Department of Mathematics
TUM School of Computation, Information and Technology
Technical University of Munich

Explicit Methods for Time Integration with Extended Stability Domain

Bachelor's Thesis in Mathematics

Magnus Saurbier

Thesis for the attainment of the academic degree

Bachelor of Science

at the TUM School of Computation, Information and Technology of the Technical University of Munich

Supervisor:
Prof. Dr. rer. nat. Johannes Zimmer

Advisors:
Dr. Dirk Zimmer
Andrea Neumayr

Submitted:
Munich, 01.08.2025

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Munich, 01.08.2025 Magnus Saurbier

Zusammenfassung

Anfangswertprobleme (AWP) für Systeme gewöhnlicher Differentialgleichungen (DGLs) treten in vielen realen Anwendungen auf. Die Motivation dieser Arbeit ist die Notwendigkeit, große mechanische Systeme in Echtzeit zu simulieren, um die Regelung von Robotern in Hardware-in-the-Loop-Umgebungen zu unterstützen.

Klassische Verfahren zur Lösung von AWPs umfassen Zeitschrittverfahren wie die explizite und implizite Euler-Methode. Während explizite Verfahren geringe Rechenkosten pro Schritt aufweisen, leiden sie bei steifen Systemen häufig unter Stabilitätsproblemen – insbesondere beim Kontakt von steifen Materialien. Implizite Verfahren bieten bessere Stabilitätseigenschaften, erfordern jedoch das Lösen großer Gleichungssysteme in jedem Zeitschritt, was bei hoher Systemdimension und Echtzeitanforderungen nicht praktikabel ist.

Diese Arbeit untersucht daher explizite Verfahren mit linearer Komplexität in Bezug auf die Systemdimension. Klassische explizite Methoden müssen bei steifen Problemen sehr kleine Zeitschritte verwenden, wodurch ihr Rechenvorteil verloren geht. Ziel dieser Arbeit ist es, explizite Methoden zu entwickeln, die auch bei größeren Zeitschritten stabil bleiben.

Moderne Hochleistungsrechner – insbesondere GPU-Architekturen – bevorzugen stark parallelisierbare Verfahren. Dies motiviert den Einsatz von Mehrfachableitungsmethoden, bei denen höhere Zeitableitungen parallel berechnet werden können und die sich gut für aktuelle Hardware eignen.

Kapitel 1 führt grundlegende Begriffe zu AWPs, Steifigkeit und Anforderungen an numerische Verfahren ein. Die Steifigkeit wird dabei über die Eigenwerte der Jacobi-Matrix beurteilt. Kapitel 2 gibt einen Überblick über klassische Verfahren wie Runge-Kutta-Methoden und sogenannte General Linear Methods (GLMs), einschließlich ihrer Stabilitätseigenschaften. Kapitel 3 stellt Mehrfachableitungsmethoden wie Taylor- und MDRK-Verfahren (Multiderivative Runge-Kutta) vor und beschreibt die parallele Berechnung von Zeitableitungen. MDRK-Verfahren erlauben die näherungsweise Berechnung von Zeitableitungen. Dies ist nützlich für den Fall, dass die Berechnung genügend vieler Zeitableitungen in einer bestehenden Modellierung eines Systems noch nicht implementiert wurde.

Der zentrale Beitrag dieser Arbeit wird in Kapitel 4 präsentiert: eine auf Krylov-Unterräumen basierende Technik zur effizienten Approximation der Jacobi-Matrix mit linearer Rechenkomplexität. Da die Jacobi-Matrix die lokalen linearen Dynamiken des Systems bestimmt, ermöglicht diese Approximation den Einsatz anspruchsvoller Verfahren, ohne die vollständige Berechnung der Matrix durchführen zu müssen.

Wir untersuchen dabei zwei Klassen solcher Verfahren: die *Time-Accurate Stabilized Explicit* (TASE)-Methoden von Bassenne, Fu und Mani [2] sowie exponentielle Rosenbrock-Integratoren nach Hochbruck, Ostermann und Schweitzer [15]. Diese Verfahren werden in Kapitel 5 in Kombination mit der Krylov-Jacobi-Approximation untersucht. Wir bezeichnen sie als *Krylov-Jacobi-Verfahren.*

Die Stabilität dieser Verfahren hängt von der Genauigkeit der Approximation und von der globalen Eigenwertverteilung ab. Es genügt daher nicht, die Eigenwerte eines Systems getrennt zu betrachten. Daher lässt sich der Begriff des Stabilitätsbereichs nicht direkt auf diese Verfahren beziehen. Es gibt keinen Bereich, der anhand der Position einzelner Eigenwerte die Stabilität der Methode vorhersagt. Systeme mit wenigen dominanten steifen Eigenwerten können mit großen Zeitschritten stabil integriert werden. Sind hingegen viele mäßig steife Eigenwerte breit gestreut, müssen die Zeitschritte verkleinert werden. Trotz begrenzter theoretischer Aussagen für nichtlineare Systeme zeigen unsere Ergebnisse, dass kleine nichtlineare Terme – obwohl sie große Fehler in der Operatornorm verursachen können – in der Praxis oft wenig Einfluss auf die Stabilität haben. Wichtig ist jedoch, dass im verwendeten Modell keine Unstetigkeiten oder Singularitäten in höheren Zeitableitungen auftreten.

Unter diesen Voraussetzungen erweisen sich Krylov-Jacobi-Verfahren als leistungsfähiges Werkzeug, um explizite Integratoren mit stabilitätsfördernden Eigenschaften auszustatten – ohne den Rechenaufwand vollständiger Jacobi-Auswertungen. In zahlreichen Testfällen ermöglichen sie deutlich größere Zeitschritte als klassische Verfahren wie die bekannte Runge-Kutta-Methode vierter Ordnung (RK4).

Abstract

Initial value problems (IVPs) for systems of ordinary differential equations (ODEs) appear in many real-world applications. This thesis is motivated by the need for real-time simulation of large-scale mechanical systems to support control of robots in hardware-in-the-loop environments.

Classical IVP solvers include time marching schemes such as the Euler method, available in both explicit and implicit variants. While explicit methods are computationally inexpensive, they suffer from stability issues when applied to stiff systems — especially during contact between stiff materials. Implicit methods handle stiffness more robustly but require solving large systems of equations at every time step, which is impractical under real-time constraints with high-dimensional systems.

To meet these constraints, we investigate explicit methods that scale linearly in computational cost with system size. However, classical explicit solvers require prohibitively small time steps on stiff systems, negating their efficiency. Therefore, the focus of this thesis is on developing explicit solvers that remain stable at larger time steps.

Modern high-performance computing — especially GPU architectures—favors algorithms that are parallelizable rather than sequential. This motivates the use of multiderivative methods, which allow parallel computation of higher-order time derivatives and are well-suited to modern hardware.

Chapter 1 introduces key concepts related to IVPs, stiffness, and solver requirements, and explains how stiffness is characterized through the eigenvalues of the system's Jacobian. Chapter 2 reviews classical solvers such as Runge-Kutta and general linear methods, with a focus on their stability properties. Chapter 3 introduces multiderivative methods such as Taylor and multiderivative Runge-Kutta (MDRK) integrators, and outlines how to compute multiple time derivatives in parallel. MDRK methods can be used to approximate higher time-derivatives, where their calculation is not yet implemented in the model.

Chapter 4 presents the central contribution of this thesis: a Krylov subspace-based technique for approximating the system Jacobian with linear computational complexity. Because the Jacobian determines the system's local linear dynamics, this approximation enables the use of advanced methods that incorporate Jacobian information without incurring the full computational cost of evaluating or storing it.

We focus on two classes of such methods: the time-accurate stabilized explicit (TASE) methods proposed by Bassenne, Fu, and Mani [2], and the exponential Rosenbrock integrators introduced by Hochbruck, Ostermann, and Schweitzer [15]. In chapter 5, we explore these methods in combination with the Krylov Jacobian approximation, referring to the resulting schemes as *Krylov Jacobian methods*.

The stability properties of Krylov Jacobian methods depend on the accuracy of the Jacobian approximation and differ significantly from classical methods such as Runge-Kutta schemes. They are determined by the global distribution of eigenvalues, rather than their individual positions. Therefore, the notion of the stability domain does not apply to these methods in the same sense as it does to classical methods. Krylov Jacobian methods support large time steps in systems with a few dominant stiff modes but require smaller steps on large systems where stiffness is more evenly distributed. Although theoretical guarantees are limited in nonlinear cases, our results indicate that small nonlinear terms, while leading to large operator-norm errors, have limited practical effect. However, care must be taken in system modelling to avoid discontinuities or singularities in higher derivatives used for the Krylov approximation.

When these conditions are met, Krylov Jacobian methods provide a powerful means to enhance the stability of explicit solvers without incurring the cost of full Jacobian evaluation. In many test cases, they permit significantly larger time steps than classical methods like the fourth-order Runge-Kutta (RK4) scheme.

Contents

1 Introduction

Numerical simulations have become a vital tool in analyzing and predicting the time-dependent behavior of complex systems. Research groups at the German Aerospace Center (DLR) have been using numerical simulations to model the behavior of complex systems such as aircraft, spacecraft, and other dynamical systems. Numerical simulations can be used simultaneously while operating robotic systems to predict the behavior of the system from current sensor data. These predictions can be fed back into the system to refine control decisions. Such a use is called 'hardware-in-the-loop'. The system will be modeled into an initial value problem (IVP) and a solution will be obtained through a numerical integration method.

1.1 Initial Value Problems

Initial value problems (IVPs) arise in many applications in science and engineering, where we want to model the evolution of a system over time starting from an initial value [11, ch.1, 2]. The state x may take values in an open domain $\Omega \subseteq \mathbb{R}^d$ and the process of interest might happen within the open time interval $T \subseteq \mathbb{R}$. We can model this mathematically as an ordinary differential equation (ODE) of the form

$$\frac{\partial}{\partial t}x(t) = f(t,x), \quad x(t_0) = x_0 \in \Omega \tag{1.1}$$

Where $x : T \longrightarrow \Omega$ is the sought solution curve, $f : T \times \Omega \longrightarrow \mathbb{R}^d$ is differentiable, describes the local evolution of the system and stems from the physical properties of the system. Furthermore, $t_0 \in T$ is the time at which we know the initial value $x_0 \in \Omega$ of the system. While in most applications, we are only interested in future values of $x(t)$ for $t \geq t_0$, this model equally allows us to define the evolution x into the past for $t < t_0$.

The natural question arises whether for any initial value problem, there exists a solution curve x such that eq. (1.1) is satisfied and whether this solution is unique.

This question can be answered affirmatively if f satisfies the local Lipschitz condition [11, Ch. 2]:

Definition 1.1 (Local Lipschitz Continuity) *A function $f : T \times \Omega \longrightarrow \mathbb{R}^d$ is called locally Lipschitz continuous in the state variable if for every $(t_0, x_0) \in T \times \Omega$ there exists an open cylinder $U_{(t_0,x_0)} = (t_0 - \epsilon, t_0 + \epsilon) \times B_\delta(x_0)$ such that f is Lipschitz continuous on $U_{(t_0,x_0)}$ in the second argument, i.e. there exists a constant $L > 0$ such that*

$$\|f(t,x_1) - f(t,x_2)\| \leq L \|x_1 - x_2\| \quad \forall (t,x_1),(t,x_2) \in U_{(t_0,x_0)} \tag{1.2}$$

$B_\delta(x_0) := \{x \in \mathbb{R}^d : \|x - x_0\| < \delta\}$ is the open ball of radius δ around x_0.

The following theorem gives the affirmative answer to the question of existence and uniqueness of solutions to the initial value problem (1.1) [11, Ch. 2].

Theorem 1.2 (Existence and Uniqueness of solutions of initial value problems) *Given an IVP of the same form as eq.* (1.1). *Let f be locally Lipschitz continuous in the second argument as in definition 1.1 and continuous in the first argument on the time interval T. Then there exists a unique solution $x : T \longrightarrow \mathbb{R}^d$ such that $x(t_0) = x_0$.*

Deuflhard and Bornemann [11, Ch.2] refer to the work of Walter [27] and Arnold [1] for a proof of this theorem.

We now know that the state x_0 at time t_0 uniquely defines the state $x(t)$ for all $t \in T$. Hence, we can follow Deuflhard and Bornemann [11, Ch.2] and define the evolution of the system. For any starting time $t_1 \in T$ and target time $t_2 \in T$, we obtain the evolution operator ϕ, which fulfills

$$\Phi^{t_2,t_1}x(t_1) = x(t_2) \tag{1.3}$$

The operator describes how the state $x(t_1)$ evolves from time t_1 to time t_2.

Through the reparametrization $t \mapsto t - t_0$, we can without loss of generality use $t_0 = 0$ for the rest of this thesis.

1.2 Time Marching Methods

We will employ time marching methods to obtain numerical approximations to the solution of 1.1. Following the notation in Deuflhard and Bornemann [11, ch.2], these calculate approximations to discrete points $x_n \approx x(t_n)$ on the grid $\Delta := (0, t_1, \dots, t_N)$ of the solution x by the recursion

$$x_{n+1} := \Psi^{t_{n+1},t_n}(x_n) \tag{1.4}$$

Where the discrete evolution $\Psi : \mathbb{R}^d \to \mathbb{R}^d$ is given by the specific time-marching-method used.

While other approaches for obtaining a global solution x like the Picard-Iteration [16] exist, these are infeasible for our application since we want to use the calculated results in real-time in hardware-in-the-loop applications. Thus, we need the approximation to $x(1)$ before we need the approximation to $x(10)$. Additionally, the function f might depend on measurements obtained during the runtime of the system and thus aren't available from the start. Hence, it only makes sense for us, to start from the initial value x_0 and march forward in time with a time marching method. While it is possible and in many cases useful to vary the stepsize h_n adaptively with the simulated system, we will use constant stepsizes denoted with h throughout this thesis. Refraining from using variable stepsizes is explained in section 1.4.

1.2.1 The Euler Method

The simplest time marching method is the Euler method, which was introduced by Leonhard Euler in 1768 [7, Ch.2]. Given an approximation x_n to the solution $x(t_n)$ at time t_n, it calculates the next approximation x_{n+1} at time $t_{n+1} = t_n + h_n$ as follows:

$$x_{n+1} = \Psi^{t_{n+1},t_n}(x_n) := x_n + hf(t_n, x_n). \tag{1.5}$$

Deuflhard and Bornemann [11, ch.4] prove that the approximations x_n converge to the actual values $x(t_n)$ with an error of $\|x_n - x(t_n)\| \in O(h)$ as $h \to 0$.

Stability of the Euler Method

While it is desirable that the error, of a method goes to zero as the stepsize goes to zero, it is equally important to see how errors of previous steps are propagated throughout multiple steps. If errors grow exponentially, the computed results are likely unusable. To illustrate this, we apply the Euler method to Test System 1.

Test System 1 (One Dimensional Linear Test System) *The one-dimensional linear test system is given by the equation*

$$\frac{\partial}{\partial t}x = \lambda x, \quad x(0) = x_0 \in \mathbb{R},$$

where $\lambda \in \mathbb{C}$ is a parameter. The analytic solution is well known to be

$$x(t) = e^{t\lambda}x_0$$

On Test System 1, the discrete evolution of the Euler method simplifies as follows:

$$x_{n+1} = x_n + h \cdot f(x_n) = x_n + h\lambda x_n = (1 + h\lambda)x_n \tag{1.6}$$

As the factor $(1 + h\lambda)$ is independent of n, we now get a closed form expression for x_n. We use the abbreviation $z := h\lambda$:

$$x_n = (1+z)^n x_0 \tag{1.7}$$

The factor $1 + z =: R_{\text{Euler}}(z)$ is known as the stability polynomial of the Euler method [7, Ch.2]. If x_0 was perturbed by an error $e_0 \neq 0$ such that $\hat{x}_0 = x_0 + e_0$, we would get

$$\hat{x}_n = (1+z)^n(x_0 + e_0) = x_n + (1+z)^n e_0. \tag{1.8}$$

We immediately see that if $|1 + z| > 1$, $\hat{x}_n$ deviates from x_n by an unacceptable amount. Hence, the Euler method can only be used reliably if $|1 + z| \leq 1$. We will generalize this finding under the notion of stability in section 1.3.2.

1.2.2 The Implicit Euler Method

The Euler method from eq. (1.5) is an explicit method. That means, its discrete evolution can be evaluated explicitly and does not require solving a system of equations. It is also known as the explicit Euler method. Contrastingly, one can formulate the implicit Euler method as follows.

$$x_{n+1} = x_n + hf(x_{n+1}). \tag{1.9}$$

The key difference is that the updated value x_{n+1} depends on itself. Hence, a system of (possibly nonlinear) equations has to be solved in each step.

Stability of the Implicit Euler Method

We conduct the same stability analysis as for the explicit Euler method above. Consider the Test System 1 using the abbreviation $z := h\lambda$

$$\begin{aligned} x_{n+1} &= x_n + hf(x_{n+1}) \\ &= x_n + h\lambda x_{n+1} \\ &= x_n + z x_{n+1} \end{aligned}$$

For $z \neq 1$, this is equivalent to

$$x_{n+1} = (1-z)^{-1} x_n$$

Thus, the approximation produced by the implicit Euler method is stable if and only if $|1 - z|^{-1} \leq 1$. The solution of Test System 1 remains bounded if and only if $\text{re}(\lambda) \leq 0$. For all such λ and $h > 0$, we get $\text{re}(z) \leq 0$. For all such z, it holds $|1 - z|^{-1} \leq 1$ and thus the implicit Euler method is stable. This property is generally favorable over the explicit Euler method, where only λ from a finite area of the complex plane lead to a stable simulation.

The downside of the implicit Euler method is the greater computational complexity. Calculating the update $x_n \longrightarrow x_{n+1}$ requires solving the implicit equation eq. (1.9). For large systems, this is computationally significantly more expensive than evaluating eq. (1.5). These properties carry over to more sophisticated methods and their explicit and implicit variants. The decision whether to use explicit or implicit methods has to be made in accordance with the application. This decision mainly depends on a property known as (numerical) stiffness.

1.3 Definitions and Terminology

1.3.1 Stiffness

There is no precise definition of the notion of stiffness in the literature. Most authors describe stiffness of a system through…

1. …the modeled physical properties i.e. the presence of stiff materials [29]
2. …the mathematical properties of the system, i.e. the presence of large (in absolute value) eigenvalues in the Jacobian $J_f(t,x)$ [7, sec. 112]
3. …the mathematical properties of the solution i.e. the presence of fast and slow components in the solution [10, sec. 3.4 #3]
4. …the requirements to numerical solvers i.e. the presence of stability issues for explicit methods in contrast to the relative advantage of implicit methods. [14][11, sec. 4.1.3]

While these properties largely overlap and could thus be understood as practically equivalent, we will for the course of this thesis have (2.) and (4.) in mind when we talk about stiffness.

Since we will focus on the linear stability analysis 1.3.2, it will be useful to quantify stiffness of a system in terms of the eigenvalues of the Jacobian $J_f(t,x)$ as in (2.). This will also allow us to quantify relative stiffness (e.g. "ten times more stiff than…")

It is not possible to precisely separate between stiff and non-stiff systems because there is no precise definition of "large absolute value" or "significant advantage". As Deuflhard and Bornemann [11, ch.4] put it, this classification always has to be made "pragmatically" depending on the context of the application. We will be looking for methods, for which this pragmatic evaluation of (4.) will classify the systems of interest as "not too stiff" and the eigenvalues mentioned in (2.) as "not too large".

The pragmatic evaluation especially depends on the focus of the application. If the fastest dynamics in the system need to be resolved accurately, the stepsize h must be chosen small enough regardless of stiffness. If the fast dynamics are not of interest, the stepsize can be chosen larger, as long as stability is guaranteed. This is the case for our application. Due to the real-time nature of our application, it will be necessary to use larger stepsizes h to ensure that the simulation can keep up with real-time. The goal is to maximize the stepsize h while maintaining stability.

1.3.2 Stability

When talking about stability, we need to differentiate between stability of the system itself and stability of the numerical solution produced by a time marching method. We follow the definitions made in [11, Def. 3.19] for an IVP.

Definition 1.3 (Stability of a System) *We call the solution curve starting at x_0 of* (1.1)…

*a …**stable** if $\forall \epsilon > 0 \exists \delta > 0$ such that*

$$\Phi^{t,0}x \in B_\epsilon(\Phi^{t,0}x_0) \quad \textit{for all } t \geq 0, x \in B_\delta(x_0) \tag{1.10}$$

*b …**unstable** if it is not stable*

$B_r(x) := \{y \in \mathbb{R}^n : \|y - x\| < r\}$

For a numerical solution we proceed analogously with the discrete evolution:

Definition 1.4 (Stability of a time marching method) *We call the solution produced by a time marching method to an IVP* (1.1) …

*a …**stable** if $\forall \epsilon > 0 : \exists \delta > 0$ such that*

$$\Psi^{t_n,0}x \in B_\epsilon(\Psi^{t_n,0}x_0) \quad \textit{for all } n \in \mathbb{N}, x \in B_\delta(x_0) \tag{1.11}$$

*b …**unstable** if it is not stable*

A central part of this work will be analyzing for which systems a specific time marching method will produce a stable solution. For general systems this is as hard as computing the method's solution for each IVP and thus infeasible. We will therefore restrict our stability analysis to Test System 1 or its multidimensional counterpart Test System 2.

Test System 2 (Multidimensional Linear Test System) *The multidimensional linear test system is the initial value problem*

$$\frac{\partial}{\partial t}x = Lx, x(0) = x_0 \in \mathbb{R}^d. \tag{1.12}$$

Its analytic solution is well known to be

$$x(t) = e^{tL}x_0, \tag{1.13}$$

where $e^L := \sum_{k=0}^{\infty} \frac{L^k}{k!}$ *denotes the matrix exponential of a matrix* L.

Using the simple linear Test System 1, we can analyze, for which values of λ a time marching method produces a stable solution. This leads us to the notion of the stability domain to characterize a time marching method [7, p.81].

Definition 1.5 (Stability Domain) *The stability domain of a numerical method is the set*

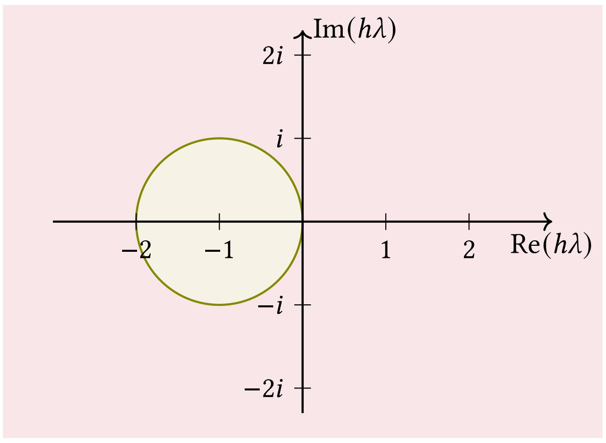
$$S_h = \{\lambda \in \mathbb{C} : the\ numerical\ solution\ of TestSystem\ 1\ with\ \Psi for\lambda\ and\ h\ is\ stable.\}, \tag{1.14}$$

where Ψ *is the discrete evolution and* h *the stepsize of the method.*

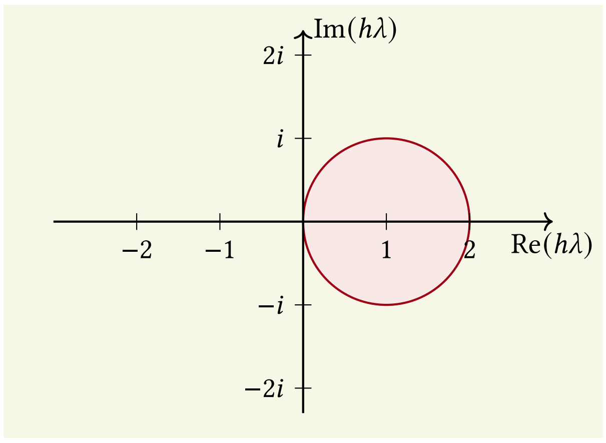
Example 1.6 (Stability Domain of the Euler Method) *The stability domain of the Euler method is given by*

$$S_h = \{\lambda \in \mathbb{C} : |1 + h\lambda| \leq 1\}. \tag{1.15}$$

This domain is visualized for the explicit and implicit Euler methods in Figure 1.1. Comparing this to the stability of the analytic solution $x(t) = e^{t\lambda}x_0$, *which is stable for all* $\lambda in\mathbb{C}$ *where* $re(\lambda) \leq 0$, *we find a serious limitation of the Euler method. The stability region of the Euler method only covers a small part of the region, in which the eigenvalue* λ *might fall for the Test System 1 to be stable.*



(a) Explicit Euler stability region



(b) Implicit Euler stability region

Figure 1.1 Stability regions for explicit and implicit Euler methods.

green Describes stable regions.

explicit Euler $\mathbb{S}_h = \{\lambda \in \mathbb{C} : |1 + h\lambda| \leq 1\}$.

implicit Euler $\mathbb{S}_h = \{\lambda \in \mathbb{C} : |1 - h\lambda| \geq 1\}$.

Similarly to the Euler method, we find that in the linear case of Test System 1, the discrete evolution Ψ of most standard time marching methods simplifies to the multiplication

$$x_{n+1} = R(h\lambda)x_n \tag{1.16}$$

With an adequate $R(h\lambda)$. The function $R : \mathbb{C} \to \mathbb{C}$ is known as the stability function of the time marching method. It allows us to write

$$x_n = R(h\lambda)^n x_0. \tag{1.17}$$

Thus, the stability function R characterizes the stability domain of a method. A time marching method is stable if and only if

$$|R(h\lambda)| \leq 1 \tag{1.18}$$

In the multidimensional case Test System 2, there is a similar result [11, Thm 3.33]. The stability function $R : \mathbb{R}^{d\times d} \to \mathbb{R}^{d\times d}$ is matrix valued in this case.

Theorem 1.7 (Stability of Linear Iterations) *The iteration $x_{n+1} = R(hL)x_n$ is stable for all $x_0 \in \mathbb{R}^d$ if and only if for all eigenvalues λ in the spectrum $\sigma(R(hL))$ one of the following conditions holds:*

1. $|\lambda| < 1$
2. $|\lambda| = 1$ *and λ is a simple eigenvalue.*

An eigenvalue λ is called simple if all Jordan Blocks in the Jordan normal form of $R(hL)$ corresponding to λ are of size 1.

Proof. *See Deuflhard and Bornemann [11, Thm 3.33].* □

Using the stability domain, we can more precisely define the desired stability properties of a time marching method. We follow the definitions made in [10, Ch. 3] and [23].

Definition 1.8 (Linear Stability Properties of a Time Marching Method) *A time marching method is called...*

a ***A-stable*** *if its stability domain contains the entire left half of the complex plane*

$$\{\lambda \in \mathbb{C} : Re(\lambda) \leq 0\} \subseteq S_h$$

b ***F-stable*** *if it is A-stable and its S_h contains no part of the right half of the complex plane.*

$$\{\lambda \in \mathbb{C} : Re(\lambda) \leq 0\} = S_h$$

1.3.3 Order of accuracy

The order of accuracy of a time marching method describes the asymptotic decay of the error of the method as the stepsize h goes to zero. Deuflhard and Bornemann [11, Ch.4] distinguish between the consistency error, which occurs in a single step and the grid error, which describes the total deviation of the numerical solution from the analytic one. We apply those definitions using the constant stepsize h.

Definition 1.9 (Consistency) *Let Φ be the continuous evolution of a system as given in eq.* (1.1) *and Ψ be the discrete evolution used to approximate the system's solution. We call*

$$\epsilon(t,x,h) = \Psi^{t+h,t}x - \Phi^{t+h,t}x \tag{1.19}$$

the consistency error at the point (t,x). We call a method consistent of order p if its consistency error can be bounded by

$$\epsilon(t,x,h) \leq Ch^{p+1} \tag{1.20}$$

for small enough h and a constant C, which is independent of $(t,x) \in T \times \Omega$.

Definition 1.10 (Convergence) *Let $(x_0, \dots, x_N)$ be the approximation to the points $(x(0), \dots, x(t_N))$ of the solution x to an IVP as in eq.* (1.1)*. We call*

$$\epsilon_{t_N}(h) = \max_{n=1\dots N} \|x_n - x(t_n)\| \tag{1.21}$$

the discretization error of the approximation. A time marching method defines an approximation to an IVP for any stepsize h through its discrete evolution. We call a family of such approximations convergent of order $p > 0$ if

$$\epsilon_{t_N}(h) \leq Ch^p \tag{1.22}$$

for small enough h and an adequate constant C.

Deuflhard and Bornemann [11, Theorem 4.10] prove the following implication

Theorem 1.11 (Order of Consistency and Convergence) *If a time marching method is consistent of order p, the approximation it produces, converges with order p.*

Proof. *We refer to Deuflhard and Bornemann [11, Theorem 4.10].* □

1.4 Problem statement

We aim to simulate large systems of ordinary differential equations (ODEs) in real-time for the use in hardware-in-the-loop applications. These ODEs typically describe mechanical systems in robotics and involve stiff contacts like gripping motions. The computation will be done on modern high-performance computers enabling large scale parallel computing and efficient matrix-vector operations.

1.4.1 Constraints posed by the application

This situation poses the following requirements on integration methods:

1. The use in real-time applications, where the simulated machines rely on inputs that depend on the running simulation, poses a strict constraint on the computation time. The calculated simulation results must be processed by a machine controller and fed to the system's actors before the simulated process happens in reality. This means, the simulation time must always be slightly ahead of real-time. Therefore, when running a time marching method with stepsize h, the calculation of one step must never take more than h seconds.

2. Constraint 1 combined with the very large expected system dimension $d \approx 10^6$ requires suitable solvers to scale well with d preferably scaling as $O(d)$ with the system dimension.

3. The use of parallel computing architecture gives methods, which can be split into several compute tasks an advantage in terms of computation time. Compute tasks can only be split between several compute cores and processed in parallel if they do not depend on each other's results. Thus, we prefer methods, which consist of several independent compuational tasks.

