{2}\right) \\
a_{31} &= 3\varphi_2 - \frac{9}{2}\varphi_2\left(-\frac{hL}{2}\right) - \frac{5}{2}\varphi_2\left(-\frac{hL}{6}\right) + 6a_{33} + a_{21} \\
a_{32} &= 6\varphi_3 + 3\varphi_3\left(-\frac{hL}{2}\right) - 2a_{33} + a_{22} \\
a_{43} &= \frac{7}{9}\varphi_2 - \frac{10}{3}\varphi_3, \qquad a_{44} = \frac{4}{3}\varphi_3 - \frac{1}{9}\varphi_2
\end{aligned}
$$

Table 4.7: ERK43ZB tableau

difficulties for some problems. In contrast, by construction our ERK43ZB method does not suffer from any of these issues. Consequently, it appears to be a promising contribution to the field.

Chapter 5

Examples and Implementation

5.1 Implementation details

The general system of ODEs that we have considered thus far has been

$$\frac{dy}{dt} = F(y) - Ly. \tag{5.1}$$

Applying an exponential Runge–Kutta method to such a system requires the evaluation of functions $\varphi_k(c_j hL)$ for some k and j. Since L is assumed to be a general matrix, then these φ_k are matrix functions related to the matrix exponential. Methods for efficiently and robustly calculating such matrix functions, and the matrix exponential in particular, are an active field of research. Certain of them methods are robust and less prone to error. In our testing, we chose to implement a scaling and squaring algorithm, followed by a Padé approximant, for calculating the matrix exponential [Moler & Van Loan 2003][Higham 2008]. Although slow, this is a robust technique for calculating the φ_k functions for a general matrix L. In particular cases, special properties of L may be used to devise more computationally efficient

methods. For example, if L is a sparse matrix, it is worth implementing a Krylov subspace method. A short description of Krylov subspace methods (as well as other methods such as Chebyshev methods, Leja interpolation, and contour integrals) in relation to exponential RK methods is given in [Hochbruck & Ostermann 2010]. We will not spend more time on the variety of methods already in use, but will argue for a transformation to be done to the equation such that the matrix in the term that is treated exactly by the ERK method is diagonal.

An important practical application of exponential integrators are PDEs containing a linear term Ly, where L is a Laplacian. If spectral transforms are used to convert spatial derivatives to algebraic expressions, the Laplacian becomes a diagonal matrix. Calculating the exponential of a diagonal matrix is straightforward to implement and computationally inexpensive. To avoid loss of accuracy due to finite numerical precision, truncations of Taylor series should be used when evaluating $\varphi_k(x)$ near 0 for $k > 0$ [Bowman 2005]. Other applications use finite differences to approximate the Laplacian as a nondiagonal discretized spatial operator. In these cases, the difficulty of robustly computing the various matrix φ_k functions has discouraged many researchers from using adaptive exponential integrators.

Recognizing the huge computational advantages of the diagonal case, it would seem reasonable when L is diagonalizable to compute a one-time change of basis that diagonalizes L; that basis can then be reused for computing matrix functions of $c_j hL$ for arbitrary values of h. However, not all matrices are diagonalizable. In the general case, instead of trying to diagonalize L, we can find its Schur decomposition

$$L = UTU^{\dagger}, \tag{5.2}$$

where $U^{\dagger}$ denotes the conjugate transpose of the *unitary* matrix U (so that $U^{\dagger} = U^{-1}$)

and T is an upper triangular matrix. Further, we can write $T = D + B$ where D is a diagonal matrix and B is a strictly upper triangular matrix. Equation 5.1 becomes

$$\frac{dy}{dt} + U(D + B)U^{\dagger}y = F(t, y). \tag{5.3}$$

On multiplying by $U^{\dagger}$ on the left we obtain

$$\frac{d(U^{\dagger}y)}{dt} + (D + B)U^{\dagger}y = U^{\dagger}F(t, y), \tag{5.4}$$

or, in terms of the transformed variable $Y = U^{\dagger}y$,

$$\frac{dY}{dt} + DY = U^{\dagger}F(t, UY) - BY, \tag{5.5}$$

By doing this transformation, we avoid working with exponentials of a full matrix in favour of exponentials of a diagonal matrix. The main trade-off is that we have to compute the Schur decomposition for the matrix L, but that is only done once in the beginning and the longer the interval for the time integration, the more it is clear that paying this price is worth it. The second drawback is that some part that could have been treated exactly is now treated numerically and this could have an addition to the error of the method. In addition, we have to do two matrix multiplications at each step because the nonlinearity $F(t, y)$ is evaluated in the initial space. Although the efficiency gained in calculating the φ_k functions is more than enough to compensate for the drawback of two added multiplications per step and the potential of added numerical error, there is another advantage to implementing exponential RK methods in this way. The φ_k functions are now diagonal matrices and can thus be stored as vectors. This is a large improvement in memory usage as even for sparse matrices, the matrix φ_k functions are general full matrices requiring

extra storage. Furthermore, the ERK methods (and classical RK methods) work by multiplying the weights of the method by the vector of y approximated at a previous step. In the case of ERK methods, the weights are linear combinations of matrix functions and hence matrices themselves. By implementing the Schur decomposition and being able to work with weights that are diagonal matrices, we have replaced all the needed matrix-vector multiplications with computationally cheap vector dot products.

Until now, embedded ERK methods that allow for step size adjustment based on a predetermined error tolerance have been regarded as something only worth mentioning for the sake of completeness. However, with the optimizations afforded by Schur decomposition, the practical use of embedded ERK methods becomes computationally viable. An adaptive exponential method requires recalculating the weights (and corresponding φ_k functions) every time that the step size is adjusted. However, since these are now functions of diagonal matrices, there is no longer a huge computational cost to bear. As in the case of fixed step size, the Schur decomposition of L only needs to be calculated once in the beginning, so depending how long the integration takes place, the cost of the decomposition will typically be negligible.

An important point about the Schur decomposition is that the term By does not incorporate any of the stiffness inherent in the linear term Ly. Consider the following equation, where B is a $m \times m$ strictly upper triangular matrix:

$$\frac{dy}{dt} = -By. \tag{5.6}$$

This means that

$$\frac{dy_i}{dt} = \sum_{j=i+1}^{m} b_{ij} y_j \text{ for } i = 1, \ldots, m-1 \quad \text{and} \quad \frac{dy_m}{dt} = 0, \tag{5.7}$$

which means that we can easily find the exact solution of this system. Each of the components of the general solution will be polynomials in t. Going back to the idea of stiffness being the phenomenon when nearby solution curves approach the solution curve of interest at an exponentially fast rate, we deduce that since polynomials cannot approach each other at a fast rate, Equation 5.6 cannot be stiff. Such ODE systems can even be solved exactly by suitable classical Runge–Kutta methods if the order of the RK method is equal to or higher than the order of the highest-order polynomial among the components of the solution.

We can see this by a simple example. Consider the equation

$$\begin{bmatrix} y_1' \\ y_2' \\ y_3' \end{bmatrix} = - \begin{bmatrix} 0 & a & b \\ 0 & 0 & c \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}, \tag{5.8}$$

where L is a strict upper triangular matrix and $F = 0$. The general solution of this system of ODEs is

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} ack_3\frac{t^2}{2} - (ak_2 + bk_2)t + k_1 \\ -ck_3 t + k_2 \\ k_3 \end{bmatrix}, \tag{5.9}$$

where the constants k_1, k_2 and k_3 are fixed by the initial condition. All the components of the solution vector are polynomials in t; hence classical Runge–Kutta

methods of second- or higher-order will solve this problem exactly. It is straightforward to see that this idea generalizes to any strictly triangular matrix of arbitrary size.

Since many matrices encountered in practice are normal, the following result shows that the Schur decomposition technique not only removes linear stiffness from the problem, but in many cases will handle the linear term exactly (since $B = 0$).

Theorem 5.1. *The triangle matrix in the Schur decomposition of a normal matrix is diagonal.*

Proof. Assume L is a *normal* matrix:

$$LL^\dagger = L^\dagger L. \tag{5.10}$$

The *Schur decomposition* of L and $L^\dagger$ are

$$L = U^\dagger T U \quad \text{and} \quad L^\dagger = U^\dagger T^\dagger U, \tag{5.11}$$

where T is a triangular matrix and U is a *unitary* matrix, so that $U^\dagger = U^{-1}$. Hence

$$U^\dagger T U U^\dagger T^\dagger U = U^\dagger T^\dagger U U^\dagger T U, \tag{5.12}$$

which reduces to

$$TT^\dagger = T^\dagger T. \tag{5.13}$$

This means that the triangular matrix T resulting from the Schur decomposition of L is normal. An inductive argument shows that it must be diagonal [Prasolov 1994]. □

5.2 Examples and applications

Let us have a look at an example where the matrix L is upper triangular. First, we will solve the system by treating the full linear term by an exponential RK method. Then, we will split L into the sum of a diagonal matrix D and a strictly upper triangular matrix B, allowing us to treat the diagonal term exactly and the rest numerically. This system of ODEs is relatively small, hence this test concerns more with showing that the stiffness will be isolated in the diagonal term. Consider the equation below:

$$\frac{dy}{dt} = -Ly, \tag{5.14}$$

which is equivalent to

$$\frac{dy}{dt} = -Dy - By, \tag{5.15}$$

where

$$L = -\begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & f \end{bmatrix}, D = -\begin{bmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & f \end{bmatrix}, B = -\begin{bmatrix} 0 & b & c \\ 0 & 0 & e \\ 0 & 0 & 0 \end{bmatrix}. \tag{5.16}$$

