Universität Ulm
Fakultät für Mathematik und Wirtschaftswissenschaften

On Numerical Methods for Stiff Ordinary Differential Equation Systems

Masterarbeit

in Mathematik

vorgelegt von
Pascal Frederik Heiter
am 26.11.2012

Gutachter

Prof. Dr. Dirk Lebiedz
Prof. Dr. Franz Schweiggert

Acknowledgment

First of all, I would like to express my gratitude to my advisor, Prof. Dr. Dirk Lebiedz, for giving me the opportunity to write this thesis within his work group and for his support, instructions and patience during my work. I achieved an interesting insight in a mathematical field, which was completely new to me.

I would also like to extend my gratitude to Prof. Dr. Franz Schweiggert for serving on my thesis committee.

Further, I would like to thank the whole working group, namely Marc Fein, Jochen Siehr, Dominik Skanda, Marcel Rehberg and Jonas Unger, for all fruitful discussions and for editing my thesis. Special thanks go to Sebastian Kestler for all fruitful conversations about `C++` and technical stuff.

In addition, I would like to thank Vincent Breitenberger, Martin Geiger and Lukas Hanssler for editing my thesis, too.

Last but not least, I would like to thank Annika Laser for editing my thesis and supporting my in every sense and furthermore, I would like to thank my parents. Without their support, it would never have been possible to write this thesis and thereby, to finish my studies.

Thank you so much!

Ulm, November 2012.

Contents

Chapter 1
Introduction

1.1 Motivation

Ordinary differential equation systems (ODEs) are useful for modeling natural processes for example chemical reactions, plant growth or in general a change of a magnitude. Even though, the existence and uniqueness theory of those systems is advanced, in many cases the analytical solution is not known. Due to that, numerical methods for solving ordinary differential equation systems are very important. The first method to solve initial value problems occurred in 1768 and was developed by Leonard Euler. Euler's main idea was to approximate the derivatives with a linear term, the difference quotient. However, this method is not applicable to all ordinary differential equation systems which is demonstrated by the following example:

Example 1.1.1 Consider the following problem

$$\begin{aligned}
\dot{y}_1(t) &= -y_1(t) \\
\dot{y}_2(t) &= -\gamma y_2(t) + \frac{(\gamma-1)y_1(t) + \gamma y_1^2(t)}{(1+y_1(t))^2} \\
y_1(0) &= 2 \\
y_2(0) &= 1.5
\end{aligned} \tag{1.1}$$

with $\gamma > 1$. Figure 1.1 illustrates the behavior of the numerical solution calculated by Euler's Method (Forward Euler) with $h = 0.0476$ and $\gamma = 40$ and by Matlab's `ode23s`.

We notice, that the Forward Euler becomes instable, because the solution trajectory begins to oscillate. The ordinary differential equation system (1.1) is an example for a stiff ODE. One characteristic of those stiff problems is the existence of multiple time-scales which means that some magnitudes change very fast and do not affect the macroscopic behavior. Unfortunately, we can not exactly define the stiffness of an ODE. Curtis and Hirschfeld described the stiffness of ODEs in [4] (1952) as

> *"Stiff equations are equations where certain implicit methods, in particular BDF*[1]*, perform better, usually tremendously better, than explicit ones."*

[1]Backward Differentiation Formulas

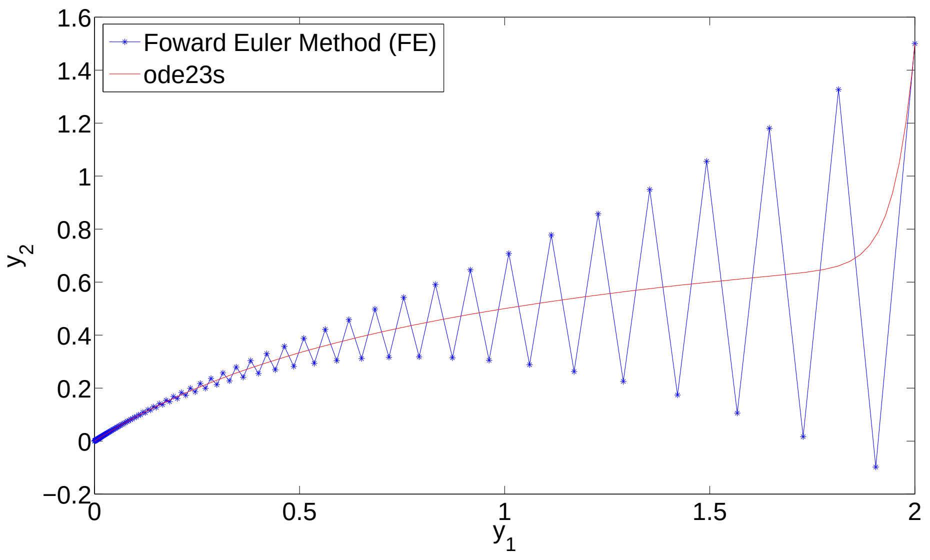


Figure 1.1: Plot of solutions for the problem (1.1) in Example 1.1.1 calculated with Forward Euler (FE) and `ode23s`.

In contrast to this opinion, Lee and Gear presented in [27] an efficient *explicit* method which is indeed able to deal with stiff systems.

1.2 Aim of this Thesis

The aim of this work is to discuss explicit methods for solving stiff ordinary differential equation systems, especially projective integrators based on ideas of Lee and Gear, cf. [27]. The focus is on the investigation of the theory and an efficient implementation in MATLAB and `C++` of these methods. Furthermore, we compare projective integrators with implicit methods, in particular with Backward Differentiation Formula (BDF) integrators. Moreover, we use a model reduction technique as provided by Lebiedz in [19] and integrate such a reduced system in order to decide if it is worthwhile to integrate the full or the reduced system with regards to the effort, runtime, integration steps and function evaluations.

1.3 Outline

The thesis is structured into five chapters.

A short overview of the existence and uniqueness theory of ordinary differential equations and their singularly perturbed forms is given in Chapter 2. Moreover, we explain the main idea of a model reduction software and introduce two models as examples for stiff ODE systems. Especially, we take a look on one model called *Simplified Six Species Hydrogen Combustion mechanism* which is a showpiece of a chemical multi-scale problem.

Chapter 3 deals with explicit integration methods solving stiff differential equation systems providing the theory of projective integrators and their implementation in Matlab. In particular, we explain their idea and give a detailed analysis of a Projective Forward Euler and a Projective Runge–Kutta Method. Further, we give a detailed proof of the second-order accuracy of the Projective Runge–Kutta Method based on ideas of Lee and Gear, cf. [26].

In Chapter 4, we discuss the numerical behavior of projective integrators and compare them to implicit methods, i.e. to BDF integrators. Furthermore, we deal with a model reduction tool and explain, how to represent a reduced model as an ODE of lower dimension.

The conclusion involves a table listing advantages and disadvantages of projective integrators compared to BDF integrators and some decision guidance in which cases it would be beneficial to use either a projective or a BDF integrator.

Chapter 2
Theory of Ordinary Differential Equation Systems and Models

In order to discuss numerical methods for solving stiff ordinary differential equation systems, we give a short overview of the existence and uniqueness theory of those. Furthermore, a few ideas of the singular perturbation theory are collected to gain a better understanding of fast and slow dynamics of multi-scale problems. Besides, the main idea of a model reduction software is discussed in this chapter, too. Afterwards, we take a look at two different nonlinear models, one well-known model called *Davis–Skodje model* and one simplified realistic chemical kinetic model, called *Simplified Six Species Hydrogen Combustion mechanism.*

2.1 Ordinary Differential Equation Systems

Ordinary differential equations are useful to describe time-dependent processes, e.g. chemical kinetics, plant growth or market behavior.

Definition 2.1.1 (Ordinary Differential Equation System) *Let $\Omega \subset \mathbb{R}^n$ be an open subset, $f : \Omega \to \mathbb{R}^n$ a vector-field and $t \in I$ with an interval $I \subset \mathbb{R}$. Then*

$$\dot{y}(t) = f\big(y(t)\big) \tag{2.1}$$

is an ***autonomous Ordinary Differential Equation (ODE) system****. Futhermore, if the vector-field f depends explicit on t, i.e.*

$$\dot{y}(t) = f\big(t, y(t)\big)$$

the system is said to be a ***nonautonomous ODE system****.*

In the following, we focus on autonomous systems, because any nonautonomous system can be written as an autonomous system with $y \in \mathbb{R}^{n+1}$ by defining $y_{n+1} := t$ and $\dot{y}_{n+1} = 1$. A solution of (2.1) is a map

$$\begin{aligned} y : I &\to \mathbb{R}^n \\ t &\mapsto y(t) \end{aligned}$$

such that y satisfies (2.1) for all $t \in I$. Note, that the solution is a curve in $\mathbb{R}^n$, called *trajectory.*

Definition 2.1.2 (Initial Value Problem) *Let $\Omega \subset \mathbb{R}^n$ be an open subset, $f : \Omega \to \mathbb{R}^n$ a vector-field, $t, t_0 \in I \subset \mathbb{R}$ and $y_0 \in \Omega$. Then*

$$\begin{aligned} \dot{y}(t) &= f\big(y(t)\big) \\ y(t_0) &= y_0 \end{aligned}$$

is said to be an ***Initial Value Problem*** *(IVP).*

Before establishing the existence-uniqueness theorem for nonlinear autonomous ODE systems, we need more definitions.