4. Due to the real-time nature of our application, the system might be influenced external events such as human controller inputs. These will be incorporated into the system's function f at the time t, at which they occur. This can change the system's dynamics and render function evaluations from previous steps useless in predicting future steps.

5. Because the simulated mechanical systems regularly contain stiff contacts, the resulting ODEs contain numerical stiffness. The aim of this thesis is to find methods, which allow the use of larger stepsizes h without becoming unstable from stiffness.

1.4.2 Dialectic mechanics

Regarding stiffness, a useful technique has been developed by Zimmer and Oldemeyer [29, 30]. The modelling technique called dialectic mechanics (DM) is a reliable way to manipulate the Eigenvalues of a system. Figure 1.2 illustrates the effect of this manipulation. The eigenvalues of the original system are marked in green and the ones of the manipulated system in red.

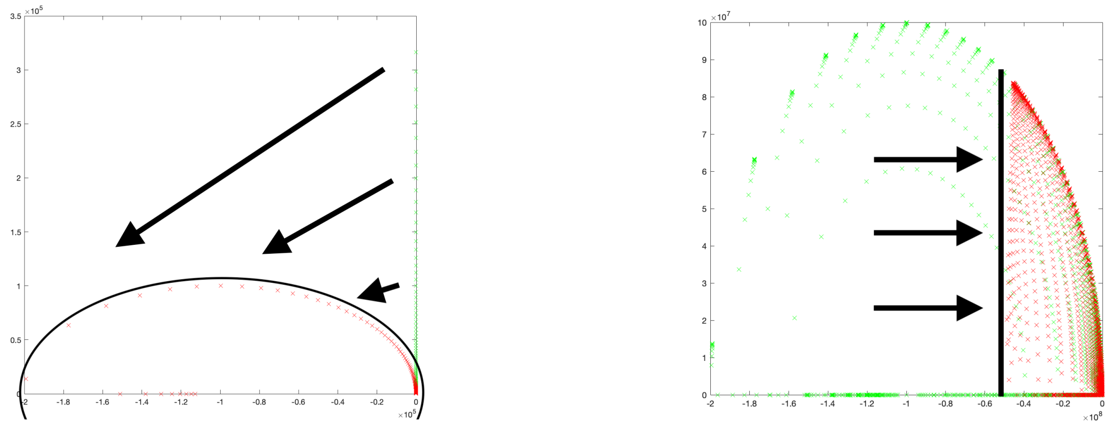


Figure 1.2 Manipulation of eigenvalues through dialectic mechanics

The parameter T_D forces the eigenvalues to be within a circle in the left half of the complex plane that touches the imaginary axis in the origin. The parameter d_{el} enforces a lower bound on the real part of the eigenvalues. Going into the details of this modelling technique is beyond the scope of this thesis. We refer to the original papers [29, 30].

The manipulation mainly concerns eigenvalues with larger absolute value and leaves those corresponding to slower dynamics relatively unaltered. Through the use of DM, we gain a bound on the region, in which our eigenvalues might fall, while mostly preserving the slower (and in our case more relevant) dynamics of the system. Regarding a sequence λ_i of eigenvalues, whose stiffness diverges to infinity as i goes to ∞, the sequence of manipulated eigenvalues $\hat{\lambda}_i$ will converge towards an attractor $\hat{\lambda} \in \mathbb{C}$. Hence, the manipulated eigenvalues of previously stiff systems will be closely neighboring each other. The position of the attractor $\hat{\lambda}$ depends on the parameters T_D and d_{el}.

The manipulated systems are in many cases more suitable for the integration with explicit methods. Hence, many of the methods studied in this thesis allow larger stepsizes while remaining stable when applied to the through DM manipulated system instead of the original.

1.4.3 Suitable candidates

We seek to find methods with improved stability characteristics. We are especially interested in accurately simulating slow dynamics of a system governed by non-stiff eigenvalues. We view fast dynamics caused by stiff eigenvalues as an obstacle rather than a phenomenon of interest. Constraint 1 forces us to use stepsizes, which are too large to resolve fast dynamics anyway, so it is not a problem if a method completely damps the fast, ‘uninteresting’ dynamics and only resolves the slow ones accurately. Most fast dynamics such as vibrations in stiff contacts are of very small amplitude, such that they do not play a significant role in the overall error of the system. The only requirement towards the fast dynamics caused by stiff eigenvalues, which we pose, is for them to remain bounded in amplitude.

Regarding the position of the system’s eigenvalues in the complex plane, it would of course be ideal to achieve F-stabiliy (see definition 1.8). This property will not be achieved with explicit methods realistically. It is however, vital to accurately distinguish between stable and unstable dynamics within the slow frequencies. That is, for an eigenvalue λ with small absolute value, the corresponding dynamic shall be

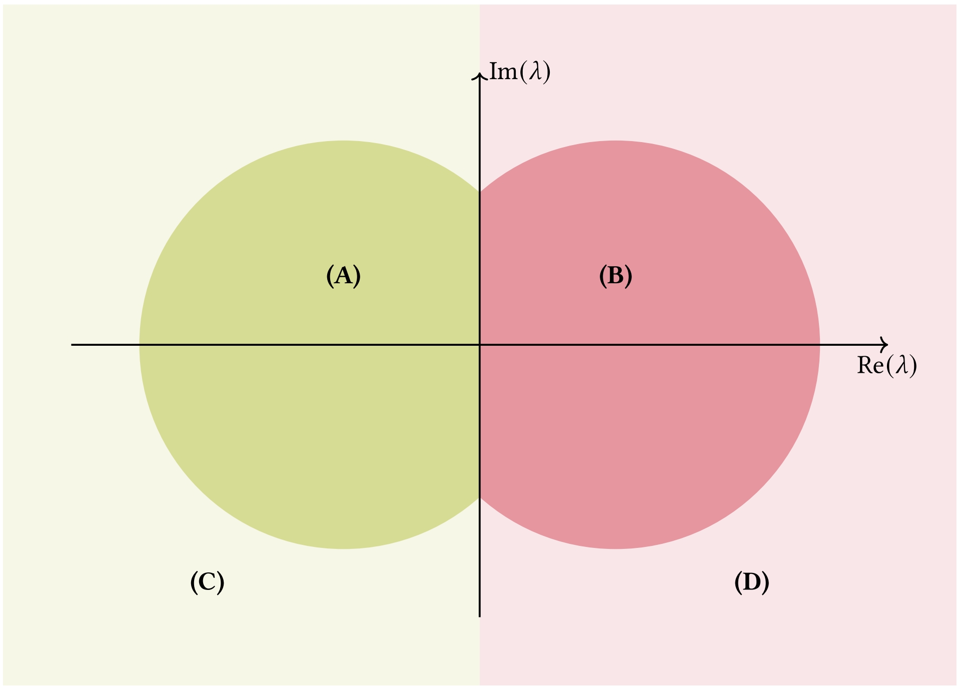


Figure 1.3 Desired stability domain of a time marching method.

simulated in a stable manner if and only if the real part $\mathrm{re}(\lambda) \leq 0$, which is equivalent to stability of the analytical solution.

To achieve this, it is necessary that for eigenvalues with small absolute value, a method correctly splits the left and right half of the complex plane in stable and unstable region. Ideally, the stability follows the description of Figure 1.3 where the stability of the method depends on the region, in which the eigenvalue lies.

(A) Must necessarily be stable. These dynamics regularly occur in the systems, which we want to simulate and must not destabilize the integration method.

(B) Must necessarily be unstable. These dynamics may occur in the tested systems and are unstable in reality. An integration method must reliably detect this instability.

(C) Is preferred to be stable. These dynamics can be avoided through adequate modelling of the system. It is preferred that solvers integrate them in a stable manner, but not necessary to the successful application of a method.

(D) Is preferred to be unstable. These dynamics are unlikely to occur in the modelled system and thus, are of secondary concern.

There is no scale to Figure 1.3 since it describes the desired stability domain of methods qualitatively. We aim to find methods, whose stability domain matches Figure 1.3 on as large as possible of a scale.

Due to constraint 2, we cannot use implicit methods. These would require solving a (possibly nonlinear) system of equations at each time step. These solves typically require at least $O(d^3)$ operations, which is unfeasible for systems dimensions in the order of $d = 10^6$. Thus, we will focus on explicit methods. We will however allow solving small linear systems of equations when using linear implicit methods in chapter 5. The dimension of these systems will be $K < 10$ making these solves computationally feasible.

To satisfy constraint 3, we seek to avoid iterative methods. This is another reason to not use implicit methods, since these typically need iterative algorithms to solve nonlinear systems of equations. It also makes Runge-Kutta methods less attractive since these require to sequentially evaluate several substages. We refer to parallelizable multistage peer methods in section 2.5 as an alternative approach to multistage methods, which allow parallelization. This constraint is not strict and can be violated if the added computational cost of sequential calculations is justified through an increase in allowed stepsize.

Because of constraint 4, function evaluations at previous points are not reliable information in predicting future points. Hence, we cannot use multistep methods such as the ones presented in section 2.3.

Since we expect stiffness to be the relevant bottleneck on step size, accuracy respectively order of accuracy of the evaluated methods will be of secondary concern.

We will for the course of this thesis restrict ourselves to the analysis of stability only in the linear sense. That is, prove stability for the linear Test System 2 of the form $\frac{\partial}{\partial t}x = Lx$. To assess nonlinear stability, we will perform numerical tests with common examples of nonlinear systems.

1.4.4 Constant Stepsize

We have seen in section 1.2.1 that decreasing the stepsize h also decreases the truncation error made at each evolution step. Noting from the evolution (1.4) that $t_N = \sum_{n=0}^{N-1} h_n$, we see that we must evaluate Ψ a number of $N-1$ times and N is inversely correlated to the average stepsize h. If we choose an equidistant grid $\Delta = (0, \frac{1}{N}, \frac{2}{N}, \dots, \frac{N-1}{N}, 1)$ we get $N = 1/h$.

It is natural to inspect whether it is beneficial to vary the stepsize h_n depending on the local truncation errors to save computational resources where the truncation error is low anyway and improve accuracy where smaller h_n significantly reduce errors [11, Ch.5].

For our application however, we cannot shift computational load to those times when it has the greatest impact on the produced error because the simulation speed must never be slower than real-time. Otherwise, the results would have no use to real-time applications.

On the other hand, we also do not have much use for simulation speeds much greater than real-time because we cannot 'build up a lead' by simulating ahead of real-time due to possible current-time sensor data necessary to know the physical properties of the system. The compute hardware would need to idle until the latest sensor data becomes available.

Thus, we will always choose time steps h slightly larger than the computation time it takes to evaluate one step of a time marching method. Therefore, there is no need for variable stepsizes. We will write h for the constant stepsize.

1.5 Notation

Notation 1.12 (variables, parameters and derivatives) *We write …*

1. *h for the stepsize*
2. *S for the number of stages in a method*
3. *s to iterate over the stages*
4. *K for the number of time-derivatives used in a method*
5. *k to iterate over the time-derivatives*
6. *N for the number of time steps in a simulation*
7. *n to iterate over the time steps*
8. *L for the linear test system matrix in (2)*
9. *$z = h\lambda$ to abbreviate the stability polynomial argument*
10. *$R(z)$ for the evaluation of the stability polynomial of a method*
11. *$\rho(M) := \max\{|\lambda| : \lambda \text{ is an eigenvalue of } M\}$ for the spectral radius of a matrix M*

12. $\mathbb{S}_h := \{\lambda \in \mathbb{C} : \rho(R(h\lambda)) \leq 1\}$ *for the stability domain of a method with stepsize* h
13. a_{ij}, b_i, c_i *for the coefficients of a Runge-Kutta method*
14. $x^{(k)}(t) = \frac{\partial^k}{\partial t^k} x(t)$ *to denote the* k*-th time derivative of the solution curve* $x(t)$.
15. $x^{(0)}(t) = x(t)$ *to conveniently denote the solution's value itself within iterations.*
16. $x^{(k)} = x^{(k)}(t)$ *and* $x = x(t)$ *where the argument is clear from the context.*
17. $f^{(k)}(t, x) = x^{(k)}(t)$ *to denote the evaluation of the* k*-th time derivative of the solution curve.*
18. $f^{(k)}$ *where the arguments are clear from the context*
19. J_f *or* $J_f(t, x)$ *to denote the Jacobian of the function* f

2 General Linear Methods

There exists a plethora of methods designed to solve different types of ODEs.

This chapter reviews the class of general linear methods (GLMs) including a detailed focus on Runge-Kutta methods, which are a subclass of GLMs. These methods can be viewed as more sophisticated versions of the Euler methods, are amongst the most popular explicit integration methods and represent the benchmark, against which we will compare advanced methods in the following chapters. We will analyze the linear stability of these methods using their respective stability polynomials.

2.1 Enhancing the Euler Method

At each time step, the Euler method only uses the information of one function evaluation $f(t_n, x_n)$ to calculate the next step. Significantly more accurate results can be achieved by using more information in the following ways:

1. **Multiple Stages:** Evaluate f at multiple points during one step.

2. **Multiple Previous Steps:** Use the information of the current step and multiple previous steps.

3. **Multiple Derivatives:**

   a) Calculate not only values for $x^{(1)}(t_n) = f(t_n, x_n) = f^{(1)}(t_n, x_n)$,
   but also for $x^{(2)}(t_n) = f^{(2)}(t_n, x_n),\ x^{(3)}(t_n) = f^{(3)}(t_n, x_n)$ and so on.

   b) Use derivates of the function f for example it's Jacobian $J_f(t_n, x_n) = \frac{\partial f}{\partial x}(t_n, x_n)$.

These three approaches enable the construction of generalizations to the Euler method and have been structured in the following diagramm by Butcher in [5, Fig.1]. We can hereby organize the most common methods in a three dimensional diagramm with the Euler method at the origin. The multiderivative axis is split into two parts representing the approaches a) and b).

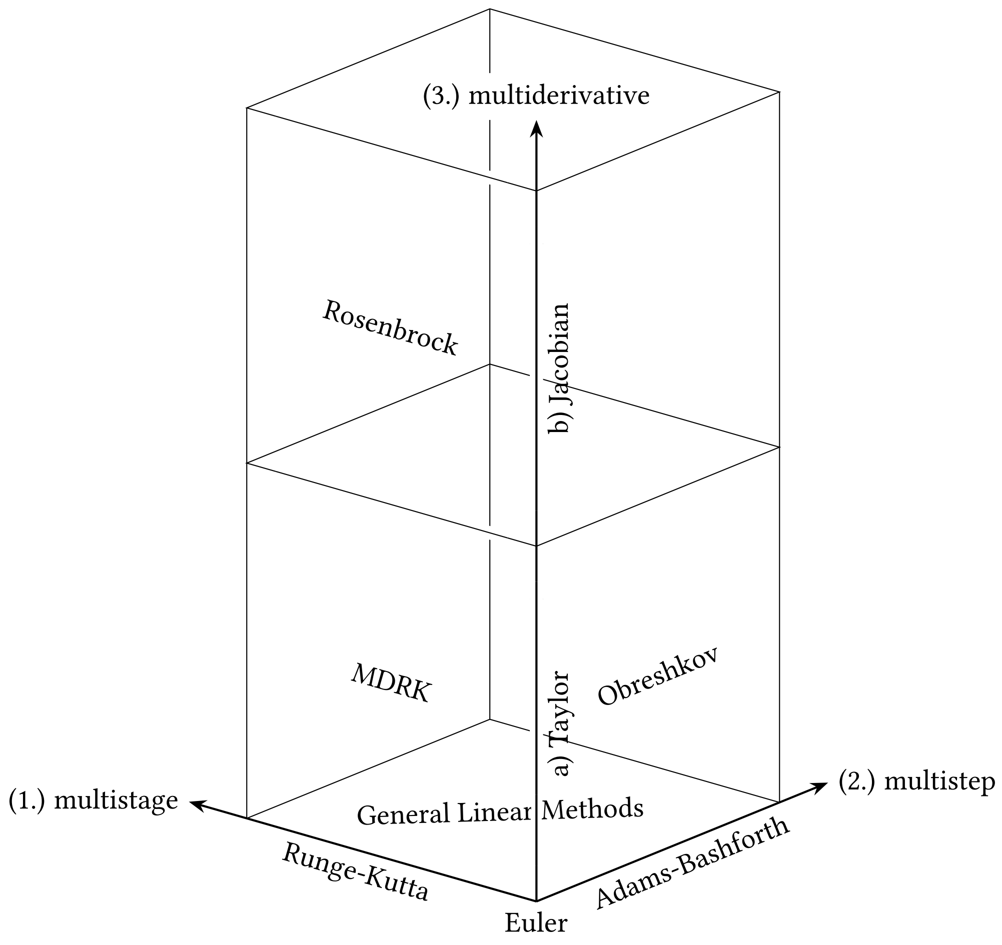


Figure 2.1 general linear methods (GLM) as a 3D Cube

The term general linear method (GLM) was first used around 1965 by (amongst others) Butcher [5]. It describes the class of ODE integrators found on the bottom plane of Figure 2.1. These methods make use of multiple previous steps and multiple stages. We will first consider each of these directions on their own and then review the combined class of GLMs. Methods along the multiderivative axis (3.) will be explored in the following chapters.

2.2 Runge-Kutta Methods

When focusing on the multistage-axis in 2.1, we find the well known Runge-Kutta methods, which were introduced by C. Runge and M.W. Kutta around 1900 [4, 25]. I have already investigated these methods and their stability properties in my internship at the DLR from April to June 2024. This section is a brief summary of the findings from that internship. Runge-Kutta methods are usually noted as a Butcher tableau [6], which is a matrix representation of the method:

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \end{array} = \begin{array}{c|c} c & A \\ \hline & b^T \end{array} \tag{2.1}$$

The method calculates the intermediate stages $\mathbf{k} := (k_1, k_2, \dots, k_S)^T \in \mathbb{R}^{d \cdot S}$, where each k_s is given by:

$$k_s = f\left(t_n + c_s h, y_n + h \sum_{j=1}^{S} a_{sj} k_j\right), \quad s = 1, 2, \dots, S. \tag{2.2}$$

The final solution is then updated as:

$$y_{n+1} = y_n + h \sum_{s=1}^{S} b_s k_s. \tag{2.3}$$

Using the Kronecker product $\otimes$, and the pointwise evaluation $\mathbf{f}$ of the function f, we can simplify these expressions.

Notation 2.1 (Kronecker Product) *$\otimes$ denotes the Kronecker product where for matrices $M \in \mathbb{R}^{n\times m}$ and $N \in \mathbb{R}^{p\times q}$ the Kronecker product is defined as the $(np)\times(mq)$ block matrix*

$$M \otimes N = \begin{pmatrix} m_{11}N & m_{12}N & \cdots & m_{1m}N \\ m_{21}N & m_{22}N & \cdots & m_{2m}N \\ \vdots & \vdots & \ddots & \vdots \\ m_{n1}N & m_{n2}N & \cdots & m_{nm}N \end{pmatrix} \tag{2.4}$$

Notation 2.2 (Entry Wise Function Evaluation) *Let $\mathbf{x} = (x_1, \dots, x_s)^T \in R^{d\cdot s}$ be a vector of (possibly vector valued) entries. We write $\mathbf{f}(\mathbf{x}) := (f(x_1), \dots f(x_s))^T$ to denote the entry wise evaluation of f with the vector of arguments $\mathbf{x}$. Analogously, we write $\mathbf{f}(t,\mathbf{x}) := (f(t,x_1), \dots f(t,x_s))^T$ for non-autonomous problems.*

In this notation, we can simplify a step of a Runge-Kutta method as

$$\mathbf{k} = \mathbf{f}(t_n + ch, y_n + hA\mathbf{k}) \tag{2.5}$$

$$y_{n+1} = y_n + (b \otimes I_d)^T \mathbf{k} \tag{2.6}$$

Where $A = (a_{ij})$, $b = (b_i)$ and $c = (c_i)$ are the coefficients of the method. It can be computationally advantageous to use special structures for the matrix A [7, Ch.3]:

- **General:** No restrictions on A. (2.5) has to be solved simultaneously (solve one system of dimension $n \cdot s$). The method is called “Implicit Runge-Kutta” (IRK) method.
- **Lower Triangular:** $a_{ij} = 0$ for $i < j$. The k_i can be implicitly calculated by solving (2.2) (solve s systems of dimension n). The method is called “Diagonally implicit Runge-Kutta” (DIRK) method.
- **Lower Triangular with same diagonal element**: $a_{ii} = \lambda$ (solve s systems of dimension n, but the Jacobian and it’s LU decomposition can be reused). The method is called “Singly diagonally implicit Runge-Kutta” (SDIRK) method.
- **Strictly Lower Triangular:** $a_{ij} = 0$ for $i \le j$. The k_{i+1} can be explicitly calculated from $k_1, \dots, k_i (i = 1 \dots s-1)$. Thus the method is called “Explicit Runge-Kutta” (ERK) method.

Since we are aiming at applications with very large system dimension n, we want to avoid solving systems of this dimension. Thus, we will focus on ERK methods.

Stability of Runge-Kutta Methods

Just like the factor $R_{\text{Euler}}(z) = 1 + z$ (see section 1.2.1), any explicit Runge-Kutta method has a stability polynomial, which characterizes whether errors will decay or blow up over many iterations.

The stability Polynomial can be computed from the coefficients in the Butcher tableau through the use of trees [7, ch.3]. We calculate the stability polynomial of a method with $S = 3$ stages as an example:

$$\begin{aligned} k_1 &= Lx_n \\ k_2 &= L(x_n + a_{21}hk_1) = L(x_n + a_{21}hLx_n) \\ k_3 &= L(x_n + a_{31}hk_1 + a_{32}hk_2) = L(x_n + a_{31}hLx_n + a_{32}hLh(x_n + a_{21}hLx_n)) \\ x_{n+1} &= x_n + h(b_1k_1 + b_2k_2 + b_3k_3) = x_n + b_1hLx_n + b_2hL(x_n + a_{21}hLx_n) + b_3hL(x_n + a_{31}hLx_n + a_{32}hL(x_n + a_{21}hLx_n)) \\ &= (I_n + (b_1 + b_2 + b_3)hL + (b_2a_{21} + b_3(a_{31} + a_{32}))(hL)^2 + b_3a_{32}a_{21}(hL)^3)x_n \end{aligned} \tag{2.7}$$

Method	Stability Function
Euler	$R(z) = 1 + z$
Heun	$R(z) = 1 + \frac{z}{2} + \frac{z^2}{2}$
RK4	$R(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}$

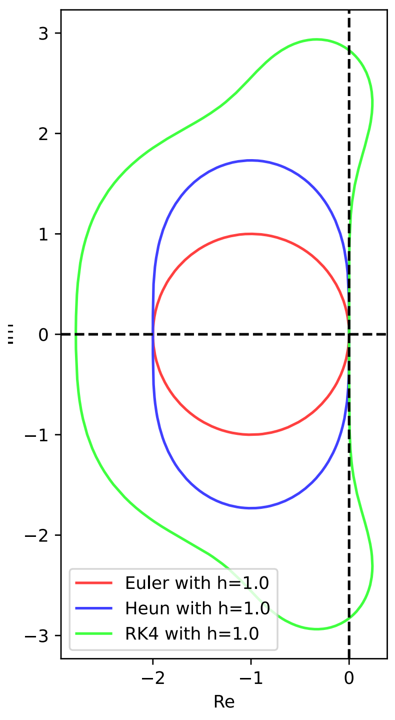


Figure 2.2 Stability functions and stability regions of common time integration methods.

Hence, the general $S = 3$ stability polynomial is given by using the abbreviation $z = hL$.

$$R(z) = 1 + (b_1 + b_2 + b_3)z + (b_2 a_{21} + b_3(a_{31} + a_{32}))z^2 + b_3 a_{32} a_{21} z^3 \tag{2.8}$$

The stability polynomials of standard explicit Runge-Kutta methods equal the Taylor polynomials of the respective degrees. These are given in Figure 2.2.

If we compute n steps of the Runge-Kutta method, we get $x_n = R(hL)^n x_0$, so the method yields a stable solution if $\lim_{k\to\infty} \left\| R(hL)^k x_0 \right\| < \infty$. This is the case if the spectral radius of $R(hL)$ is less than 1, i.e. $\rho(R(hL)) < 1$. If $\rho(R(hL)) > 1$, but x_0 is orthogonal to all eigenvactors with eigenvalues greater than 1, the method is theoretically still stable. In practice however, numerical errors will resolve the orthogonality and the method will become unstable. For the edge case $\rho(R(hL)) = 1$ the method is stable if and only if the Jordan form of L has no Jordan blocks of size greater than 1. See Theorem 8.4 for details.

The spectral mapping theorem yields that this is equeivalent to having

$$|R(h\lambda)| < 1 \text{ for all Eigenvalues } \lambda \text{ of } A \tag{2.9}$$

So the Stability domain of a Runge-Kutta method is the set

$$S = \{\lambda \in \mathbb{C} : |R(h\lambda)| < 1\}$$

This leads us to the following approach to find Runge-Kutta methods with desirable stability properties:

Designing suitable stability polynomials

Instead of immediately searching for the coefficients of a Runge-Kutta method, we can first search for the coefficients of a stability polynomial that satisfies (2.9) for h as large as possible. We use a clever method described by Ketcheson and Ahmadia [18] to find the optimal stability polynomial for a given number of stages s. Given a number of stages s, and set of Eigenvalues $\Lambda = \{\lambda_1, \dots, \lambda_n\}$ of L, we want to solve the following optimization problem:

Problem 1 $\max_{\alpha_i \in \mathbb{R}(i=1,\dots,s), h \in \mathbb{R}_{\geq 0}}$ h
subject to $|R(h\lambda)| - 1 < 0$ *for all* $\lambda \in \Lambda$

Since this is not a convex optimization problem, solving it is not feasible in general. However, by fixing the step size h, we obtain the following problem:

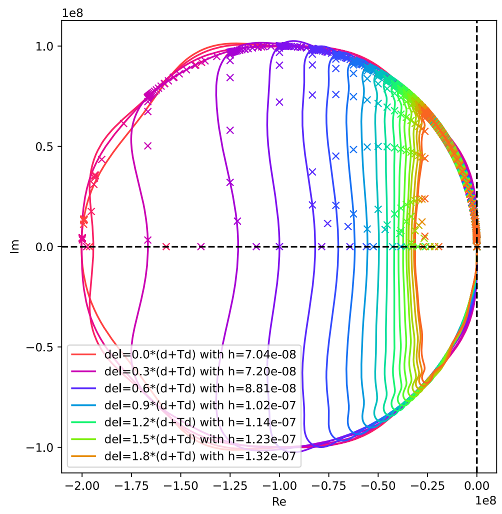
Problem 2 $\min_{\alpha_i \in \mathbb{R}(i=1,\dots,s)}$ $\max_{\lambda \in \Lambda} |R(h\lambda)| - 1$

By $r(h, \Lambda)$ we denote the minimal value achieved in this problem. Now we can find the optimal stability polynomial by solving

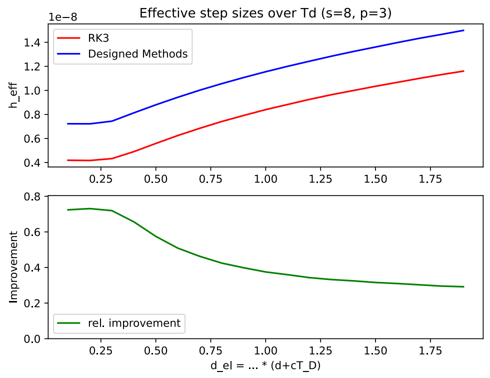
Problem 3 $\max_{h \in \mathbb{R}_{\geq 0}}$ h
subject to $r(h, \Lambda) < 0$

This optimization problem has been implemented in the Matlab library `RKOpt` by Ketcheson et al. [19].

With this approach, we can design Runge-Kutta methods with a stability domain that contains a specified set of Eigenvalues Λ. The modelling approach of dialectic mechanics presented in section 1.4.2 can be used to guarantee all eigenvalues of a given system of ODEs to fall into a confined region within the left half of the complex plane. In my internship, I tested the approach of Ketcheson and Ahmadia [18] to design Runge-Kutta methods with a stability domain that contains the set of eigenvalues that can arise in systems manipulated with the dialectic mechanics technique. The stability domains for these methods are designed to fit the Eigenvalues obtained with h as large as possible for different values of the dialectic mechanics parameters T_d and d_{el}. They are shown in Figure 2.3. The crosses represent the eigenvalues that are present in the manipulated system. Figure 2.3b compares the effective stepsizes (i.e. the stepsize normalized by $\frac{3}{8}$ to account for the larger number of function evaluations and the incurring computational cost) of third order optimized Runge-Kutta methods to the possible stepsizes of the well known Runge-Kutta-3 method. The conclusion of the analysis conducted in my internship was that optimized Runge-Kutta methods offer a stepsize advantage of about 20-70% compared to standard Runge-Kutta methods depending on the shape of the desired stability domain. This advantage is in most applications not worth the cost of specifically designing a method for a system.



(a) Stability domains of dialectic mechanics optimized Runge-Kutta methods



(b) Comparison of effective stepsizes (i.e. $h_{eff} = \frac{3}{8}h$) of optimized RK methods vs. RK3

Figure 2.3 Optimized third order Runge-Kutta methods

2.3 Linear Multistep Methods

Along the multistep-axis in 2.1 we find the class of linear multistep methods, such as the Adams-Bashforth methods [7, Ch.2]. These methods use previous function evaluations to calculate the next step. Adams-Bashforth methods are of the general form:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i f(t_n - ih, y_{n-i}). \tag{2.10}$$

While these "are amongst the most commonly used classes of linear multistep methods" Butcher [7, Ch.2], one can further generalize them to also depend on the previous $y_{n-i}, i = 1, \dots, s$ as follows [7, Ch.2]:

$$y_{n+1} = \sum_{i=1}^{s} a_i y_{n-i} + h \sum_{i=1}^{s} b_i f(t_n - ih, y_{n-i}). \tag{2.11}$$

If one replaces the starting index $i = 1$ with $i = 0$ in (2.10) or (2.11) to include the current step y_n, one receives the implicit Adams-Moulton methods or implicit multistep methods respectively. Note that since multistep methods depend on $y_{n-1}, y_{n-2}, \dots, y_{n-k}$, they can only be used after the first k steps have been calculated with an adequate starting procedure. We want the first k approximations to be of the same order of accuracy as the method itself.