The general solution of this system of ODEs is

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} k_1 e^{-at} + \frac{bek_3e^{-ft}}{(f-a)(f-d)} + \frac{bk_2e^{-dt}}{d-a} + \frac{ck_3e^{-ft}}{f-a} \\ k_2e^{-dt} + \frac{ek_3e^{-ft}}{f-d} \\ k_3e^{-ft} \end{bmatrix}, \tag{5.17}$$

where the constants k_1, k_2 and k_3 are fixed by the chosen initial condition. This algebraic example allows us to run many tests with ease. For demonstration, we pick the following matrix L and initial condition $y(0)$:

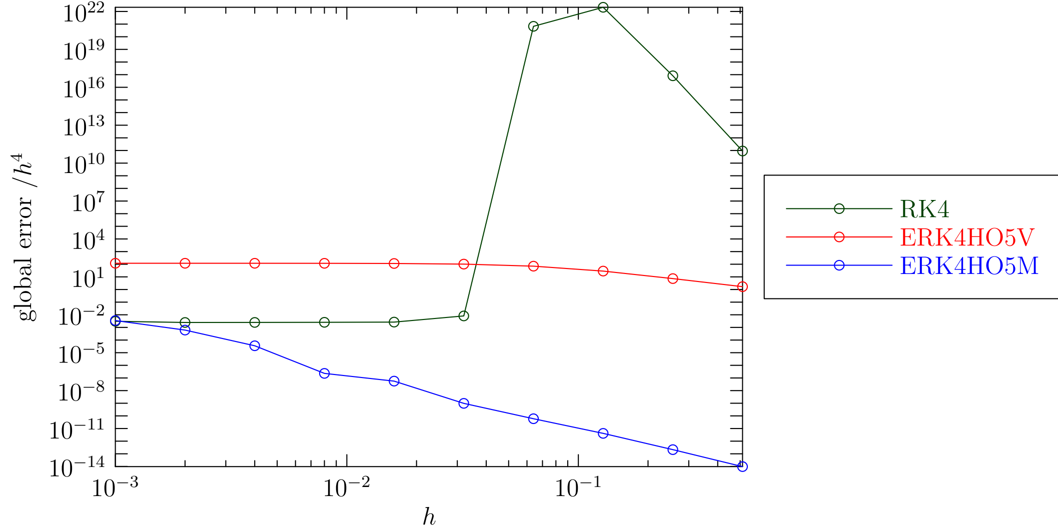


Figure 5.1: Error at $t = 1$ when solving Equation 5.18 with ERK4HO5M, ERK4HO5V, and RK4.

$$L = -\begin{bmatrix} 1 & 2 & 7 \\ 0 & 75 & 8 \\ 0 & 0 & 15 \end{bmatrix}, \quad y(0) = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}. \tag{5.18}$$

We now compare in Figure 5.1 the error in ERK4HO5M, which is a matrix implementation of ERK4HO5 [Hochbruck & Ostermann 2005] and ERK4HO5V, which is an implementation of the same method with only the diagonal part treated exactly. Here V stands for "vector" since the matrix in the linear term is taken to be the diagonal matrix D and M stands for "matrix" since the matrix in the linear term is the complete matrix L. The full implementation ERK4HO5M is supposed to solve the problem exactly (since $F = 0$) but is susceptible to floating point precision error. Both exponential methods behave as expected for large time steps. In comparison, we show how the classical RK4 method (tableau 2.2) fails at large h.

We now compare popular exponential integrators with our proposed integrator

ERK43ZBM on the matrix problem in Example 6.2 of Hochbruck & Ostermann [2005]:

$$\frac{\partial y}{\partial t}(x,t) - \frac{\partial^2 y}{\partial x^2}(x,t) = \int_0^1 y(\bar{x},t)\,d\bar{x} + \Phi(x,t), \tag{5.19}$$

for $x \in [0,1]$ and $t \in [0,1]$, subject to homogeneous Dirichlet boundary conditions. The function Φ is chosen by substituting in the equation the exact solution which is taken to be

$$y(x,t) = x(1-x)e^t. \tag{5.20}$$

This problem can be transformed to a system of ODEs by performing a spatial discretization of the Laplacian term. Since we are not focusing as much in spatial discretizations on this work, we do a quick finite difference discretization for the Laplacian term with 200 spatial grid points:

$$\frac{\partial^2 y}{\partial x^2} \approx L = \frac{1}{dx^2}\begin{bmatrix} -2 & 1 & 0 & 0 & \dots & \dots & \dots & 0 \\ 1 & -2 & 1 & 0 & \vdots & \vdots & \vdots & \vdots \\ 0 & 1 & -2 & 1 & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & 1 & -2 & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & 1 & -2 & 1 \\ 0 & \dots & \dots & \dots & 0 & 0 & 1 & -2 \end{bmatrix}. \tag{5.21}$$

Hochbruck & Ostermann [2005] used the discrete version of Problem 5.19 to demonstrate the order reduction of the Krogstad method ERK4KM and the Cox and Matthews method ERK4CMM. We replicate their results, while adding the error produced by ERK43ZBM, in Figure 5.2. As in Hochbruck & Ostermann [2005],

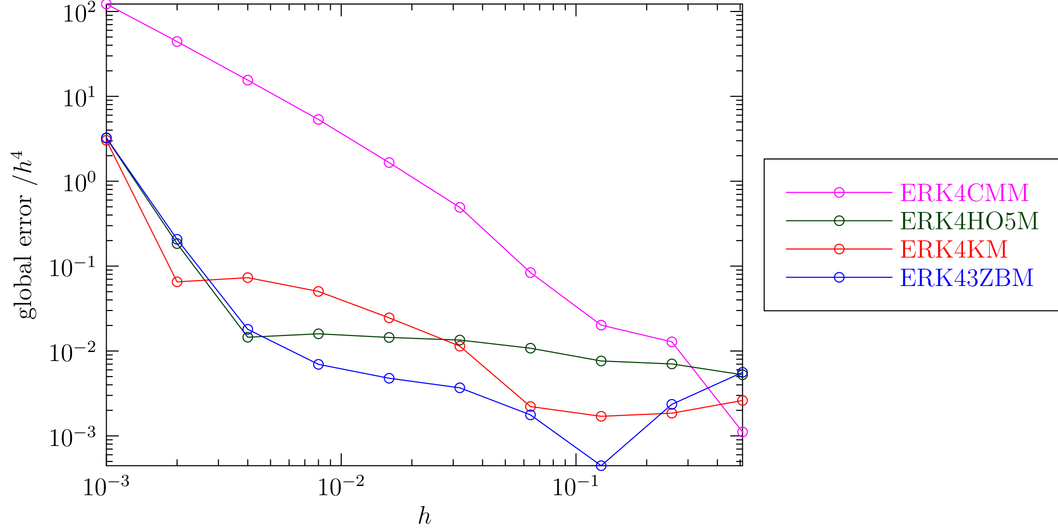


Figure 5.2: Error when solving Equation 5.19 with ERK4KM, ERK4CMM, ERK4HO5M, and ERK43ZBM.

we calculated the matrix φ_k functions with the help of Padé approximants. The plots show the L^2-norm of the error at the time $t = 1$. They appear different from the plots in the original paper only because we first dividing the error by h^4 (since the methods in consideration are supposed to be fourth order), allowing us to examine the constant C in front of h^4 in Equation 2.6. This gives us a better grasp of which among various methods of the same order is better for the given problem.

We could use this exact problem in our tests, but we noticed a small issue. The numerical treatment of the integral term is done with the Simpson method, which for the particular integral in Equation 5.19 can be written as a matrix-vector multiplication. This means it is a linear term and hence could be fused with the linear term coming from the discretized Laplacian. Therefore, all exponential integrators would solve this problem exactly. Since treating numerically a part of the equation that can be treated exactly is not a fair comparison, we propose to modify Equation 5.19 to

$$\frac{\partial y}{\partial t}(x,t) - \frac{\partial^2 y}{\partial x^2}(x,t) = \int_0^1 y^4(\bar{x},t)\, d\bar{x} + \Phi(x,t). \tag{5.22}$$

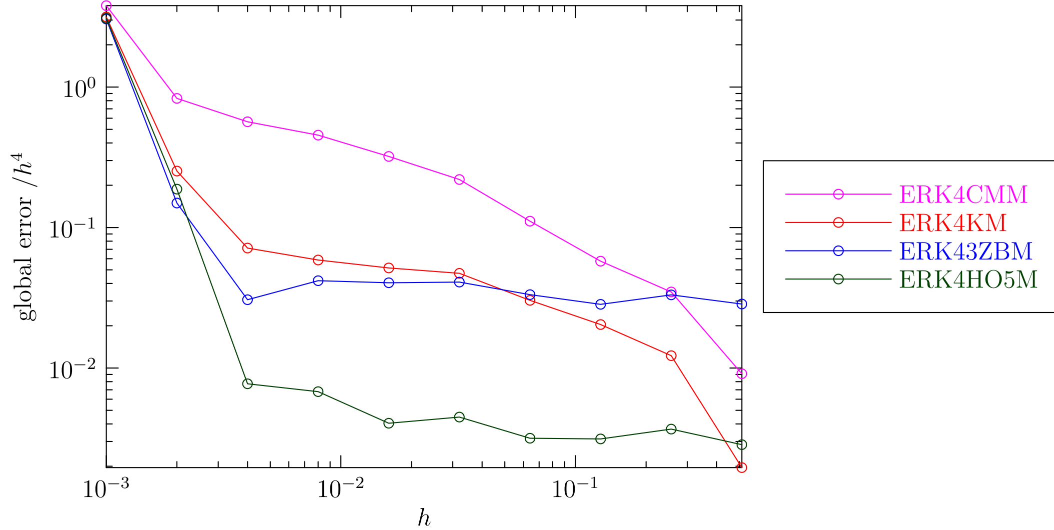


Figure 5.3: Error when solving Equation 5.22 with ERK4KM, ERK4CMM, ERK4HO5M, and ERK43ZBM.