Definition 2.1.3 (Lipschitz condition) *Let $\Omega \subset \mathbb{R}^n$ be an open subset. A function $f : \Omega \to \mathbb{R}^n$ is said to satisfy a* ***Lipschitz condition****, if*

$$\exists\, K > 0\ \forall\, x, y \in \Omega : \quad ||f(x) - f(y)|| \le K\, ||x - y||\, .$$

The function f is said to be ***locally Lipschitz****, if*

$$\forall\, x_0 \in \Omega\ \ \exists N_\varepsilon(x_0), K_0 > 0\ \forall x, y \in N_\varepsilon(x_0) : \quad ||f(x) - f(y)|| \le K_0\, ||x - y||\, .$$

where

$$N_\varepsilon(x_0) := \{x \in \mathbb{R} : ||x - x_0|| < \varepsilon\}.$$

Therefore, a function f is locally Lipschitz, if f satisfies a Lipschitz condition on an ε-neighborhood of any point in Ω. The following result is useful to decide, if a function is locally Lipschitz.

Lemma 2.1.4 *Let $\Omega \subset \mathbb{R}^n$ be an open subset and $f : \Omega \to \mathbb{R}^n$. There it holds*

$$f \in C^1(\Omega) \qquad \Rightarrow \qquad f \text{ is locally Lipschitz on } \Omega,$$

where

$$C^1(\Omega) := \{f : f \text{ is continuously differentiable on } \Omega\}.$$

Proof. cf. [24], p. 71. □

Now, we are able to formulate the (local) existence and uniqueness theorem for nonlinear systems.

Theorem 2.1.5 (Existence-Uniqueness Theorem) *Let $\Omega \subset \mathbb{R}^n$ be an open subset, $y_0 \in \Omega$, $t_0 \in I \subset \mathbb{R}$ and assume that $f \in C^1(\Omega)$. Then, there exists an $a > 0$ such that the IVP*

$$\begin{aligned} \dot{y}(t) &= f\big(y(t)\big) \qquad , t \in I \\ y(t_0) &= y_0 \end{aligned}$$

has a unique solution on the interval $[t_0 - a, t_0 + a]$.

Proof. cf. [24], p. 74 ff. □

In our test models, cf. Section 2.4 and 2.5, the right-hand side is always continuously differentiable and based on the last theorem, a unique solution exists. Further, we discuss numerical methods for solving **stiff** problems. Unfortunately, there does not exist a unique definition of a stiff ODE, but as mentioned in the introduction, Curtis and Hirschfeld describes the stiffness of ODEs in [4] (1952) as follows

> *"Stiff equations are equations where certain implicit methods, in particular BDF, perform better, usually tremendously better, than explicit ones."*

and Hairer and Wanner mentioned in their first chapter in [9]

> *"Stiff equations are problems for which explicit methods don't work."*

In fact, explicit methods work for stiff problems, but they become inefficient through a tiny choice of the step size such that the method stays stable. Lee and Gear derived in [27] an efficient explicit method for solving those stiff problems. We can describe the behavior of a stiff system as follows.

Definition 2.1.6 (Stiff system) *A system of ODEs*

$$\dot{y}(t) = f(y(t))$$

is said to be ***stiff****, if there exist both fast and slow dynamics, e.g. in chemical kinetics very fast reactions and slow reactions can occur within one dynamical system, leading to a stiff ODE.*

In many cases the macroscopic behavior of the solution trajectory is more of interest than the microscopic one.

2.2 Singularly Perturbed Ordinary Differential Equation Systems

Assuming the existence of a diffeomorphism transforming the ODE system into a **singularly perturbed form**, the problem (2.1) can be rewritten (cf. [30]) in the two following ways, on the one hand the *fast system*

$$\begin{aligned} \dot{y}_{\mathrm{f}} &= f_1(y_{\mathrm{f}}, y_{\mathrm{s}}; \varepsilon) \quad , y_{\mathrm{f}}(t) \in \mathbb{R}^{n_{\mathrm{f}}} \\ \dot{y}_{\mathrm{s}} &= \varepsilon f_2(y_{\mathrm{f}}, y_{\mathrm{s}}; \varepsilon) \quad , y_{\mathrm{s}}(t) \in \mathbb{R}^{n_{\mathrm{s}}} \end{aligned}$$

where $0 < \varepsilon \ll 1$ is a measure of the separation of time scales and on the other hand, with defining the *slow time* $\tau := \varepsilon t$, the *slow system*

$$\begin{aligned} \varepsilon \frac{d}{d\tau} y_{\mathrm{f}} &= f_1(y_{\mathrm{f}}, y_{\mathrm{s}}; \varepsilon) \quad , y_{\mathrm{f}}(\tau) \in \mathbb{R}^{n_{\mathrm{f}}} \\ \frac{d}{d\tau} y_{\mathrm{s}} &= f_2(y_{\mathrm{f}}, y_{\mathrm{s}}; \varepsilon) \quad , y_{\mathrm{s}}(\tau) \in \mathbb{R}^{n_{\mathrm{s}}}. \end{aligned}$$

We consider the limit $\varepsilon \to 0$ and obtain two reduced systems. An n_{f}-dimensional *reduced fast system*

$$\begin{aligned} \dot{y}_{\mathrm{f}} &= f_1(y_{\mathrm{f}}, y_{\mathrm{s}}; 0) \\ \dot{y}_{\mathrm{s}} &= 0 \end{aligned} \tag{2.2}$$

whereby y_{s} is constant and in contrast to this, the differential-algebraic *reduced slow system* with a decrease of dimension from $n_{\mathrm{s}} + n_{\mathrm{f}}$ to n_{s} is

$$\begin{aligned} 0 &= f_1(y_{\mathrm{f}}, y_{\mathrm{s}}; 0) \\ \frac{d}{d\tau} y_{\mathrm{s}} &= f_2(y_{\mathrm{f}}, y_{\mathrm{s}}; 0). \end{aligned} \tag{2.3}$$

Consider the reduced system (2.3). Then,

$$\mathcal{W}_0 := \left\{ (y_{\mathrm{f}}, y_{\mathrm{s}}) \in V \subset \mathbb{R}^{n_{\mathrm{f}}} \times \mathbb{R}^{n_{\mathrm{s}}} \ : \ f_1(y_{\mathrm{f}}, y_{\mathrm{s}}; 0) = 0 \right\}.$$

is called **slow manifold**. Assuming that all eigenvalues of the reduced system Jacobian $D_{y_{\mathrm{f}}} f_1$ w.r.t. y_{f} have negative real part, the implicit function theorem guarantees the existence of a smooth function $h(\cdot)$ mapping from a compact domain $K \subset \mathbb{R}^{n_{\mathrm{f}}}$ to $\mathbb{R}^{n_{\mathrm{s}}}$, i.e.

$$h : K \to \mathbb{R}^{n_{\mathrm{s}}}$$

representing the slow manifold by

$$h(y_{\mathrm{s}}) = y_{\mathrm{f}}.$$

Thereby, the reduced slow system (2.3) can be written as

$$\frac{d}{d\tau} y_{\mathrm{s}} = f_2(h(y_{\mathrm{f}}), y_{\mathrm{s}}; 0).$$

Note that $\mathcal{W}_0$ is locally invariant.

Fenichels Geometric Singular Perturbation Theory [10, 11, 12, 13, 17] and some additional assumptions (cf. [30], pp 18-19) leads to an existence theorem for locally invariant manifolds $\mathcal{W}_\varepsilon$ for perturbed systems, which are close to $\mathcal{W}_0$. This locally invariant manifold $\mathcal{W}_\varepsilon$ is called slow, if $0 < \varepsilon \ll 1$.

2.3 Model Reduction Methods

In this section, we give a short overview of model reduction methods and focus on a method which bases on ideas of Lebiedz, cf. [19]. A detailed discussion of those

methods can be found in [30].
In general, model reduction methods for ODEs modeling chemical kinetics have been developed in the last century. Many methods deal with the occurrence of a Slow Invariant attracting Manifold (SIM) within the phase space which attracts nearby trajectories and leads to a lower dimensionality. Based on the stiffness of the high-dimensional dynamical system and consequently the existence of multiple time scales, we assume that our systems have a singularly perturbed form.
Some model reduction techniques are listed in the following

- *Quasi Steady–State Assumption (QSSA)*, cf. [3, 6, 23]
- *Partial Equilibrium Assumption (PEA)*, cf. [18]
- *Invariant Constrained equilibrium Edge PreImage Curve (ICE-PIC)*, cf. [33]
- *Zero Derivative Principle (ZDP)*, cf. [1, 5]