Stability of Linear Multistep Methods

The stability of linear multistep methods is a special case of the stability of general linear methods (see section 2.4).

2.4 General Linear Methods

In this section, we present a generalization introduced by Butcher [8] to unify Runge-Kutta and linear multistep methods into a single framework. The aim of this section is to provide an overview of an important class of time marching methods and their stability characteristics. From an application standpoint however, GLMs are not the focus of this thesis due to their use of multiple previous steps. As outlined in section 1.4.3, information from previous steps cannot be used to predict future steps when encountering external events, which might alter the system's dynamics. Thus, we will focus on theoretically understanding GLMs rather than working towards their application. For convenience, we will restrict our studies in this section to the autonomous case $x^{(1)}(t) = f(x(t))$.

A general linear method is defined by its extended $(S + R) \times (S + R)$ Butcher tableau [5][7, Ch.5]

$$\begin{bmatrix} A & U \\ B & V \end{bmatrix} \tag{2.12}$$

Where $A = (a_{ij}) \in \mathbb{R}^{S \times S}, U = (u_{ij}) \in \mathbb{R}^{S \times R}, B = (b_{ij}) \in \mathbb{R}^{R \times S}, V = (v_{ij}) \in \mathbb{R}^{R \times R}$ are matrices of coefficients.

Each step uses the information of R previous function evaluations $\mathbf{y_{n-1}} = (y_{(n-1,1)}, \dots, y_{(n-1,R)})^T \in \mathbb{R}^{R \cdot d}$ and computes S stages $\mathbf{x} = (x_1, \dots, x_S)^T \in \mathbb{R}^{S \cdot d}$ as well as R output values $\mathbf{y_n} = (y_{(n,1)}, \dots, y_{(n,R)})^T \in \mathbb{R}^{S \cdot d}$, which are transferred to the next step.

The stages and outputs are calculated as follows:

$$\begin{aligned} \mathbf{x}_i &= \sum_{j=1}^{s} a_{ij} h f(x_j) + \sum_{j=1}^{r} u_{ij} y_{(n-1,j)}, \quad i = 1, 2, \dots, s, \\ y_{(n,i)} &= \sum_{j=1}^{s} b_{ij} h f(x_j) + \sum_{j=1}^{r} v_{ij} y_{(n-1,j)}, \quad i = 1, 2, \dots, r. \end{aligned} \tag{2.13}$$

Or in compact form using the Kronecker product $\otimes$ (see notation 2.1) and the entry wise evaluation $\mathbf{f}$ of the function f (see notation 2.2).

$$\begin{pmatrix} \mathbf{x} \\ \mathbf{y_n} \end{pmatrix} = \begin{pmatrix} A \otimes I_d & U \otimes I_d \\ B \otimes I_d & V \otimes I_d \end{pmatrix} \begin{pmatrix} h\mathbf{f}(t_n, \mathbf{x}) \\ \mathbf{y_{n-1}} \end{pmatrix} \tag{2.14}$$

Note that when trying to interpret the output values $\mathbf{y^{(n)}}$, they can represent previous points, previous function evaluations or any linear combination of the two.

Stability of General Linear Methods

To analyze the stability of general linear methods, we again turn to the linear Test System 1 and search for the stability function $R(z)$, which describes the propagation of values over many iterations. As the GLM propagates the values of $\mathbf{y^{(n)}} \in \mathbb{R}^{R \cdot d}$, $R(z)$ will be an $R \times R$ matrix.

We insert the test equation into the GLM to obtain $R(z)$:

$$\begin{pmatrix} x \\ \mathbf{y}_n \end{pmatrix} = \begin{pmatrix} A & U \\ B & V \end{pmatrix} \begin{pmatrix} hF(X) \\ \mathbf{y}_{n-1} \end{pmatrix} = \begin{pmatrix} A & U \\ B & V \end{pmatrix} \begin{pmatrix} h\lambda x \\ \mathbf{y}_{n-1} \end{pmatrix} \tag{2.15}$$

$$\implies x = zAx + U\mathbf{y}_{n-1} \tag{2.16}$$

$$\implies (I - zA)x = U\mathbf{y}_{n-1} \tag{2.17}$$

$$\implies x = (I - zA)^{-1}U\mathbf{y}_{n-1} \tag{2.18}$$

$$\implies \mathbf{y}_n = B(I - zA)^{-1}U\mathbf{y}_{n-1} + V\mathbf{y}_{n-1} \tag{2.19}$$

Hence, we get

$$R(z) = B(I - zA)^{-1}U + V \tag{2.20}$$

We now see that a GLM is stable if and only if all eigenvalues of the matrix $R(z)$ lie within the unit circle, i.e. $\rho(R(z)) \leq 1$. While we can now determine the stability properties of a GLM given it's extended Butcher tableau, the search for suitable GLMs is still cumbersome. Finding GLMs with extended stability domains lies beyond the scope of this thesis.

2.5 Parallelizable Multistage Peer Methods

As seen with Runge-Kutta methods, evaluating the function f several times during one step yields accuracy and stability benefits. Leveraging modern parallel compute hardware, it is advantageous to be able to compute these function evaluations independently of one another. For Runge-Kutta methods, this is not possible because the points at which the evaluations of later stages take place depend on earlier stage evaluations. Parallelizable Multistage Methods as described by Pagano [23] offer the ability to evaluate several stages independently and thus leverage parallelization. The key is - as suggested by the name **Peer** Method - to use $s \in \mathbb{N}$ equally accurate stages and reuse all the stages of the previous step to calculate the new stages. Using the stages $Y_{n,i}$ for $i = 1, \ldots, s$ of the n-th step, we get the propagation

$$Y_{n+1,i} = \sum_{j=1}^{s}, b_{ij} Y_{nj} + h \sum_{j=1}^{s} a_{ij} f(t_{n,j}, Y_{n,j}), \quad i = 1, \ldots, s \tag{2.21}$$

The coefficients b_{ij}, a_{ij} ideally satisfy the applicable order conditions, which can be obtained from the solution of a nonlinear system of equations. Details are given by Pagano [23, section 3.1]. If all the previous stages $Y_{n,i}$ were of p-th order accuracy, all the new stages can be calculated to p-th order as well allowing for a p-th order scheme, for which all the function evaluations can be calculated in parallel. Such a method would then select one of the stage values as the n-th step approximation at time t_n. This could for example be $Y_{n,s}$. Note that the method relies on more than one initial value and thus needs to be paired with an adequate starting procedure to calculate the first stage's values $Y_{1,i} \quad i = 1, \dots, s$, which guarantees the $Y_{1,i}$ to be of p-th order of accuracy.

While the previous stages aren't values of whole time steps t_n, these methods can still be expressed in the sense of general linear methods as in section 2.4 by comparing the equations (2.13) and (2.21). Hence, their stability analysis suffers from the same complexity through the curse of dimension.

To achieve better stability properties, than simply using explicit Multistage Peer Methods, Pagano [23] propose to combine them with the linear implicit TASE operator introduced by Bassenne, Fu, and Mani [2] thus combining the improved stability of TASE methods with the parallel and thus, possibly faster evaluation of Parallel Multistage Peer Methods.

3 Multiderivative Methods

Recalling the constraints posed by the problem outlined in 1.4, we search for time integration methods to solve eq. (1.1), which …

1. emit predictable computation time.
2. scale as $O(n)$ with the system dimension, especially avoid computing the Jacobian J_f.
3. preferably can make use of modern parallel computing architecture.
4. don't depend on information before the last time step to be compatible with external inputs.
5. work well with moderately stiff systems. (i.e. have a larger stability domain than popular solvers such as RK4)

Requirement 4 rules out multistep methods such as the ones presented in section 2.3. Hence, we will not study multistep methods further in this thesis. To fulfill 3, we want to avoid calculating many sequential stages per step. When using multiple function evaluations per step, we prefer methods where the stages don't depend on each other and can thus be evaluated in parallel such as the approach presented in section 2.5. Recalling the overview of possible enhancements of the Euler method in fig. 2.1, leveraging multiple derivatives seems to be a promising direction in the search for methods with better stability properties.

From now on, we assume that the system's function $f : T \times \Omega \longrightarrow \mathbb{R}^d$ is K continuously differentiable. In this case, the solution $x : T \longrightarrow \mathbb{R}^d$ is $K+1$ times continuously differentiable. In section 3.1, we study the Taylor method as an example of a multiderivative integration scheme. In section 3.2, we investigate how to efficiently calculate the time-derivatives $\frac{\partial^k x}{\partial t^k}(t)$ of the solution $x(t)$, which we will denote as $x^{(k)}(t)$ or short $x^{(k)}$ where the argument is clear from the context. We will also write $f^{(k)}(t,x) = x^{(k)}(t)$ to evaluate the time-derivatives or short $f^{(k)}$ where the arguments are clear from context. We will use the convention $x^{(0)} = f^{(0)} = x$ whereas the zeroth derivative of a function denotes the function value itself.

3.1 The Taylor Method

The simplest use of higher derivatives $x^{(k)}(t)$ leads us to the method known as the Taylor method [7, Ch.2]. The idea is to use the truncated Taylor series of

$$x(t+h) = x_n + \sum_{k=1}^{K} \frac{x^{(k)}(t_n)}{k!} h^k + \tilde{R}(t, t+h), \tag{3.1}$$

where $\tilde{R}(t, t+h)$ denotes the small remainder term, which asymptotically goes to zero with $\tilde{R}(t, t+h) \in O(h^{K+1})$ as h goes to zero. Using the standard choice of coefficients $\alpha_k = \frac{1}{k!}$ for $k = 1 \ldots K$, a step of the Taylor method goes

$$x_{n+1} = x_n + \sum_{k=1}^{K} \alpha_k x^{(k)}(t_n) h^k. \tag{3.2}$$

We note that since the remainder $\tilde{R}(t, t+h) \in O(h^{K+1})$, the Taylor method using the standard choice of coefficients is consistent of order K and hence, by Theorem 1.11 produces an approximation, which is convergent of order K. While the choice $\alpha_k = \frac{1}{k!}$ for $k = 1 \ldots K$ is common, it can make sense to deviate from it for $k \geq p$. This limits the order of convergence of the produced approximations to p, but creates some degrees of freedom to select the coefficients α_k to optimize stability.

3.1.1 Stability of Taylor Methods

We first notice, that for the linear test equation $x^{(1)} = Lx$ from Test System 2, one step of the Taylor method (3.2) can be written as

$$x_{n+1} = x_n + h\sum_{k=1}^{K} \alpha_k (hL)^k x_n = (I + h\sum_{k=1}^{K} \alpha_k (hL)^k) x_n. \tag{3.3}$$

From eq. (3.3) we can see that using the abbreviation $z = hL$, the stability polynomial of the Taylor method is given by

$$R(z) = I + \sum_{k=1}^{K} \alpha_k z^k. \tag{3.4}$$

In choosing coefficients $\alpha_k \neq \frac{1}{k!}$ for $k > p$, the additional degrees of freedom can be used to optimize the stability domain of the Taylor method. To find suitable coefficients α_k, one can use the same methods as for Runge-Kutta methods (See section 2.2), since the stability polynomial of Taylor methods determines the stability domain of the method in exactly the same way as the stability polynomial of Runge-Kutta methods does.

3.2 Calculating higher derivatives

3.2.1 Required order of accuracy of the higher derivatives

We observe in eq. (3.2) that each derivative $x^{(k)}$ is multiplied with a factor h^k. To produce an asymptotic truncation error of $O(h^{K+1})$, it is therefore not necessary to know each $x^{(k)}$ exactly. Instead, it would suffice to have an approximation

$$\tilde{x}^{(k)} = x^{(k)} + O(h^{K+1-k}). \tag{3.5}$$

These inaccuracies would not amount to larger asymptotic errors than the error already committed through truncation of the Taylor series. We can use a multiderivative method such as the Taylor method to approximations the higher derivatives at future time steps in the same way as we approximate the state variable at future times.

$$\tilde{x}^{(k)}_{n+1} := x^{(k)}_n + \sum_{j=1}^{K-k} \frac{x^{(k+j)}(t)}{j!} h^j, \quad k = 0, 1, \dots, K-1 \tag{3.6}$$

These methods would be consistent of order $K - k$ and hence, by Theorem 1.11, the approximations $\tilde{x}^{(k)}_{n+1}$ would be accurate of order $K - k$, which is one order of accuracy less than is required by eq. (3.5). So far, we have not used the function f to calculate the higher derivatives. This will be the key to increasing the order of accuracy of the higher derivative approximations by one and enable us to evaluate the equations eq. (3.6) in parallel to efficiently calculate the higher derivatives in a way suitable for modern compute architecture. First we will look into the classical way of computing higher derivatives.

3.2.2 Standard approaches to calculating higher derivatives

Methods using $x^{(k)}(t) = \frac{d^k}{dt^k} x(t)$ have not found nearly as much attention as for example Runge-Kutta methods because traditionally, the calculation of these derivatives has been the bottleneck to their use. According to Butcher [7, sec. 250], "Most serious investigations of [the Taylor Method] have been concerned, above all, with the automatic generation of procedures for generating the second, third, … derivative functions $f_2, f_3, \dots$ from a given first derivative function f."

Hairer and Wanner [13] calculated the well-known formulas to compute the higher derivatives $x^{(k)}$ for the autonomous case $x^{(1)} = f(x)$. We use the notation $f_x = \frac{\partial}{\partial x} f, f_{xx} = \frac{\partial^2}{\partial x^2} f, \dots$. We omit the state argument, i.e. write f instead of $f(x)$ and denote the arguments of multilinear operators in brackets e.g. $f_{xx}(y,z)$ is the bilinear map f_{xx} operating on the vectors $y, z \in \mathbb{R}^d$.

$$\begin{aligned}
x^{(0)} &= x \\
x^{(1)} &= f \\
x^{(2)} &= f_x(f) \\
x^{(3)} &= f_{xx}(f,f) + f_x(f_x(f)) \\
x^{(4)} &= f_{xxx}(f,f,f) + f_{xx}(f_x(f),f) + 2f_{xx}(f,f_x(f)) + f_x(f_{xx}(f,f)) + f_x(f_x(f_x(f))) \\
&\vdots
\end{aligned}$$

From Schwarz's theorem [20] and the assumption that f is K times continuously differentiable, we know that the partial derivatives $\partial_i \partial_j f$ commute. Hence, the multilinear operators $f_{xx}, f_{xxx}, \dots$ are symmetric, i.e. $f_{xx}(y,z) = f_{xx}(z,y)$ and $f_{xxx}(y,z,w) = f_{xxx}(w,y,z) = f_{xxx}(z,w,y)$ etc. This means that the order of the arguments does not matter. Using this and inserting the previous derivatives into the multilinear operators recursively lets us simplify the formulas to the recursive formulation

$$\begin{aligned}
x^{(0)} &= x \\
x^{(1)} &= f \\
x^{(2)} &= f_x(x^{(1)}) \\
x^{(3)} &= f_{xx}(x^{(1)},x^{(1)}) + f_x(x^{(2)}) \\
x^{(4)} &= f_{xxx}(x^{(1)},x^{(1)},x^{(1)}) + 3f_{xx}(x^{(2)},x^{(1)}) + f_x(x^{(3)}) + f_x(x^{(2)})(x^{(2)}) \\
&\vdots
\end{aligned} \tag{3.7}$$

3.2.3 Calculating higher derivatives in parallel

We assume that all $f_{xx}, f_{xxx}, \dots$ in eq. (3.7) are Lipschitz continuous with Lipschitz constant L in each of their arguments. If we plug in the approximations $\tilde{x}_n^{(k)}$ obtained by eq. (3.6) with accuracy $O(h^{K-k})$ into the right-hand sides, the left-hand sides $x_n^{(k)}$ will be of the desired accuracy $x_n^{(k)} = x^{(k)}(t_n) + O(h^{K-k+1})$, since each $x_n^{(k)}$ for $k = 1 \dots K$ only depends on the values $\tilde{x}^{(j)}$ for $j \le k-1$, which are accurate of at least order $K - (k-1) = K - k + 1$.

Equation (3.8) summarizes the approach for the parallel computation of the higher derivatives.

$$\begin{pmatrix} x_n^{(0)} \\ x_n^{(1)} \\ \vdots \\ x_n^{(K)} \end{pmatrix} \overset{\text{eq. (3.6)}}{\longmapsto} \begin{pmatrix} \tilde{x}_{n+1}^{(0)} \\ \tilde{x}_{n+1}^{(1)} \\ \vdots \\ \tilde{x}_{n+1}^{(K-1)} \end{pmatrix} \overset{\text{eq. (3.7)}}{\longmapsto} \begin{pmatrix} x_{n+1}^{(0)} \\ x_{n+1}^{(1)} \\ \vdots \\ x_{n+1}^{(K)} \end{pmatrix} \tag{3.8}$$

This approach can be viewed as replacing the state information $x(t_n)$ with the information contained in the Taylor Polynomial of $x(t)$ constructed at the time t_n and then propagating the new extended state information $\hat{x}_n = (x_n^{(0)}, x_n^{(1)}, \dots, x_n^{(K-1)})$ forward in time using the Taylor method. Instead of using instances of the Talyor method, we could use any multiderivative time marching method, which works with derivatives $\tilde{x}^{(k)}$ approximated with order of accuracy $K - k + 1$.

The idea stems from an approach studied by Cellier and Kofman [10, ch.11] regarding Quantized State Systems. As discussed with my advisor Zimmer [31], the implementation of calculating higher derivatives $x^{(k)}$ will be done in the simulation framework (e.g. Dymola [26]) and not in the solver itself. Hence, it will not be the focus of this thesis. For sparse systems, the implementation within the simulation framework also allows for the calculation of higher derivatives $x^{(k)}$ without instantiating the multilinear operators $f_{xx}, f_{xxx}, \dots$. Thus, when studying the computational complexity of the methods, we will assume the calculation of higher derivatives to be $O(d \cdot K)$ in the total number of operations and the runtime to be $O(d)$ when leveraging parallel computing architectures.

3.3 Multiderivative-Runge-Kutta

It is sketched in [7, Ch.2] to combine the use of multiple stages and multiple time-derivatives.

These methods are known as multiderivative Runge-Kutta (MDRK) methods and are generally given by the form

$$\begin{aligned} x_{n+c_s} &:= x_n + \sum_{k=1}^{K} \frac{(c_s h)^k}{k!} \sum_{j=1}^{s-1} a_{ij} f^{(k)}(t_n + hc_j, x_{n+c_j}) \quad \text{for} \quad s = 1 \dots S \\ x_{n+1} &:= x_n + \sum_{k=1}^{K} \frac{h^k}{k!} \sum_{j=1}^{S} b_j f^{(k)}(t_n + hc_j, x_{n+c_j}), \end{aligned}$$

where the coefficients $c = (c_s)_{s=1\dots S}$, $A^{(k)} = (a_{sj})^{(k)}_{sj=1\dots S}$, $b = (b_i)^{(k)}_{s=1\dots S}$ for $k = 1 \dots K$ can be arranged in the extended Butcher tableau

$$\begin{array}{c|c|c|c|c|} c & A^{(1)} & A^{(2)} & \cdots & A^{(K)} \\ \hline & (b^{(1)})^T & (b^{(2)})^T & \cdots & (b^{(K)})^T \end{array} \tag{3.9}$$

For linear systems $x^{(1)} = Lx$, similar to Runge-Kutta methods, these formulas simplify to the evaluation of polynomials in the matrix hL. We give the example with $K = 2$ and $S = 2$.

Example 3.1 (MDRK on linear system)

$$\begin{aligned} x_{n+c_1} &= x_n \\ x_{n+c_2} &= x_n + (c_2 hL) a_{21}^{(1)} x_n + \frac{(c_2 hL)^2}{2} a_{21}^{(2)} x_n \end{aligned} \tag{3.10}$$

$$\begin{aligned} x_{n+1} &= x_n + (hL)\left(b_1^{(1)} x_n + b_2^{(1)} x_{n+c_2}\right) + \frac{(hL)^2}{2}\left(b_1^{(2)} x_n + b_2^{(2)} x_{n+c_2}\right) \\ &= x_n + \left((hL) b_1^{(1)} + \frac{(hL)^2}{2} b_1^{(2)}\right) x_n + \left((hL) b_2^{(1)} + \frac{(hL)^2}{2} b_2^{(2)}\right)\left(x_n + (c_2 hL) a_{21}^{(1)} x_n + \frac{(c_2 hL)^2}{2} a_{21}^{(2)} x_n\right) \\ &= (I + (b_1^{(1)} + b_2^{(1)})(hL) + (b_1^{(2)} + 2 b_2^{(1)} c_2 a_{21}^{(1)}) \frac{(hL)^2}{2} \\ &+ (b_2^{(1)} c_2^2 a_{21}^{(2)} + b_2^{(2)} c_2 a_{21}^{(1)}) \frac{(hL)^3}{2} + (b_2^{(2)} c_2^2 a_{21}^{(2)}) \frac{(hL)^4}{4}) x_n \end{aligned} \tag{3.11}$$

Thus, the linear stability properties of MDRK methods can be assessed through the same methods as outlined in section 2.2 regarding the stability polynomials of Runge-Kutta methods.

Furthermore, the stability polynomial entirely determines the discrete evolution of MDRK methods on linear systems. Thus, an MDRK method, whose stability polynomial agrees with the Taylor polynomial in the first p coefficients, is - just like Taylor methods - consistent of order p. For nonlinear systems, verifying the order conditions is more complicated. To achieve nonlinear order of accuracy p, the coefficients of the method must solve a system of nonlinear equations. The derivation of such a method through the solution of a nonlinear system of equations is demonstrated by Wusu, Akanbi, and Okunuga [28].

Remark 3.2 *MDRK methods unify Runge-Kutta and Taylor methods into a single larger class of methods. Runge-Kutta methods form the special case of $K = 1$. Taylor methods form the special case of $S = 1$. Setting $S = 1$ and $K = 1$, we find the Euler method. MDRK methods span the lower left face of Figure 2.1.*

Calculating higher derivatives from MDRK methods

We study only the linear case and start by returning to example 3.1. The following equalities follow directly from eq. (3.10) and let us calculate all $x^{(k)}(t_n)$ for $k = 1 \dots (K \cdot S)$ from the evaluations of $f_{k,s} := f^{(k)}(t_n + c_s, x_{n+c_s})$ for $s = 1 \dots S, k = 1 \dots K$

$$\begin{aligned}
x^{(1)}(t_n) &= Lx_n = f^{(1)}(t_n, x_{n+c_1}) \\
x^{(2)}(t_n) &= L^2 x_n = f^{(2)}(t_n, x_{n+c_1}) \\
x^{(3)}(t_n) &= L^3 x_n = \frac{2}{a_{21}^{(1)} c_2^2 h^2} \left(f^{(1)}(t_n + hc_2, x_{n+c_2}) - x^{(1)}(t_n) - h a_{21}^{(1)} c_2 h x^{(2)}(t_n) \right) \\
x^{(4)}(t_n) &= L^4 x_n = \frac{2}{a_{21}^{(1)} c_2^2 h^2} \left(f^{(2)}(t_n + hc_2, x_{n+c_2}) - x^{(2)}(t_n) - h a_{21}^{(1)} c_2 h x^{(3)}(t_n) \right)
\end{aligned}$$

In the same way as illustrated for the case $K = 2$ and $S = 2$, it is for any linear system and set of parameters S and K, possible to calculate the derivatives up to $k = K \cdot S$ as linear combinations of the function evaluations $f_{k,i}$. For larger K and S, determining the correct coefficients for the linear combination of $f_{k,i}$ can be done by solving a linear system of equations.

This lets us compute the higher derivatives even if no explicit formula for the higher derivatives has been implemented in the model. This technique can bridge the gap towards the applicability of multiderivative methods when the calculation of multiple time-derivatives has not been implemented to sufficient order in an existing current model.

For nonlinear systems, finding expressions to calculate higher derivatives from the evaluations $f_{k,i}$, requires the solution of a nonlinear system of equations related to the one solved by Wusu, Akanbi, and Okunuga [28] for the derivation of a method with high nonlinear consistency order. This lies beyond the scope of this thesis. If successfully derived, such an approach could greatly improve the performance of other multiderivative methods such as the Krylov Jacobian approximation presented in chapter 4 by providing more derivatives than are available from the provided model. It might even be possible to adapt multiderivative Runge-Kutta methods to work with parallel function evaluations like parallelizable peer methods. Such a method, that can calculate high order derivatives through parallel function evaluations could enable multiderivative methods with remarkable performance on parallel computers.

We end this section with two tests of multiderivative Runge-Kutta methods. One on a linear system and one on a nonlinear system. These tests verify that MDRK methods can achieve remarkable accuracy on linear systems, but the accuracy on nonlinear systems is lower since the used scheme does not fulfill the nonlinear order conditions.

3.3.1 Accuracy of MDRK methods

Numerical Test 1 *We test MDRK schemes with $K = 4$ and $S = 2, 4$ on a linear Test System 2 $x^{(1)} = Lx, x_0 = (1, 1, 0, 0)^T$ where*

$$L = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix} \tag{3.12}$$

The extended Butcher tableau of the method with $(S = 2, K = 4)$ is

$$
(S=2,K=4):\;\begin{array}{c|cc|cc|c}
0 & 0 & 0 & 0 & 0 & \cdots \\
1 & 1 & 0 & 1 & 0 & \cdots \\
\hline
0 & 0.9095238095238096 & 0.7857142857142857 & 0.6142857142857143 & 0.3666666666666668 & \cdots
\end{array}
$$

$$
\begin{array}{c|cc|cc}
\cdots & 0 & 0 & 0 & 0 \\
\cdots & 1 & 0 & 1 & 0 \\
\hline
\cdots & 0.0904761904761904 & 0.0333333333333333 & 0.0142857142857143 & 0.0142857142857143
\end{array}
\tag{3.13}
$$

Table 3.1 Extended Butcher tableau of **MDRK(S=2,K=4)**

The code for these methods can be found in the appendix. These schemes are of consistency order 8 *and* 16 *respectively for linear systems and thus, achieve remarkable accuracy. In Figure 3.1, we compare them to the well known Runge-Kutta-4 scheme with Butcher tableau*

$$
\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
\frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
0 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
\tag{3.14}
$$

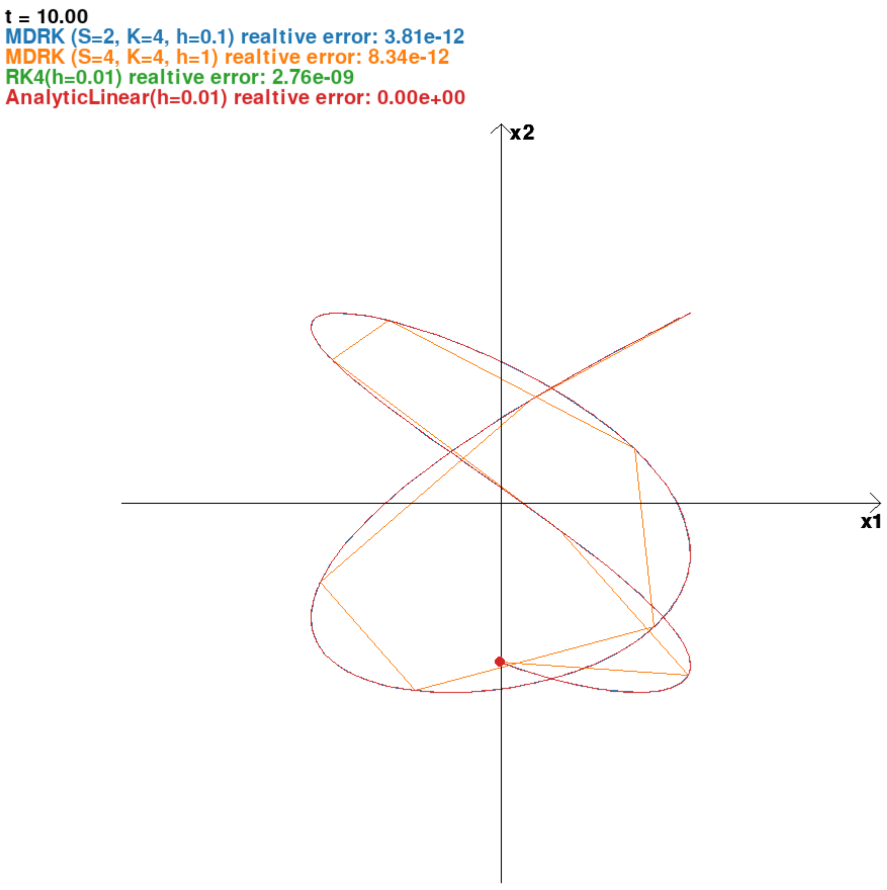
.

Next, we test both MDRK schemes on a nonlinear system. These methods have not been designed to fulfill the nonlinear order conditions found by Wusu, Akanbi, and Okunuga [28]. Hence, their order of accuracy is reduced to $p = K = 4$, so they are barely more accurate than the respective Taylor Method. Note that MDRK($S = 1, K = 4$) = TaylorMethod($K = 4$).