Both the space and time domain remain the same as in Problem 5.19. The Laplacian term is discretized like in 5.21 and the function Φ is also calculated by substituting the exact solution in Equation 5.22, which again is

$$y(x,t) = x(1-x)e^t. \tag{5.23}$$

The error plots after solving the discretized version of Problem 5.22 with ERK4KM, ERK4CMM, ERK4HO5M, and ERK43ZBM are displayed in Figure 5.3. The results of this modified test are in line with the results from the original problem from Hochbruck & Ostermann [2005]: ERK4K and ERK4CM suffer from order reduction.

The next test that we will do on Equation 5.22 will be the implementation of the Schur decomposition idea. Even though the only two stiff-order four ERK methods that we have are ERK4HO5 and the fourth estimate in ERK43ZB, we will compare these two implementations for all methods used in Figure 5.3. Figure 5.4a shows

the two error plots for ERK4HO5: the implementation involving the full discretized Laplacian matrix and the implementation where the Laplacian is reduced to a diagonal matrix via Schur decomposition. Figure 5.4b shows the same situation for the fourth-order estimate of ERK43ZB. Figure 5.5a and Figure 5.5b emphasize that even when the ERK4K and ERK4CM methods are applied to systems with a diagonal linear term, they can still suffer from order reduction.

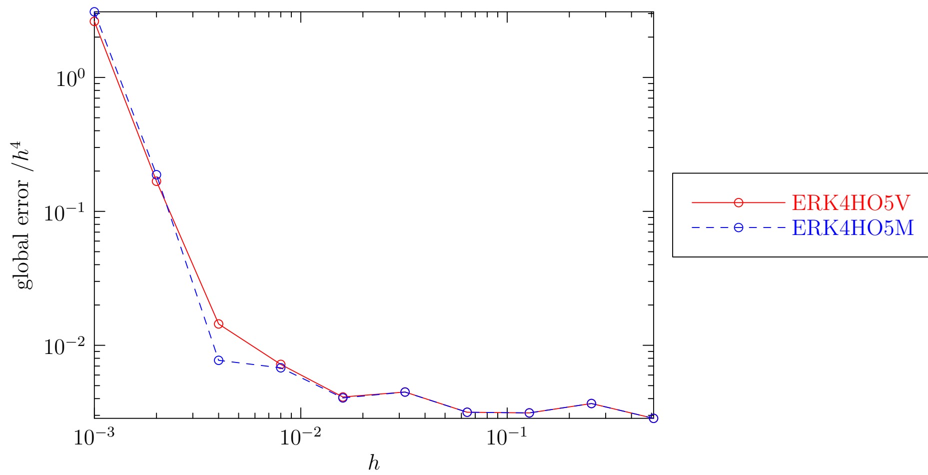
The last numerical test that we will do concerns the capability of our method ERK43ZBV to adjust the step size as the numerical solution is evolved in order for the error to remain within a specified tolerance. We revisit another example from Hochbruck & Ostermann [2005]:

$$\frac{\partial y}{\partial t}(x,t) - \frac{\partial^2 y}{\partial x^2}(x,t) = \frac{1}{1+y(x,t)^2} + \Phi(x,t), \tag{5.24}$$

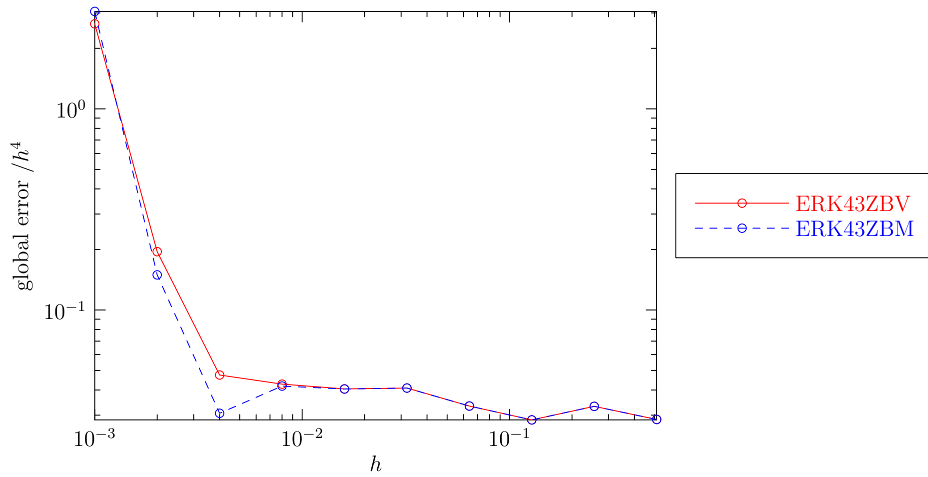
for $x \in [0,1]$. Again, we discretize in space using 200 grid points, although this time we continue the time integration from $t_0 = 0$ to $t_n = 3$ in order to show how ERK43ZBV performs over a longer time. The exact solution of Equation 5.24 is again taken to be

$$y(x,t) = x(1-x)e^t \tag{5.25}$$

and the term $\Phi(x,t)$ is calculated by substituting the exact solution into the equation. Our plan was to compare our stiff $(4,3)$ pair with the stiff $(4,3)$ pair ERK43DKV, but as we explained at the end of the previous chapter, the third-order estimate in ERK43DKV is actually fourth order for some problems, as shown in Figure 5.6. This makes it falsely conclude that the difference between its fourth-order estimate and its (supposed) third-order estimate is extremely small and hence the method does not properly adapt the step size. In Figure 5.7 we show the evolution of the

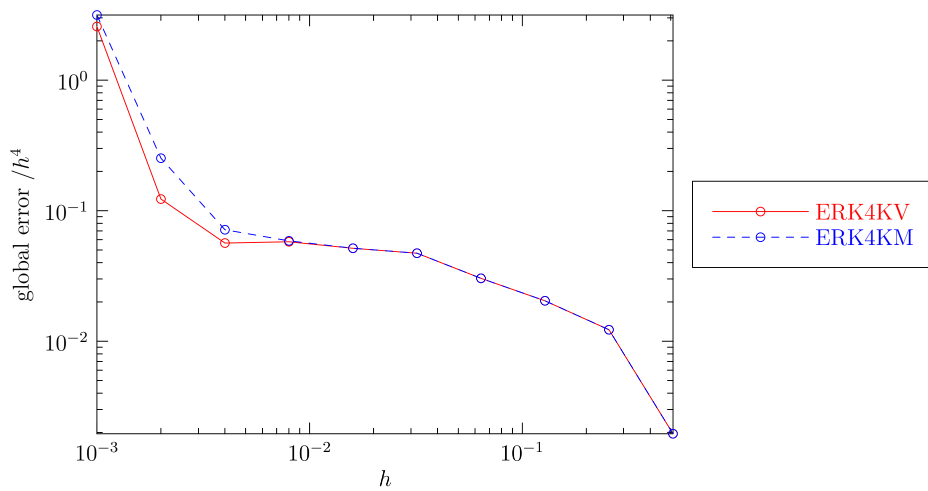


(a) Error comparison for ERK4HO5.

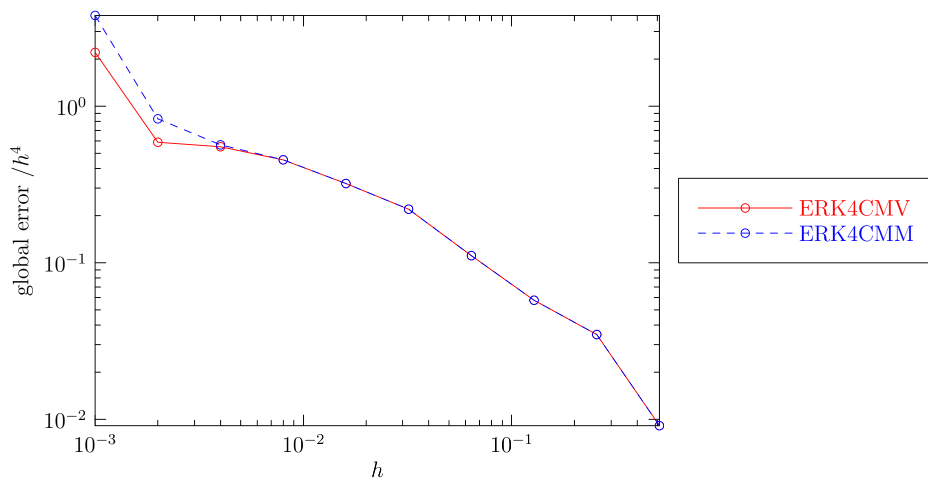


(b) Error comparison for ERK43ZB.

Figure 5.4: Error comparison when solving Equation 5.22 with and without the Schur decomposition.



(a) Error comparison for ERK4K.



(b) Error comparison for ERK4CM.

Figure 5.5: Error comparison when solving Equation 5.22 with and without the Schur decomposition.