Nevertheless, we focus on a different model reduction technique, a **Trajectory-Based Optimization Approach** which is introduced by Lebiedz in [19]. The main idea is to minimize occurring relaxing (chemical) forces along reaction trajectories. Thus, an optimization problem wants to identify a SIM via minimization of an objective function including information about the behavior of trajectories.
SIMs can be described as a solution of an initial value problem

$$\dot{c}(t) = f(c(t)), \quad c(t) \in \mathbb{R}^n \tag{2.4}$$

$$c(0) = c^0. \tag{2.5}$$

with an initial value $c^0 \in \mathbb{R}^n$. The general *trajectory-based optimization approach* is formulated as

$$\min_{c} \int_0^{t_\mathrm{f}} \Phi(c(t))\,\mathrm{d}t \tag{2.6a}$$

subject to

$$\dot{c}(t) = f(c(t)) \tag{2.6b}$$

$$0 = g(c(0)) \tag{2.6c}$$

$$c_j(0) = c_j^0, \qquad j \in I_\text{fixed} \tag{2.6d}$$

whereas $c : [0, t_\mathrm{f}] \to \mathbb{R}^n$ denotes the state vector containing the concentration of chemical species. Equation (2.6b) describes the system dynamics, e.g. chemical kinetics determined by the reaction mechanism. This dynamics enter the optimization problem as an equality constraint. Additional constraints, e.g. chemical mass conservation relations as a consequence from the law of mass conservation, are represented by a function $g \in C^\infty(\mathbb{R}^n)$ in (2.6c). The index set I_fixed contains the indices of state variables, denoted as *reaction progress variables*, which parameterize the reduced model with fixed values at $t = 0$. Due to that, the other state variables c_j, $j \notin I_{\text{fixed}}$ represent the degrees of freedom. The solution of the optimization problem (2.6) represents a trajectory, which is in the best case close to a SIM and thus, we gain a point near the attracting SIM while evaluating this solution at $t = 0$. Simultaneously, we reconstruct the full species composition from given values c_j^0, $j \in I_{\text{fixed}}$. This process is called *species reconstruction.*

We use the following relaxation criterion by choosing the objective function as follows

$$\Phi(c(t)) = || \, J_f \cdot f(c(t)) \, ||_2^2$$

with the system Jacobian J_f.

Several other relaxation criteria and a software package called `MoRe` developed by Jochen Siehr have been tested over time, cf. [20, 21, 8, 7, 25, 31, 32, 30, 28, 29]. Using the software `MoRe`, especially a `MoRe-Wrapper` written by Marcel Rehberg, enables the building of a reduced right-hand side and thereby, we are able to deal with a reduced system.

2.4 Davis–Skodje Model

The Davis–Skodje model

$$\begin{aligned}
\dot{y}_1(t) &= -y_1(t) \\
\dot{y}_2(t) &= -\gamma y_2(t) + \frac{(\gamma-1)y_1(t) + \gamma y_1^2(t)}{(1+y_1(t))^2}
\end{aligned}$$

with $y(t) \in \mathbb{R}^2$ is an example of a stiff ODE system where $\gamma > 1$ is a measure of the spectral gap, i.e. γ is a measure for the stiffness of the system. The singularly perturbed form is

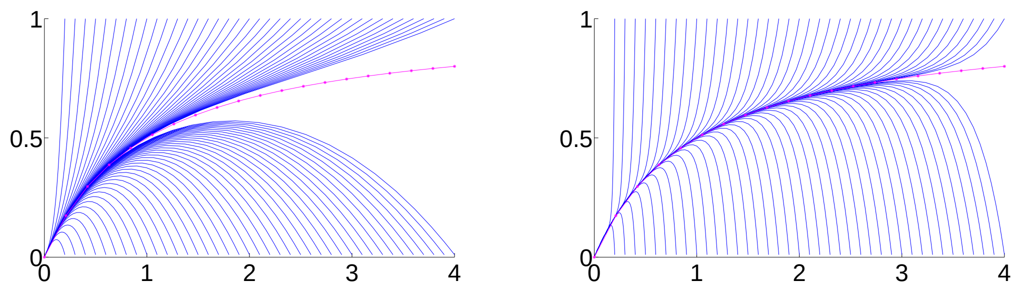
$$\begin{aligned}
\dot{y}_1(t) &= -y_1(t) \\
\varepsilon \dot{y}_2(t) &= -y_2(t) + \frac{y_1}{1+y_1} - \frac{\varepsilon y_1}{(1+y_1)^2}
\end{aligned}$$

whereas $\varepsilon := \frac{1}{\gamma}$. This model is widely used for testing model reduction methods, because the SIM is analytically computable through

$$\mathcal{W}_\varepsilon = \left\{ (y_1, y_2) \in \mathbb{R}^2 \; : \; y_2 = \frac{y_1}{1+y_1} \right\}.$$

Thus, it holds $y_{\text{s}} = y_1$ and $y_{\text{f}} = y_2$. The equilibrium of the Davis–Skodje model is the origin (0,0). Figure 2.1a and 2.1b depict the solution trajectories of various initial values

$$y_0 \in \left\{ \begin{pmatrix} 0.2 \\ 1 \end{pmatrix}, \begin{pmatrix} 0.3 \\ 1 \end{pmatrix}, \ldots, \begin{pmatrix} 4 \\ 1 \end{pmatrix}, \begin{pmatrix} 0.2 \\ 0.01 \end{pmatrix}, \begin{pmatrix} 0.3 \\ 0.01 \end{pmatrix}, \ldots, \begin{pmatrix} 4 \\ 0.01 \end{pmatrix} \right\}$$



(a) $\gamma = 3.0$ (b) $\gamma = 15.0$

Figure 2.1: Visualization of different solutions of the Davis–Skodje model for various initial values.

for $\gamma = 3.0$ and $\gamma = 15.0$. As mentioned before, the stiffness of the system depends on the value of γ. For a large value of γ, the SIM (magenta line) is more attractive because of the larger time scale separation.

2.5 Simplified Six Species Hydrogen Combustion Mechanism

The Simplified Six Species Hydrogen Combustion mechanism consists of five reactive species (O, H_2, H, OH, H_2O) and inert nitrogen (N_2). The combustion mechanism depends on the temperature and we fix the temperature at $T = 3000$K. The non-simplified mechanism was published by Lie et al. in [16] and was simplified by Ren et al. in [33] for testing their model reduction method ICE-PIC. Table 2.1 contains the specific six reactions of Arrhenius type for this mechanism whereas M represents a third body with collision efficiencies as follows

$$M = c_{\mathrm{O}} + 2.5c_{\mathrm{H_2}} + c_{\mathrm{H}} + c_{\mathrm{OH}} + 12c_{\mathrm{H_2O}} + c_{\mathrm{N_2}},$$

whereas c_s is the concentration of species s. The element mass conservation relations for this mechanism are

$$\begin{aligned} z_{\mathrm{H}} + 2z_{\mathrm{H_2}} + z_{\mathrm{OH}} + 2z_{\mathrm{H_2O}} &= 12.3400566662\,\mathrm{kg}\cdot\mathrm{mol}^{-1} \\ z_{\mathrm{OH}} + z_{\mathrm{O}} + z_{\mathrm{H_2O}} &= 4.1100136712\,\mathrm{kg}\cdot\mathrm{mol}^{-1} \\ 2z_{\mathrm{N_2}} &= 65.8102672822\,\mathrm{kg}\cdot\mathrm{mol}^{-1} \end{aligned}$$

whereas z_s is the specific mole of species s, and based on values presented by Al-Khateeb in [2]. The forward reaction rates are computable via the Arrhenius law

$$k_{\mathrm{f},i} = A\,T^b \exp\left(\frac{E_{\mathrm{a}}}{TR}\right), \quad i = 1, \ldots, 6$$

for each reaction i corresponding to the values in Table 2.1 and with the universal gas constant

$$R = 8.3144727 \frac{\mathrm{J}}{\mathrm{mol\ K}}.$$

Reaction			A / (cm,mol,s)	b	E$_a$ / kJ mol^{-1}
$O + H_2$	$\rightleftharpoons$	$H + OH$	$5.08 \cdot 10^{04}$	2.7	26.317
$H_2 + OH$	$\rightleftharpoons$	$H_2O + H$	$2.16 \cdot 10^{08}$	1.5	14.351
$O + H_2O$	$\rightleftharpoons$	$2OH$	$2.97 \cdot 10^{06}$	2.0	56.066
$H_2 + M$	$\rightleftharpoons$	$2H + M$	$4.58 \cdot 10^{19}$	-1.4	436.726
$O + H + M$	$\rightleftharpoons$	$OH + M$	$4.71 \cdot 10^{18}$	-1.0	0.000
$O + OH + M$	$\rightleftharpoons$	$H_2O + M$	$3.80 \cdot 10^{22}$	-2.0	0.000