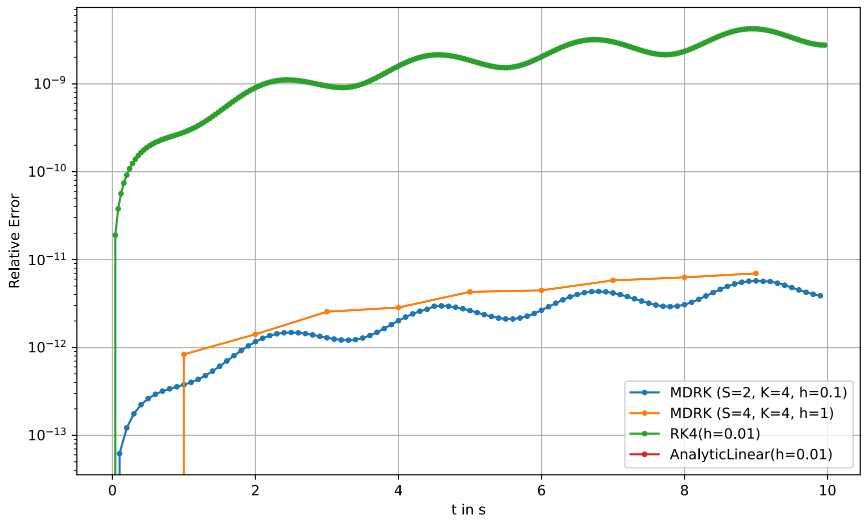
Test System 3 (Nonlinear Example System) *[32]*

$$
\begin{pmatrix} x \\ \dot{x} \end{pmatrix}^{(1)} = \begin{pmatrix} \dot{x} \\ -\frac{4}{3} r\,sign(x)\,|x|^{\frac{3}{2}} \end{pmatrix}, \quad \begin{pmatrix} x(0) \\ \dot{x}(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}
$$

Numerical Test 2 (MDRK on nonlinear system) *We compare the MDRK methods with $S = 2, K = 4$ and $S = 4, K = 4$ to a Taylor method with $K = 4$ and the RK4 method on Test System 3. See Figure 3.1 for the trace and accuracy plot of the test. We see that the multiderivative methods are similarly outperformed by the RK4 method.*

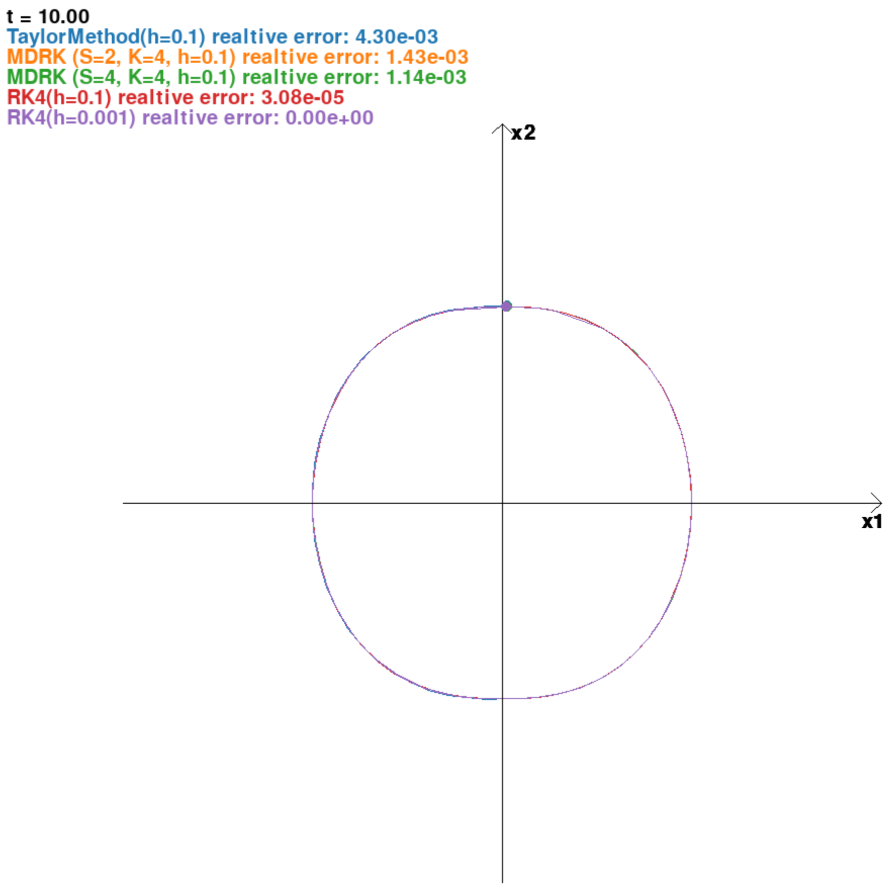


(a) Trace

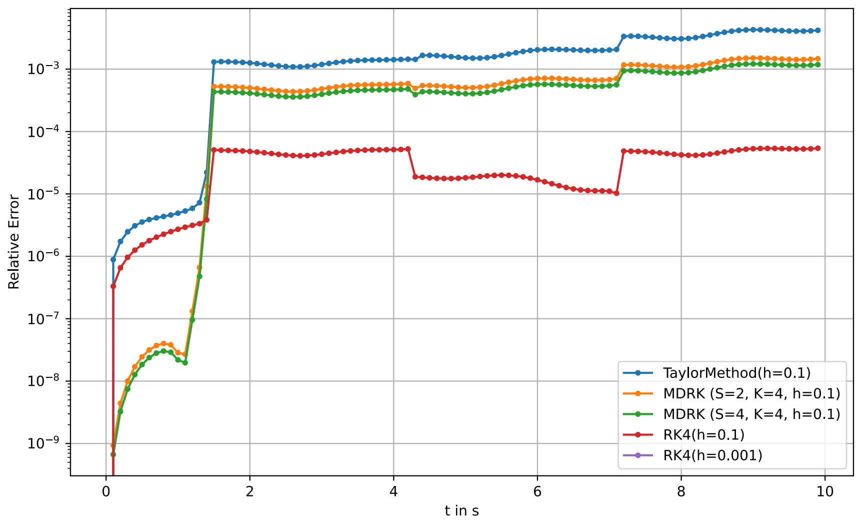


(b) Relative Errors

Figure 3.1 MDRK schemes on linear system



(a) Trace



(b) Relative Errors

Figure 3.2 MDRK schemes on nonlinear system

4 Krylov Jacobian Approximation

This chapter presents a method to obtain a low rank approximation of the Jacobian $J_f(x,t) = \frac{\partial}{\partial x} f(x,t)$ using only evaluations of multiple derivatives $\frac{\partial^k}{\partial^k} x(t)$ of the solution curve. It will be the core method enabling the use of advanced methods that rely on the Jacobian.

4.1 Krylov Subspace Reduction

According to Gutknecht [12], Krylov subspace methods are amongst the most popular methods for solving very large linear systems of equations $Lx = b$. The Conjugate Gradient (CG) for symmetric or hermitian, positive definite matrices L and the Generalized Minimal Residual (GMRES) method for general matrices are common techniques to solve large linear systems. These solvers start with an initial approximation x_0, and they successively compute the best approximation

$$x_n := \operatorname{argmin}_{x \in \mathcal{K}_n(L,x_0)} \|Lx - b\| \tag{4.1}$$

where $\mathcal{K}_n(L, x_0) := \operatorname{span}(x_0, Lx_0, L^2x_0, \ldots, L^{n-1}x_0)$ is the Krylov subspace of dimension n spanned by the initial approximation x_0 and the first n powers of the matrix L acting on x_0.

While we are not particularly interested in solving large linear systems, we can profit from the insight that the Krylov subspaces $\mathcal{K}_n(L, x_0)$ already capture enough information about the matrix L to approximate the action of L on a vector x.

This idea is used in Hochbruck and Ostermann [14] to efficiently approximate the action of the matrix exponential e^{hL} on a vector x by reducing the necessary computations to the Krylov subspace $\mathcal{K}_n(L, x_0)$.

4.2 Krylov Jacobian Approximation in the linear case

We will now use multiple time-derivatives $x^{(k)}(t)$ to build a Krylov subspace approximation of the Jacobian $J_f(t,x) = \frac{\partial f}{\partial x}(t,x)$. We start by looking at the linear test system $x^{(1)} = Lx$, $x(0) = x_0$. The analytical solution of this system is well known to be

$$x(t) = e^{Lt}x(0) = \sum_{k=0}^{\infty} \frac{L^k t^k}{k!} x(0). \tag{4.2}$$

This can be verified using the Existence and Uniqueness Theorem 1.2. Hence, the time-derivatives are of the form

$$x^{(k)}(t) = \frac{\partial^k}{\partial t^k} e^{Lt}x(0) = L^k e^{Lt} x(0) = L^k x(t). \tag{4.3}$$

Recalling the definition of the Krylov space $\mathcal{K} := \mathcal{K}_K(L, x) = \operatorname{span}(x, Lx, L^2x, \ldots, L^{K-1}x)$, we make the key observation that the time-derivatives $x^{(0)}, x^{(1)}, \ldots, x^{(K-1)}$ form a basis of the Krylov space $\mathcal{K}_K(L, x)$. Hence, the time-derivatives $x^{(k)}$ can be used to approximate the action of the operator L on the Krylov space, i.e. we have

$$Lx^{(k)} = x^{(k+1)} \quad \text{for} \quad k = 0, 1, \ldots, K-1. \tag{4.4}$$

As eq. (4.4) defines the action of L on a Krylov-basis, any operator, which fulfills eq. (4.4) acts identically to L on the entire Krylov space $\mathcal{K}$. We now want to find a low rank operator $\hat{B}$, which has this property.

Definition 4.1 (Krylov Operator) *Let $x^{(0)} \dots x^{(K)} \in \mathbb{R}^d$ be linear independent vectors. We call a Matrix $\hat{B} \in \mathbb{R}^{d\times d}$ a* Krylov operator *on the space $\mathcal{K} := span(x^{(0)} \dots x^{(K-1)})$ if it fulfills*

$$\hat{B}x^{(k)} = P_{\mathcal{K}}x^{(k+1)} \quad \textit{for} \quad k = 0 \dots K-1, \tag{4.5}$$

where $P_{\mathcal{K}} : \mathbb{R}^d \to \mathcal{K}$ is the orthogonal projection onto the Krylov space $\mathcal{K}$.

We use the projection $P_{\mathcal{K}}$ on the right hand side, because it is advantageous for the subsequent calculation using the Krylov operator, that its image space is contained in the Krylov space. i.e. $\hat{B}\mathcal{K} \subseteq \mathcal{K}$. We notice that the projection only affects the condition in eq. (4.5) with $k = K-1$. Hence, we can simplify the other conditions in eq. (4.5) through the following Lemma.

Lemma 4.2 (Simplified Krylov Property) *The following statements concerning an operator $\hat{B} \in \mathbb{R}^{d\times d}$ are equivalent:*

1. *$\hat{B}$ is a Krylov Operator on the space $\mathcal{K} = span(x^{(0)} \dots x^{(K-1)})$.*
2. *It holds that*

$$\hat{B}x^{(k)} = x^{(k+1)} \quad \textit{for} \quad k = 0 \dots K-2 \tag{4.6}$$

$$\textit{and} \quad \hat{B}x^{(K-1)} = P_{\mathcal{K}}x^{(K-1)}. \tag{4.7}$$

Proof. *The condition in eq.* (4.7) *is identical to eq.* (4.5)*. The conditions in eq.* (4.6) *are equivalent to the ones from eq.* (4.5) *because $x^{(k+1)} \in \mathcal{K}$ for $k = 0 \dots K-2$ and thus, the projection is $P_{\mathcal{K}}x^{(k+1)} = x^{(k+1)}$.* □

The remainder of this section is concerned with computing a Krylov Operator from vectors $x^{(0)} \dots x^{(K)}$. The next Lemma lets us formalize the Krylov property in terms of a matrix-vector multiplication. We use the matrices

$$T := \begin{pmatrix} I_K \\ 0 \end{pmatrix} \in \mathbb{R}^{(K+1)\times K} \quad \text{and} \quad T_Y := \begin{pmatrix} 0 \\ I_K \end{pmatrix} \in \mathbb{R}^{(K+1)\times K}, \tag{4.8}$$

where I_K is the $K \times K$ identity matrix and 0 is a row of zeros of appropriate size. Multiplying a matrix M from the right with T or T_Y removes the first or last row of M respectively. Multiplying from the left acts the same way on the rows.

Lemma 4.3 (Krylov Property in Matrix Vector Notation) *Let*

$$Z := \left(x^{(0)} \dots x^{(K)}\right) \in \mathbb{R}^{d\times(K+1)}$$

$$X := ZT = \left(x^{(0)} \dots x^{(K-1)}\right) \in \mathbb{R}^{d\times K}$$

$$Y := ZT_Y = \left(x^{(1)} \dots x^{(K)}\right) \in \mathbb{R}^{d\times K}. \tag{4.9}$$

Then, the Krylov property eq. (4.5) *can be written as*

$$\hat{B}X = P_{\mathcal{K}}Y. \tag{4.10}$$

Proof. *Comparing X and Y columnwise shows the equivalence to eq.* (4.5)*.* □

To compute the Krylov Operator $\hat{B}$, we will use the QR decomposition [3] of Z.

Lemma 4.4 (QR-Decomposition) *Let $Z \in \mathbb{R}^{d\times(K+1)}$ be a matrix with linearly independent columns. Then, there exist unique matrices $Q \in \mathbb{R}^{d\times(K+1)}$ and $R \in \mathbb{R}^{(K+1)\times(K+1)}$ such that*

$$QR = Z, \tag{4.11}$$

such that Q is column orthonormal. That is $Q^TQ = I_{K+1}$ is the identity matrix of size $(K+1) \times (K+1)$. And such that R is upper triangular, i.e. $r_{ij} = 0$ for $i > j$. The matrices Q and R are unique up to a sign change of the columns of Q.

Proof. *We refer to Bornemann [3, Theorem on p.32].* □

The following theorem lets us compute a Krylov Operator $\hat{B}$ and is the main result of this section.

Theorem 4.5 (Krylov operator) *Let $x^{(0)} \dots x^{(K)} \in \mathbb{R}^d$ be linearly independent vectors. Let T, T_Y, Z, X and Y be defined as in eq.* (4.8) *and eq.* (4.9). *Let $Z = \hat{Q}\hat{R}$ be the QR decomposition of Z.*

Let

$$R_X := T^T \hat{R} T \quad \text{and}$$
$$R_Y := T^T \hat{R} T_Y$$

be the upper left and upper right part of the matrix $\hat{R}$. Because $x^{(0)} \dots x^{(k)}$ are linearly independent, the matrix $\hat{R}$ and thus also the matrix R_X are invertible. Let

$$Q := \hat{Q}T \in \mathbb{R}^{d \times K} \tag{4.12}$$

be the first K columns of $\hat{Q}$. Then, the matrix

$$\hat{B} := QR_Y R_X^{-1} Q^T \in \mathbb{R}^{d \times d} \tag{4.13}$$

is a Krylov operator on the Krylov space $\mathcal{K} = span(x^{(0)} \dots x^{(K-1)})$.

Proof. *We will verify the equality eq.* (4.10), *which is equivalent to the Krylov property eq.* (4.5) *by lemma 4.3.*

First, we notice that Q inherits column orthonormality from $\hat{Q}$, because

$$Q^T Q = (T^T \hat{Q}^T)(\hat{Q}T) = T^T \hat{Q}^T \hat{Q} T = T^T I_{K+1} T = I_K. \tag{4.14}$$

Let $k \in \{0, \dots, K-1\}$. Then we have $\hat{Q}\hat{r}^{(k)} = x^{(k)}$ where $\hat{r}^{(k)}$ is the k-th column of $\hat{R}$. Because of the upper triangular structure of $\hat{R}$, the last entry of $\hat{r}^{(k)}$ is zero. Hence, with $r^{(k)} := T^T \hat{r}^{(k)}$ it holds

$$Qr^{(k)} = \hat{Q}TT^T \hat{r}^{(k)} = \hat{Q}\hat{r}^{(k)} = x^{(k)}. \tag{4.15}$$

Since we can write all $x^{(k)}$ as linear combinations of the columns of Q, and Q is column-orthonormal, the columns of Q form an orthonormal basis of the Krylov space $\mathcal{K}$. It is well known from linear algebra [17] that QQ^T then equals the orthogonal projection $P_{\mathcal{K}}$ onto the Krylov space $\mathcal{K}$ (see lemma 8.2 for a proof). Hence, we can write

$$\begin{aligned} QR_Y &= Q(T^T \hat{R} T_Y) \\ &= QT^T (\hat{Q}^T \hat{Q}) \hat{R} T_Y \\ &= Q(T^T \hat{Q}^T)(\hat{Q}\hat{R}) T_Y \\ &= QQ^T Z T_Y \\ &= P_{\mathcal{K}} Y. \end{aligned} \tag{4.16}$$

Analogously, we get

$$\begin{aligned} QR_X &= Q(T^T \hat{R} T) \\ &= QT^T (\hat{Q}^T \hat{Q}) \hat{R} T \\ &= Q(T^T \hat{Q}^T)(\hat{Q}\hat{R}) T \\ &= QQ^T Z T \\ &= P_{\mathcal{K}} X \\ &= X, \end{aligned} \tag{4.17}$$

where the last equality holds because all columns of X are contained in the Krylov space $\mathcal{K}$. Now we can verify the Krylov property eq. (4.10) *by inserting eqs.* (4.16) *and* (4.17).

$$\begin{aligned} \hat{B}X &= QR_Y R_X^{-1} Q^T X \\ &= QR_Y R_X^{-1} Q^T Q R_X \\ &= QR_Y \\ &= P_{\mathcal{K}} Y \end{aligned}$$

□

Remark 4.6 *It is useful to think of the operator $\hat{B}$ in two parts. Given any vector $x \in \mathbb{R}^d$ …*

1. *…$R_X^{-1}Q^Tx =: a$ extracts the coefficients a, with which $P_{\mathcal{K}}x$ can be written as a linear combination of the Krylov basis vectors $x^{(0)}, \dots, x^{(K-1)}$.*

2. *…QR_Ya constructs a linear combination from these coefficients with respect to the vectors $P_{\mathcal{K}}x^{(1)}, \dots, P_{\mathcal{K}}x^{(K)}$.*

This algorithm could be implemented through the following **python** code using the **numpy** library.

Algorithm 1 Build Krylov Operator from linearly independent vectors

```
import numpy as np
def build_operator(self, Z):
    X = Z[:, :-1]
    Q, R = np.linalg.qr(Z) # QR decomposition
    Kx = min(Z.shape[0], K-1) # Account for small d
    Rx = R[:Kx, :Kx]
    Ry = R[:Kx, 1:Kx+1]
    Q = Q[:,:Kx]
    Q = Q[:,b]
    B = Ry @ np.linalg.inv(Rx)
    return Q, B
```

It is computationally very useful to return Q and B instead of combining them to $\hat{B}$ to exploit the low dimension of B in further calculations. In section 5.2 we need to evaluate $e^{h\hat{B}}$, which is a lot cheaper after the reformulation

$$e^{h\hat{B}} = e^{Q(hB)Q^T} = Qe^{hB}Q^T \tag{4.18}$$

The main work will be cut down from a $d \times d$ matrix exponential on the left to a $K \times K$ matrix exponential on the right. We also need to mention that the explicit inversion of R_X in Algorithm 1 might become numerically unstable for larger K. Since the number of time-derivatives K is usually small (typically $K \leq 10$), it will most likely not be a problem in practice. It depends on the further use of the operator $\hat{B}$ whether it is possible to avoid the inversion of R_X entirely. Where we only need to evaluate matrix vector products of the form $\hat{B}v = QBQ^Tv$, we can use the following algorithm for the evaluation of $\hat{B}v$ given R_X, R_Y, T, Q and v without explicitly computing R_X^{-1}.

Algorithm 2 Apply Krylov Operator to a vector

```
    def apply_operator(self, Rx, Ry, T, Q, v):
        w = np.linalg.solve(Rx, T.T @ (Q.T @ v))
        return Q @ (Ry @ w)
```

4.2.1 Linear dependency

In theorem 4.5, we assumed the $x^{(k)}$ were linearly independent. This might not be the case in practice. Fortunately, **numpy**'s QR decomposition algorithm works regardless of the linear independence of the columns of Z. If $x^{(k)}$ can be expressed as a linear combination of the $x^{(j)}, j < k$, then the diagonal element[1] $r_{k,k}$ will equal 0 and if $k \leq K-1$, the matrix R_X will not be invertible, which is necessary for Algorithm 1 to work. To repair the invertibility of R_X, we can remove the row and column, in which the 0-diganoal-element occurred. If we remove all vectors $x^{(k_0)}$, for which the diagonal element r_{k_0,k_0} is zero,

[1] Here, we use zero based matrix indexing to stay consistent with the indexation of the vectors $x^{(k)}$.

the remaining vectors $x^{(k_{\neq 0})}$ will be linearly independent and consequently the matrix R_X invertible. We implement this through the boolean array **b**, which contains **True** for exactly those columns, that cannot be expressed as a linear combination of their predecessors and thus, form a basis of the Krylov space K.

Algorithm 3 Build Krylov Operator from possibly linearly dependent vectors

```
import numpy as np
def build_operator(self, Z):
    """creates a linear operator, which maps the columns of Z
       like z_1 -> z_2 -> z_3 -> ... -> z_K"""
    # Assumes Z of shape (d x K)
    # Returns Q, B such that (Q@B@Q.T)z_k = z_{k+1}
    X = Z[:, :-1]
    K = Z.shape[1]  # Number of time-derivatives
    Kx = min(Z.shape[0], K-1) # dimension of Krylov Space
    Q, R = np.linalg.qr(Z) # QR decomposition
    Rx = R[:Kx, :Kx]
    Ry = R[:Kx, 1:Kx+1]
    Q = Q[:,:Kx]
    b = np.abs(np.abs(np.diag(Rx)) > 1e-10) # small tolerance
    if not np.any(b):
        return np.zeros((Z.shape[0],1)), -np.ones((1,1))
    Rx = Rx[b,:][:,b]
    Ry = Ry[b,:][:,b]
    Q = Q[:,b]
    B = Ry @ np.linalg.inv(Rx)
    lams = np.linalg.eigvals(B)
    for lam in lams: self.eigvals.append(lam)
    return Q, B # Q @ B @ Q.T = ^B
```

4.2.2 Test on Linear System

Test System 4 (Linear System) *A linear System is of the form*

$$x^{(1)} = Lx, \quad x(0) = x_0 \tag{4.19}$$

where $L \in \mathbb{R}^{d\times d}$ and $x_0 \in \mathbb{R}^d$. Its analytical solution is well known to be

$$x(t) = e^{tL}x_0. \tag{4.20}$$

From the analytical formula, we can design a simple time marching method, which produces the analytic solution of eq. (4.20) at all grid points $t_n = n \cdot h$. The discrete evolution of this analytic linear method is given by

$$\Psi^{t+h,t}x = e^{Lh}x \tag{4.21}$$

We will use this method as a reference solution when testing on linear systems.

Since our work of chapter 4 lets us approximate the Jacobian of the system $J_f(t,x) = L \approx \hat{B}$, we can use $\hat{B}$ instead of L in eq. (4.21) to approximate the analytic solution of eq. (4.19). Using the matrix $\hat{B} = QBQ^T$ obtained from Algorithm 5, we obtain a linear exponential integrator.

$$\Psi^{t+h,t}x = e^{h\hat{B}}x = e^{hQBQ^T}x = Qe^{hB}Q^Tx \tag{4.22}$$

It can be implemented in the following way:

Algorithm 4 Linear Krylov step

```
import numpy as np
import scipy.linalg as la
def __call__(self, t, x, fk):
    Q, B = self.krylov_operator(t, x, fk,
       correct_excitement=False)
    return Q @ (la.expm(self.h * B) @ (Q.T @ x))
```

Numerical Test 3 (Linear System with $d = 4$ and $K = 4$) *We test this method on a Linear System with the matrix*

$$L = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -100 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, \quad x_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 2 \end{pmatrix} \tag{4.23}$$

L has the eigenvalues $\lambda_{1\ldots4} = i, -i, 10i, -10i$. One can interpret this system as two independent springs - one horizontally and one vertically oriented - moving an object in the 2D-plane. The state vector represents the phase space of $(x - position, y - position, x - velocity, y - velocity)$.

The trace of the $(x - position, y - position)$ is shown in Figure 4.1

We see that the linear Krylov method performs very well on this system and produces an error only slightly larger than the machine precision of 10^{-16}. This should not surprise us, as the method is designed to produce the analytic solution using the Krylov approximation of the Jacobian as the underlying linear operator. We can expect the Krylov Jacobian approximation to -up to machine precision- return the exact Jacobian, since the underlying Krylov space is 4-dimensional in this case and therefore spans the entire 4-dimensional space of the system. For this test, at time $t = 4.5$, the Krylov approximation of the Jacobian is

$$\begin{bmatrix} -3.48e-15 & -1.20e-14 & 1.00e+00 & 5.89e-14 \\ 9.02e-15 & 4.14e-15 & 2.67e-16 & 1.00e+00 \\ -1.00e+02 & -2.62e-14 & -2.37e-15 & -1.80e-13 \\ 3.22e-15 & -1.00e+00 & -2.80e-17 & -2.18e-15 \end{bmatrix} \tag{4.24}$$

This deviates from the desired matrix L by $\left\|L - \hat{B}\right\|_\infty = 2.801e-13$. This small error is also not surprising, since we used a 4 dimensional Krylov space to approximate the 4×4 Jacobian of the system.

Numerical Test 4 (Normalization of Magnitudes) *When running Algorithm 3, we observe that the magnitudes of the time-derivatives i.e. the columns of the matrix Z may differ greatly in magnitude. Especially for systems, in which the Jacobian has eigenvalues of large magnitude, the higher time-derivatives will blow up in magnitude. From the derivatives of the linear Test System 2, which are $x^{(k)}(t) = L^k x(t)$ (see eq. (4.3)), we see that the derivatives grow exponentially with the derivative number k and the rate of growth is determined by the eigenvalues of L. Since we are aiming to solve stiff systems, that have large eigenvalues as defined in section 1.3.1, this growth is likely to be significant. To avoid numerical instabilities, we normalized the vectors $x^{(k)}$.*

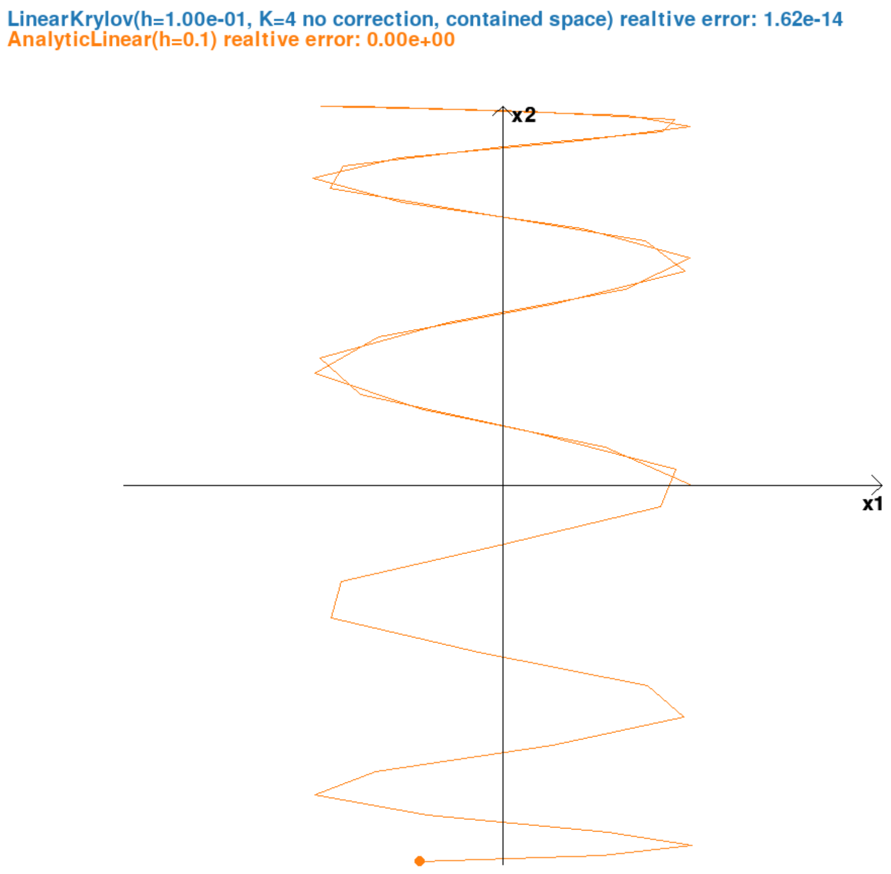
$$Z = \left(\frac{x^{(0)}}{\|x^{(0)}\|} \quad \frac{x^{(1)}}{\|x^{(1)}\|} \quad \cdots \quad \frac{x^{(K)}}{\|x^{(K)}\|} \right) \in \mathbb{R}^{d\times(K+1)}. \tag{4.25}$$

To keep the information about the derivatives' magnitudes, we used the diagonal matrix D, which has the diagonal elements[2] $d_{k,k} = \frac{x^{(k)}}{x^{(k-1)}}$ for $k = 1\ldots K$. Then we defined the Krylov Operator as

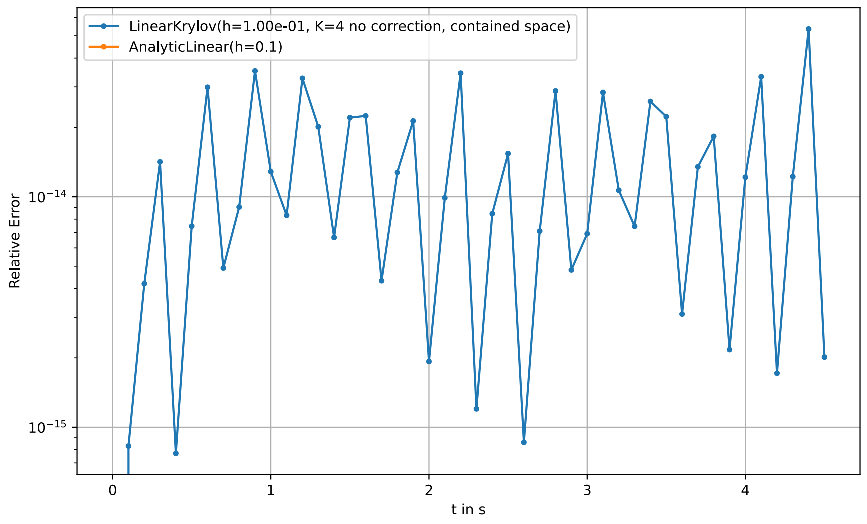
$$\hat{B} = QR_Y D R_X^{-1} Q^T \tag{4.26}$$

where $Q, R_Y and R_X$ are defined in the same way as in Theorem 4.5 but from the QR-Decomposition of the matrix Z from eq. (4.25).