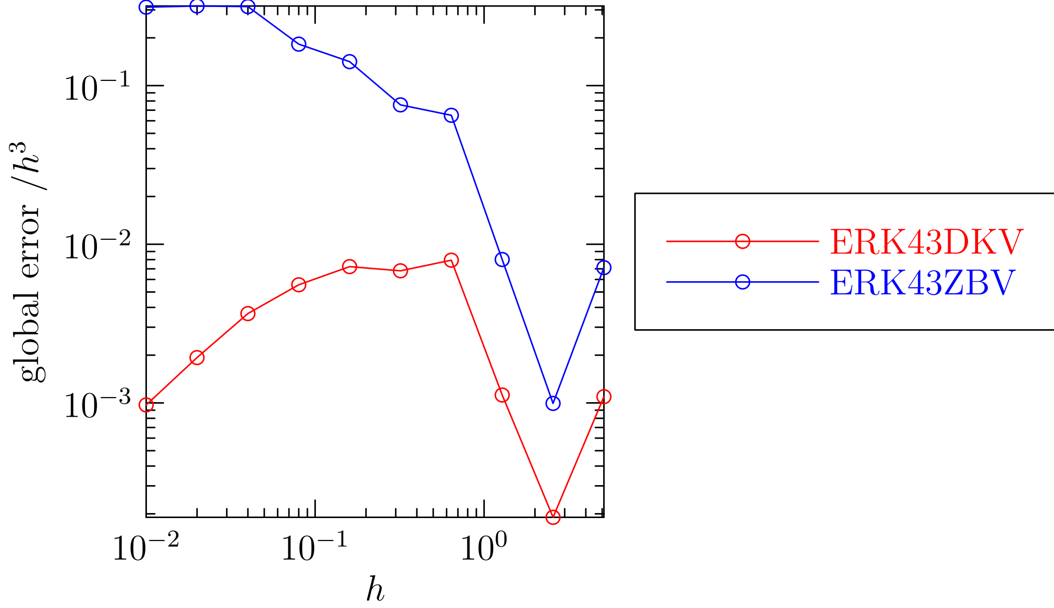


Figure 5.6: Error comparison of the third-order estimates for Equation 5.24 from $t = 0$ to $t = 3$.

L^∞ error over the time interval $[0, 3]$. We note that ERK43ZBV successfully adapts the time step to keep the error small, whereas ERK43DKV erroneously continues to increase the time step without bound!

Perhaps the best way to showcase the ability of our method ERK43ZBV to adjust the step size over the course of a long time integration is by trying to approximate a solution that is periodic in time. Consider again the equation:

$$\frac{\partial y}{\partial t}(x,t) - \frac{\partial^2 y}{\partial x^2}(x,t) = \frac{1}{1 + y(x,t)^2} + \Phi(x,t), \tag{5.26}$$

for $x \in [0, 1]$. We discretize in space using 200 grid points and integrate in time from $t_0 = 0$ to $t_n = 30$. The exact solution of Equation 5.26 is taken this time to be

$$y(x,t) = 10(1 - x)x(1 + \sin(t)) + 2 \tag{5.27}$$

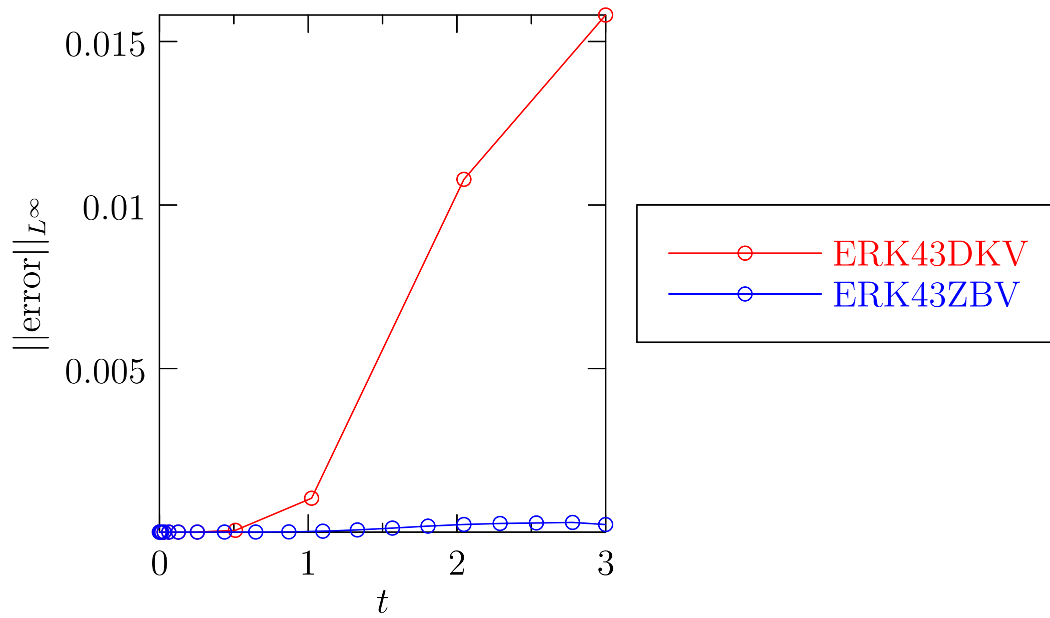


Figure 5.7: Evolution of the error for Equation 5.24 from $t = 0$ to $t = 3$.

and the term $\Phi(x, t)$ is calculated by substituting the exact solution into the equation. In Figure 5.8 we plot the L^∞ norm of the error at each time step accepted by our embedded method.

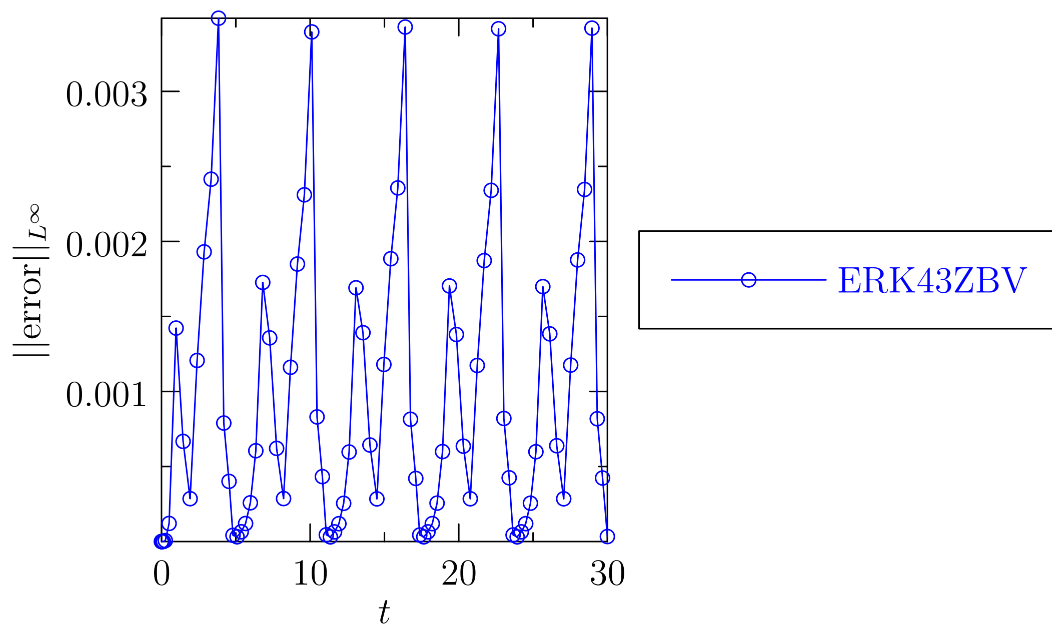


Figure 5.8: Evolution of the error for Equation 5.26 from $t = 0$ to $t = 30$.

Chapter 6

Conclusion and Future Work

We have been introduced to

$$\frac{dy}{dt} = F(t, y) - Ly; \tag{6.1}$$

a type of equation that arises frequently in scientific problems. In attempting to numerically solve equations like 6.1, we saw that explicit methods fail when using large step sizes, but implicit methods showed no signs of failing at any step size (although they may still be inaccurate for large time steps). This has been recognized in the literature as the phenomenon of stiffness, although defining it that way is not quantitative at all. We saw that there exist a number of attempts at defining stiffness and after showing that most of them were inadequate, we managed to properly define stiffness by making use of local Lyapunov exponents and the curvature of the solutions. Despite being able to accurately solve stiff systems like 6.1, implicit methods compute the solution by iteratively finding a root of a nonlinear equation at each time step. This makes implicit methods computationally expensive when the size of the system is large.

In hopes of being able to achieve high accuracy for large step sizes while still using an explicit method for computing a solution, we introduced exponential Runge–Kutta (ERK) methods. We saw that when the numerical stiffness comes from the linear term in 6.1, ERK methods can deal with it by treating the linear term exactly. After giving a brief history of this family of explicit methods, we discussed the stiff-order conditions that were derived by Hochbruck & Ostermann [2005] to construct high-order ERK methods. We developed our own fourth-order exponential Runge–Kutta method, ERK43ZB, by solving these conditions with Mathematica. This new method was designed such the second-last stage gives a third-order approximation to the solution, making it an embedded ERK method that allows for step-size adjustment during the time integration of Equation 6.1. Moreover, ERK43ZB was optimized such that the fourth-order estimate has minimal error in general and the third-order estimate is never fourth order, avoiding the problems with step size adjustment that we witnessed with ERK43DKV in Figure 5.7.

Even though ERK methods do not employ an iterative method at each time step like implicit methods do, they are still computationally intensive. ERK methods treat the linear term in 6.1 by calculating exponentials and related functions of L. This is not a problem when L is just a number or if L is a diagonal matrix, but it is a computational burden when L is a general matrix (even if it is sparse). A brief discussion about how the exponential of a matrix has been dealt with in the literature of exponential Runge–Kutta methods was given in Chapter 5. We then introduced the idea of computing the Schur decomposition of the general matrix L in order to transform Equation 6.1 to an equation where the linear term is a triangular matrix. The diagonal part of that new linear term is treated exactly by the ERK method and the strictly triangular part is treated explicitly, together with the nonlinear term. We saw that this would be particularly useful for embedded ERK methods, because

otherwise, every time that the step size was adjusted, the matrix functions would have be recalculated. With Schur decomposition, the functions recalculated at every adjusted step size would only be functions of a diagonal matrix. Since the Schur decomposition algorithm only needs to be run once at the very beginning, this greatly optimizes the embedded ERK methods and makes them a viable choice for high-performance computing.