Tab. 2.1: Simplified six species hydrogen combustion mechanism

The ODE system can be derived as proposed in [28] and we obtain the following ordinary differential equation system

$$\begin{aligned}
\rho \dot{z}_{\mathrm{O}} &= - k_{\mathrm{f},1}\, c_{\mathrm{O}} c_{\mathrm{H_2}} + k_{\mathrm{r},1}\, c_{\mathrm{H}} c_{\mathrm{OH}} \\
&\quad - k_{\mathrm{f},3}\, c_{\mathrm{O}} c_{\mathrm{H_2O}} + k_{\mathrm{r},3}\, c_{\mathrm{OH}}^2 \\
&\quad - k_{\mathrm{f},5}\, c_{\mathrm{O}} c_{\mathrm{H}} M + k_{\mathrm{r},5}\, c_{\mathrm{OH}} M \\
\rho \dot{z}_{\mathrm{H_2}} &= - k_{\mathrm{f},1}\, c_{\mathrm{O}} c_{\mathrm{H_2}} + k_{\mathrm{r},1}\, c_{\mathrm{H}} c_{\mathrm{OH}} \\
&\quad - k_{\mathrm{f},2}\, c_{\mathrm{H_2}} c_{\mathrm{OH}} + k_{\mathrm{r},2}\, c_{\mathrm{H_2O}} c_{\mathrm{H}} \\
&\quad - k_{\mathrm{f},4}\, c_{\mathrm{H_2}} M + k_{\mathrm{r},4}\, c_{\mathrm{H}}^2 M \\
\rho \dot{z}_{\mathrm{H}} &= k_{\mathrm{f},1}\, c_{\mathrm{O}} c_{\mathrm{H_2}} - k_{\mathrm{r},1}\, c_{\mathrm{H}} c_{\mathrm{OH}} \\
&\quad + k_{\mathrm{f},2}\, c_{\mathrm{H_2}} c_{\mathrm{OH}} - k_{\mathrm{r},2}\, c_{\mathrm{H_2O}} c_{\mathrm{H}} \\
&\quad + 2k_{\mathrm{f},4}\, c_{\mathrm{H_2}} M - 2k_{\mathrm{r},4}\, c_{\mathrm{H}}^2 M \\
&\quad - k_{\mathrm{f},5}\, c_{\mathrm{H}} c_{\mathrm{O}} M + k_{\mathrm{r},5}\, c_{\mathrm{OH}} M \\
&\quad - k_{\mathrm{f},6}\, c_{\mathrm{H}} c_{\mathrm{OH}} M + k_{\mathrm{r},6}\, c_{\mathrm{H_2O}} M \\
\rho \dot{z}_{\mathrm{OH}} &= k_{\mathrm{f},1}\, c_{\mathrm{O}} c_{\mathrm{H_2}} - k_{\mathrm{r},1}\, c_{\mathrm{H}} c_{\mathrm{OH}} \\
&\quad - k_{\mathrm{f},2}\, c_{\mathrm{H_2}} c_{\mathrm{OH}} + k_{\mathrm{r},2}\, c_{\mathrm{H_2O}} c_{\mathrm{H}} \\
&\quad + 2k_{\mathrm{f},3}\, c_{\mathrm{H_2O}} c_{\mathrm{O}} - 2k_{\mathrm{r},3}\, c_{\mathrm{OH}}^2 \\
&\quad + k_{\mathrm{f},5}\, c_{\mathrm{H}} c_{\mathrm{O}} M - k_{\mathrm{r},5}\, c_{\mathrm{OH}} M \\
&\quad - k_{\mathrm{f},6}\, c_{\mathrm{H}} c_{\mathrm{OH}} M + k_{\mathrm{r},6}\, c_{\mathrm{H_2O}} M \\
\rho \dot{z}_{\mathrm{H_2O}} &= k_{\mathrm{f},2}\, c_{\mathrm{H_2}} c_{\mathrm{OH}} - k_{\mathrm{r},2}\, c_{\mathrm{H_2O}} c_{\mathrm{H}} \\
&\quad - k_{\mathrm{f},3}\, c_{\mathrm{H_2O}} c_{\mathrm{O}} - k_{\mathrm{r},3}\, c_{\mathrm{OH}}^2 \\
&\quad + k_{\mathrm{f},6}\, c_{\mathrm{H}} c_{\mathrm{OH}} M - k_{\mathrm{r},6}\, c_{\mathrm{H_2O}} M \\
\rho \dot{z}_{\mathrm{N_2}} &= 0.
\end{aligned}$$

involving the concentrations c_s and the corresponding specific moles z_s by converting them as follows

$$c_s = \rho z_s$$

with

$$\rho = \frac{p}{RT} \left(\sum_{s \in \{\mathrm{O},\mathrm{H}_2,\mathrm{H},\mathrm{OH},\mathrm{H}_2\mathrm{O},\mathrm{N}_2\}} z_s \right)^{-1}$$

and $p = 101325$ Pa. Note, that the reverse rates $k_{\mathrm{r},i}$, which depend on the temperature, have to be computed for every reaction i as proposed in [28]. As long as no diffeomorphism, which transforms the system above in a singularly perturbed form, is known the choice of the reaction progress variable, i.e. the *slow variable*, is arbitrairly. In our case, we choose $z_{\mathrm{H}_2\mathrm{O}}$.

Chapter 3
Projective Integrators for Stiff Ordinary Differential Equations

In this chapter, we introduce explicit methods for solving stiff ordinary differential equation systems. The idea of projective integrators considered in this work was published by Gear et al. in [14, 15, 27]. One aspect of developing those integrators is their black-box use, independent of the choice of the inner integrator. For example, the microscopic behavior can be described by a Monte Carlo simulation. However, we are only interested in long term behavior, i.e. macroscopic behavior. Lee and Gear motivated the integrators in [27] as follows

> *"If the stiff differential equations are not directly available, our formulations and stability analysis are general enough to allow the combined outer-inner projective integrators to be applied to black-box legacy codes or perform a coarse-grained time integration of microscopic systems to evolve macroscopic behavior, for example."*

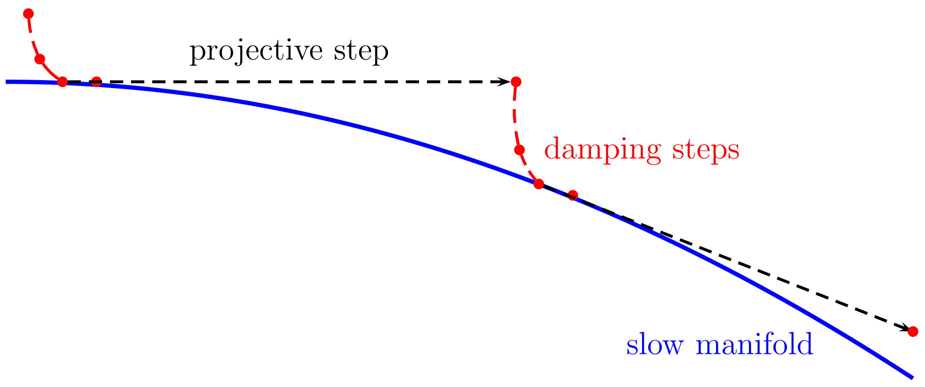


Figure 3.1: Idea of projective integrators.

The conventional Forward Euler Method and other conventional explicit methods are inefficient for solving stiff initial value problems, because the stability depends on the choice of the step size, i.e. the stiffer the system the smaller the step size. Therefore, a long term behavior observation becomes very expensive, because we need a large number of integration steps. The main difficulty is that the fast dynamics affect the explicit method adversely. It would be beneficial if these fast dynamics were damped in every integration step and after this, a larger projective step can be performed.

The main idea of projective integrators, which are explicit methods exploiting the multi-scale features of stiff systems, is straightforward. An *inner integrator* damps the fast dynamics with a constant step size, which is small enough to guarantee stability of the algorithm that means following stably the fast transients towards the slow manifold. After a few damping steps a chord slope is determined based on two previous calculated solution values which now describe the behavior of the slow manifold. Using this chord slope, a large projective step can be performed.

Figure 3.1 shows the idea of damping and projective steps relative to a slow manifold. The blue line represents the slow attracting manifold. The red dots results from damping the fast dynamic. The black dashed arrow illustrates a projective step using two previous calculated values.

Based on ideas of Lee and Gear in [27], in the following a (Tele-)Projective Forward Euler Method (PFE) and a second-order accurate Projective Runge–Kutta Method (PRK) are presented. Both algorithms are available in MATLAB and the projective Runge–Kutta Method is also implemented in `C++`.

3.1 Projective Forward Euler Method

Consider an initial value problem as defined in Section 2.1.2

$$\begin{aligned} \dot{y}(t) &= f(y(t)) \quad , t \in [t_0, t_f] \\ y(t_0) &= y_0 \end{aligned}$$

with $y_0 \in \mathbb{R}^n$. The **Projective Forward Euler Method** (PFE) extends the idea of conventional Forward Euler.

(i) Choose a suitable inner integrator (e.g. conventional Forward Euler Method) which is at least of first-order accuracy, a projective factor M, a number of damping steps k and a step size h_0 such that the inner integrator is stable. Note that for the conventional Forward Euler Method the best choice of the step size is

$$h_0 := \frac{1}{\max_i |\lambda_i|}$$

whereas λ_i are the eigenvalues of the system Jacobian $\partial f / \partial y$.

(ii) Start from $y_n = y(t_n)$. Perform k damping steps to obtain $y_{n+1}, \ldots, y_{n+k}$.

(iii) Perform one more damping step to obtain y_{n+k+1} and use this value to approximate the chord slope

$$v'_{n+k+1,n+k} = \frac{y_{n+k+1} - y_{n+k}}{h_0}.$$

(iv) Perform the projective step

$$y_{n+s} = y_{n+k+1} + Mh_0\, v'_{n+k+1,n+k} = y_{n+k+1} + M\left(y_{n+k+1} - y_{n+k}\right).$$

whereby $s = k + 1 + M$ is the length of this PFE step. Note that the calculations above are all vector operations which are cheap to compute. This method can be applied efficiently to stiff systems having a clear time scale separation, i.e. the eigenvalues of the system Jacobian $\partial f/\partial y$ are well clustered. The eigenvalues with the most negative real parts correspond to the fast time scales and the eigenvalues with real parts being relative close to the origin correspond to the slow ones. If there exists a large gap between the clusters, projective integrators can be applied to this stiff system. The length of the projective step depending on the choice of M is strongly related to the size of this gap. If the time scales are not clearly separated, telescopic projective, i.e. teleprojective integrators are efficient methods for carrying out the time integration, cf. Section 3.2. Lee and Gear also introduced an on-the-fly local error estimator for PFE in [26].