[2]using 1-based matrix indices



(a) Trace



(b) Relative Errors

Figure 4.1 Linear Krylov on linear test system

Through equalizing the magnitudes of the columns of the matrix Z, this normalization improved the numerical accuracy of the Linear Krylov Integrator. For example, in the very stiff Linear System of the form as eq. (4.19) *with the matrix*

$$L = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -10^{16} & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{pmatrix}, \tag{4.27}$$

which has the eigenvalues $i, -i, 10^8 i, -10^8 i$, the relative error was reduced from approximately 10^{-1} to approximately 10^{-5}. Although this improvement is very welcome, we can see in the trace of Figure 4.2a that whilst the dominating very stiff horizontal dynamic was simulated more accurately, the slow vertical dynamic was ignored. Only calculating the errors for the vertical (x_2, x_4) components reveals that normalization significantly worsened the method's accuracy on the slow dynamic.

Since with regard to applications, we are mostly interested in the accurate simulation of the slow dynamics, this behavior is unacceptable. Thus, we will not normalize the magnitudes of the derivatives in future tests.

4.3 External Forces

So far, we have been working under the assumption that the given system is entirely linear as in test System 2 and that the time-derivatives $x^{(k)}, k = 0 \dots K$ stem entirely from multiplying the state x with the matrix L. In practice, this might not be the case. Let us take a look at a system with linear eigendynamics excited by an external force:

$$x^{(1)} = f(x,t) = Lx + g(t) \tag{4.28}$$

We see, that the time-derivatives $x^{(k)} = f^{(k)}$ are given as

$$\begin{aligned} f^{(0)} &= x \\ f^{(1)} &= Lf^{(0)} + g^{(0)} \\ f^{(2)} &= Lf^{(1)} + g^{(1)} \\ &\vdots \\ f^{(k)} &= Lf^{(k-1)} + g^{(k-1)} \quad k \geq 1, \end{aligned} \tag{4.29}$$

where $g^{(k)} = \frac{\partial}{\partial t} g(t)$ denotes the k-th derivative of the external-force term $g(t)$. Again, we use $g(t) = g^{(0)}(t)$ for convenience and abbreviate $g^{(k)}(t) =: g^{(k)}$ where the argument is clear from context.

If we apply Algorithm 3 to the test system in eq. (4.28) by setting the columns of the matrix Z as $z^{(k)} = f^{(k)}$ for $k = 0 \dots K$, we would get an operator $\hat{B}$, which suffices the following mapping.

$$\hat{B} f^{(k)} = f^{(k+1)} = Lf^{(k)} + g^{(k)} \neq Lf^{(k)} \tag{4.30}$$

This is not the desired result, since the images of the derivatives $f^{(k)}$ are disturbed by the external exciting term rather than stemming purely from the linear eigendynamics. We would like to find expressions $z^{(1)} \dots z^{(K)}$, which fulfill

$$Lz^{(k)} = z^{(k+1)} \quad for \quad k = 1 \dots K-1. \tag{4.31}$$

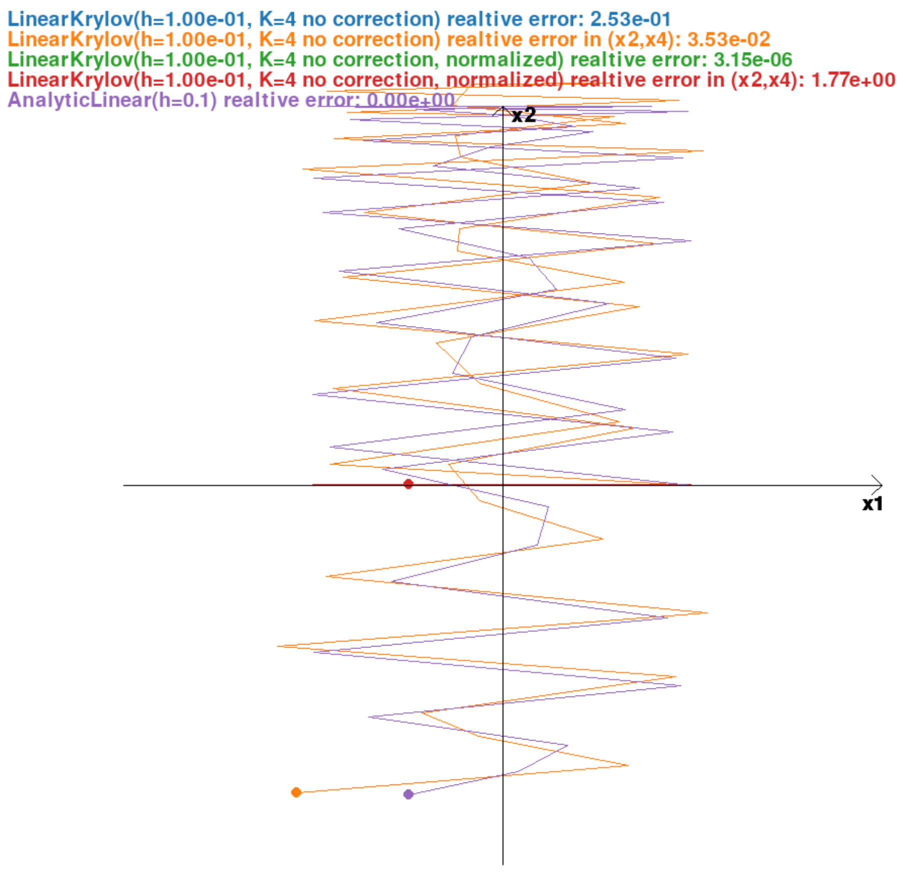
That is, we want to find the Krylov Space

$$\mathcal{K}_K(L, z^{(1)}) = \operatorname{span}(z^{(1)}, Lz^{(1)}, L^2 z^{(1)}, \dots, L^{K-1} z^{(1)}) \tag{4.32}$$

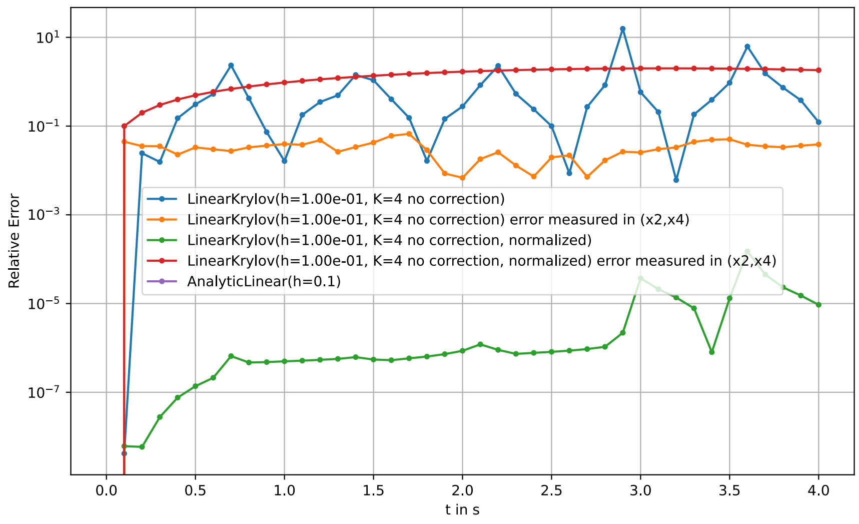
for an adequate $z^{(1)}$. The following theorem lets us compute such vectors $z^{(1)} \dots z^{(K)}$ from the derivatives $f^{(k)}(t,x)$

Theorem 4.7 (External-Force-Corrected Krylov Subspace) *Using the terms*

$$f_t^{(k)}(x,t) = \frac{\partial f^{(k)}}{\partial t}(x,t) = \lim_{r \to 0} \frac{f^{(k)}(t+r, x) - f^{(k)}(t,x)}{r}, \quad k \geq 0, \tag{4.33}$$



(a) Trace



(b) Relative Errors

Figure 4.2 Normalization on very stiff linear test system

where x is fixed and does not depend on the derivation variable t, we use the notation $f^{(0)}(t,x) = x$. Let

$$z^{(k)} = f^{(k)}(t,x) - f_t^{(k-1)}(t,x), \quad k = 1 \dots K, \tag{4.34}$$

then

$$\operatorname{span}(z^{(1)}, Lz^{(1)}, L^2 z^{(1)}, \dots, L^{K-1} z^{(1)}) = \mathcal{K}_K(L, f^{(1)}). \tag{4.35}$$

Proof. We are required to prove that $z^{(k)} = L^{k-1} f^{(1)}$ for $k = 1 \dots K$. From eq. (4.29), we can find analytic expressions for the terms $f_t^{(k)}(x,t)$, which we will abbreviate as $f_t^{(k)}$.

$$\begin{aligned} f_t^{(0)} &= 0 \\ f_t^{(k)} &= L f_t^{(k-1)} + g^{(k)} \quad k \geq 1 \end{aligned} \tag{4.36}$$

Inserting the results from eq. (4.29) and eq. (4.36), we get

$$\begin{aligned} z^{(1)} &= f^{(1)} - f_t^{(0)} = f^{(1)} - 0 = f^{(1)} \quad \text{and} \\ z^{(k+1)} &= f^{(k+1)} - f_t^{(k)} \\ &= (Lf^{(k)} + g^{(k)}) - (Lf_t^{(k-1)} + g^{(k)}) \\ &= L(f^{(k)} - f_t^{(k-1)}) \\ &= Lz^{(k)} \end{aligned} \tag{4.37}$$

for $k = 1 \dots K$. The claim follows from induction.

□

The terms $f_t^{(k)}(t,x)$ can in practice be approximated using the second order finite difference formula

$$f_t^{(k)}(x,t) \approx \frac{f^{(k)}(t+r,x) - f^{(k)}(t-r,x)}{2r}, \tag{4.38}$$

where $r > 0$ is a small parameter. Since r can be chosen arbitrarily small, the error of the finite difference will be negligible compared to the consistency error of most methods. If we apply Theorem 4.5 (Algorithm 3) to the matrix

$$Z = \begin{pmatrix} z^{(1)} & z^{(2)} & \cdots & z^{(K)} \end{pmatrix} \in \mathbb{R}^{d \times K}, \tag{4.39}$$

we obtain an operator $\hat{B}$, which fulfills

$$\hat{B}|_{\mathcal{K}} = P_{\mathcal{K}} L|_{\mathcal{K}} \tag{4.40}$$

on the Krylov space $\mathcal{K} = \mathcal{K}_{K-1}(L, f^{(1)}) = \operatorname{span}(f^{(1)}, Lf^{(1)}, L^2 f^{(1)}, \dots L^{K-2} f^{(1)})$. Note, that the dimension of the Krylov space $\mathcal{K}$ was reduced by one compared to the formulation for the purely linear case presented in Theorem 4.5. This is because computing $z^{(0)}$ is not possible from eq. (4.34). Observe that for the linear case of test System 2 studied in section 4.2, the function f does not depend on time, so all the $f_t^{(k)}$ terms become zero and the corrected matrix Z reduces to the initial selection of Z eq. (4.9) except for the omitted first column. We can implement this new external-force-correction mechanism by calling Algorithm 3 on the corrected matrix Z as defined in eq. (4.39) When the system contains external forces, we want to use the corrected matrix Z, but when simulating purely linear systems, it is advantageous to include the first column as done in eq. (4.9). In this case, there is no need to compute the $f_t^{(k)}$ terms. We use the function $\hat{f} : \mathbb{R} \times \mathbb{R}^d \times \mathbb{N} \to \mathbb{R}^d, \hat{f}(t,x,k) = f^{(k)}(t,x)$ and set $r = 10^{-8}$ for the finite difference in Algorithm 5

Algorithm 5 Krylov Jacobian corrected for external forces

```
def krylov_operator(self, t, x, fk, correct_excitement=True):
    eps = 1e-8 # finite difference
    k_range = range(1 if correct_excitement else 0, self.K+1)
    Z = np.array([fk(t, x, k) for k in k_range]).T
    if correct_excitement:
        H = np.array([(fk(t+eps, x, k)-fk(t-eps, x, k))/(2*eps)
            for k in range(self.K)]).T
        Z = Z - H
    Q, B = self.build_operator(Z)
    return Q, B
```

Remark 4.8 *This approach is not directly compatible with the parallel computing of multiple derivatives. That is because the calculation of the terms $f^{(k)}(t \pm r, x)$ in the finite difference of the term $f_t^{(k)}(t, x)$ involves recursively evaluating the terms $f^{(m)}(t \pm r, x)$ for all $m \leq k \leq K-1$ to evaluate eq.* (3.7)*, which in the case of the system from eq.* (4.28) *is of the form from eq.* (4.29)*. If we were to use the approximations $\tilde{x}^{(m)}$, which do not depend on t, calculated at the previous integration step instead, in eq.* (4.29)*, we would compute*

$$\tilde{f}_t^{(k)}(t, x) = \lim_{r \to 0} \frac{(L\tilde{x}^{(k-1)} + g^{(k-1)}(t+r)) - (L\tilde{x}^{(k-1)} + g^{(k-1)})}{r} = \frac{\partial}{\partial t} g^{(k-1)}(t) = g^{(k)}(t) \tag{4.41}$$

instead of the term $f_t^{(k)} = Lf_t^{(k-1)} + g^{(k)}$ from eq. (4.36)*. This would break the recurrence relation eq.* (4.37)*, but the new term would still allow us to correct for the external forcing term $g(t)$ with the recurrence relation*

$$z^{(k+1)} = x^{(k+1)} - \tilde{f}^{(k)}(t, x) = Lx^{(k)}. \tag{4.42}$$

This recurrence relation (4.42) *could be used to build a Krylov-like operator similarly to the approach presented in Theorem 4.5. The key difference is that the property*

$$z^{(k)} = L^{k-1} z^{(1)}, \quad k = 1 \dots K \tag{4.43}$$

which we had in Theorem 4.7 is lost. Therefore, we would need to adapt Theorem 4.5 in such a way that the matrices X and Y do not necessarily have to be shifted versions of one another. This increases the computational complexity, since we cannot expect to find (all but one of) the columns of Y in the span of X anymore.

To adequately test the external-force-correction mechanism of Theorem 4.7, we need an integration scheme, which uses the Jacobian approximation and is suitable for nonlinear systems. In Chapter 5, we will present such methods and test them in conjunction with the external-force-correction mechanism from this section.

4.4 Nonlinear Eigendynamics

Since the Krylov Jacobian approximation is based upon an observation from linear systems and has been designed with systems of linear eigendynamics in mind, it is natural that we encounter problems when applying it to systems with nonlinear eigendynamics.

We identified two main types of problems, which we will discuss in the following.

4.4.1 Effect of small perturbations in the derivatives on the Krylov Operator

For general systems with large nonlinear parts in the eigendynamics, we cannot make any guarantees about the quality of the Krylov Operator as an approximation to the Jacobian. This is because if the derivatives $x^{(k)}$ are dominated by the nonlinear part of the system, they do not hold enough information about the eigendynamics to compute a good approximation of the Jacobian. For systems where the linear part of the eigendynamics is dominant on the other hand, we can expect the Krylov Operator to approximately capture the linear eigendynamics of the system.

This is quantified in the following Theorem for autonomous systems.

Theorem 4.9 (Krylov Operator on Nonlinear Autonomous Systems) *Consider the nonlinear autonomous system of the form*

$$x^{(1)} = f(x) = Lx + r(x). \tag{4.44}$$

Let $L \in \mathbb{R}^{d\times d}$ be arbitrary and $r : \mathbb{R}\times\mathbb{R}^d \to \mathbb{R}^d$ be K-times continuously differentiable. For $k \in \{1,\dots,K\}$, let $r^{(k)}(x)$ denote the nonlinear remainder of the k-th time derivative of f, i.e.

$$f^{(k)}(t,x) = L^k x + r^{(k)}(x). \tag{4.45}$$

Let

$$\begin{aligned} X &= \begin{pmatrix} x^{(1)} & \dots x^{(K-1)} \end{pmatrix}, \\ Y &= \begin{pmatrix} x^{(2)} \dots x^{(K)} \end{pmatrix}, \\ R &= \begin{pmatrix} r^{(2)} \dots r^{(K)} \end{pmatrix}. \end{aligned} \tag{4.46}$$

Then the Krylov Operator $\hat{B}$ fulfills

$$\left\|\hat{B}X - P_{\mathcal{K}}LX\right\| \leq C\left\|R\right\|. \tag{4.47}$$

The constant C depends on the used norm and the system dimension d. For $\|\cdot\| = \|\cdot\|_2$ the claim holds with $C = 1$.

Proof. *By Definition 4.1 of the Krylov Operator we have $\hat{B}X = P_{\mathcal{K}}Y$. By definition of the nonlinear remainder terms $r^{(k)}$, we have $Y = LX + R$. Then we can write*

$$\hat{B}X - P_{\mathcal{K}}LX = P_{\mathcal{K}}Y - P_{\mathcal{K}}LX = P_{\mathcal{K}}(LX + R) - P_{\mathcal{K}}LX = P_{\mathcal{K}}R \tag{4.48}$$

It follows

$$\left\|\hat{B}X - P_{\mathcal{K}}LX\right\| = \left\|P_{\mathcal{K}}R\right\| \leq C\left\|R\right\|. \tag{4.49}$$

Where $C = \|P_{\mathcal{K}}\|$ does not depend on $\epsilon > 0$. If we use the Euclidean norm $\|\cdot\| = \|\cdot\|_2$, we get $C = 1$ because $P_{\mathcal{K}}$ is an orthogonal Projection. (see lemma 8.3 for a proof) □

Remark 4.10 *While we found a bound on the error $\left\|\hat{B}x^{(k)} - P_{\mathcal{K}}x^{(k)}\right\|$ for all $k = 1\dots K-1$ this does not translate into a bound on the operator norm $\left\|\hat{B} - P_{\mathcal{K}}L\right\|$ in terms of $\|R\|$ because the columns of X can be almost linearly dependent. This is illustrated by the following example.*

Test System 5 (Almost linear system)

$$x^{(1)} = -x + \begin{pmatrix} 0 \\ \frac{\epsilon}{2}(x)_1^2 \end{pmatrix}, \quad x(0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \tag{4.50}$$

where $(x)_1$ denotes the first component of x and $\epsilon > 0$ is a small parameter.

Example 4.11 (Almost linearly dependent columns in almost linear system) *Consider the Test System 5. We can calculate the derivatives*

$$\begin{aligned} x^{(1)} &= -x + \begin{pmatrix} 0 \\ \frac{\epsilon}{2}(x)_1^2 \end{pmatrix} = \begin{pmatrix} -1 \\ \frac{\epsilon}{2} \end{pmatrix}, \\ x^{(2)} &= -x^{(1)} + \begin{pmatrix} 0 \\ \epsilon(x)_1(x^{(1)})_1 \end{pmatrix}, = \begin{pmatrix} 1 \\ -\frac{3}{2}\epsilon \end{pmatrix}, \\ x^{(3)} &= -x^{(2)} + \begin{pmatrix} 0 \\ \epsilon((x)_1(x^{(2)})_1 + (x^{(1)})_1^2) \end{pmatrix} = \begin{pmatrix} -1 \\ \frac{7}{2}\epsilon \end{pmatrix}. \end{aligned}$$

This system can be expressed in the form of eq. (4.44) *with the matrices*

$$L = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \quad \text{and} \quad R = \begin{pmatrix} 0 & 0 & 0 \\ \frac{\epsilon}{2}(x)_1^2 & \epsilon(x)_1(x^{(1)})_1 & \epsilon((x)_1(x^{(2)})_1 + (x^{(1)})_1^2) \end{pmatrix}. \tag{4.51}$$

We get

$$X = \begin{pmatrix} -1 & 1 \\ \frac{1}{2}\epsilon & -\frac{3}{2}\epsilon \end{pmatrix} \tag{4.52}$$

$$X^{-1} = \begin{pmatrix} -\frac{3}{2} & -\frac{1}{\epsilon} \\ -\frac{1}{2} & -\frac{1}{\epsilon} \end{pmatrix} \tag{4.53}$$

$$Y = \begin{pmatrix} 1 & -1 \\ -\frac{3}{2}\epsilon & \frac{7}{2}\epsilon \end{pmatrix}. \tag{4.54}$$

Because X *has full rank, we get* $P_{\mathcal{K}} = I_2$ *for* $\mathcal{K} = span(X)$ *and* $\hat{B} = YX^{-1}$. *Thus,*

$$\hat{B} - P_{\mathcal{K}}L = \begin{pmatrix} 1 & -1 \\ -\frac{3}{2}\epsilon & \frac{7}{2}\epsilon \end{pmatrix} \begin{pmatrix} -\frac{3}{2} & -\frac{1}{\epsilon} \\ -\frac{1}{2} & -\frac{1}{\epsilon} \end{pmatrix} - L \tag{4.55}$$

$$= \begin{pmatrix} -1 & 0 \\ \epsilon & -2 \end{pmatrix} - \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix} \tag{4.56}$$

$$= \begin{pmatrix} 0 & 0 \\ \epsilon & -1 \end{pmatrix} \tag{4.57}$$

We see that $\left\|\hat{B} - P_{\mathcal{K}}L\right\|_1 \geq 1$ *for arbitrary small* $\epsilon > 0$. *Through the equivalence of norms for finite dimensional vector spaces, we see that we cannot bound the operator norm* $\left\|\hat{B} - P_{\mathcal{K}}L\right\|$ *in terms of* $\|R\|$, *which is proportional to* ϵ.

Example 4.11 shows that even small perturbations of a linear system can lead to big differences in the approximated operator. Through Theorem 4.9 however, we know that the the products $\hat{B}x^{(k)}$ only suffer from little errors under small disturbances for $k = 1 \ldots K - 1$. Since depending on the later use of $\hat{B}$ in an integration scheme, we mostly only evaluate these products, we can expect such integration schemes to still perform well. The following test verifies this.

Numerical Test 5 (Linear Krylov on Almost Linear System) *We test the Linear Krylov method on the Almost linear system studied in example 4.11 with* $\epsilon = 0.1$. *We use a Runge-Kutta-4 scheme with* $h = 0.001$ *as reference and compare against a Runge-Kutta-4 scheme with* $h = 0.1$. *In Figure 4.3, we see that the Linear Krylov Method significantly outperforms the RK4 scheme of the same stepsize. Hence we see that regardless of the large error* $\left\|\hat{B} - P_{\mathcal{K}}L\right\| = 1$, *the accuracy of* $\left\|\hat{B}X - P_{\mathcal{K}}LX\right\| \leq C\,\|R\|$, *which is guaranteed by Theorem 4.9, still leads to accurate time marching methods.*

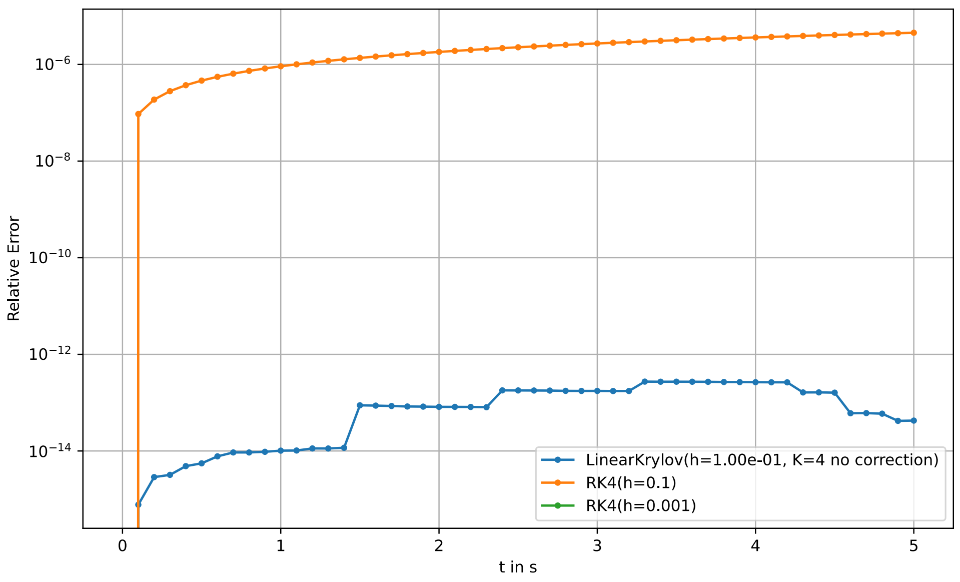


Figure 4.3 Linear Krylov on almost linear system

4.4.2 Discontinuities in higher derivatives

Since we use multiple derivatives for the calculation of an integration step, it is important to consider the smoothness of the higher derivatives $x^{(k)}$ of the solution curve x.

Example 4.12 *In Test System 3 [32],*

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}^{(1)} = \begin{pmatrix} \dot{x} \\ -\frac{4}{3} r\, sign(x)\, |x|^{\frac{3}{2}} \end{pmatrix}, \quad \begin{pmatrix} x(0) \\ \dot{x}(0) \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

the higher derivatives are

$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}^{(k)} = \begin{pmatrix} x^{(k)} \\ x^{(k+1)} \end{pmatrix} \quad for \quad k = 0 \dots 4,$$

where

$$\begin{aligned} x^{(0)} &= x \\ x^{(1)} &= \dot{x} \\ x^{(2)} &= -\frac{4}{3} r \cdot sign(x)\, |x|^{\frac{3}{2}} \\ x^{(3)} &= -2r\, |x|^{\frac{1}{2}}\, \dot{x} \\ x^{(4)} &= -sign(x) r (|x|^{-\frac{1}{2}}\, (\dot{x})^2 + \frac{8}{3} r\, |x|^2) \\ x^{(5)} &= \frac{r}{2}\, |x|^{-\frac{3}{2}} + 8r^2\, |x|\, \dot{x}. \end{aligned}$$

Note that $x^{(4)}$ and $x^{(5)}$ have a singularity at $x = 0$. This means, multiderivative methods using $K \geq 3$ derivatives are not defined at that point. We can still apply them as long as there is no value of x exactly equal to 0. But the produced approximations will suffer from large errors whenever an integration step close to $x = 0$ is evaluated. We see in Figure 4.4, where we tested the system from eq. (4.58) with $r = 1$, that both Taylor methods and methods using the Krylov Jacobian approximation incur spikes in their respective errors when passing through the singularity at times $t \approx 1.4$, $t \approx 4.3$ and $t \approx 7.2$. In Figure 3.2, we can observe the same problem at the singularities for multiderivative Runge-Kutta schemes.

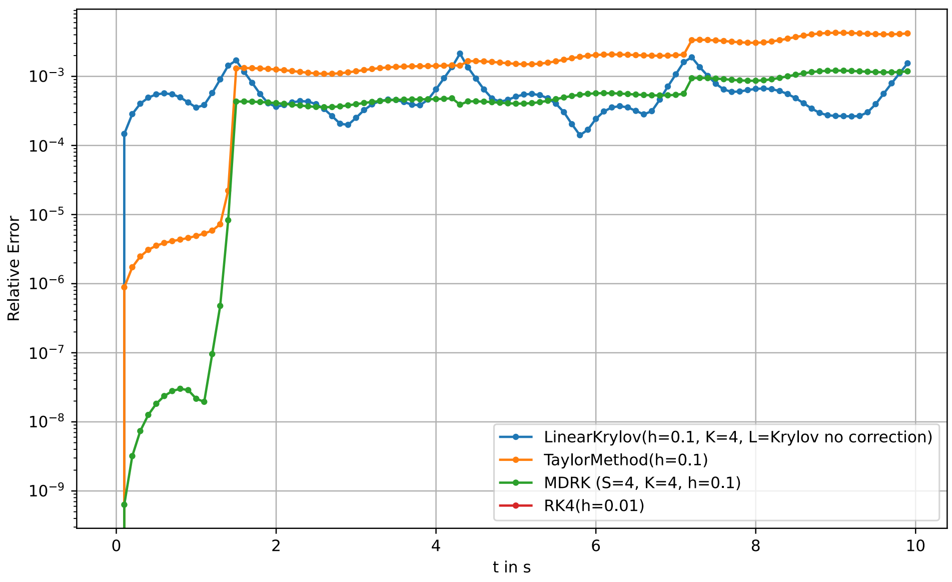


Figure 4.4 Multiderivative methods passing through singularity of nonlinear system

The issue lies within the use of multiple derivatives on systems, whose function f is not sufficiently often continuously differentiable. This issue has to be solved on the modelling side. A possible solution would be to linearize the function f in a small region around $x = 0$ to avoid the singularity.

In Chapter 5, we will introduce methods suited for nonlinear systems and conduct further numerical experiments. Without addressing the issue of discontinuities in the higher derivatives from the modelling side, the problem will persist.