Lastly, we would like to remark that while we have defined stiffness in general, we can only circumvent *linear* stiffness (the one that comes from the linear term) with exponential Runge–Kutta methods. If there is stiffness coming from $F(t, y)$ too or if there is no linear term at all in the equation 6.1, then extra processing has to be done before we apply an ERK method. Hochbruck & Ostermann [2010] suggests that if $L = 0$, then one can linearize the equation around a certain state and artificially get a linear term. Dendumrongsup & Tartakovsky [2021] consider a few possible states around which to linearize, but their testing has been done with ERK methods that are not proven to satisfy the stiff-order conditions by Hochbruck & Ostermann [2005]. It remains as a plan for future work to test different linearizing strategies. Furthermore, we think that it is worth testing the idea of linearizing at each step independently. However, by doing this, the matrix L in the resulting linear part is not guaranteed to be diagonal and we would have to perform a Schur decomposition at each step. The testing of this idea also remains for a future work, but it is a promising thought nonetheless, considering the computational cost that comes with the alternative being to simply use implicit methods. Another potential improvement to consider is to account for the off-diagonal terms of the triangular matrix from the Schur decomposition by using optimized algorithms for calculating functions of triangular matrices.

Bibliography

[Balac & Mahé 2013] S. Balac & F. Mahé, Computer Physics Communications, **184**:1211, 2013.

[Bogacki & Shampine 1989] P. Bogacki & L. F. Shampine, Appl. Math. Letters, **2**:1, 1989.

[Bowman *et al.* 2006] J. C. Bowman, C. R. Doering, B. Eckhardt, J. Davoudi, M. Roberts, & J. Schumacher, Physica D, **218**:1, 2006.

[Bowman 2005] J. C. Bowman, Robust efficient routines to compute $\phi_n(x)$ for n=1 to 4., https://github.com/dealias/triad/blob/master/phi.h, 2005.

[Boyd 2001] J. P. Boyd, *Chebyshev and Fourier spectral methods*, Courier Corporation, 2001.

[Butcher 2009] J. C. Butcher, Canadian Applied Mathematics Quarterly, **17**:433, 2009.

[Butcher 2016] J. C. Butcher, *Numerical methods for ordinary differential equations*, John Wiley & Sons, 2016.

[Cartwright 1999] J. H. Cartwright, Physics Letters A, **264**:298, 1999.

[Cash & Karp 1990] J. R. Cash & A. H. Karp, ACM Transactions on Mathematical Software (TOMS), **16**:201, 1990.

[Certaine 1960] J. Certaine, Mathematical methods for digital computers, **1**:128, 1960.

[Cox & Matthews 2002] S. Cox & P. Matthews, J. Comp. Phys., **176**:430, 2002.

[Curtiss & Hirschfelder 1952] C. F. Curtiss & J. O. Hirschfelder, Proceedings of the National Academy of Sciences, **38**:235, 1952.

[Dendumrongsup & Tartakovsky 2021] N. Dendumrongsup & D. M. Tartakovsky, Computational Geosciences, **25**:1667, 2021.

[Ding & Kang 2017] X. Ding & S. Kang, arXiv preprint arXiv:1703.09622, 2017.

[Dormand & Prince 1980] J. R. Dormand & P. J. Prince, Journal of computational and applied mathematics, **6**:19, 1980.

[Henry 2006] D. Henry, *Geometric theory of semilinear parabolic equations*, volume 840, Springer, 2006.

[Higham 2008] N. J. Higham, *Functions of matrices: theory and computation*, SIAM, 2008.

[Hochbruck & Ostermann 2005] M. Hochbruck & A. Ostermann, SIAM J. Numer. Anal., **43**:1069, 2005.

[Hochbruck & Ostermann 2010] M. Hochbruck & A. Ostermann, Acta Numerica, **19**:209, 2010.

[Kassam & Trefethen 2005] A.-K. Kassam & L. N. Trefethen, SIAM Journal on Scientific Computing, **26**:1214, 2005.

[Krogstad 2005] S. Krogstad, Journal of Computational Physics, **203**:72, 2005.

[Kutta 1901] W. Kutta, Z. Math. Phys., **46**:435, 1901.

[Lambert 1991] J. D. Lambert, *Numerical methods for ordinary differential systems: the initial value problem*, John Wiley & Sons, Inc., 1991.

[Luan & Ostermann 2014] V. T. Luan & A. Ostermann, Journal of Computational and Applied Mathematics, **256**:168, 2014.

[Lunardi 1996] A. Lunardi, Applications of Mathematics-New York, **41**:79, 1996.

[Moler & Van Loan 2003] C. Moler & C. Van Loan, SIAM review, **45**:3, 2003.

[Prasolov 1994] V. V. Prasolov, *Problems and theorems in linear algebra*, volume 134, American Mathematical Soc., 1994.

[Runge 1895] C. Runge, Mathematische Annalen, **46**:167, 1895.

[Tokman & Loffeld 2010] M. Tokman & J. Loffeld, Procedia Computer Science, **1**:229, 2010.

[Whalen *et al.* 2015] P. Whalen, M. Brio, & J. V. Moloney, Journal of Computational Physics, **280**:579, 2015.

Appendix A

Mathematica script to check order conditions

In[1]:=

```
(* Package for Non-Commutative Albegra *)
Needs["NCAlgebra‘"]
(* To clear all variables *)
Clear["Global‘*"]
(*Modify the number of stages*)
stages = 4
s=stages;
(*Building the Butcher tableau and vector of c_i*)
a=Table[Symbol["a"<>ToString[i]<>ToString[j]],
        {i,0,stages-1},{j,0,i}]
c=Table[Symbol["c"<>ToString[i]],{i,0,stages-1}]
(*Evaluating each phi function at each c_i*)
phi1c=Table[phi1[-c[[i]]*L*h],{i,1,stages}]
phi2c=Table[phi2[-c[[i]]*L*h],{i,1,stages}]
```

```
phi3c=Table[phi3[-c[[i]]*L*h],{i,1,stages}]
```

```
Out[2]= {{a00},
          {a10,a11},
          {a20,a21,a22},
          {a30,a31,a32,a33}}
```

```
Out[3]= {c0,c1,c2,c3}
```

```
In[4]:= (* We only need to set L to be non-commutative*)
        (* and the phi functions will be so automatically.*)
        SetNonCommutative[J,K,L,a]
        (*Stiff conditions as in 05 Hochbruck paper*)
        order1 = {Sum[a[[s]][[i]],{i,stages}]==phi1[-L*h]}
        order2a = {Sum[a[[s]][[i]]*c[[i]],
                    {i,stages}] == phi2[-L*h]}
        order2b = Table[Sum[a[[i]][[k]],{k,1,i}]
                    ==phi1c[[i+1]]*c[[i+1]],{i,1,stages-1}]
        order3a = {Sum[a[[s]][[i]]*c[[i]]^2/(2!),
                    {i,1,stages}]==phi3[-L*h]}
        order3b={Sum[a[[s]][[i]]**J**(phi2c[[i]]*c[[i]]^2
                  -Sum[a[[i-1]][[j]]*c[[j]],{j,2,i-1}]),
                  {i,2,stages}] == 0}
        order4a = {Sum[a[[s]][[i]]*c[[i]]^3/(3!),{i,stages}]
                    ==phi4[-L*h]}
```

```
order4b={Sum[a[[s]][[i]]**J**(phi3c[[i]]*c[[i]]^3
          -Sum[a[[i-1]][[j]]*c[[j]]^2/(2!),
              {j,2,i-1}]),{i,2,stages}] == 0}
order4c = {Sum[a[[s]][[i]]**J**Sum[a[[i-1]][[j]]**J
              **(phi2c[[j]]*c[[j]]^2
              -Sum[a[[j-1]][[k]]*c[[k]],{k,2,j-1}]),
                  {j,2,i-1}],{i,2,stages}] == 0}
order4d={Sum[a[[s]][[i]]*c[[i]]**K
          **(phi2c[[i]]*c[[i]]^2
          -Sum[a[[i-1]][[j]]*c[[j]],
              {j,2,i-1}]),{i,2,stages}] == 0}
```

```
In[-13]:= (*Testing the method ERK4K*)
c0=0;c1=1/2;c2=1/2;c3=1;
a00=1/2*phi1[-c1*L*h];
a10=1/2*phi1[-c2*L*h] -phi2[-c2*L*h];
a11=phi2[-c2*L*h];
a20=phi1[-c3*L*h]-2*phi2[-c3*L*h];
a21=0;
a22=2*phi2[-c3*L*h];
a30=phi1[-L*h]-3*phi2[-L*h]+4*phi3[-L*h];
a31=2*phi2[-L*h]-4*phi3[-L*h];
a32=2*phi2[-L*h]-4*phi3[-L*h];
a33=4*phi3[-L*h]-phi2[-L*h];
```

In[-3]:=
```
(*  Checking whether conditions hold  *)
(*  exactly for orders 1,2,3  *)
allorder1 = order1;
FullSimplify[allorder1]
allorder2 = Join[order2a,order2b];
FullSimplify[allorder2]
allorder3 = Join[order3a,order3b];
FullSimplify[allorder3]
```

Out[-2]=
```
{True}
```

Out[0]=
```
{True,True,True,True}
```

Out[2]=
```
{True,True}
```

In[3]:=
```
(*  Checking whether condition 4 holds weakly   *)
(*  i.e. b_i(0) instead      of b_i(-hL)  *)
(*  Re-read Theorem 4.7 in the Ho&Os05 paper *)
(*  when applying this step, "holding weakly" *)
(*  has a very  particular meaning *)
a30=a30/.L->0;
a31=a31/.L->0;
a32=a32/.L->0;
a33=a33/.L->0;
```

```
phi1[0]=1;phi2[0]=1/2;phi3[0]=1/(3!);
phi4[0]=1/(4!);phi5[0]=1/(5!);
(* The following line takes care of  *)
(* psi_p(0)=0 in "holding weakly" *)
order4a = order4a/.{L->0};
allorder4=Join[order4a,order4b,order4c,order4d];
FullSimplify[allorder4]
```