In order to discuss the errors within projective integrators only up to third-order, we ignore terms of higher order. Hence, the error involves multiplies of h^2y'', h^3y''' and h^3Jy'' where J is the system Jacobian and the prime represents differentiation w.r.t. t. Consider a bounded number of steps (independent of h) such that the exact time at which y''' and Jy'' are evaluated does not matter. Let

$$e_j(h) := y_j - y(t_j)$$

be the *global error* starting from a correct value y_0, i.e. $e_0 = 0$ and

$$d_j(h) := y_{j+1} - y(t_{j+1})$$

be the *local error* starting from a correct value y_j and performing one integration step to y_{j+1}. Define

$$C_j(h) := \left(-\frac{h^2}{2}y''_j, -\frac{h^3}{6}y''', -\frac{h^3}{2}Jy''\right), D_j := \begin{pmatrix}\xi_j\\ \gamma_j\\ \eta_j\end{pmatrix}, E_j := \begin{pmatrix}\psi_j\\ \phi_j\\ \theta_j\end{pmatrix}$$

and the translation operator T

$$T(q) := \begin{pmatrix}1 & 0 & 0\\ -3q & 1 & 0\\ 0 & 0 & 1\end{pmatrix}.$$

Note that

$$C_n(h) = T(m-n)C_m(h)$$

holds for all $m, n \in \mathbb{N}$. The following lemma presents a formula for the error coefficients of the global error after one PFE step. Thus, the error can be computed on-the-fly via recurrence formulas as presented in [26].

Lemma 3.1.1 (Global Error for a PFE Step) *For one PFE step it holds*

$$e_s(h) = C_s(h)E_s + \mathcal{O}(h^4)$$

whereas

$$E_s = \begin{pmatrix} (M+1)\psi_{k+1} - M\psi_k + M(M+1) \\ 3M(M+1)(\psi_k - \psi_{k+1}) - M\phi_k + (M+1)\phi_{k+1} - M(M+1)(2M+1) \\ (M+1)\theta_{k+1} - M\theta_k \end{pmatrix}$$

Thus

$$\begin{aligned}
\psi_s &= (M+1)\psi_{k+1} - M\psi_k + M(M+1) \\
\phi_s &= 3M(M+1)(\psi_k - \psi_{k+1}) - M\phi_k + (M+1)\phi_{k+1} - M(M+1)(2M+1) \\
\theta_s &= (M+1)\theta_{k+1} - M\theta_k.
\end{aligned}$$

Proof. We prove this lemma with ideas and results from [26]. The global and local error can be represented by

$$e_j(h) = C_j(h)E_j + \mathcal{O}(h^4) \qquad \text{and} \qquad d_j(h) = C_{j+1}(h)D_j + \mathcal{O}(h^4)$$

and the error amplifier of the conventional Forward Euler Method is $(I+hJ)$, because

$$\begin{aligned}
e_{n+1}(h) &= y_{n+1} - y(t_{n+1}) = y_n + hf(t_n, y_n) - y(t_n) - hy'(t_n) + \mathcal{O}(h^2) \\
&= y_n - y(t_n) + h\left(f(t_y, y_n) - f(t_y, y(t_n))\right) + \mathcal{O}(h^2) \\
&\overset{\text{Mean value theorem}}{=} e_n + h(\underbrace{f_y(t_n, \xi_n)}_{=J}(y_n - y(t_n))) + \mathcal{O}(h^2) \\
&= e_n + hJe_n + \mathcal{O}(h^2) = (I+hJ)e_n + \mathcal{O}(h^2).
\end{aligned}$$

Further, there it holds

$$\begin{aligned}
e_{n+1}(h) &= (I+hJ)e_n(h) + d_n(h) + \mathcal{O}(h^4) = (I+hJ)C_n(h)E_n + C_{n+1}(h)D_n + \mathcal{O}(h^4) \\
&= (I+hJ)C_{n+1}(h)T(1)E_n + C_{n+1}(h)D_n + \mathcal{O}(h^4) \\
&= C_{n+1}(h)T(1)E_n + hJC_{n+1}(h)T(1)E_n + C_{n+1}(h)D_n + \mathcal{O}(h^4) \\
&= C_{n+1}(h)\begin{pmatrix} \psi_n \\ -3\psi_n + \phi_n \\ \theta_n \end{pmatrix} + hJC_{n+1}(h)\begin{pmatrix} \psi_n \\ -3\psi_n + \phi_n \\ \theta_n \end{pmatrix} + C_{n+1}(h)\begin{pmatrix} \xi_n \\ \gamma_n \\ \eta_n \end{pmatrix} + \mathcal{O}(h^4)
\end{aligned}$$

$$
\begin{aligned}
&= \psi_n \left(-\frac{h^2}{2}y''_{n+1}\right) + (-3\psi_n + \phi_n)\left(-\frac{h^3}{6}y'''\right) + \theta_n\left(-\frac{h^3}{2}Jy''\right) \\
&\quad + \psi_n\left(-\frac{h^3}{2}Jy''_{n+1}\right) + \underbrace{(-3\psi_n + \phi_n)\left(-\frac{h^4}{6}Jy'''\right) + \theta_n\left(-\frac{h^4}{2}J^2y''\right)}_{\mathcal{O}(h^4)} \\
&\quad + \xi_n\left(-\frac{h^2}{2}y''_{n+1}\right) + \gamma_n\left(-\frac{h^3}{6}y'''\right) + \eta_n\left(-\frac{h^3}{2}Jy''\right) + \mathcal{O}(h^4) \\
&= C_{n+1}(h)\begin{pmatrix} \psi_n + \xi_n \\ -3\psi_n + \phi_n + \gamma_n \\ \theta_n + \psi_n + \eta_n \end{pmatrix} + \mathcal{O}(h^4).
\end{aligned}
$$

Therefore, this leads to

$$
\begin{aligned}
\psi_{n+1} &= \psi_n + \xi_n \\
\phi_{n+1} &= -3\psi_n + \phi_n + \gamma_n \\
\theta_{n+1} &= \theta_n + \psi_n + \eta_n.
\end{aligned}
$$

Assuming that the local error coefficient are constant, i.e. $\xi_n = \xi$, $\gamma_n = \gamma$ and $\eta_n = \eta$ for $n = 1, \ldots, k$, the global error coefficient can be rewritten to

$$
\begin{aligned}
\psi_{n+1} &= n\xi \\
\phi_{n+1} &= -3\frac{n(n-1)}{2}\xi + n\gamma \\
\theta_{n+1} &= \frac{n(n-1)}{2}\xi + n\eta.
\end{aligned}
$$

For a projective step from $\{t_k, t_{k+1}\}$ to t_s, there it holds

$$
e_s(h) = (M+1)e_{k+1}(h) + Me_k(h) + d_k^{\mathrm{PFE}}(h) + \mathcal{O}(h^4)
$$

involving the local error for the extrapolation

$$
d_k^{\mathrm{PFE}}(h) = C_{k+1}(h)\begin{pmatrix} M(M+1) \\ M(M^2-1) \\ 0 \end{pmatrix}.
$$

We prove the representation of this extrapolation error. Assuming $y(t_k) = y_k$ and

$y(t_{k+1}) = y_{k+1}$ and using Taylor expansion yields

$$\begin{aligned}
y_s - y(t_s) &= y_{k+1} + M(y_{k+1} - y_k) - y(t_{k+1} + Mh) \\
&= y_{k+1} + M(y_{k+1} - y_k) \\
&\quad - \left(y(t_{k+1}) + Mhy'(t_{k+1}) + \frac{M^2h^2}{2} y''(t_{k+1}) + \frac{M^3h^3}{6} y'''(t_{k+1}) + \mathcal{O}(h^4) \right) \\
&= M(y(t_{k+1} - y(t_{k+1} - h)) \\
&\quad -Mhy'(t_{k+1}) - \frac{M^2h^2}{2} y''(t_{k+1}) - \frac{M^3h^3}{6} y'''(t_{k+1}) + \mathcal{O}(h^4) \\
&= M \left[y(t_{k+1}) - \left(y(t_{k+1}) - hy'(t_{k+1}) + \frac{h^2}{2} y''(t_{k+1}) - \frac{h^3}{6} y'''(t_{k+1}) + \mathcal{O}(h^4) \right) \right] \\
&\quad -Mhy'(t_{k+1}) - \frac{M^2h^2}{2} y''(t_{k+1}) - \frac{M^3h^3}{6} y'''(t_{k+1}) + \mathcal{O}(h^4) \\
&= -\frac{h^2}{2} M(M+1) y''(t_{k+1}) - \frac{h^3}{6} M(M^2-1) y'''(t_{k+1}) + \mathcal{O}(h^4).
\end{aligned}$$