4.5 Computational complexity

Our aim is to use the Krylov Jacobian approximation to profit from the improved stability properties of methods involving the Jacobian without incurring the computational cost of order $O(d^2)$, which comes with instantiating the full Jacobian. Thus, it is important to analyze the computational complexity of Algorithm 5, and the call of Algorithm 3

We start with the computational complexity of Algorithm 3.

```
1  def build_operator(self, Z):
2      """creates a linear operator, which maps the columns of
         Z like z_1 -> z_2 -> z_3 -> ... -> z_K"""
3      # Assumes Z of shape (d x K)
4      # Returns Q, B such that (Q@B@Q.T)z_k = z_{k+1}
5      K = Z.shape[1]  # Number of time-derivatives
6      Kx = min(Z.shape[0], K-1) # dimension of Krylov Space
7      Q, R = np.linalg.qr(Z) # QR decomposition
8      Rx = R[:Kx, :Kx]
9      Ry = R[:Kx, 1:Kx+1]
10     Q = Q[:,:Kx]
11     b = np.abs(np.abs(np.diag(Rx)) > 1e-10)
12     if not np.any(b):
13         return np.zeros((Z.shape[0],1)), -np.ones((1,1))
14     Rx = Rx[b,:][:,b]
15     Ry = Ry[b,:][:,b]
16     Q = Q[:,b]
17     B = Ry @ np.linalg.inv(Rx)
18     return Q, B
```

Theorem 4.13 (Compuational Complexity of Krylov Operator) *The Krylov Operator $\hat{B}$ can be computed via Algorithm 3 with no more than $5dK^2 \in O(dK^2)$ floating point operations, where d is the system dimension and K is the number of time-derivatives.*

Proof. *We can assume $K \leq d$ because a larger Krylov Space than system dimension does not make sense. Reviewing the code, we find that the only lines containing floating point operations are the QR-Decomposition of Z in line 7 and the matrix-matrix multiplication and matrix inversion in line 17. According to Bornemann [3, Ch. 3, p.35], the QR-Decomposition of a $d \times K$ matrix can be computed via the Householder algorithm in $4dK^2 - \frac{4}{3}K^3$ operations. It is well known that the inversion of the upper triangular matrix R_X can be done in less than K^3 operations as well as the matrix-matrix multiplication $R_Y \cdot R_X^{-1}$, which can be done in K^3 operations. In total, we have*

$$4dK^2 - \frac{4}{3}K^3 + K^3 + K^3 < 4dK^2 + K^3 \leq 5dK^2 \in O(dK^2) \tag{4.58}$$

floating point operations. All other operations are merely storage manipulations of at most $d \times K$ or $K \times K$ matrices. These are asymptotically $O(d \cdot K)$ storage operations. □

Next, we analyze the computational complexity of Algorithm 5. This algorithm is used to compute the matrix Z, from which the Krylov Operator is constructed via Algorithm 3. The main work lies in the evaluation of the functions $f^{(k)}(t, x)$.

```
def krylov_operator(self, t, x, fk):
    eps = 1e-8 # finite difference
    k_range = range(1, self.K+1)
    Z = np.array([fk(t, x, k) for k in k_range]).T
    if self.correct_excitement:
```

```
6             H = np.array([(fk(t+eps, x, k)-fk(t-eps, x,
                k))/(2*eps) for k in range(self.K)]).T
7             Z = Z - H
8         Q,B = self.build_operator(Z)
9         return Q, B
```

Lemma 4.14 (Computational Complexity of Derivatives)
Generating the matrix Z for external force-correction via Algorithm 5 (excluding the call to Algorithm 3) can be done in at most $3(C+d)K \in O(dK)$ floating point operations, where C is the maximum number of floating point operations required to evaluate the function $f^{(k)}(t,x)$ for any $k \in \{0,\dots,K\}$.

Proof. *As explained in section 3.2, we assume that the cost of the function evaluations is linear in the system dimension i.e. $C \in O(d)$. Hence, we find*

$$3(C+d)K \in O(dK). \tag{4.59}$$

It remains to prove that the left side of eq. (4.59) is an upper bound for the floating point operations of Algorithm 5 excluding the call to Algorithm 3. Reviewing the code, we observe that the main work is carried out in the lines 4 and 6 through the function evaluations $f^{(k)}(t,x)$. Counting the indices $k = 1\dots K$ in line 4 and $k = 0\dots K-1$ in line 6, we get $3K$ function evaluations. These cost $3CK$ floating point operations. Calculating the finite difference in line 6 requires another $2d$ operations for subtraction and division. Combined with the d operations for the subtraction in line 7, we get

$$(3C+3d)K \tag{4.60}$$

floating point operations. □

Combining these results, we can summarize the asymptotic runtime of constructing the Krylov Jacobian Approximation.

Corollary 4.15 (Computational Complexity of Krylov Jacobian Approximation)
Building the Krylov Jacobian Approximation using Algorithm 5 and Algorithm 3 requires at most

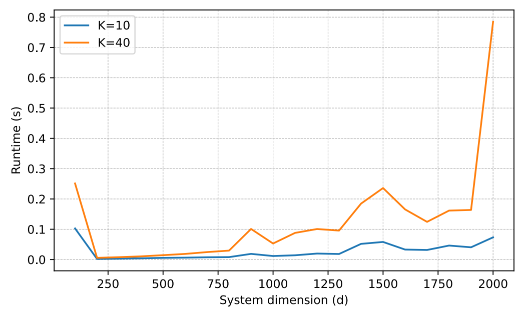
$$5dK^2 + 3(C+d) \in O(dK^2) + O(dK) = O(dK^2) \tag{4.61}$$

floating point operations, where C is defined as in Lemma 4.14.

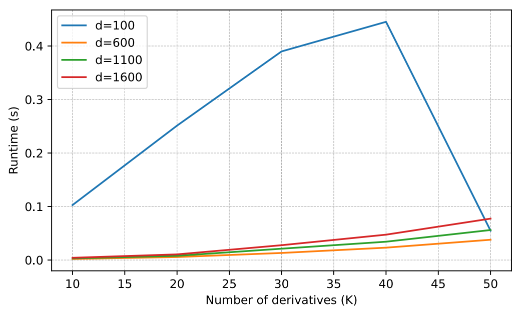
Numerical Test 6 (Compute time to build Krylov operator) *We tested the runtime of Algorithm 3 and Algorithm 5 on randomly generated instances the linear Test System 2 with different dimensions d for Krylov Operators with different dimensions K. The test was conducted on a MacBook Pro with an M1 Pro CPU. The compute times are averaged over 20 runs.*

In Figure 4.5, we see that the calculation of the Krylov operator through Algorithm 3 takes up only a fraction of $\approx$ 1% of the runtime of Algorithm 5. Note that Figure 4.5c and Figure 4.5d use significantly increased ranges for the values of d and K Figure 4.5a and Figure 4.5b. The overwhelming majority of the work of Algorithm 5 goes into calculating the time-derivatives of the solution. This is in part because the parallel evaluation of the time-derivatives as outlined in section 3.2 has not been implemented for this test case. Thus, the calculation of the time-derivatives was done by dense matrix-vector multiplication of $L^k x$, which is computationally expensive for large d. This is to be resolved in future work with an implementation of the parallel evaluation of time-derivatives and can be drastically reduced for sparse systems.

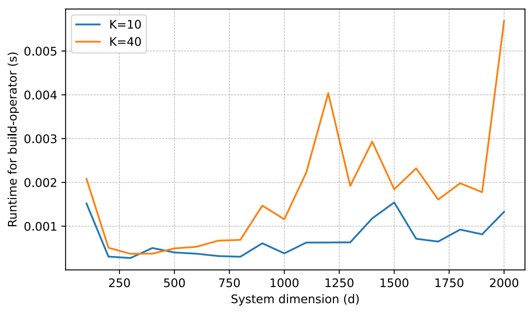
Furthermore, we see that the sole execution of Algorithm 3 is computationally feasible for values as large as $d = 2 \cdot 10^3$ and $K = 50$. We chose such large values for K just for the purpose of this demonstration. For real application, it would be highly unfeasible to compute this many time-derivatives of the solution. Realistically, one would choose $K \approx 4$ or slightly larger for exceptionally smooth systems [31].



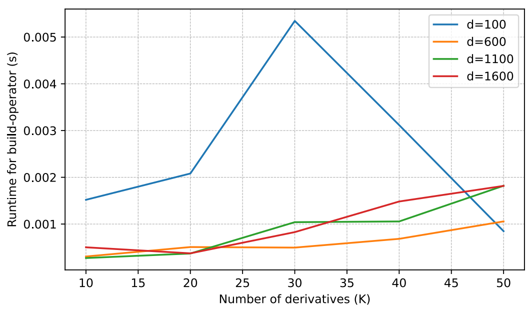
(a) Runtime of Algorithm 5 by d



(b) Runtime of Algorithm 5 by K



(c) Runtime of Algorithm 3 by d



(d) Runtime of Algorithm 3 by K

Figure 4.5 Compute time to construct Krylov Operator

The expected trends of linear growth in the system dimension d and quadratic growth in the number K of derivatives are barely visible due to noise in the measured runtimes. This might be related to varying thermal throttling of the compute hardware. We can still confidently tell from the scale of the measured times, that the calculation of the Krylov operator from given time-derivatives by Algorithm 3 will not be the limiting factor to the use of the Krylov Jacobian approximation in terms of calculation time.

5 Jacobian Methods

With the ability to cheaply approximate the Jacobian of a system, we can use methods, which use the Jacobian to create a stable integration scheme. Two of those schemes are linearly implicit integrators and exponential Rosenbrock integrators. Both of these schemes theoretically achieve A-stability (i.e. they produce stable results on linear systems with arbitrary eigenvalues in the left half of the complex plane, see definition 1.8) if the exact Jacobian is used. Since the Krylov Jacobian Approximation is of limited rank, it cannot capture every eigendynamic of a high dimensional system and thus, A-stability will only be achieved if the system contains only few very stiff eigenvalues. We will explore the stability of these methods in section 5.3 and section 5.4.

5.1 Linearly Implicit Methods

Starting from the Backwards Euler formula [7, Ch.2]

$$x_{n+1} = x_n + hf(t, x_{n+1}),$$

which can only be solved implicitly in the general case, one can derive an explicit formula for the linear case $f(t,x) = Lx$ [10, sec 10.4].

$$\begin{aligned} x_{n+1} &= x_n + hLx_{n+1} \\ \implies (I - hL)x_{n+1} &= x_n \\ \implies x_{n+1} &= (I - hL)^{-1}x_n \end{aligned}$$

This lets us explicitly calculate the propagation of the implicit Euler method. We observe that multiplying the right-hand side of a linear system of ODEs $x^{(1)} = Lx$ with the matrix $(I - hL)^{-1}$ and then solving it with the explicit Euler method, yields the same result as solving the system with the implicit Euler method.

$$\begin{aligned} x_{n+1} &= x_n + h(I - hL)^{-1}Lx_n \\ &= (I - hL)(I - hL)^{-1}x_n + h(I - hL)^{-1}Lx_n \\ &= (I - hL + hL)(I - hL)^{-1}x_n \\ &= (I - hL)^{-1}x_n \end{aligned} \tag{5.1}$$

Bassenne, Fu, and Mani [2] use this observation to develop the so-called "time accurate and stabilized explicit (TASE)" preconditioners, which can be used to stabilize explicit methods and make them applicable to stiff systems. The authors define the preconditioners as follows:

$$T_L^{(p)}(\alpha, h) = \begin{cases} (1 - \alpha hL)^{-1}, & \text{if } p = 1 \\ \frac{2^{p-1}T_L^{(p-1)}(\alpha, \frac{h}{2}) - T_L^{(p-1)}(\alpha, h)}{2^{p-1}-1}, & \text{if } p \geq 2 \end{cases} \tag{5.2}$$

For $\alpha = 1$, this is exactly the implicit Euler preconditioner from above in eq. (5.1). The higher order TASE operators are designed using Richardson's Extrapolation [22]. Preconditioning a system of ODEs can turn explicit methods into A-stable (see definition 1.8) integration schemes.

The TASE operator $T_L^{(p)}$ converges to the identity matrix I as $O(h^p)$. Thus, an explicit method of order $p' \leq p$ will retain its order of accuracy. That is, its result will converge to the analytical solution of the system $x^{(1)} = f(t,x)$ with order p if applied to the preconditioned system

$$x^{(1)} = T_{L(t,x)}^{(p)} f(t, x), \quad x(0) = x_0, \tag{5.3}$$

	TASE order			
	$p = 1$	$p = 2$	$p = 3$	$p = 4$
RK1 ($C = 2.00$)	**0.50**			
RK2 ($C = 2.00$)	0.50	**1.50**		
RK3 ($C = 2.50$)	0.40	1.20	**2.80**	
RK4 ($C = 2.79$)	0.36	1.08	2.51	**5.38**

Table 5.1 Values of α_{min}

where $L(t, x)$ is an adequate approximation to the Jacobian $J_f(t, x) = \frac{\partial f}{\partial x} f(t, x)$. We will use the Krylov operator from chapter 4 in place of the Jacobian to avoid the computational cost of a full jacobian instantiation.

Bassenne, Fu, and Mani [2] prove, that applying an explicit method on a system preconditioned with the TASE operator of order p becomes A-stable if

$$\alpha \geq \alpha_{\text{min}} = \frac{2^p - 1}{C} \tag{5.4}$$

Where C is a constant depending on the stability region of the used explicit method. The authors recommend using $\alpha = \alpha_{\text{min}}$ as this choice leads to the most accurate amongst the A-stable methods. The values of α_{min} for popular Runge-Kutta methods are given in Table 5.1.

If given the Jacobian $J_f(x, t)$, we can apply any explicit scheme together with the TASE operator of the corresponding order of accuracy and achieve a cheap, stable scheme. We refrain from promising an A-stable scheme, because achieving this property requires us to use an approximation of L, that contains all the eigenvalues of J_f, which can't be guaranteed with a low rank approximation. We will explore this further in section 5.3 and section 5.4.

We use the Krylov Jacobian Approximation with correction for externeal excitement from chapter 4 to approximate the Jacobian $\hat{B} \approx J_f(t, x)$ of the system and apply two versions of the TASE method:

1. We apply the Euler Method on a system preconditioned with the first order TASE operator $T_{\hat{B}}^{(1)}$ with parameter $\alpha = 1.0$. Here we use stepsize $h = 0.2$ and $h = 0.01$.

2. We apply the Runge-Kutta-4 scheme with the Butcher tableau from Equation (3.14) on a system preconditioned with the fourth order Tase operator $T_{\hat{B}}^{(4)}$ and parameter $\alpha = 5.4$. Here we use stepsize $h = 0.01$.

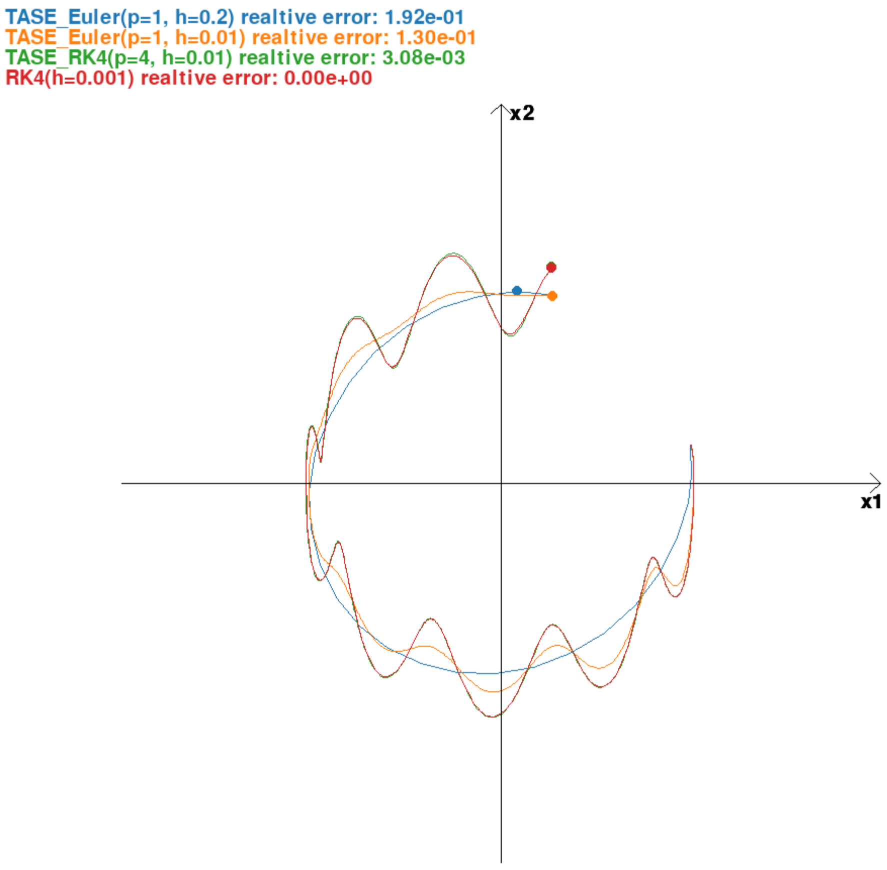
We denote these methods in the form **TASE[p]-[Explicit]** where the order p of the Tase operator and the name of the used explicit method is inserted.

Test System 6 (Oscillating System) *With the parameters ω, α, β we define the system*

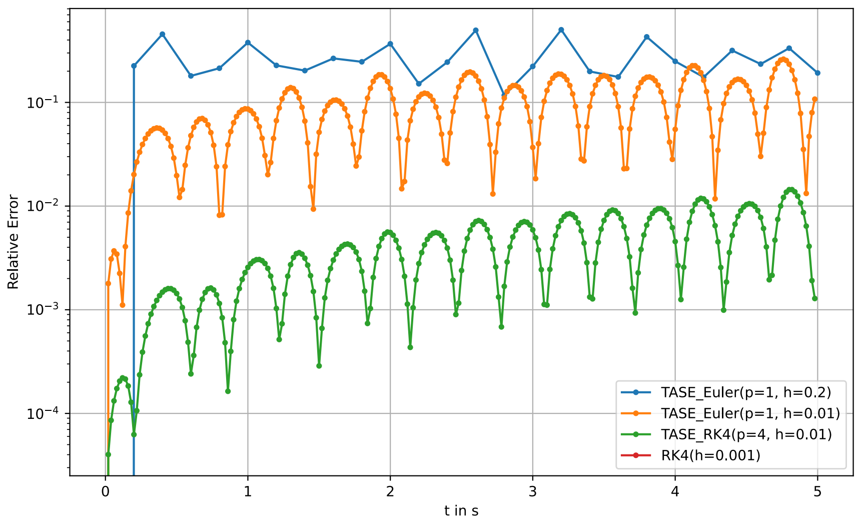
$$\begin{pmatrix} x \\ \dot{x} \end{pmatrix}^{(1)} = \begin{pmatrix} 0 & 1 \\ -\omega & 0 \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \end{pmatrix} \cdot \beta \cos(\alpha t), \quad \begin{pmatrix} x(x) \\ \dot{x}(0) \end{pmatrix} = x_0 \tag{5.5}$$

Numerical Test 7 (TASE methods on oscillating system) *We test these methods on an Oscillating System with the parameters $\omega = 100, \alpha = 1, \beta = 100$ and $x_0 = (1, 0.3)^T$.*

*We can see from the trace in Figure 5.1a, that only **TASE4-RK4** with $h = 0.01$ follows the fine oscillations well. Both instances of **TASE1-Euler** damp the fine oscillations quickly as we would expect from low order implicit methods. It is remarkable that even **TASE1-Euler** with the large stepsize $h = 0.2$ is able to follow the general dynamic while not being destabilized by the high frequency oscillations. We chose the moderately stiff system with parameter $\omega = 100$ to visualize the damping of the oscillations and to be able to use the Runge-Kutta-4 method with a reasonable stepsize $h = 0.001$ as a reference solution.*



(a) Trace



(b) Relative Errors

Figure 5.1 TASE methods on oscillating system ($\omega = 100, \alpha = 1, \beta = 100$)

We could however use parameters up to $\omega = \beta = 10^{16}$ and still obtain stable results before seeing significant deterioration in the smoothness of the solutions. These likely stem from arithmetic inaccuracies at very high values of ω and β.

Numerical Test 8 (TASE methods on very stiff oscillating system) *We test the methods **TASE1-Euler** and **TASE4-RK4** with stepsizes $h = 0.1$ and $h = 0.001$ on a very stiff oscillating system (see eq. (5.5)) with parameters $\omega = 10^{16}, \alpha = 1, \beta = 10^{16}$. The Jacobian has the eigenvalues $\lambda_{1/2} = \pm 10^8 i$. We use **TASE1-Euler** with $h = 0.001$ as reference solution.*

*We see in Figure 5.2 that all tested methods remain stable. While the first order method **TASE1-Euler** follows the system's equilibrium state well, the fourth order method **TASE4-RK4** significantly deviates from this equilibrium.*

Reducing its stepsize reduces the deviation, but it remains offset significantly. This offset from an equilibrium solution of a very stiff spring is noticeable throughout many tests of the higher order TASE methods. The first order TASE method never showed this deficiency.

Pagano [23] proposes to use TASE operators to obtain stabilized parallellizable explicit peer (SEPP) methods, which achieve similar stability properties to the combination of TASE methods with Runge-Kutta methods while avoiding the sequential function evaluation necessary with Runge-Kutta methods. These methods seem to be a promising topic for further investigation since they can do more computational work in parallel an thus, potentially reduce the duration of the calculations.

5.2 Exponential Integrators

Exponential Integrators make use of the analytic solution to linear systems of ODEs like Test System 2, which is well known to be given by the matrix exponential.

$$x(t) = e^{tL} x(t_0) \tag{5.6}$$

and correct for the nonlinearity via the variation of constants formula [21]. The formula states that the solution to

$$x^{(1)}(t) = f(t, x(t)) = Lx(t) + g(t, x(t)), \quad x(0) = x_0 \tag{5.7}$$

is given by the integral equation [9]

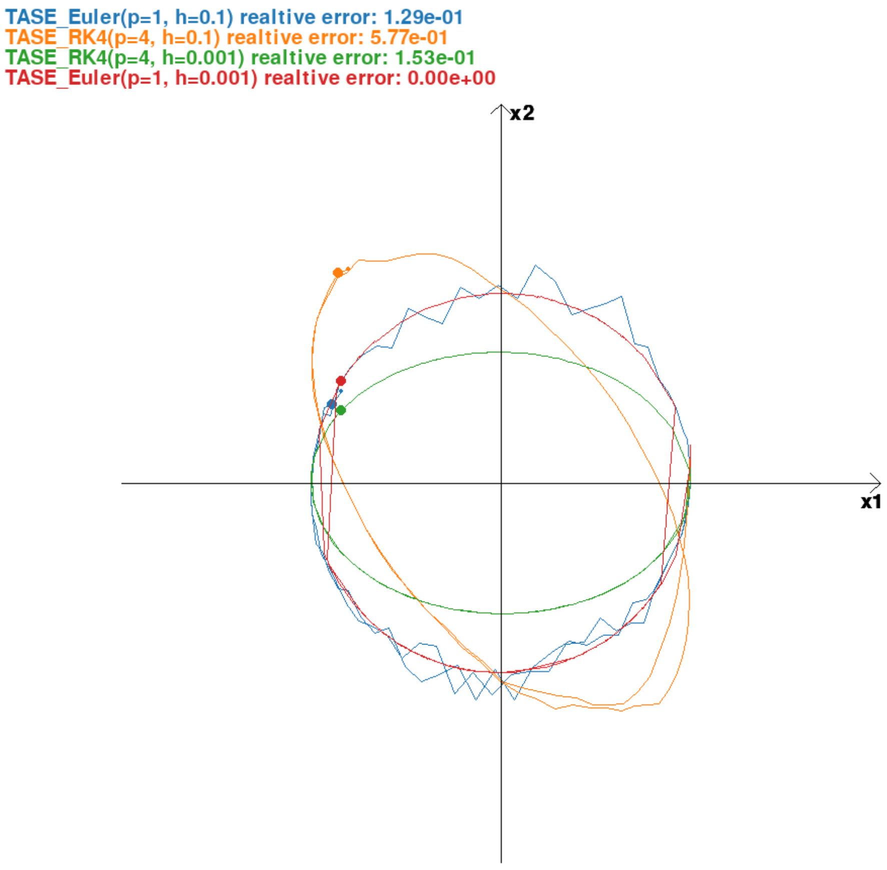
$$x(t) = e^{tL} x(0) + \int_0^t e^{(t-s)L} g(s, x(s)) ds. \tag{5.8}$$

This has the obvious advantage, that when the system is dominated by the linear part, g is small and hence the produced solution is dominated by the first matrix exponential term, which is the exact solution to the linear system. Furthermore, by computing the exact analytical solution of linear systems, the method is stable on a linear system if and only if the system is stable, thus, achieving F-stability (See definition 1.8). We use the Krylov Jacobian Approximation from chapter 4 and thus, must expect slight inaccuracies. Nevertheless, these methods exhibit excellent stability properties especially when applied to systems of linear eigendynamics.

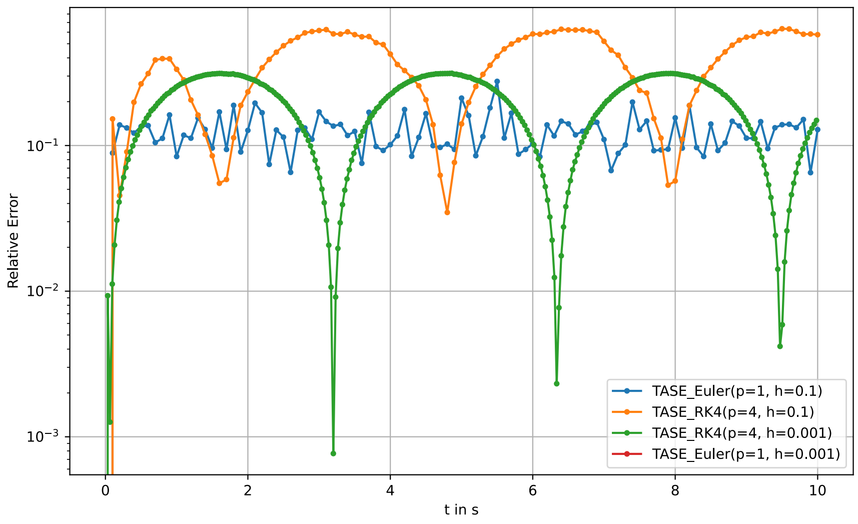
5.2.1 Exponential Rosenbrock Methods

Rosenbrock methods generally describe the combination of using the system's Jacobian and multiple stages per step [5]. They can be found in the span of axes (1.) and (3.) of Figure 2.1.

The integral term in the right hand side of eq. (5.8) is approximated by a quadrature rule. To achieve higher order of accuracy in the quadrature term, one has to evaluate the integrand $e^{(t-s)L} g(s, x(s))$ multiple times. This leads to the use of multiple function evaluations per step, just like in Runge-Kutta methods. Such methods, which use multiple stages in exponential integrators are called **Exponential Rosenbrock Methods** [15]. These make use of the following terms:



(a) Trace



(b) Relative Errors

Figure 5.2 TASE methods on very stiff oscillating System ($\omega = 10^{16}, \alpha = 1, \beta = 10^{16}$)

- The functions $\varphi_k(z)$, which are defined as

$$\varphi_k(z) = \int_0^1 e^{(1-s)z} \frac{s^{k-1}}{(k-1)!} ds, \quad k \geq 1, \tag{5.9}$$

  and satisfy the recurrence relations

$$\varphi_k(z) = \begin{cases} e^z, & k = 0 \\ z^{-1}\left(\varphi_{k-1}(z) - \varphi_{k-1}(0)\right), & k \geq 1 \end{cases} \tag{5.10}$$

  The method parameters a_{ij} and b_i are typically chosen as linear combinations of the functions φ_k and the related functions $\varphi_k(c_i \cdot)$.

- The residuals D_{nj} are given by

$$D_{nj} = g(t_n + c_j h, X_{n,j}) - g(t_n, x_n) \tag{5.11}$$

  where $g(t,x) = f(t,x) - Lx$ is the nonlinear part of the system.

- The term

$$v_n = \frac{\partial}{\partial t} f(t_n, x_n) \tag{5.12}$$

  is used to account for the possible non-autonomous nature of the system.

Exponential Rosenbrock methods can then be written in the form

$$X_{n,i} = x_n + hc_i\varphi_1(c_ihL)f(t_n,x_n) + (hc_i)^2\varphi_2(c_ihL)v_n + h\sum_{j=2}^{i-1} a_{ij}(hL)D_{nj}, \quad i = 1,\dots,s \tag{5.13}$$

$$x_{n+1} = x_n + h\varphi_1(hL)f(t_n,x_n) + h^2\varphi_2(hL)v_n + h\sum_{i=2}^{s} b_i(hL)D_{ni} \tag{5.14}$$

where the stages $X_{n,i}$ approximate the solution at the times $t_{n,i} = t_n + c_ih$ corresponding to the nodes c_i of the method.

Hochbruck, Ostermann, and Schweitzer [15] present the two-stage method **ExpRb32** and the three-stage method **ExpRb43**, which are of order $p = 3$ and $p = 4$ respectively and come with an error estimate of order $p-1$. The error estimate is obtained by using the coefficents $\hat{b}_i$ instead of b_i. The parameters a_{ij} and b_i, $\hat{b}_i$ and c_i can be noted in a Butcher tableau just like for Runge-Kutta methods.

c	A
	b^T
	$\hat{b}^T$

Table 5.2 General Exponential Rosenbrock Method Butcher tableau

0		
1	φ_1	
	$\varphi_1 - 2\varphi_3$	$2\varphi_3$
	φ_1	

Table 5.3 Butcher tableau of **ExpRb32**

0			
$\frac{1}{2}$	$\frac{1}{2}\varphi_1(\frac{1}{2}\cdot)$		
1	0	φ_1	
	$\varphi_1 - 14\varphi_3 + 36\varphi_4$	$16\varphi_3 - 48\varphi_4$	$-2\varphi_3 + 12\varphi_4$
	$\varphi_1 - 14\varphi_3$	$16\varphi_3$	$-2\varphi_3$

Table 5.4 Butcher tableau of **ExpRb43**

It is not hard to see that for linear systems $x^{(1)} = Lx$, the functions g and the remainders $D_{n,i}$ are both zero. Thus, the increment (5.14) simpliefies to

$$x_{n+1} = x_n + h\varphi_1(hL)Lx_n + 0 = x_n + hL(hL)^{-1}(e^{hL} - I_d)x_n = e^{hL}x_n$$

This agrees with the analytic solution of the system and therefore emits A-stabiliy.
For general systems of ODEs of the form $x^{(1)} = f(t,x)$, the Linear part L is replaced by the Jacobian $J_f(t_n, x_n) = \frac{\partial f}{\partial x}(t_n, x_n)$ or a suitable approximation of it. We will use the Krylov operator $\hat{B}$ in the place of L. Its matrix exponential can be evaluated through the lown rank representation $e^{h\hat{B}} = e^{Q(hB)Q^T} = Qe^{hB}Q^T$ from eq. (4.18). Because $B \in \mathbb{R}^{(K-1)\times(K-1)}$ is a very small operator ($K \approx 4$), the evaluation of its matrix exponential does not present a significant compuational challenge[1].