Out[3]=

```
{True,
phi2[-h L]**J**phi2[-h L]
+2 (-2 phi2[-h L]**J**phi3[-h L]
+phi2[-h L]**J**phi3[-(h L)/2]
-2 phi3[-h L]**J**phi2[-h L]
+phi3[-h L]**J**phi2[-(h L)/2]
+8 phi3[-h L]**J**phi3[-h L]
-2 phi3[-h L]**J**phi3[-(h L)/2])
==phi2[-h L]**J**phi2[-(h L)/2],
phi2[-h L]**J**phi2[-h L]**J**phi2[-(h L)/2]
+phi2[-h L]**J**phi2[-(h L)/2]**J**phi2[-(h L)/2]
==4 phi3[-h L]**J**phi2[-h L]**J**phi2[-(h L)/2]
+2 phi3[-h L]**J**phi2[-(h L)/2]**J**phi2[-(h L)/2],
True}
```

Appendix B

Mathematica script to construct stiff embedded ERK methods

In[4]:=

```
(* NCAlgebra to allow for non commutative expr *)
Needs["NCAlgebra`"]
(* To clear all variables *)
Clear["Global`*"]
(*Modify the number of stages*)
stages = 5
s=stages;
(*Building the Butcher tableau and vector of c_i*)
a=Table[Symbol["a"<>ToString[i]<>ToString[j]],
                        {i,0,stages-1},{j,0,i}]
c=Table[Symbol["c"<>ToString[i]],{i,0,stages}]

phi1c=Table[phi1[-c[[i]]*L*h],{i,1,stages}]
phi2c=Table[phi2[-c[[i]]*L*h],{i,1,stages}]
```

```
phi3c=Table[phi3[-c[[i]]*L*h],{i,1,stages}]

(* Set the NonCommutative variables *)
(* We only need to make L non-commutative *)
(* and every function of L is automatically *)
(* non-commutative (i.e. Phi functions) *)
SetNonCommutative[J,K,L,a];
```

```
Out[4]= {{a00},
         {a10,a11},
         {a20,a21,a22},
         {a30,a31,a32,a33},
         {a40,a41,a42,a43,a44}}
        {c0,c1,c2,c3,c4}
```

```
In[6]:= (* Specifying c_i and evaluating phi funcs at 0 *)
        c0=0;c1=1/6;c2=1/2;c3=1/2;
        c4=1;
        c5=1;
        phi1[0]=1;phi2[0]=1/2;phi3[0]=1/(3!);
        phi4[0]=1/(4!);phi5[0]=1/(5!);
```

In the cell below we build each weight a_{ij} as a linear combination of $\varphi_k(-c_j hL)$

functions. For example:

$$\begin{aligned} a_{00} = {} & a_{0011}\varphi_1(-c_1hL) + a_{0012}\varphi_1(-c_2hL) + a_{0013}\varphi_1(-c_3hL) \\ & + a_{0014}\varphi_1(-c_4hL) + a_{0015}\varphi_1(-c_5hL) + a_{0021}\varphi_2(-c_1hL) \\ & + a_{0022}\varphi_2(-c_2hL) + a_{0023}\varphi_2(-c_3hL) + a_{0024}\varphi_2(-c_4hL) \\ & + a_{0025}\varphi_2(-c_5hL) + a_{0031}\varphi_3(-c_1hL) + a_{0032}\varphi_3(-c_2hL) \\ & + a_{0033}\varphi_3(-c_3hL) + a_{0034}\varphi_3(-c_4hL) + a_{0035}\varphi_3(-c_5hL). \end{aligned}$$

However, we have implemented an optimizer such that in case there are some equal c_i, we will not have repeating terms in the linear combinations for the weights.

In[10]:=

```
(* Joining all the phi functions as lists *)
(* based on whether they are phi1, phi2 etc *)
phiFunctions = {Join[phi1c,{phi1[-L*h]}],
                Join[phi2c,{phi2[-L*h]}],
                Join[phi3c,{phi3[-L*h]}]};
(* Variables to solve for *)
vars = {};
(* When some c_i are the same *)
optimizer = {};
(* Looping over weights a_ij using i and j *)
For[i=0,i<stages,i++,
  tempListA = {};
  For[j=0,j<i+1,j++,
    tempA = 0;
    (* Looping over phi_m evaluated *)
```

```
  (* at c_n using m and n *)
  For[m=1,m<Length[phiFunctions]+1,m++,
  optimizer = {};
    (* n starts at 2 to avoid having *)
    (* phi_m at c_0 since c_0=0 *)
    For[n=2,n<Length[phiFunctions[[m]]]+1,n++,
      (* Forming the linear combinations *)
      (* of phi for a_ij *)
      If[MemberQ[optimizer,c[[n]]],,
      tempA+=Symbol["a"<>ToString[i]<>ToString[j]
                    <>ToString[m]<>ToString[n-1]]
                    *phiFunctions[[m,n]];
      SetCommutative[Symbol["a"<>ToString[i]<>
        ToString[j]<>ToString[m]<>ToString[n-1]]];
      AppendTo[vars,Symbol["a"<>ToString[i]<>
                    ToString[j]<>ToString[m]<>
                    ToString[n-1]]];
      AppendTo[optimizer,c[[n]]];
      ];
    ];
  ];
  AppendTo[tempListA,tempA];
];
(* Assigning the linear combinations to *)
(* weights a_ij. Crucial step so we can *)
(* use them as variables later. *)
```

```
  MapThread[Set,{a[[i+1]],tempListA}];
];
```

In[14]:=

```
(* To get all coefficients in front of any *)
(* phi_i eval at each c_j. We do not  *)
(* include phi_i at c_0 since c_0=0. *)
phiFunctionsAllTogetherNoc0 =
  {Join[Delete[phi1c,1],{phi1[-L*h]}],
   Join[Delete[phi2c,1],{phi2[-L*h]}],
   Join[Delete[phi3c,1],{phi3[-L*h]}]};
phiFlattened =
      Union[Flatten[phiFunctionsAllTogetherNoc0]];
(* To gather all possible terms involving *)
(* the operators J and K. *)
terms = {};
termsJ = {J};
termsK={K};
termsJJ={J**J};
For[i = 1, i<Length[phiFlattened]+1,i++,
  AppendTo[termsJ,J**phiFlattened[[i]]];
  AppendTo[termsK,K**phiFlattened[[i]]];
  For[j=1,j<Length[phiFlattened]+1,j++,
    AppendTo[termsJ,phiFlattened[[i]]
                        **J**phiFlattened[[j]]];
    AppendTo[termsK,phiFlattened[[i]]
```

```
                          **K**phiFlattened[[j]]];
    ];
];
For[i = 1, i<Length[termsJ]+1,i++,
  For[j=1,j<Length[phiFlattened]+1,j++,
    AppendTo[termsJJ,termsJ[[i]]
                          **J**phiFlattened[[j]]];
  ];
];

termTypes = Join[phiFlattened,termsJ,
                 termsJJ,termsK];
(* To form a list with all Coefs *)
(* for all types of terms *)
allCoefs = {};
```

The stiff-order conditions up to order three for a method with $\tilde{s}$ stages are

$$\sum_{i=0}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL) = \varphi_1(-hL), \tag{2.1}$$

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)c_i = \varphi_2(-hL), \tag{2.2}$$

$$\sum_{j=0}^{i-1} a_{ij}(-hL) = c_i\varphi_1(-c_ihL), \tag{2.3}$$

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)c_i^2 = 2\varphi_3(-hL), \tag{2.4}$$

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)J(\varphi_2(-c_ihL)c_i^2 - \sum_{j=1}^{i-1} a_{ij}(-hL)c_j) = 0. \tag{2.5}$$

The reader is reminded that we have been using 0-indexing in the Butcher tableaux, but Mathematica uses 1-indexing for its arrays. Therefore, there are some discrepancies between how the equations for the stiff-order conditions are written and how they are implemented.

```
In[10]:=  (* To append coefs from order1lower *)
          (* Equal->Subtract gets all terms on one side*)
          order1lower = {Sum[a[[s-1]][[i]],{i,stages-1}]
                         ==phi1[-L*h]};
          allorder1lower = order1lower;
          allorder1Coeflower = CoefficientRules[
                         allorder1lower/.Equal->Subtract,
                         termTypes];
          For[i = 1, i<Length[allorder1Coeflower]+1,i++,
            For[j=1,j<Length[allorder1Coeflower[[i]]]+1,j++,
              AppendTo[allCoefs,allorder1Coeflower[[i,j,2]]];
```

```
  ];
];
```

```
In[3]:= (* To append coefs from order2lower *)
order2alower = {Sum[a[[s-1]][[i]]*c[[i]],
                    {i,stages-1}] == phi2[-L*h]};
order2blower = Table[Sum[a[[i]][[k]],{k,1,i}]
          ==phi1c[[i+1]]*c[[i+1]],{i,1,stages-2}];
allorder2lower = Join[order2alower,order2blower];
allorder2Coeflower = CoefficientRules[
      allorder2lower/.Equal->Subtract,termTypes];
For[i = 1, i<Length[allorder2Coeflower]+1,i++,
  For[j=1,j<Length[allorder2Coeflower[[i]]]+1,j++,
    AppendTo[allCoefs,allorder2Coeflower[[i,j,2]]];
  ];
];
```

```
In[8]:= (* Saving a_(s-2)i before evaluating their *)
(* phi functions at 0 a_(s-2)i will be used *)
(* again for the higher-order method. *)
asoriginal = a[[s-1]];
a30=a30/.L->0;
a31=a31/.L->0;
a32=a32/.L->0;
a33=a33/.L->0;
```

```
(* To append coefs from order3lower *)
(* Applying conditions for order 3 weakly: *)
(* a_(s-2)i(0) instead of a_(s-2)i(-hL) *)
(* Always reread Theorem 4.7 in the Ho&Os05 paper *)
(* when applying this step, "holding weakly" has *)
(* a very particular meaning *)
order3alower = {Sum[a[[s-1]][[i]]*c[[i]]^2/(2!),
                  {i,stages-1}]==phi3[-L*h]};
order3blower={ExpandNonCommutativeMultiply[
               Sum[a[[s-1]][[i]]**J**(phi2c[[i]]
                *c[[i]]^2-Sum[a[[i-1]][[j]]*c[[j]],
                {j,2,i-1}]),{i,2,stages-1}]] == 0};
(* The following line takes care of psi_p(0)=0 *)
(* in "holding weakly" *)
order3alower = order3alower/.{L->0};
allorder3lower = Join[order3alower,order3blower];
allorder3Coeflower = CoefficientRules[
        allorder3lower/.Equal->Subtract,termTypes];
For[i = 1, i<Length[allorder3Coeflower]+1,i++,
  For[j=1,j<Length[allorder3Coeflower[[i]]]+1,j++,
  AppendTo[allCoefs,allorder3Coeflower[[i,j,2]]];
  ];
];
```