Finally, we obtain

$$\begin{aligned}
e_s(h) &= C_s(h)E_s + \mathcal{O}(h^4) \\
&= (M+1)C_{k+1}(h)E_{k+1} - MC_k(h)E_k + C_{k+1}(h) \begin{pmatrix} M(M+1) \\ M(M^2-1) \\ 0 \end{pmatrix} + \mathcal{O}(h^4) \\
&= (M+1)C_s(h)T(M)E_{k+1} - MC_s(h)T(M+1)E_k + C_s(h)T(M) \begin{pmatrix} M(M+1) \\ M(M^2-1) \\ 0 \end{pmatrix} + \mathcal{O}(h^4) \\
&= C_s(h) \left(T(M) \left((M+1)E_{k+1} + \begin{pmatrix} M(M+1) \\ M(M^2-1) \\ 0 \end{pmatrix} \right) - MT(M+1)E_k \right) + \mathcal{O}(h^4)
\end{aligned}$$

and this leads to

$$\begin{aligned}
E_s &= T(M) \left((M+1)E_{k+1} + \begin{pmatrix} M(M+1) \\ M(M^2-1) \\ 0 \end{pmatrix} \right) - MT(M+1)E_k \\
&= \begin{pmatrix} 1 & 0 & 0 \\ -3M & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \left((M+1) \begin{pmatrix} \psi_{k+1} \\ \phi_{k+1} \\ \theta_{k+1} \end{pmatrix} + \begin{pmatrix} M(M+1) \\ M(M^2-1) \\ 0 \end{pmatrix} \right) \\
&\quad -M \begin{pmatrix} 1 & 0 & 0 \\ -3(M+1) & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \psi_k \\ \phi_k \\ \theta_k \end{pmatrix}
\end{aligned}$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ -3M & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} (M+1)\psi_{k+1} + M(M+1) \\ (M+1)\phi_{k+1} + M(M^2-1) \\ (M+1)\theta_{k+1} \end{pmatrix} - \begin{pmatrix} M\psi_k \\ -3(M+1)M\psi_k + M\phi_k \\ M\theta_k \end{pmatrix}$$

$$= \begin{pmatrix} (M+1)\psi_{k+1} - M\psi_k + M(M+1) \\ 3M(M+1)(\psi_k - \psi_{k+1}) - M\phi_k + (M+1)\phi_{k+1} - M(M+1)(2M+1) \\ (M+1)\theta_{k+1} - M\theta_k \end{pmatrix}.$$

Thus, we obtain

$$\begin{aligned} \psi_s &= (M+1)\psi_{k+1} - M\psi_k + M(M+1) \\ \phi_s &= 3M(M+1)(\psi_k - \psi_{k+1}) - M\phi_k + (M+1)\phi_{k+1} - M(M+1)(2M+1) \\ \theta_s &= (M+1)\theta_{k+1} - M\theta_k. \end{aligned}$$

□

3.2 Teleprojective Forward Euler Method

As Gear and Lee presented in [27], the projective integration process can be iterated by using the outer integrator as an inner integrator within yet another outer integrator. This can be repeated as many times as desired. Figure 3.2 shows an illustration of an Teleprojective Forward Euler with $k = 3$ damping steps, the projective factor $M = 6$ and overall 2 layers, that generates a telescopic PFE step of $100h_0$ at layer 2.

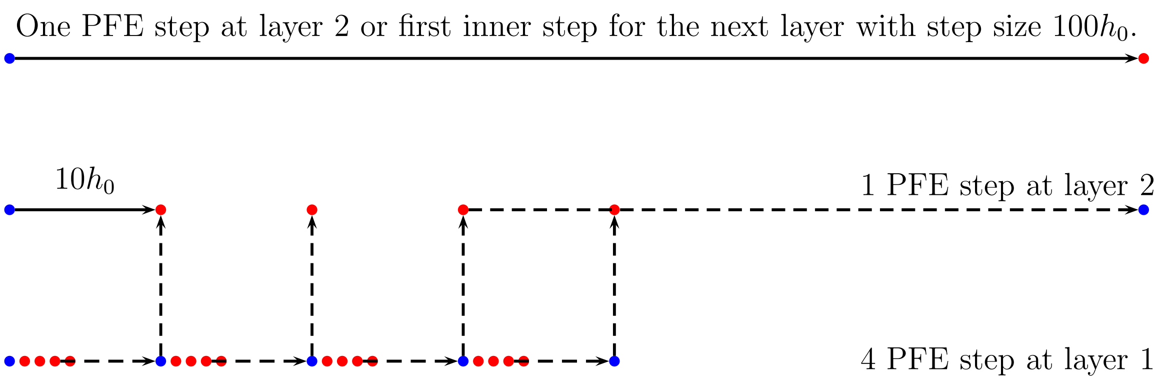

One PFE step at layer 2 or first inner step for the next layer with step size $100h_0$.
$10h_0$
1 PFE step at layer 2
4 PFE step at layer 1

Figure 3.2: PFE with 2 layers, $k = 3$ and $M = 6$.

The stability and accuracy of those (tele-)projective integrators depend on a suitable choice of the parameters k, M, h_0 and the maximal number of layers L for each stiff system. Approximating the direction of the projective step by chord slopes simplifies

Tab. 3.1: Critical values for [0,1]-stable PFE.

k	M_0 (PFE with $L = 1$)	M_∞ (Telescopic PFE with $L > 1$)
1	4.8284	2
2	8.4435	3
3	12.0446	6.6560
4	15.6411	8.3172
5	19.2357	12.2147

the study of stability and additionally, the properties of the outer integrator can be analyzed independently of the choice of the inner integrator, cf. [15]. We assume for simplicity that at each layer q, i.e. q denotes the current layer, the parameters k and M are constant and that all eigenvalues are close to the real axis. This allows us to consider stability only along the real axis and infer instability in its neighborhood by continuity. The choice of h_0 has to satisfy

$$|\rho(h_0\lambda)| < 1$$

for all eigenvalues λ of the system Jacobian $\partial f/\partial y$ and $\rho(h_0\lambda)$ is the error amplifier of the innermost integrator. The linear stability polynomial for one PFE step using only one layer ($L = 1$) is given by Equation (6) in [27], i.e.

$$\sigma_1(\rho) = \rho^{k+1} + M(\rho^{k+1} - \rho^k) = ((M+1)\rho - M)\rho^k.$$

For PFE with $L > 1$ layers, the stability polynomial is

$$\sigma_{j+1}(\rho) = ((M+1)\sigma_j(\rho) - M)\sigma_j^k(\rho)$$

for $j = 1, \ldots, L-1$, cf. Equation (7) in in [27]. The stability region for given parameters k and M can be obtained by plotting $|\sigma(\rho)| = 1$. If this region includes all $\rho \in [0,1]$, the integrator is said to be $[0,1]$-stable. Note that the stability analysis in [27] is sufficient for parabolic problems and for real values of ρ. The major advantage of [0,1]-stable integrators is that no clear time scale separation is required. Lee and Gear provided values of M depending on the number of damping steps to obtain a [0,1]-stable integrator, cf. Table 3.1 and [27]. For $0 \leq M < M_0$ the PFE with $L = 1$ layer is [0,1]-stable and for $0 \leq M < M_\infty$ the PFE with $L > 1$ layers is [0,1]-stable being completely independent of the number of layers. A detailed analysis of the [0,1]-stability of the Teleprojective Forward Euler Method is given in [15]. The implementation of a Teleprojective Forward Euler Method in MATLAB is listed in Listing 3.1 using a function `innerInt()` which is listed in Listing 3.2 representing the inner integrator.

Listing 3.1: pfe.m

```
function [T,Y] = pfe(f,tstart,tend,y0,M,h0,nk,L)

%set problem parameters
nofelem = ceil(tend/((M+nk+1)^L*h0)) +1;
dim = max(size(y0,1),size(y0,2));
nearEquilibrium = false;
tol = 10e-17;

%allocate memory and set initial values
Y = zeros(nofelem,dim);
if (size(y0,1) ~= 1)
    Y(1,:) = y0';
else
    Y(1,:) = y0;
end
T = zeros(1,nofelem);
T(1) = tstart;

%allocate step
step = zeros(nk+2,dim);

for j=1:nofelem-1
    %check if current point is near equilibrium
    if (~nearEquilibrium)
    step(1,:) = Y(j,:);
    t = T(j);

    %perform nk+1 damping steps
    for i = 1:nk+1
      [t,step(i+1,:)] = innerInt(f,t,step(i,:),M,h0,nk,L-1);
    end

    %perform a projective step using chord slope
    T(j+1) = t + M*(M+nk+1)^(L-1)*h0;
    Y(j+1,:) = (1+M)*step(end,:) - M*step(end-1,:);
    if( norm(abs(Y(j+1,:)-Y(j,:))) < tol )
        nearEquilibrium = true;
    end

    else
       Y(j+1,:) = Y(j,:);
```

```
42          T(j+1) = T(j) + (nk+1+M)^L*h0;
43      end
44  end
```

Line 1: Calling the function `pfe()` with the following parameters:

`f` - function handle, i.e. the right-hand side of the ODE system.

`tstart`,`tend` - time interval [`tstart`,`tstart`] in which the integration will be performed.

`y0` - initial value.

`M` - projective factor.

`h0` - innermost step size h_0.

`nk` - number of damping steps k.

`L` - number of layers L.

Line 3-20: Set initial value and allocate memory for speed up.

Line 24,36-38: If the current point is close to the equilibrium, we do not continue calculating new values.

Line 29-31: Performing $k+1$ damping steps using an inner integrator `innerInt()`.

Line 35: Performing one projective step

$$y_{n+s} = y_{n+k+1} + M(y_{n+k+1} - y_{n+k})$$

whereas $s = k + 1 + M$.