5.3 Further Numerical Tests

In this section, we conduct numerical tests on different test systems to

5.3.1 Accuracy of Krylov Jacobian Methods

Numerical Test 9 (Accuracy of Jacobian methods on stiff oscillating system) *We compare the methods **ExpRb32**, **ExpRb43**, **TASE1-Euler**, **TASE4-RK4** with stepsize $h = 0.1$ on the stiff oscillating Test System 6 with parameters $\omega = 10^4, \alpha = 1, \beta = 10^4, x_0 = (1, 0.1)^T$. The Jacobian of the system is $J_f(t,x) = \begin{pmatrix} 0 & 1 \\ -10^4 & 0 \end{pmatrix}$ and has the eigenvalues $\lambda_{1/2} = \pm 100i$. The results in Figure 5.3 show that all tested methods remain stable even though the relatively large stepsize $h = 0.1$ was used for the Jacobian methods. The Runge-Kutta-4 scheme could only be run with the maximum stepsize $h = 0.027$ before becoming unstable at $h > 0.02785$.*

To conduct further tests on linear forced systems, we use the Prothero Robinson example system [24].

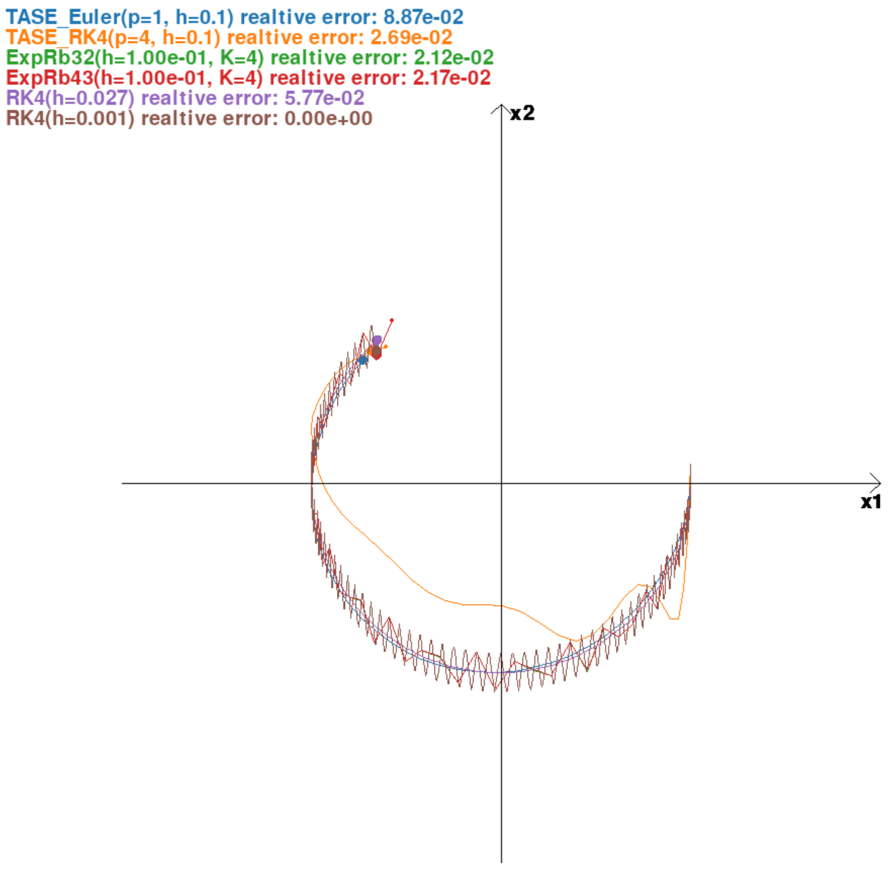
Test System 7 (Prothero Robinson)

$$x^{(1)}(t) = L(x - \varphi(t)) + \varphi^{(1)}(t), \quad x(0) = x_0 \tag{5.15}$$

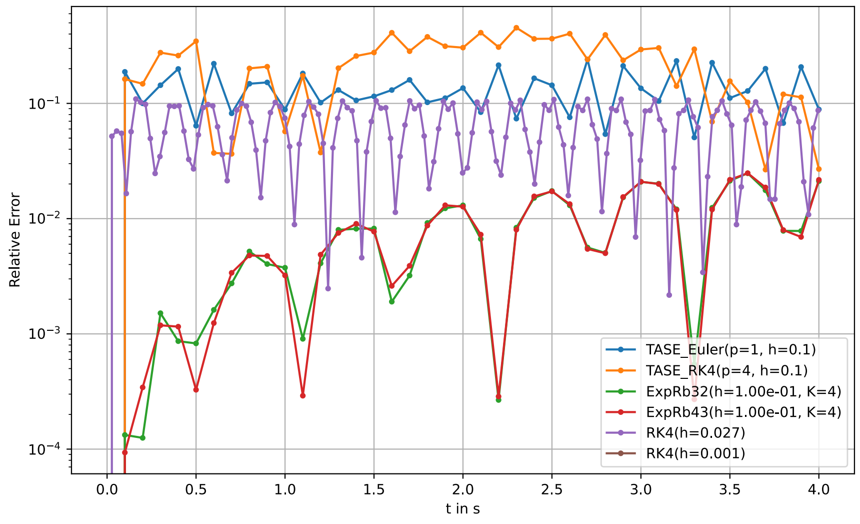
$L \in \mathbb{R}^{d\times d}$ describes an oscillatory behaviour. One can imagine a spring. $\varphi : T \longrightarrow \mathbb{R}^d$ shifts the equilibrium position of the spring.

Numerical Test 10 (Exact Jacobian vs. Krylov Jacobian) *We test the methods **TASE1-Euler** and **ExpRb32** on Test System 7 where the matrix $L \in \mathbb{R}^{4\times 4}$ has the eigenvalues $\lambda_{1/2} = -10 \pm 10i$ and $\lambda_{3/4} = -10 \pm 5500i$, $\varphi(t) = (sin(t), sin(2t), sin(3t), sin(4t))^T$ and $x_0 = 0$. with $h = 0.01$. In both cases we compare the version of the method using the Krylov Jacobian approximation to the version using the exact Jacobian L. We see in Figure 5.4, that using the exact Jacobian instead of the Krylov Jacobian approximation is slightly advantageous for **ExpRb32**, but shows almost no difference when paired with **TASE1-Euler**. The spike in relative error at $t = \pi$ is likely due to the analytic solution passing through 0 at this time.*

[1]Without using external excitement it will be $B \in \mathbb{R}^{K\times K}$.

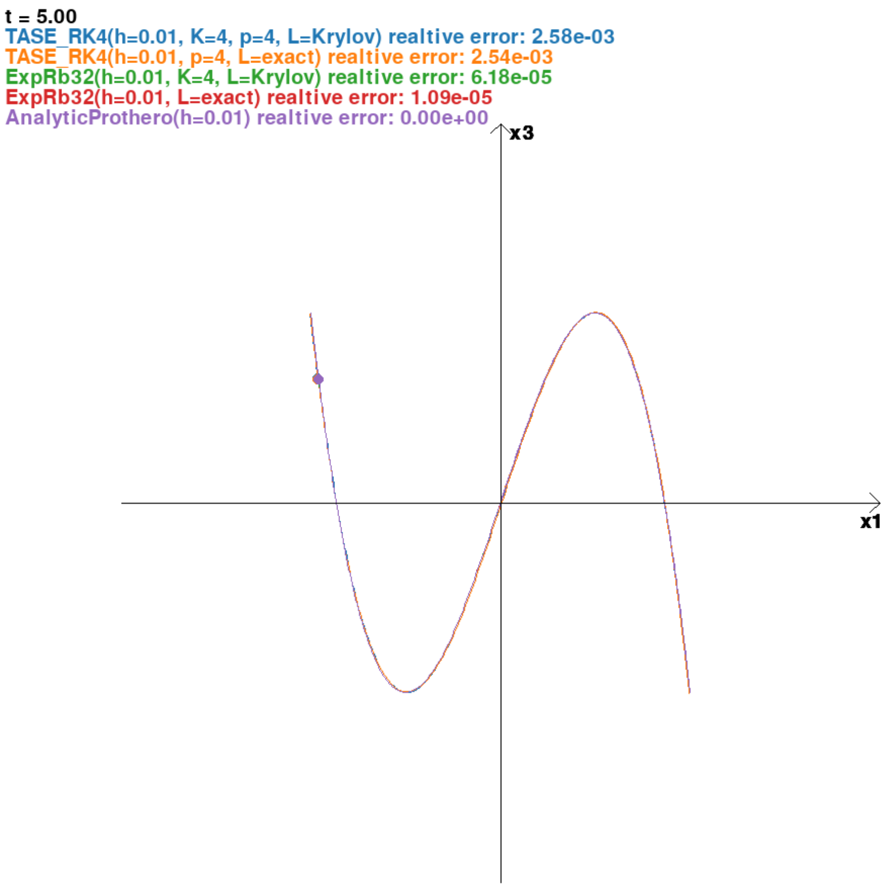


(a) Trace

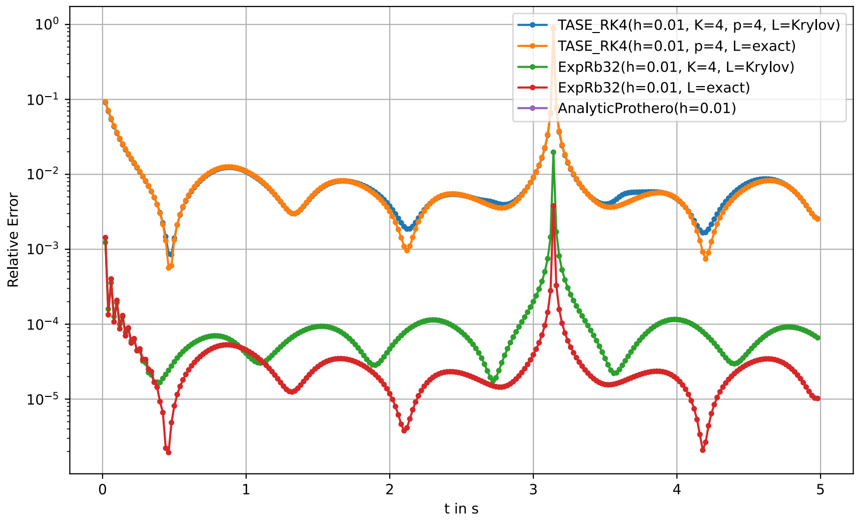


(b) Relative Errors

Figure 5.3 Krylov Jacobian methods on stiff oscillating system



(a) Trace



(b) Relative Errors

Figure 5.4 Krylov Jacobian methods on Prothero Robinson system

5.3.2 Large Systems

In this section, we will study the influence of the placement of the eigenvalues of large matrices ($d > K$) on the stability of Krylov Jacobian methods. We use the following notation to describe rectangular subsets of the complex plane.

Notation 5.1 (Complex rectangles) *Let $\lambda, \mu \in \mathbb{C}$. We denote by*

$$[\lambda, \mu] := \{\lambda' \in \mathbb{C} : re(\lambda') \in [\min(re(\lambda), re(\mu)), \max(re(\lambda), re(\mu))] \\ \text{and } im(\lambda') \in [\min(im(\lambda), im(\mu)), \max(im(\lambda), im(\mu))]\}$$

the rectangular subset of the complex plane spanned by λ and μ. We abbreviate as

$$[\lambda] := [\lambda, 0] \tag{5.16}$$

the rectangle spanned by λ and the origin. For a given set $\Lambda \subset \mathbb{C}$, we write

$$\pm\Lambda := \left\{\lambda, \bar{\lambda} | \lambda \in \Lambda\right\} \tag{5.17}$$

to denote the union of the set and its element-wise complex conjugate.

In the following, we will sample random eigenvalues from rectangular subsets of the complex plane. The underlying distribution will be uniform. First, we analyze the eigenvalues of the Krylov Jacobian approximation for large systems. The goal is to understand how well the approximation captures the spectral properties of the original Jacobian.

Numerical Test 11 (Eigenvalues of Krylov Jacobian Approximation) *We use $x_0 = 0$ and a randomly*[2] *generated matrix $L \in \mathbb{R}^{16\times16}$ with the eigenvalues $\lambda_{1\dots8} = -400\pm200i, -300\pm200i, -200\pm200i, -100\pm 200i$ and 8 more uniformly sampled eigenvalues $\lambda_{9\dots16} \in [-300+100i, -200+50i] \cup [-300-100i, -200-50i]$. We used the **TASE1-Euler** method with stepsize $h = 0.01$ and $K = 4, 8$ and 12. Figure 5.5 shows all eigenvalues that were present in the Krylov Jacobian approximations throughout the interval $T = [0, 10]$.*

[2]We constructed $L = QDQ^T$, where

1. Q is a random unitary matrix obtained from the QR-decomposition of a uniformly sampled matrix $[1.0]^{16\times16} \ni M = QR$.
2. D is the block diagonal matrix with the real matrix embeddings of the eigenvalues $D_i = \begin{pmatrix} \mathrm{re}(\lambda_i) & \mathrm{im}(\lambda_i) \\ -\mathrm{im}(\lambda_i) & \mathrm{re}(\lambda_i) \end{pmatrix}$ on its diagonal..

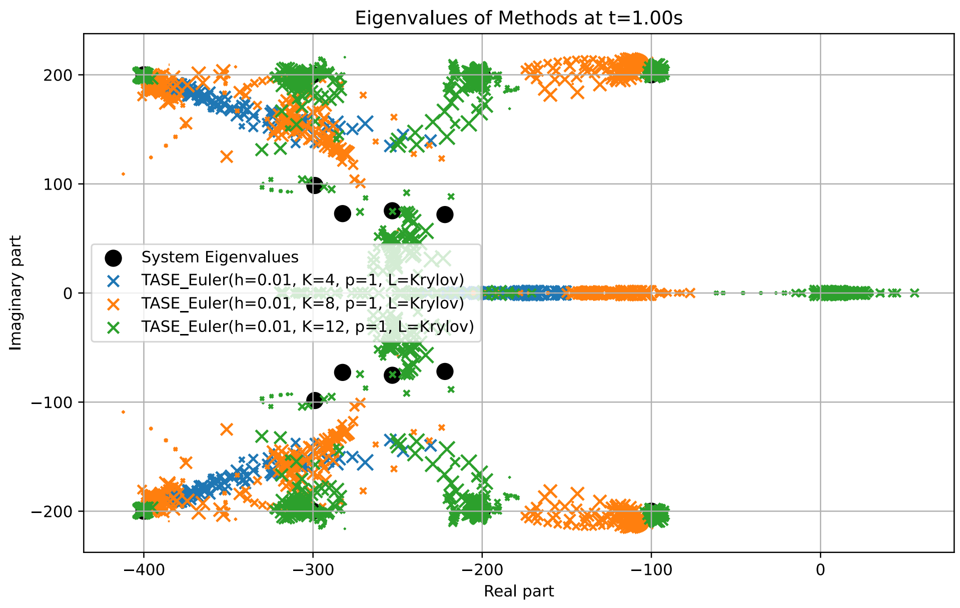


Figure 5.5 Eigenvalues of Krylov Jacobian approximation for large system

We see that …

1. *… the operator with $K = 12$ is able to approximately capture all $\lambda_{1\dots8}$ and form a cluster between the eigenvalues $\lambda_{9\dots16}$.*
2. *… the operator with $K = 8$ manages to approximately find the most extreme eigenvalues $\lambda_{1,2} = -400 \pm 200i$ and $\lambda_{7,8} = -100 \pm 200i$. We see two clusters of eigenvalues around $-300 \pm 150i$ that slightly miss the eigenvalues $\lambda_{3,4} = -300 \pm 200i$ respectively.*
3. *… the method with $K = 4$ seems to only find a trace towards the eigenvalues $\lambda_{1,2}$.*

*All three operators contain some eigenvalues on the real axis despite no purely real eigenvalues being present in the original matrix L. Remarkably, despite not nearly capturing all eigenvalues of the original Jacobian, all three **TASE1-Euler** methods remain stable. It is also interesting to note that the $K = 12$ operator contains some eigenvalues with positive real part.*

A rigorous analysis of the eigenvalues of Krylov Jacobian approximations is beyond the scope of this thesis. Through empirical tests, we found that the Krylov operator tends to predominantely capture the most extreme eigenvalues in the spectrum of L when $K < d$. A possible explanation is that these eigenvalues tend to amplify their respective eigenvectors the most. Thus, these compontents of the time-derivatives $x^{(1)} \dots x^{(K)}$ gain the most weight in the construction of $\hat{B}$.
We see, that the stability of Krylov Jacobian methods cannot be determined by the position of individual eigenvalues. Rather, we have to take into account the entire spectrum together when analyzing the stability of these methods. In the next test, we explore what effect the system dimension d and with it the number of eigenvalues in a system has on the stability of Krylov Jacobian methods. As a representative method, we choose the **TASE1-Euler** method paired with the Krylov Jacobian approximation using $K = 4$, $K = 8$ and $K = 12$ derivatives.

Numerical Test 12 (Largest possible stepsize on stiff systems of increasing dimension) *We use Test System 7 with a randomly initialized vector $x_0 \in [1.0]^d$ and a matrix $L \in \mathbb{R}^{d\times d}$ where the spectrum*

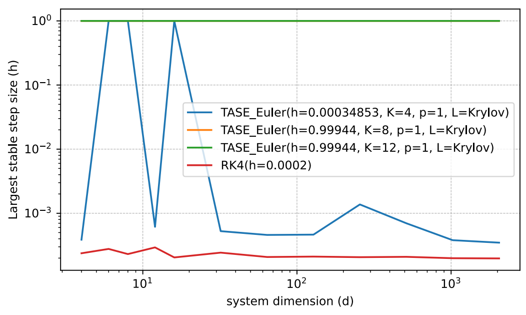
a) $\sigma(L) \subset \pm[-10^4 + 10^4 i]$ contains damped and oscillatory dynamics from a rectangular subset of the left half of the complex plane.

b) $\sigma(L) \subset \pm[-10^4 + 10^4 i, -0.5 \cdot 10^4 + 0.5 \cdot 10^4 i]$ contains mixed damped and oscillatory dynamics from a restricted region inside the left half of the complex plane.

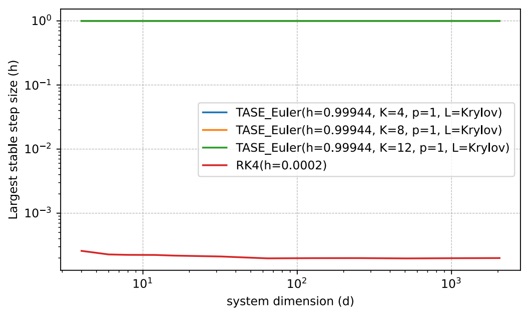
c) $\sigma(L) \subset \pm\left\{-10^5 + 10^5 i\right\} \cup \pm[-10^4 + 10^4 i]$ contains one very stiff eigenvalue[3] and many randomly sampled moderately stiff eigenvalues.

d) $\sigma(L) \subset \pm\left\{-10^5 + 10^5 i\right\} \cup \pm[-10^4 + 10^4 i, -0.5 \cdot 10^4 + 0.5 \cdot 10^4 i]$ contains one very stiff eigenvalue and many randomly sampled moderately stiff eigenvalues from a restriced region inside the left half of the complex plane.

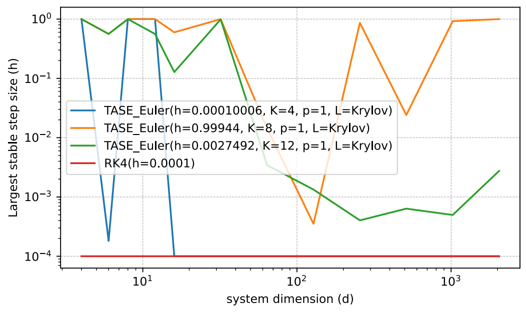
We use the heuristic $\|x_{50}\| < 10\,\|x_0\|$ to determine whether a method produced a 'stable' solution. We will call this property heuristic-stbility or short h-stability. While this heuristic may not entirely align with true stability of the methods, it is withouth a deeper understanding of the stability properties of Krylov Jacobian methods nontrivial to find a more reliable and computationally feasible alternative. For each method and all system dimensions $d = 4, 6, 8, 12, 16, 32, 64, 128, 256, 512, 1024, 2048$, we used a binary search algorithm to determine the largest stepsize h with which the method produced an h-stable solution. Note that h-stability must not be monotone in the stepsize h. Hence, the binary search algorithm might not exactly find the maximal stepsize h. Throughout many repeated tests, the results stayed roughly consistent suggesting their reliability regardless of possible random deviations. We only searched for stepsizes in the range $h \in [10^{-4}, 1]$. This methods showing a maximum stable stepsize $h = 10^{-4}$ might in reality require even smaller stepsizes. Methods reaching the h-stable stepsize $h = 1$ might in reality allow even larger stepsizes.



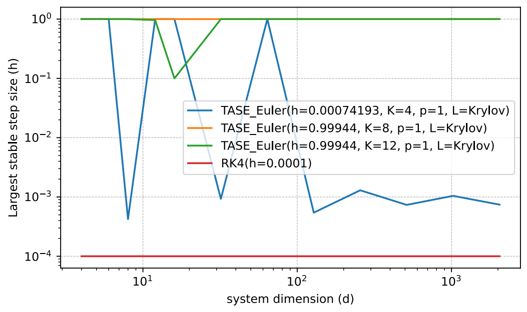
(a) $\sigma(L) \subset \pm[-10^4 + 10^4 i]$



(b) $\sigma(L) \subset \pm[-10^4 + 10^4 i, -0.5 \cdot 10^4 + 0.5 \cdot 10^4 i]$



(c) $\sigma(L) \subset \pm\left\{-10^7 + 10^5 i\right\} \cup \pm[-10^4 + 10^4 i]$



(d) $\sigma(L) \subset \pm\left\{-10^7 + 10^5 i\right\} \cup \pm[-10^4 + 10^4 i, -0.5 \cdot 10^4 + 0.5 \cdot 10^4 i]$

Figure 5.6 Largest possible stepsize on stiff systems of increasing dimension

*Throughout all tests, the RK4 method ist forced to take steps $h < 10^{-3}$ to remain h-stable. In the case of Figure 5.6a the **TASE1-Euler** method with $K = 4$ is forced to reduce its stepsize after exceeding $d > 10$. Both other methods with $K = 8$ and $K = 12$ however, remain h-stable even for the stepsize $h = 1.0$ for all tested dimensions up to $d = 2048$.*

[3]The terms 'very stiff', 'stiff' and 'moderately stiff' are to be understood in a comparative sense and do not have an absolute definition.

*In Figure 5.6b, we restricted the eigenvalues to all be in the same order of magnitude of stiffness. Here, all tested **TASE1-Euler** methods allowed the stepsize $h = 1.0$. In Figure 5.6c and Figure 5.6d, we used the same sampling regions as in Figure 5.6a and Figure 5.6b respectively, but added one very stiff pair of eigenvalues at $-10^7 \pm 10^5 i$. Here all methods were forced to reduce their stepsize at some point to remain h-stable.*
*Throughout all tests, we notice that increasing the stability of **TASE1-Euler** methods is influenced by a change in system dimension for small d, but plateaus eventually. Combined with the finding that reducing the area of the sampling region improves the methods' performance, this suggests that Krylov Jacobian methods will be applicable to systems of very large dimension*[4] *if the eigenvalues are restricted to be positioned close to each other in the complex plane. On systems where the eigenvalues are spread out evenly however, Krylov Jacobian methods seem to encounter significant stepsize restrictions.*
The modelling approach of dialectic mechanics studied in section 1.4.2, is capable of manipulating the eigenvalues in a way that moves very stiff ones towards a moderately stiff attractor. It is reasonable to assume that Krylov Jacobian methods will perform well on adequately manipulated systems because we can expect clusters of eigenvalues to lie close to an attractor. Testing this is beyond the scope of this thesis, but seems to be a worthy investigation for future analyses.

Numerical Test 13 (Largest possible stepsize on large systems of increasing stiffness) *We use Test System 7 with a randomly initialized vector $x_0 \in [1]^d$ and a matrix $L \in \mathbb{R}^{d\times d}$ where the spectrum*

a) $\sigma(L) \subset [-r]$ describes a highly damped stiff system.

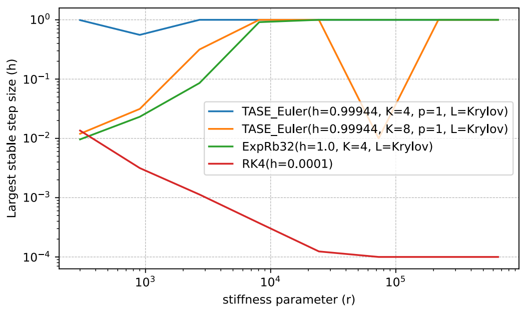
b) $\sigma(L) \subset [ri]$ describes a highly oscillatory stiff system.

c) $\sigma(L) \subset [-r + ri]$ describes as highly damped and oscillatory stiff system.

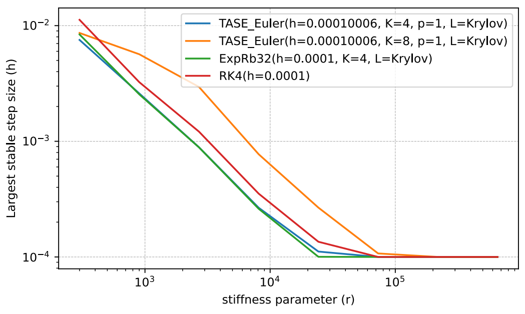
d) $\sigma(L) \subset [-r+ri, -r+ri+100-100i]$ describes a highly damped and oscillatory stiff system with closely neighboring dynamics.

The stiffness parameter $r > 0$ is used to vary from moderately stiff to very stiff system. Again, we use the heuristic $\|x_{50}\| < 10\,\|x_0\|$ to determine whether a method produced an h-stable solution.