Next, we implement the order conditions for the higher-order method. Below are the

conditions that the method needs to satisfy for stiff-order four. We implement these conditions weakly in the sense of Theorem 3.1, together with strongly implementing the conditions up to order three given explicitly above. Again, because of the different indexing that Mathematica uses for its arrays, there are some discrepancies between how the conditions are written and how they are implemented.

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)c_i^3 = 6\varphi_4(-hL), \quad (2.6)$$

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)J(\varphi_3(-c_ihL)c_i^3 - \frac{1}{2}\sum_{j=1}^{i-1} a_{ij}(-hL)c_j^2) = 0, \quad (2.7)$$

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)J\sum_{j=1}^{i-1} a_{ij}(-hL)J(\varphi_2(-c_jhL)c_j^2 - \sum_{k=1}^{i-1} a_{jk}(-hL)c_k) = 0, \quad (2.8)$$

$$\sum_{i=1}^{\tilde{s}-1} a_{(\tilde{s}-1)i}(-hL)c_iK(\varphi_2(-c_ihL)c_i^2 - \sum_{j=1}^{i-1} a_{ij}(-hL)c_j) = 0. \quad (2.9)$$

```
In[20]:=  (* Restoring the weights a_(s-2)i *)
          a[[s-1]] = asoriginal;
          (* To append coefs from order1higher *)
          order1higher = {Sum[a[[s]][[i]],
                        {i,stages}]==phi1[-L*h]};
          allorder1higher = order1higher;
          allorder1higherCoef = CoefficientRules[
                allorder1higher/.Equal->Subtract,termTypes];
          For[i = 1, i<Length[allorder1higherCoef]+1,i++,
            For[j=1,j<Length[allorder1higherCoef[[i]]]+1,j++,
            AppendTo[allCoefs,allorder1higherCoef[[i,j,2]]];
            ];
          ];
```

```
In[21]:=  (* To append coefs from order2higher *)
order2ahigher = {Sum[a[[s]][[i]]*c[[i]],
              {i,stages}] == phi2[-L*h]};
order2bhigher = Table[Sum[a[[i]][[k]],{k,1,i}]
          ==phi1c[[i+1]]*c[[i+1]],{i,1,stages-1}];
allorder2higher = Join[order2ahigher,order2bhigher];
allorder2higherCoef = CoefficientRules[
      allorder2higher/.Equal->Subtract,termTypes];
For[i = 1, i<Length[allorder2higherCoef]+1,i++,
  For[j=1,j<Length[allorder2higherCoef[[i]]]+1,j++,
  AppendTo[allCoefs,allorder2higherCoef[[i,j,2]]];
  ];
];
```

```
In[22]:=  (* To append coefs from order3higher *)
order3ahigher = {Sum[a[[s]][[i]]*c[[i]]^2/(2!),
                  {i,stages}]==phi3[-L*h]};
order3bhigher={ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]**J**(phi2c[[i]]*c[[i]]^2
          -Sum[a[[i-1]][[j]]*c[[j]],
          {j,2,i-1}]),{i,2,stages}]] == 0};
allorder3higher = Join[order3ahigher,order3bhigher];
allorder3higherCoef = CoefficientRules[
      allorder3higher/.Equal->Subtract,termTypes];
```

```
For[i = 1, i<Length[allorder3higherCoef]+1,i++,
  For[j=1,j<Length[allorder3higherCoef[[i]]]+1,j++,
  AppendTo[allCoefs,allorder3higherCoef[[i,j,2]]];
  ];
];
```

```
In[22]:= (* Saving a_(s-1)i before evaluating their *)
(* phi functions at 0 a_(s-1)i will be used *)
(* again for the higher-order method *)
asOriginal = a[[s]];
a40=a40/.L->0;
a41=a41/.L->0;
a42=a42/.L->0;
a43=a43/.L->0;
a44=a44/.L->0;
(* To append coefs from order4higher *)
(* Applying conditions for order 4 weakly on last *)
(* intermediate stage: *)
(* a_(s-1)i(0) instead of a_(s-1)i(-hL) *)
(* Always reread Theorem 4.7 in the Ho&Os05 paper *)
(* when applying this step, "holding weakly" has *)
(* a very particular meaning *)
order4ahigher = {Sum[a[[s]][[i]]*c[[i]]^3/(3!),
                    {i,stages}]==phi4[-L*h]};
order4bhigher={ExpandNonCommutativeMultiply[
```

```
          Sum[a[[s]][[i]]**J**(phi3c[[i]]*c[[i]]^3
          -Sum[a[[i-1]][[j]]*c[[j]]^2/(2!),
          {j,2,i-1}]),{i,2,stages}]] == 0};
order4chigher = {ExpandNonCommutativeMultiply[
          Sum[a[[s]][[i]]**J**Sum[a[[i-1]][[j]]
          **J**(phi2c[[j]]*c[[j]]^2
          -Sum[a[[j-1]][[k]]*c[[k]],{k,2,j-1}]),
          {j,2,i-1}],{i,2,stages}]] == 0};
order4dhigher={ExpandNonCommutativeMultiply[
          Sum[a[[s]][[i]]*c[[i]]
          **K**(phi2c[[i]]*c[[i]]^2
          -Sum[a[[i-1]][[j]]*c[[j]],
          {j,2,i-1}]),{i,2,stages}]] == 0};
(* The following line takes care of psi_p(0)=0 *)
(* in "holding weakly" *)
order4ahigher = order4ahigher/.{L->0};
allorder4higher = Join[order4ahigher,order4bhigher,
                       order4chigher,order4dhigher];
allorder4higherCoef = CoefficientRules[
      allorder4higher/.Equal->Subtract,termTypes];
For[i = 1, i<Length[allorder4higherCoef]+1,i++,
  For[j=1,j<Length[allorder4higherCoef[[i]]]+1,j++,
  AppendTo[allCoefs,allorder4higherCoef[[i,j,2]]];
  ];
];
```

In[1107]:=
```
(* Solving the constraints. *)
solution=Solve[allCoefs==0];
```

In[1107]:=
```
(* Selecting the method *)
method=solution[[1]];
a[[s]]=asOriginal;
(* Displaying the Butcher tableau with the free *)
(* variables still in it. *)
a = a/.method;
```

In[1224]:=
```
(* Using the freedom in selecting coefficients *)
(* 3rd order method should never be 4th order *)
order4alower = {Sum[a[[s-1]][[i]]*c[[i]]^3/(3!),
                  {i,stages-1}]!=phi4[-L*h]};
order4blower={Sum[a[[s-1]][[i]]*(phi3c[[i]]*c[[i]]^3
              -Sum[a[[i-1]][[j]]*c[[j]]^2/(2!),
              {j,2,i-1}]),{i,2,stages-1}] != 0};
order4clower = {Sum[a[[s-1]][[i]]*Sum[a[[i-1]][[j]]
          *(phi2c[[j]]*c[[j]]^2
          -Sum[a[[j-1]][[k]]*c[[k]],{k,2,j-1}]),
          {j,2,i-1}],{i,2,stages-1}] != 0};
order4dlower={Sum[a[[s-1]][[i]]*c[[i]]
          *(phi2c[[i]]*c[[i]]^2
          -Sum[a[[i-1]][[j]]*c[[j]],
          {j,2,i-1}]),{i,2,stages-1}] != 0};
```

```
(* Stiff Conditions reduce to Classical *)
(* Conditions when L=0, so we make sure that the *)
(* 3rd order stiff method does NOT satisfy *)
(* Classical Conditions of order 4. *)
order4alower=order4alower/.{L->0};
order4blower=order4blower/.{L->0};
order4clower=order4clower/.{L->0};
order4dlower=order4dlower/.{L->0};

allorder4lower = Join[order4alower, order4blower,
              order4clower,order4dlower]/.method;
(* Alerts if one of the 4th order conditions *)
(* is satisfied by the third order method *)
(* without choosing the free variables*)
For[i=1,i<Length[allorder4lower]+1,i++,
  If[allorder4lower[[i]],,
    Print["Condition ", i, " is satisfied by third
          order methid. Check why."];
  ];
];
```