Listing 3.2: innerInt.m

```
function [t,y] = innerInt(f,t0,y0,M,h0,nk,q)

if (q == 0)
    %innermost layer: performing conventional forward euler
    y = y0 + h0*f(t0,y0)';
    t = t0 + h0;
elseif (q > 0)
    y = y0; t = t0;

```

```
10     %perform nk damping steps
11     y_nk = y;
12     for i = 1:nk
13         [t, y_nk] = innerInt(f,t,y_nk,M,h0,nk,q-1);
14     end
15     %calculate y(t_{nk+1})
16     [t, y_nkp1] = innerInt(f,t,y_nk,M,h0,nk,q-1);
17 
18     %perform a projective step using chord slope
19     t = t + M*(nk+1+M)^(q-1)*h0;
20     y = y_nkp1 + M*(y_nkp1 - y_nk);
21 end
```

Line 1: Calling the function `innerInt()` with the following parameters:

> `f,y0,M,h0,nk` - same as in `pfe()`.
>
> `t0` - start time.
>
> `q` - current layer.

Line 11-16: Performing $k+1$ damping steps.

Line 19-20: Performing a projective step. Hence, the overall step size of one step at current layer q is

$$(k+1+M)^q h_0.$$

Similar to the error analysis for the PFE with $L=1$, we give an analogical result for $L>1$. Before, we take a look at the local error at layer $q+1$. There it holds

$$C_1^{q+1}(sh) = C_s^q(h)R(s)$$

whereas the superscript q resp. $q+1$ corresponds to the current layer and the scaling operator R defined as

$$R(s) := \begin{pmatrix} s^2 & 0 & 0 \\ 0 & s^3 & 0 \\ 0 & 0 & s^3 \end{pmatrix}.$$

Lemma 3.2.1 (Global Error for a PFE Step on Layer $L > 1$) *Assume that at each layer the local error coefficients are constant, i.e. $\xi_n^q = \xi^q$, $\gamma_n^q = \gamma^q$ and $\eta_n^q = \eta^q$ for all $q = 0, 1, \dots, L$ and $n = 1, \dots, k$. Then the following formulas for computing the error coefficients at each layer $q = 0, 1, \dots, L$ hold*

$$\psi_0^q = \phi_0^q = \theta_0^q = 0$$

and

$$\begin{aligned}
\psi_{n+1}^q &= \psi_n^q + \xi^q \\
\phi_{n+1}^q &= -3\psi_n^q + \phi_n^q + \gamma^q \\
\theta_{n+1}^q &= \theta_n^q + \psi_n^q + \eta^q \\
\psi_s^q &= (M+1)\psi_{k+1}^q - M\psi_k^q + M(M+1) \\
\phi_s^q &= 3M(M+1)(\psi_k^q - \psi_{k+1}^q) - M\phi_k^q + (M+1)\phi_{k+1}^q - M(M+1)(2M+1) \\
\theta_s^q &= (M+1)\theta_{k+1}^q - M\theta_k^q.
\end{aligned} \tag{3.1}$$

With these values, the local error coefficients on the next layer can be computed via

$$\xi^{q+1} = \frac{\psi_s^q}{s^2}, \qquad \gamma^{q+1} = \frac{\phi_s^q}{s^3}, \qquad \eta^{q+1} = \frac{\theta_s^q}{s^3}.$$

Proof. The identities in (3.1) can be derived immediately from Lemma 3.1.1. Moreover, it holds

$$\begin{pmatrix} \xi^{q+1} \\ \gamma^{q+1} \\ \eta^{q+1} \end{pmatrix} = R^{-1}(s)E_s^q = \begin{pmatrix} \frac{1}{s^2} & 0 & 0 \\ 0 & \frac{1}{s^3} & 0 \\ 0 & 0 & \frac{1}{s^3} \end{pmatrix} \begin{pmatrix} \psi_s^q \\ \phi_s^q \\ \theta_s^q \end{pmatrix}.$$

□

Note that if the innermost integrator is Forward Euler, then it holds $\xi_j^0 = 1$, $\gamma_j^0 = -2$ and $\eta_j^0 = 0$.

3.3 Projective Runge–Kutta Method

The previously presented algorithms are only of first-order accuracy. Analogical to the conventional trapezoidal method for ODEs, Lee and Gear derived a second-order accurate projective integrator in [27]. The main idea is to perform a predictor-corrector pattern. One step at the outermost layer L with step size H of the *Projective Runge–Kutta Method* (PRK) using PFE as an inner integrator can be performed as follows

(i) Start from $y_n = y(t_n)$. Perform $k + 1$ damping steps using an inner integrator with step size $h = H/s$ to obtain y_{n+k} and y_{n+k+1}.

(ii) Perform one projective step to gain a predicted value

$$y^P_{n+s} = y_{n+k+1} + M(y_{n+k+1} - y_{n+k}).$$

(iii) Start from y^P_{n+s} and perform $k_1{+}1$ damping steps to gain $y^{\mathrm{P}}_{n+s+k_1}$ and $y^{\mathrm{P}}_{n+s+k_1+1}$.

(iv) Perform a corrector step via

$$y_{n+s} = y_{n+k+1} + M\left(\alpha(y_{n+k+1} - y_{n+k}) + (1-\alpha)(y^P_{n+k_1+1} - y^P_{n+k_1})\right)$$

with $s = M + k + 1$, a weighted average of chord slopes using a real scalar α and k and k_1 being the number of damping steps starting from y_n resp. y^P_{n+s}. M is the projective multiplier, cf. the previous sections. Note that we always choose $k_1 = k$ in our implementation.

Figure 3.3 illustrates one PRK step. The red dashed arrow shows a projective step. Additionally, after this projective step more damping steps are performed to gain information about the future behavior of the solution trajectory. Hence, a PRK step (green line) can be performed with these values (green dots). The blue dots depict the start points and the red dots the solutions points after a damping step (black dashed arrow).

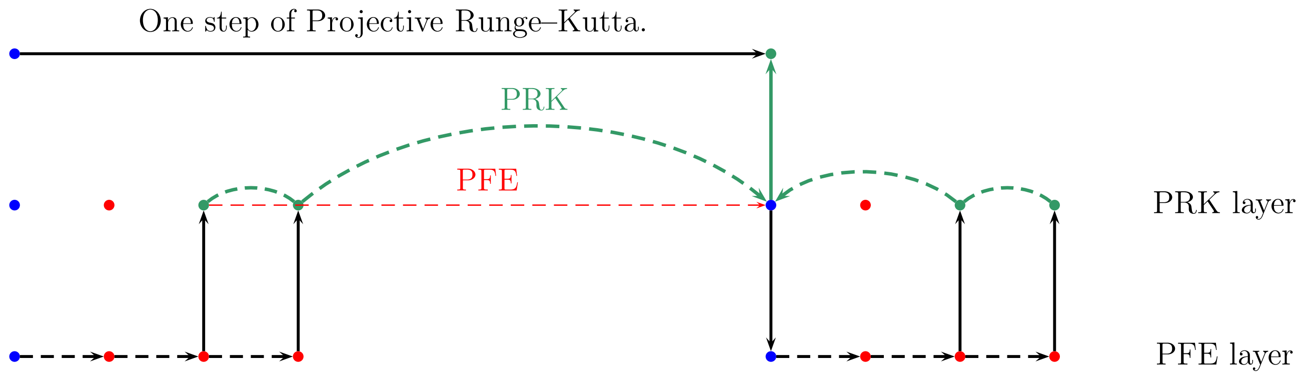


Figure 3.3: PRK as an outer integrator for PFE with $k = 2$ damping steps.

Such predictor-corrector patterns are useful to estimate an error because we can make a difference between the predicted and corrected value. The stability polynomial is given by

$$\sigma_{\mathrm{PRK}}(\rho) = \rho^{k+1} + M\left(\alpha(\rho^{k+1} + \rho^k) + (1-\alpha)(\rho^{k_1+1} - \rho^{k_1})\sigma_{\mathrm{PFE}}(\rho)\right)$$

as provided in [27], Equation (12). Note that $\sigma_{\mathrm{PFE}}(\rho)$ is the stability polynomial of the PFE Method. Lee and Gear provide values of M depending on the number of damping steps to obtain a [0,1]-stable PRK integrator with Forward Euler as an inner integrator, cf. Table 3.2. This means at $q = 1$ we perform PRK and at $q = 0$ we perform the conventional Forward Euler. Thus, if we choose $M < M_0$, we obtain a [0,1]-stable PRK Method.

We give a detailed proof based on ideas of Lee and Gear as presented in [26] of the following result that guarantees the second-order accuracy depending on the choice of α.

Tab. 3.2: Critical values for [0,1]-stable PRK with $L = 1$.

k	1	2	3	4	5
M_0	7.7958	14.1501	20.4726	26.7848	33.0924

Lemma 3.3.1 (General Choice of α) *Consider a PRK integrator at the layer q with step size H and let $h = H/s$ be the step size of the damping steps performed by a PFE at layer $q-1$ and choose*

$$\alpha = \frac{-\psi_{k+1}^{q-1} - M(\psi_{k_1+1}^{q-1} - \psi_{k_1}^{q-1}) + M(M+1+2k_1)}{M(\psi_{k+1}^{q-1} - \psi_{k}^{q-1} - \psi_{k_1+1}^{q-1} + \psi_{k_1}^{q-1} + 2(M+1+k_1))}.$$

Then, the outer PRK integrator is of second-order accuracy.