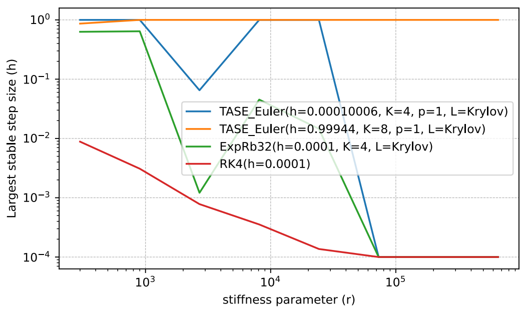
[4]The aim is to simulate systems in the order of $d = 10^6$, which is beyond the computational limits of my personal hardware for these tests



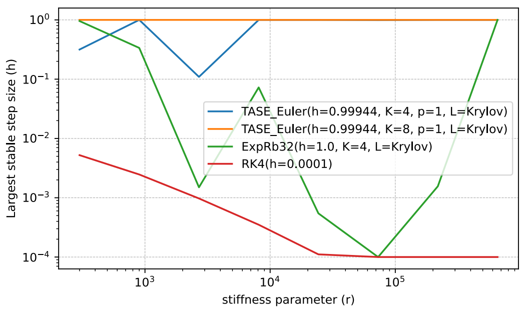
(a) $\sigma(L) \subset [-r]$



(b) $\sigma(L) \subset [ri]$



(c) $\sigma(L) \subset [-r + ri]$



(d) $\sigma(L) \subset [-r + ri, -r + ri + 100 - 100i]$

Figure 5.7 Largest possible stepsize on large systems of increasing stiffness

Figure 5.7 shows the largest possible stepsize h determined through binary search like in Numerical Test 12 for the spectra a) to d). We see in a) that Krylov Jacobian methods are able to h-stably integrate arbitrarily stiff problems with $h = 1.0$ when considering purely damped systems. Contrary, for purely oscillatory problems in b), they encounter the same stepsize restrictions as the standard explicit Runge-Kutta-4 method.
For problems with damped and oscillatory dynamics in c), we observe a difference between the methods with $K = 4$ and the one with $K = 8$. This suggests that the larger Krylov operator is able to better capture the largely varying eigenvalues resulting in the improved stability characteristics.
For the closely neighboring eigenvalues used in d), we see that both TASE methods are able to h-stably integrate the entire tested range of $r \in [3 \cdot 10^2, 6.5 \cdot 10^5]$. It is interesting to note that the ***ExpRb32*** *method was forced to take small steps for moderately stiff $r \in [3 \cdot 10^3, 3 \cdot 10^5]$, but was h-stable with large h again for $r \geq 3 \cdot 10^5$. This non-monotone stability in the stiffness parameter might be explained by the heuristic approach to determine stability, which was used in this plot. However, since the reduced stepsize for moderately stiff systems seen in Figure 5.7d appears consistent throughout many iterations of the test. It is therefore more likely that the more damped eigenvalues for larger r cause better stability of the exponential integrator. It is worthy to note that the relative spread of the eigenvalues in Figure 5.7d decreases as r increases.*

5.4 Interpreting Stability of Krylov Jacobian methods

Understanding the stability characteristics of the Krylov Jacobian methods on large systems is key to determining their usability. There is no analytic understanding of the behavior of the Krylov Jacobian approximation large systems yet. Therefore it is hard to definitively determine the gains in stepsize that come from using Krylov Jacobian methods over classical explicit methods such as the standard Runge-Kutta-4 (RK4) method. However, we can see that Krylov Jacobian methods are consistently on par with and in many cases outperform RK4. We learn from Numerical Test 12, that Krylov Jacobian methods perform significantly better when the eigenvalues of a system are mostly confined to a smaller area as in case (b) and (d) rather than spread out over a bounded region in the left half of the complex plane as in (a) and (c). We also learn that introducing single very stiff eigenvalues into the system does not significantly reduce

the methods' performance in most cases. Using many moderatley stiff eigenvalues on the other hand forces significantly smaller stepsizes. From Numerical Test 13, we learn that Krylov Jacobian methods perfrom great on purely damped systems (a), while being only on par with RK4 in purely oscillatory systems (b). In the damped and oscillatory case, the tested methods allow stepsizes as large as $h = 1.0$ up until eigenvalues of real or imaginary part greater than 10^4. Beyond this point, the stepsize was reduced infeasibly low.
It is important to note that these methods respond to stiffness in a significantly different way than Runge-Kutta or Taylor methods. For these methods, stability on linear systems entirely depends on the products $h\lambda$ for all $\lambda \in \sigma(L)$ individually. Here, a single eigenvalue for which the product $h\lambda$ lies outside of the stability domain of the method, leads to unstable results. Having more eigenvalues of the same stiffness however, has no impact on the stability of Runge-Kutta methods. As we have seen in Numerical Test 12, having more similarly stiff eigenvalues can force a method to take smaller steps to remain stable. For a single eigenvalue-pair however, we had already seen in numerical Test 8, that TASE methods were capable of stably integrating a system with eigenvalues up to $10^8 i$ with stepsize $h = 10^{-1}$. Contrary for larger systems with $d = 16$, oscillatory eigenvalues in the order of $10^5 i$ already forced the stepsize below $h = 10^{-4}$ as seen in Figure 5.7b. We can conclude, that it is not the position of individual eigenvalues that destabilizes Krylov Jacobian methods, but rather the number and variety of stiff eigendynamics that forces stepsize reductions.
For Runge-Kutta and Taylor methods, it also holds that if an eigenvalue λ can be stably simulated with stepsize h, then 2λ can be stably simulated with $\frac{h}{2}$. This does not apply to Krylov Jacobian methods in general. The only case where this property could be observed is shown in Figure 5.7b. In other cases, the dependency of the largest stable stepsize on stiffness was nonlinear. In Figure 5.7d, it was not even monotone for **ExpRb32**.
It also seems to be the case that different Jacobian methods differ in their sensitivity to errors in the Jacobian approximation they use. The **TASE1-Euler** method seems to be very forgiving to errors of the Krylov Jacobian as long as the eigenvalues of L are roughly matched. **ExpRb32** on the other hand does not work well with the closely neighboring eigenvalues in Figure 5.7d.
It generally seems advantageous to use larger Krylov spaces, but the advantage methods with $K = 12$ over those with $K = 4$ is by no means consistent. From Figure 5.5, we see that the larger Krylov space clearly leads to a more accurate representation of the eigenvalues of L in the approximation $\hat{B}$. In most testcases, the method with the larger Krylov space ($K = 8$) did not have to reduce its stepsize regardless of the increasing system dimension or stiffness. But in Figure 5.6c and Figure 5.7b, it suffers from the same limitations as the methods with the smaller Krylov space of $K = 4$.
We see, that the stability properties of Krylov Jacobian methods differ drastically from those of classical explicit methods such as RK4. For the application of the new methods however, it is vital to gain a better understanding of the limitations they face. Especially understanding the limmits for the stiffness of a single eigenvalue and the limits to the stiffness of many widely spread eigenvalues while maintaining stability is important. Furthermore it seems to be desirable to develop strategies to confidently detect unstable behavior in the methods.

6 Conclusion

The aim of this thesis was to explore explicit integration methods suitable for the real-time simulation of stiff mechanical systems. In Chapter 1, we introduced initial value problems (IVPs) and formalized essential concepts such as stiffness, stability, and accuracy. We discussed different formulations of the notion of stiffness and decided to use the eigenvalues of the system's Jacobian as the primary indicator for stiffness and the stable applicability of explicit methods as the main objective regarding stiffness. We also studied the eigenvalue manipulation in the modelling framework of dialectic mechanics to shape stiffness properties of systems to be more suitable for explicit methods.

Chapter 2 provided a review of classical time-marching schemes, emphasizing that the stability of methods such as Runge-Kutta (RK) can be analyzed via their stability polynomials. We observed that the stability of these methods on multidimensional systems can be assessed for each eigenvalue individually and entirely depends on the products $h\lambda$ for $\lambda \in \sigma(L)$. While Runge-Kutta methods can be optimized to improve their stability region for specific eigenvalue configurations, their overall stability domain remains bounded, and only minor improvements in allowable step size are achievable. We explored general linear methods (GLMs) as a unifying framework encompassing Runge-Kutta and multistep methods. However, we excluded multistep methods from further investigation due to their reliance on past values, which is unsuitable in systems affected by external events. A notable subclass of GLMs are parallelizable peer methods, which achieve Runge-Kutta-level accuracy without requiring sequential function evaluations—making them compatible with modern parallel computing architectures.

Chapter 3 focused on the parallel computation of higher derivatives, showing that such derivatives can be approximated similarly to the state variable using time-marching schemes. We studied Taylor methods and multiderivative Runge-Kutta (MDRK) methods that benefit from multiple higher derivatives. A key insight was that MDRK schemes can be used to calculate higher derivatives even when those are not provided directly by the model, presenting an opportunity for applying these methods to legacy systems.

The core contribution of this thesis was introduced in Chapter 4: a novel approach to approximating the Jacobian matrix using a Krylov subspace, based solely on the available time-derivatives. This Krylov Jacobian approximation has linear computational complexity and enables the use of sophisticated Jacobian-based methods in large-scale systems where full Jacobian evaluation is otherwise computationally infeasible. This approach is particularly suitable for linear systems but can also handle linear eigendynamics excited by external forces. However, the current implementation of the correction mechanism for external forces is not yet compatible with the parallel computation of higher derivatives, as the evaluation points required for finite differences cannot be freely adjusted. Adapting the correction mechanism to support this compatibility as outlined in remark 4.8 remains an important direction for future work.

For general nonlinear systems, the usability of the Krylov Operator cannot be guaranteed. Even for systems that contain almost linear eigendynamics disturbed by a small nonlinear term, there is no bound on the deviation of the Krylov operator from the system's Jacobian in matrix norm as example 4.11 showed. However, we were able to prove a bound on the error of the action of the Krylov operator on the system's time-derivatives. Numerical tests showed that Krylov Jacobian methods work well on almost linear systems. For systems of larger nonlinear terms, the usability remains to analyzed further. Special care must be taken in the modelling of systems where higher derivatives contain discontinuities or singularities, as this leads to large errors.

In Chapter 5, we examined explicit methods that utilize the Jacobian, such as time accurate and stabilized explicit (TASE) schemes and exponential Rosenbrock integrators. Enabled by the Krylov Jacobian, these methods can now be employed without the computational burden of full Jacobian instantiation. These Krylov Jacobian methods exhibit significantly different stability properties from traditional schemes.

A notable insight is that the effect of stiffness on Krylov Jacobian methods cannot be understood by considering each eigenvalue in isolation. For example, a single stiff eigenvalue is typically well-handled, but stability degrades as the number and variety of stiff eigenvalues increases. As such, the notion of the stability domain does not apply to Krylov Jacobian methods. System dimension had only a limited influence: while these methods are required to progressively reduce their step sizes for systems of dimension $d \leq 100$, their stability plateaued beyond that. In purely damped systems, the Krylov Jacobian methods outperformed RK4 by up to three orders of magnitude in allowed step size. However, in purely oscillatory systems, no step size advantage over RK4 was observed. Furthermore, the relationship between stiffness and step size was neither linear nor monotonic and remains an important subject of future analyses.

Using more time-derivatives generally enabled larger stable time steps, but a detailed theoretical analysis of the stability properties of Krylov Jacobian methods remains open. Future work must also assess the applicability of these methods to real-world systems to ensure that they are robust to the types of stiffness typically encountered in practice. This is vital in safety-critical environments where simulation accuracy and stability directly impact hardware behavior.

Although accuracy was not the primary focus of this thesis, we observed that Krylov Jacobian methods match or exceed the accuracy of classical methods in linear systems with external forces. Interestingly, TASE methods of order $p \geq 2$ failed to follow steady-state behavior in highly oscillatory forced systems, while first-order TASE methods ($p = 1$) successfully damped oscillations and captured the steady state. This deviation exists regardless of the use of the Krylov Jacobian approximation or the exact Jacobian and can therefore be attributed to the properties of higher order TASE methods themselves. This limitation in higher-order TASE methods warrants further investigation. In general, Exponential Rosenbrock methods showed superior accuracy but were less robust to errors in the Krylov operator than TASE methods.

Future use of Krylov Jacobian methods can also be in conjunction with the eigenvalue manipulation in the dialectic mechanics framework. Since this technique confines the system's eigenvalues to a limited region, where many eigenvalues are manipulated towards a common attractor, the resulting spectrum might be well suited for the application of Krylov Jacobian methods, which seem to work well on systems with tightly neighboring eigenmodes. Another future development might involve the use of parallelizable Peer methods in conjunction with TASE preconditioning operators known as stabilized explicit parallelizable peer (SEPP) methods. In combination with adapting multiderivative Runge-Kutta methods to use parallel stages, this approach could yield a highly parallelizable stable integration technique.

In summary, the Krylov Jacobian approximation is a promising strategy to enable Jacobian-based integration schemes in large, stiff systems without the cost of explicit Jacobian evaluation. However, the resulting methods behave fundamentally different from traditional explicit solvers, and their stability properties are not yet fully understood. A deeper theoretical understanding and empirical validation are required before these methods can be reliably used in real-time, safety-critical applications. Future development towards parallelizability can yield highly efficient stable integration schemes suited for modern compute architecture.

7 Acknowledgements

Research

To get an overview of numerical solvers for ordinary differential equations, I specifically profited from the books by Deuflhard and Bornemann [11] and Butcher [7], which contain a compact and accessible overview of the most relevant techniques and tools of analysis, which have been developed within the past centuries.

The most important approaches, which I investigated were prompted by my advisor **Dr. Dirk Zimmer**, who brought up multiderivative methods, exponential integrators and general linear methods.

To develop the mathematical foundations of the newly proposed Krylov-Jacobian-Approximation, I greatly profited from regular meetings with my advisor, **Andrea Neumayr**. She helped me understand the concept of Krylov Subspace methods by providing introductory literature on the topic, engaging in conversations about the potential use of Krylov Subspace methods in conjunction with multiderivative approaches and by checking my early sketches and calculations in the process of designing the scheme, which is presented in Chapter 4. Furthermore, she pointed me in the right direction in implementing and testing the schemes presented in this thesis.

I used OpenAI's **ChatGPT** to collect more possible approaches to the problem at hand. It gave me keywords such as "linearly implicit schemes" or "Krylov subspace methods" and short explanations of the approaches, which helped me quickly assess their relevance. To get a quick introduction to specific methods, which I wanted to investigate, I read their respective **Wikipedia** entries.

To gain concrete understanding of techniques, which were not yet covered in the standard literature, I searched for further scientific literature in the databases of **Mathscinet** and **Google Scholar**. I also found literature through the standard google search from websites such as **Springer Nature** and **Arxiv.org**.

Numerical Tests

To test the investigated methods numerically, I used the **Python** programming language, where I relied heavily on the **NumPy** library to enable vectorization and matrix calculations. I wrote and tested the code in Microsoft's **Visual Studio Code** editor. Within the editor I used Microsoft's **Github Copilot** extension, which uses a GPT language model to enable AI auto completions in most common programming languages. While not writing any code for me, this extension sped up my coding process by providing useful suggestions.

To visualize the numerical tests, I used the Python library **Pygame** to create a Graphical User Interface (GUI) and draw the solution curves calculated by the tested methods in real-time. I used the Python library **Matplotlib** to visualize error data and plot the eigenvalues and stability regions of systems and methods as well as the eigenvalues present in the Krylov-Jacobian-Approximation.

For the Pygame GUI and the implementation of Runge-Kutta methods in the numerical tests, I reused some of the code, which I wrote during my Internship at the German Aerospace Center (DLR) from April to June 2024. All the other methods were newly implemented.

Some of the most important functions of the code can be found in the Appendix.

LaTeX Graphics and Equations

Next to Python code completions, I also used Microsoft's **Github Copilot** for code completions while designing LaTeX Graphics with **tikz** and writing equations in LaTeX Math Mode. It produced some of the filling words in between calculations, but did not take part in producing any of the complete text passages.

File Version Control

I used the DLR's **GitLab** to back up and sync code and LaTeX files as well as share them with my advisors.

8 Appendix

8.1 Omitted proofs

8.1.1 Krylov Operator

Definition 8.1 (Orthogonal Projection) *An operator $P \in \mathbb{R}^{d \times d}$ is called an orthogonal projection onto the subspace $V \subseteq \mathbb{R}^d$ if $P^2 = P$ and the image $im(P) = V$ and the kernel $ker(P)$ are orthogonal to each other.*

$$x^T y = 0 \quad \textit{for all} \quad x \in ker(P), y \in im(P) \tag{8.1}$$

Lemma 8.2 (Orthogonal Projection Matrix) *Let $Q \in \mathbb{R}^{d \times K}$ be a column-orthonormal matrix, i.e. $Q^T Q = I_K$. Then*

$$P := QQ^T \tag{8.2}$$

is the orthogonal projection onto the Krylov space $\mathcal{K} = span(Q)$, i.e. the image $im(P) = \{Px\}$ is orthogonal to the kernel $ker(P) = \left\{x \in \mathbb{R}^d : Px = 0\right\}$.

Proof. *Let $x \in im(P)$, i.e. $x = Px'$ for some $x' \in \mathbb{R}^d$ and $y \in ker(P)$, i.e. $QQ^T y = 0$. Then*

$$\begin{aligned} x^T y &= (Px')^T y \\ &= (QQ^T x')^T y \\ &= (x')^T QQ^T y \\ &= (x')^T 0 \\ &= 0. \end{aligned}$$

Hence, x is orthogonal to y and $im(P)$ is orthogonal to $ker(P)$. □

Lemma 8.3 (Norm of the Orthogonal Projection) *Let $P \in \mathbb{R}^{d \times d}$ be the orthogonal projection onto a subspace $V \subseteq \mathbb{R}^d$. Then $\|P\|_2 = 1$ if and only if $V \neq \{0\}$.*

Proof. *Let $x \in \mathbb{R}^d$. We have*

$$P(x - Px) = Px - P^2 x = Px - Px = 0. \tag{8.3}$$

Hence, $x - Px \in ker(P)$. Thus, we get

$$\|x\|_2^2 = \|x - Px + Px\|_2^2 \overset{\textit{Pythagoras}}{=} \|x - Px\|_2^2 + \underbrace{2(x - Px)^T Px}_{0} + \|Px\|_2^2 \geq \|Px\|_2^2 \tag{8.4}$$

Thus, we have $\|P\|_2 \leq 1$.

If V is the zero space, $\|P\| = 0$ since all vectors $x \in \mathbb{R}^d$ are mapped to 0.

If V is not the zero space, then there exists some $x \in V \setminus \{0\}$ and some $y \in \mathbb{R}^d$ such that $Py = x$. And thus, $Px = P(Py) = P^2 y = Py = x$. Hence, $\|P\|_2 \geq 1$. □

8.1.2 Stability of Linear Iterations

Theorem 8.4 (Stability of Jordan Blocks) *Let $R \in \mathbb{R}^{d\times d}$ be arbitrary. The sequence $\|R^n\|_{n\in\mathbb{N}}$ remains bounded if and only if one of the following conditions hold:*

1. *The spectral radius $\rho(R) < 1$.*
2. *The spectral radius $\rho(R) = 1$ and all Jordan blocks where $|\lambda| = 1$ have size $m = 1$.*

Proof. *We can write $R = TJT^{-1}$ where J is the Jordan normal form of R and T an invertible matrix. Then we get*

$$\begin{aligned}\|R^n\| &= \left\|TJ^nT^{-1}\right\| \le \|T\|\,\|J^n\|\,\left\|T^{-1}\right\| = C_1 \cdot \|J^n\| \quad \textit{and}\\ \|R^n\| &= \left\|TJ^nT^{-1}\right\| \ge C_2 \cdot \|J^n\|\end{aligned}$$

Where $C_1 = \|T\|\left\|T^{-1}\right\|$ and $C_2 = \frac{1}{\|T\|\|T^{-1}\|}$. The exact values of C_1 and C_2 are not important for the proof, only that they are finite and independent of n. We have now seen, that the stability of the Runge-Kutta method is entirely determinded by the behavior of the powers J^n of the Jordan normal form of R. Since J is a block diagonal matrix, it is sufficient to consider the powers of the blocks of J. These are of the form

$$B = \begin{pmatrix} \lambda & 1 & 0 & \dots & 0 \\ 0 & \lambda & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \ddots & 1 \\ 0 & 0 & 0 & \dots & \lambda \end{pmatrix} \in \mathbb{C}^{m\times m}$$

Matrix multiplication yields

$$B^n = \begin{pmatrix} \lambda^n & \binom{n}{1}\lambda^{n-1} & \binom{n}{2}\lambda^{n-2} & \dots & \binom{n}{m-1}\lambda^{n-m+1} \\ 0 & \lambda^n & \binom{n}{1}\lambda^{n-1} & \dots & \binom{n}{m-2}\lambda^{n-m+2} \\ 0 & 0 & \lambda^n & \dots & \binom{n}{m-3}\lambda^{n-m+3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \lambda^n \end{pmatrix}$$

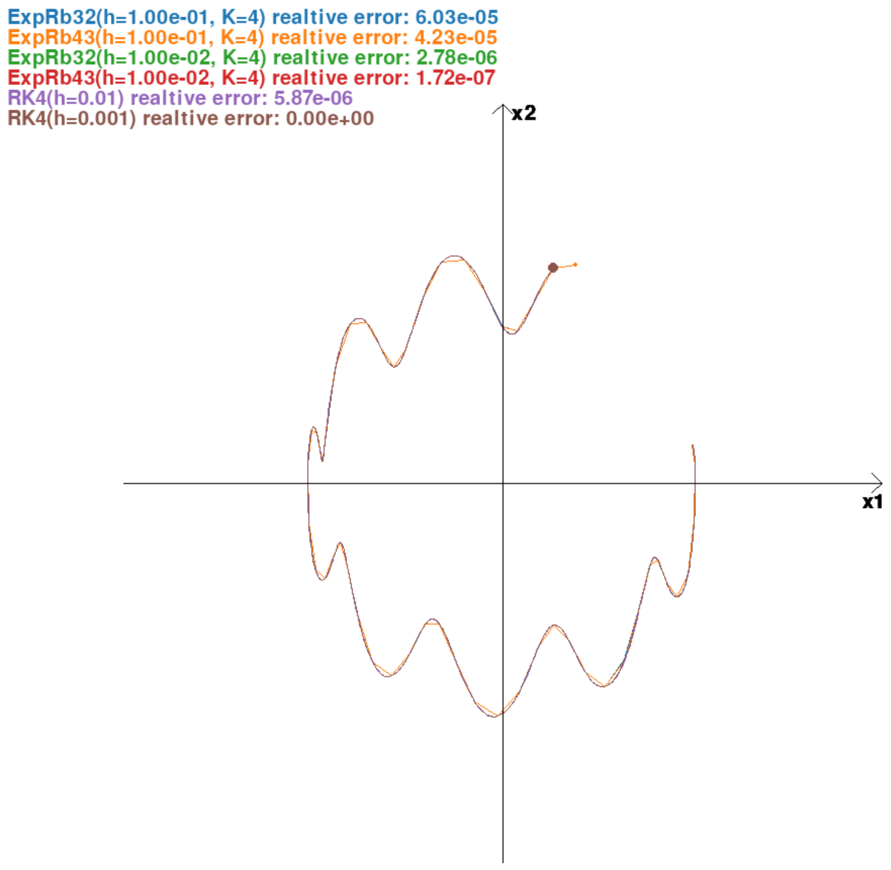
For $|\lambda| \neq 1$ the exponential growth of λ^{n-l} ($l \in \mathbb{N}$ constant) dominates the polynomial growth of $\binom{n}{l}$. We can conclude that

$$\lim_{n\to\infty} \|B^n\| = \begin{cases} \infty & \textit{if } |\lambda| > 1 \textit{ or } (|\lambda| \ge 1 \textit{ and } m > 1) \\ 0 & \textit{if } |\lambda| < 1 \\ 1 & \textit{if } |\lambda| = 1 \textit{ and } m = 1 \end{cases}$$

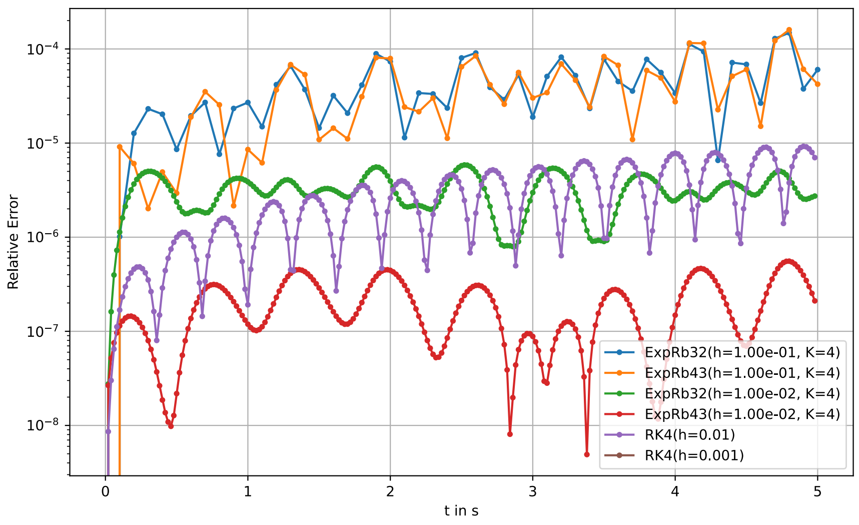
This concludes the proof. □

8.2 Exponential Rosenbrock Methods on Oscillating System

Numerical Test 14 *We test **ExpRb32** and **ExpRb43** on the same oscillating system as used in Numerical Test 7. The results are given in Figure 8.1. We see that the methods are capable of stably integrating the oscillating system and produces only small errors.*



(a) Trace



(b) Relative Errors

Figure 8.1 Exponential Rosenbrock methods on oscillating system

List of Figures

List of Tables

Bibliography

[1] V. I. Arnold. *Gewöhnliche Differentialgleichungen.* 1st ed. Berlin, Heidelberg, New York: Springer-Verlag, 1980. ISBN: 9783540668909.

[2] M. Bassenne, L. Fu, and A. Mani. "Time-Accurate and highly-Stable Explicit operators for stiff differential equations". In: *Journal of Computational Physics* 424 (2021), p. 109847. ISSN: 0021-9991. DOI: `https://doi.org/10.1016/j.jcp.2020.109847`.

[3] F. Bornemann. *Numerische lineare Algebra - Eine konzise Einführung mit MATLAB und Julia.* Wiesbaden: Springer, 2018. ISBN: 978-3-658-24431-6. DOI: `doi:10.1007/978-3-658-24431-6`.

[4] J. Butcher. "A history of Runge-Kutta methods". In: *Applied Numerical Mathematics* 20.3 (1996), pp. 247–260. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/0168-9274(95)00108-5`.

[5] J. Butcher. "General linear method: a survey". In: *Applied Numerical Mathematics* 1.4 (1985), pp. 273–284. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/0168-9274(85)90007-8`.

[6] J. Butcher. "Coefficients for the study of Runge-Kutta integration processes". In: *Journal of the Australian Mathematical Society* 3.2 (1963), pp. 185–201. DOI: `10.1017/S1446788700027932`.

[7] J. Butcher. *Numerical Methods for Ordinary Differential Equations.* John Wiley and Sons, Ltd, 2016. ISBN: 9781119121534. DOI: `10.1002/9781119121534`.

[8] J. Butcher. "On the Convergence of Numerical Solutions to Ordinary Differential Equations". In: *Math. Comp.* 20 (Jan. 1966), pp. 1–. DOI: `10.2307/2004263`.

[9] A. Carrasco and H. Leiva. "Variation of Constants Formula for Functional Parabolic Partial Differential Equations". In: *Electronic Journal of Differential Equations* 2007.130 (2007), pp. 1–20. ISSN: 1072-6691.

[10] F. Cellier and E. Kofman. *Continuous System Simulation.* Berlin, Heidelberg: Springer, 2006. ISBN: 978-0-387-26102-7.

[11] P. Deuflhard and F. Bornemann. *[Band] 2 Gewöhnliche Differentialgleichungen.* Berlin, New York: De Gruyter, 2008. ISBN: 9783110203578. DOI: `doi:10.1515/9783110203578`.

[12] M. H. Gutknecht. "A Brief Introduction to Krylov Space Methods for Solving Linear Systems". In: *Frontiers of Computational Science.* Ed. by Y. Kaneda, H. Kawamura, and M. Sasai. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 53–62. ISBN: 978-3-540-46375-7.

[13] E. Hairer and G. Wanner. "Multistep-multistage-multiderivative methods for ordinary differential equations". In: *Computing* 11 (Sept. 1973). Received: 30 April 1973, pp. 287–303. DOI: `10.1007/BF02252917`.

[14] M. Hochbruck and A. Ostermann. "Exponential integrators". In: *Acta Numerica* 19 (2010), 209–286. DOI: `10.1017/S0962492910000048`.

[15] M. Hochbruck, A. Ostermann, and J. Schweitzer. "Exponential Rosenbrock-Type Methods". In: *SIAM Journal on Numerical Analysis* 47.1 (2009), pp. 786–803. DOI: `10.1137/080717717`.

[16] M. Hutzenthaler, T. Kruse, and T. A. Nguyen. *On the speed of convergence of Picard iterations of backward stochastic differential equations.* 2022. arXiv: `2107.01840 [math.PR]`.

[17] G. Kemper and F. Reimers. *Lineare Algebra. Mit einer Einführung in diskrete Mathematik und Mengenlehre.* 1st ed. Springer-Verlag GmbH Deutschland, ein Teil von Springer Nature. Berlin, Heidelberg: Springer Spektrum, 2022, pp. IX + 375. ISBN: 978-3-662-63723-4. DOI: `10.1007/978-3-662-63724-1`.

[18] D. Ketcheson and A. Ahmadia. “Optimal stability polynomials for numerical integration of initial value problems”. In: *Communications in Applied Mathematics and Computational Science* 7.2 (Dec. 2012), pp. 247–271. ISSN: 1559-3940. DOI: `10.2140/camcos.2012.7.247`.

[19] D. I. Ketcheson et al. “RK-Opt: A package for the design of numerical ODE solvers”. In: *Journal of Open Source Software* 5.54 (2020), p. 2514. DOI: `10.21105/joss.02514`.

[20] K. Königsberger. *Analysis.* Fünfte, korrigierte Auflage. Springer-Lehrbuch. Berlin: Springer-Verlag, 2004. DOI: `10.1007/3-540-35077-2`.

[21] V. T. Luan and A. Ostermann. “Parallel exponential Rosenbrock methods”. In: *Computers & Mathematics with Applications* 71.5 (2016), pp. 1137–1150. ISSN: 0898-1221. DOI: `https://doi.org/10.1016/j.camwa.2016.01.020`.

[22] P. Moin. *Fundamentals of Engineering Numerical Analysis.* 2nd ed. Cambridge University Press, 2010.

[23] G. Pagano. “Stabilized explicit peer methods with parallelism across the stages for stiff problems”. In: *Applied Numerical Mathematics* 207 (2025), pp. 156–173. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/j.apnum.2024.08.023`.

[24] J. Rang. “An analysis of the Prothero–Robinson example for constructing new DIRK and ROW methods”. In: *Journal of Computational and Applied Mathematics* 262 (2014). Selected Papers from NUMDIFF-13, pp. 105–114. ISSN: 0377-0427. DOI: `https://doi.org/10.1016/j.cam.2013.09.062`.

[25] C. Runge. “Über die numerische Auflösung von Differentialgleichungen”. In: *Mathematische Annalen* 46.2 (1895), pp. 167–178. ISSN: 1432-1807. DOI: `10.1007/BF01446807`.

[26] D. Systèmes. *Dymola – Dynamic Modeling Laboratory, Version 2025.* `https://www.3ds.com/products-services/catia/products/dymola/`. Accessed: July 9, 2025. 2025.

[27] W. Walter. *Gewöhnliche Differentialgleichungen. Eine Einführung.* 3rd ed. Berlin, Heidelberg, New York: Springer-Verlag, 1985. DOI: `10.1007/978-3-642-57240-1`.

[28] A. Wusu, M. Akanbi, and S. Okunuga. “A Three-Stage Multiderivative Explicit Runge-Kutta Method”. In: *American Journal of Computational Mathematics* 3 (2013), pp. 121–126. DOI: `10.4236/ajcm.2013.32020`.

[29] D. Zimmer and C. Oldemeyer. “Dialectic Mechanics: Extension for Real-Time Simulation”. In: *Proceedings of the 15th International Modelica Conference.* Linköping Electronic Conference Proceedings (ECP). Aachen, Germany: Linköping University Electronic Press, 2023. DOI: `10.3384/ecp204239`.

[30] D. Zimmer and C. Oldemeyer. “Introducing Dialectic Mechanics”. In: *Proceedings of the 15th International Modelica Conference.* Linköping Electronic Conference Proceedings (ECP). Aachen, Germany: Linköping University Electronic Press, 2023. DOI: `10.3384/ecp204167`.

[31] D. D. Zimmer. *Discussion on calculation of higher derivatives in multiderivative methods.* Meeting at DLR, July 2025. Personal communication. 2025.

[32] D. D. Zimmer. *Discussion on nonlinear contact system.* Meeting at DLR, May 2025. Personal communication. 2025.