In[1234]:=

```
(* Set the NonCommutative variables for order 5*)
SetNonCommutative[W];
(* Adding a phi4c vector similar to phi3c in the *)
(* begining. Starts at 2 to avoid using c0 since *)
```

```
(* c0=0. Use Join to add phi4 at the last c_i.*)
phi4c=Join[Table[phi4[-c[[i]]*L*h],
          {i,2,stages}],{phi4[-h*L]}];
(* To gather the rest of the terms involving *)
(* the operators J, K ans W *)
termsJphi4 = {};
termsJK={J**K};
termsKJ={K**J};
termsW={};
termsJJJ={J**J**J};
For[i = 1, i<Length[phi4c]+1,i++,
  AppendTo[termsJphi4,J**phi4c[[i]]];
];
For[i = 1, i<Length[phiFlattened]+1,i++,
  AppendTo[termsW,W**phiFlattened[[i]]];
];
For[i = 1, i<Length[termsJ]+1,i++,
  For[j=1,j<Length[phiFlattened]+1,j++,
    AppendTo[termsJK,termsJ[[i]]
              **K**phiFlattened[[j]]];
    AppendTo[termsKJ,termsK[[i]]
              **J**phiFlattened[[j]]];
  ];
];
For[i = 1, i<Length[termsJJ]+1,i++,
  For[j=1,j<Length[phiFlattened]+1,j++,
```

```
      AppendTo[termsJJJ,termsJJ[[i]]
                     **J**phiFlattened[[j]]];
    ];
  ];
  termTypes = Join[termTypes,phi4c,termsJphi4,
                    termsJK,termsKJ,termsW,termsJJJ];
```

In[1246]:=
```
  (* This cell helps to minimize the the 5th *)
  (* order error coming from having to satisfy *)
  (* the stiff conditions for order 4 strongly *)
  order4ahigherO = {Sum[a[[s]][[i]]*c[[i]]^3/(3!),
                    {i,stages}]==phi4[-L*h]};
  order4bhigherO={ExpandNonCommutativeMultiply[
        Sum[a[[s]][[i]]**J**(phi3c[[i]]*c[[i]]^3
        -Sum[a[[i-1]][[j]]*c[[j]]^2/(2!),
        {j,2,i-1}]),{i,2,stages}]] == 0};
  order4chigherO = {ExpandNonCommutativeMultiply[
        Sum[a[[s]][[i]]**J**Sum[a[[i-1]][[j]]
        **J**(phi2c[[j]]*c[[j]]^2
        -Sum[a[[j-1]][[k]]*c[[k]],{k,2,j-1}]),
        {j,2,i-1}],{i,2,stages}]] == 0};
  order4dhigherO={ExpandNonCommutativeMultiply[
        Sum[a[[s]][[i]]*c[[i]]
        **K**(phi2c[[i]]*c[[i]]^2
        -Sum[a[[i-1]][[j]]*c[[j]],
```

```
      {j,2,i-1}]),{i,2,stages}]] == 0};
allorder4higher0 = Join[order4ahigher0,
      order4bhigher0,order4chigher0,order4dhigher0];
(* Deletes conditions already satisfied strongly*)
For[i=1,i<Length[allorder4higher0]+1,i++,
  If[allorder4higher0[[i]],
    allorder4higher0[[i]]=0;,
    Print["Condition ", i, " is False. Check why."];
    allorder4higher0[[i]]=0;
  ];
];
allorder4higherCoef0 = CoefficientRules[
      allorder4higher0/.Equal->Subtract,termTypes];
(* We will put the coefficients in front of all *)
(* term types involving J,K and W. normSquared *)
(* is the squared 2-norm of that vector *)
normSquared = 0;
For[i = 1, i<Length[allorder4higherCoef0]+1,i++,
  For[j=1,j<Length[allorder4higherCoef0[[i]]]+1,j++,
    normSquared=normSquared
              +(allorder4higherCoef0[[i,j,2]])^2;
  ];
];
```

In[1247]:=

```
(* This cell helps to minimize the the 5th *)
```

```
(* order error coming from having to satisfy *)
(* the stiff conditions for order 5 weakly *)
(* Saving a_(s-1)i before evaluating their phi *)
(* functions at 0 a_(s-1)i will be used again *)
(* for displaying the method at the end *)
asOriginal = a[[s]];
a[[s]][[1]]=a[[s]][[1]]/.L->0;
a[[s]][[2]]=a[[s]][[2]]/.L->0;
a[[s]][[3]]=a[[s]][[3]]/.L->0;
a[[s]][[4]]=a[[s]][[4]]/.L->0;
a[[s]][[5]]=a[[s]][[5]]/.L->0;
(* Applying conditions for order 4 holding *)
(* weakly on last intermediate stage: *)
(* a_(s-1)i(0) instead of a_(s-1)i(-hL) *)
(* Always reread Theorem 4.7 in the Ho&Os05 paper *)
(* when applying this step, "holding weakly" *)
(* has a very particular meaning *)
order5ahigher = {Sum[a[[s]][[i]]*c[[i]]^4/(4!),
                  {i,stages}]==phi5[-L*h]};
order5bhigher={ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]**J**(phi4c[[i]]*c[[i]]^4
      -Sum[a[[i-1]][[j]]*c[[j]]^3/(3!),
      {j,2,i-1}]),{i,2,stages}]] == 0};
order5chigher = {ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]**J**Sum[a[[i-1]][[j]]
      **J**(phi3c[[j]]*c[[j]]^3
```

```
      -Sum[a[[j-1]][[k]]*c[[k]]^2/(2!),
      {k,2,j-1}]),{j,2,i-1}],
      {i,2,stages}]] == 0};
order5dhigher = {ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]**J**Sum[a[[i-1]][[j]]
      **J**Sum[a[[j-1]][[k]]
      **J**(phi2c[[k]]*c[[k]]^2
      -Sum[a[[k-1]][[l]]*c[[l]],{l,2,k-1}]),
      {k,2,j-1}],{j,2,i-1}],
      {i,2,stages}]] == 0};
order5ehigher = {ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]**J
      **Sum[(a[[i-1]][[j]]*c[[j]])**K
      **((phi2c[[j]]*c[[j]]^2 )
      -Sum[a[[j-1]][[k]]*c[[k]],{k,2,j-1}]),
      {j,2,i-1}],{i,2,stages}]] == 0};
order5fhigher={ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]*c[[i]]**K
      **(phi3c[[i]]*c[[i]]^3
      -Sum[a[[i-1]][[j]]*c[[j]]^2/(2!),
      {j,2,i-1}]),{i,2,stages}]] == 0};
order5ghigher = {ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]*c[[i]]**K
      **Sum[a[[i-1]][[j]]**J
      **(phi2c[[j]]*c[[j]]^2
      -Sum[a[[j-1]][[k]]*c[[k]],{k,2,j-1}]),
```

```
      {j,2,i-1}],{i,2,stages}]] == 0};
(* Condition 17 is satisfied like it is *)
(* satisfied in the paper by Lu&Os14 *)
order5hhigher=Table[phi2c[[i]]*c[[i]]^2
          -Sum[a[[i-1]][[j]]*c[[j]],
          {j,2,i-1}]==0,{i,2,stages}] ;
order5ihigher={ExpandNonCommutativeMultiply[
      Sum[a[[s]][[i]]*c[[i]]^2**W
      **(phi2c[[i]]*c[[i]]^2
      -Sum[a[[i-1]][[j]]*c[[j]],
      {j,2,i-1}]),{i,2,stages}]] == 0};
(* The following line takes care of *)
(* psi_p(0)=0 in "holding weakly" *)
order5ahigher = order5ahigher/.{L->0};
allorder5higher = Join[order5ahigher,
      order5bhigher,order5chigher,order5dhigher,
      order5ehigher,order5fhigher,order5ghigher,
      order5ihigher];
(* Deletes conditions already satisfied strongly *)
For[i=1,i<Length[allorder5higher]+1,i++,
  If[allorder5higher[[i]],
     allorder5higher[[i]]=0;,
     Print["Condition ",i, " is False. Check why."];
     allorder5higher[[i]]=0;
  ];
];
```

```
allorder5higherCoef = CoefficientRules[
      allorder5higher/.Equal->Subtract,termTypes];
For[i = 1, i<Length[allorder5higherCoef]+1,i++,
  For[j=1,j<Length[allorder5higherCoef[[i]]]+1,j++,

normSquared=normSquared
           +(allorder5higherCoef[[i,j,2]])^2;
  ];
];
```

In[1275]:=
```
(* Minimizing the norm of the coefficients in *)
(* front of terms involving J,K,W in the stiff *)
(* order conditions for order 5 *)
norm = {Sqrt[normSquared]};
optimalParameters = NMinimize[
          Join[norm,allorder4lower],
          Variables[Values[method]]]
(* Approximating the selected parameters *)
(* with fractions *)
fractionParams=Rationalize[
                  optimalParameters[[2]],0.1]
```

In[1276]:=
```
(* Displaying the final Butcher tableau *)
a[[s]] = asOriginal;
aERK43ZB = a/.fractionParams
```

Index

blow up, 16

classical limit, 38

embedded Runge–Kutta, 13

ERK43DK, 52

ERK43DKM, 52

ERK43DKV, 52

ERK43ZB, 55, 57

ERK43ZBM, 57

ERK43ZBV, 57

ERK4CM, 41

ERK4CMM, 41

ERK4HO5, 43, 45

ERK4HO5M, 45

ERK4HO5V, 45

ERK4K, 41

ERK4KM, 41

ERK54DK, 53, 54

ERK54DKM, 54

ERKBS32, 51

fast transient, 18

linear, 78

linear stiffness, 29

local Lyapunov exponents, 33

normal, 64

numerical stiffness, 36

order conditions, 7

persistent part, 18

pseudostiff, 21

RK4, 7

Schur decomposition, 64

slow transient, 18

stiff, 14

stiff-order conditions, 2, 42

stiff-order reduction, 42

stiffness ratio, 19

System 1, 12

System 2, 13

System 3, 19

System 4, 19

transient part, 18

unitary, 60, 64