Proof. By recalling the definitions of E_s^q, D_s^q and C_s in Section 3.1, we get

$$D_P^q(H) = C_s(h)E_s^{q-1}.$$

Furthermore, after k_1 damping steps with step size h at layer $q-1$, we obtain for the local error starting from a correct value y_s^P

$$e_{k_1}^{q-1}(h) = C_{s+k_1}(h)E_{k_1}^{q-1} + \mathcal{O}(h^4).$$

Thus, the error starting from a correct value y_n is

$$y_{s+k_1} - y(t_{s+k_1}) \quad = \quad (I + hk_1J)D_P^q(H) + e_{k_1}^{q-1}(h) + \mathcal{O}(h^4)$$

because

$$\begin{aligned}
y_{s+k_1} - y(t_{s+k_1}) &= e^{q-1}_{s+k_1}(h) + \mathcal{O}(h^4) \\
&= (I + hJ)^{k_1} e^{q-1}_s(h) + \underbrace{\sum_{i=1}^{k_1} d^{q-1}_{s+k_1-i}(h)}_{=e^{q-1}_{k_1}(h)} + \mathcal{O}(h^4) \\
&= \left(\sum_{k=0}^{k_1} \binom{k_1}{k} h^k I^{k_1-k} J^k \right) e^{q-1}_s(h) + e^{q-1}_{k_1}(h) + \mathcal{O}(h^4) \\
&= (I + k_1 hJ) D^q_P(H) + e^{q-1}_{k_1}(h) + \mathcal{O}(h^4).
\end{aligned}$$

Note that the last equality holds, because terms of order 4 or higher are ignored. Analogically, we get by substituting k_1 with $k_1 + 1$

$$y_{s+k_1+1} - y(t_{s+k_1+1}) = (I + h(k_1+1)J) D^q_P(H) + e^{q-1}_{k_1+1}(h) + \mathcal{O}(h^4).$$

Now, we take a look at the formula of PRK and note that

$$\begin{aligned}
y(t_s) &= y(t_{k+1}) + M \big(\alpha(y(t_{k+1}) - y(t_k)) \\
&\quad + (1-\alpha)(y(t_{s+k_1+1}) - y(t_{s+k_1}))\big) - d^{\mathrm{PRK}}_s(h,\alpha) + \mathcal{O}(h^4), \qquad (3.2)
\end{aligned}$$

whereas the PRK discretization error $d^{\mathrm{PRK}}_s(h,\alpha)$ is given by

$$d^{\mathrm{PRK}}_s(h,\alpha) = C_s(h) \begin{pmatrix} 2M\alpha(M + k_1 + 1) - M(M + 2k_1 + 1) \\ 3M\alpha(k_1 - M)(M + k_1 + 1) + M(M^2 - 3k_1(k_1+1) - 1) \\ 0 \end{pmatrix}.$$

We verify this representation by using the Taylor expansions of the following terms

$$\begin{aligned}
y(t_{k+1}) &= y(t_s - Mh) \\
y(t_k) &= y(t_s - (M+1)h) \\
y(t_{s+k_1+1}) &= y(t_s + (k_1+1)h) \\
y(t_{s+k_1}) &= y(t_s + k_1 h).
\end{aligned}$$

Equation (3.2) and the Taylor expansions of these terms lead to

$$\begin{aligned}
d_s^{\text{PRK}}(h,\alpha) &= y(t_{k+1}) - y(t_k) + M\Big(\alpha(y(t_{k+1}) - y(t_k)) + (1-\alpha)(y(t_{s+k_1+1}) - y(t_{s+k_1}))\Big) \\
&= y(t_s) - Mhy'(t_s) + \frac{M^2h^2}{2}y''(t_s) - \frac{M^3h^3}{6}y'''(t_s) - y(t_s) \\
&\quad +\alpha M\left(y(t_s) - Mhy'(t_s) + \frac{M^2h^2}{2}y''(t_s) - \frac{M^3h^3}{6}y'''(t_s)\right) \\
&\quad -\alpha M\left(y(t_s) - (M+1)hy'(t_s) + \frac{(M+1)^2h^2}{2}y''(t_s) - \frac{(M+1)^3h^3}{6}y'''(t_s)\right) \\
&\quad +(1-\alpha)M\left(y(t_s) + (k_1+1)hy'(t_s) + \frac{(k_1+1)^2h^2}{2}y''(t_s) + \frac{(k_1+1)^3h^3}{6}y'''(t_s)\right) \\
&\quad -(1-\alpha)M\left(y(t_s) + k_1hy'(t_s) + \frac{k_1^2h^2}{2}y''(t_s) + \frac{k_1^3h^3}{6}y'''(t_s)\right) + \mathcal{O}(h^4) \\
&= hy'(t_s)\Big(-M - \alpha M^2 + \alpha M^2 + \alpha M + k_1M \\
&\quad\ + M - \alpha k_1 M - \alpha M - k_1 M + \alpha k_1 M\Big) \\
&\quad -\frac{h^2}{2}y''(t_s)\Big(-\alpha M^3 - M^2 + \alpha M^3 + 2\alpha M^2 + \alpha M \\
&\quad\ - (1-\alpha)(M(k_1^2 + 2k_1 + 1) - Mk_1^2)\Big) \\
&\quad -\frac{h^3}{6}y''(t_s)\Big(-\alpha M^4 + M^3 - \alpha M(M^3 + 3M^2 + 3M + 1) \\
&\quad\ - (1-\alpha)(M(k_1^3 + 3k_1^2 + 3k_1 + 1) - Mk_1^3)\Big) + \mathcal{O}(h^4) \\
&= -\frac{h^2}{2}y''(t_s)\Big(-M(M+2k_1+1) + 2\alpha M(M+k_1+1)\Big) \\
&\quad -\frac{h^3}{6}y'''(t_s)\Big(3\alpha M(-M^2 - M + k_1^2 + k_1) + M(M^2 - 3k_1^2 - 3k_1 - 1)\Big) + \mathcal{O}(h^4) \\
&= C_s(h)\begin{pmatrix} 2M\alpha(M+k_1+1) - M(M+2k_1+1) \\ 3M\alpha(k_1-M)(M+k_1+1) + M(M^2 - 3k_1(k_1+1) - 1) \\ 0 \end{pmatrix} + \mathcal{O}(h^4).
\end{aligned}$$

Note, that terms of higher order than 3 are ignored. Besides, it holds

$$\begin{aligned}
e_s^{q-1}(h) &= y_s - y(t_s) \\
&= e_{k+1}^{q-1}(h) + M\Big(\alpha(e_{k+1}^{q-1}(h) - e_k^{q-1}(h)) \\
&\quad + (1-\alpha)(e_{s+k_1+1}^{q-1}(h) - e_{s+k_1}^{q-1}(h))\Big) + d_s^{\text{PRK}}(h,\alpha) + \mathcal{O}(h^4). \quad (3.3)
\end{aligned}$$

Thus, we only have to find representations for $e_{k+1}^{q-1}(h)$, $e_k^{q-1}(h)$, $e_{s+k_1+1}^{q-1}(h)$ and

$e^{q-1}_{s+k_1+1}(h)$ to get a formula for the error of one PRK step. There it holds

$$\begin{aligned}
e^{q-1}_{k+1}(h) &= C_{k+1}(h)E^{q-1}_{k+1} + \mathcal{O}(h^4) = C_s(h)T(M)E^{q-1}_{k+1} + \mathcal{O}(h^4) \\
&= C_s(h)\begin{pmatrix} \psi^{q-1}_{k+1} \\ \phi^{q-1}_{k+1} - 3M\psi^{q-1}_{k+1} \\ \theta^{q-1}_{k+1} \end{pmatrix} + \mathcal{O}(h^4), \\
e^{q-1}_{k}(h) &= C_{k}(h)E^{q-1}_{k} + \mathcal{O}(h^4) = C_s(h)T(M+1)E^{q-1}_{k} + \mathcal{O}(h^4) \\
&= C_s(h)\begin{pmatrix} \psi^{q-1}_{k} \\ \phi^{q-1}_{k} - 3(M+1)\psi^{q-1}_{k} \\ \theta^{q-1}_{k} \end{pmatrix} + \mathcal{O}(h^4)
\end{aligned}$$

together with

$$\begin{aligned}
e^{q-1}_{s+k_1}(h) &= y_{s+k_1} - y(t_{s+k_1}) \\
&= (I + hk_1J)D^q_P(H) + e^{q-1}_{k_1}(h) + \mathcal{O}(h^4) \\
&= (I + hk_1J)C_s(h)E^{q-1}_s + C_{s+k_1}(h)E^{q-1}_{k_1} + \mathcal{O}(h^4) \\
&= C_s(h)E^{q-1}_s + hk_1JC_s(h)E^{q-1}_s + C_s(h)T(-k_1)E^{q-1}_{k_1} + \mathcal{O}(h^4) \\
&= C_s(h)E^{q-1}_s + C_s(h)\begin{pmatrix} 0 \\ 0 \\ k_1\psi_s \end{pmatrix} + C_s(h)T(-k_1)E^{q-1}_{k_1} + \mathcal{O}(h^4) \\
&= C_s(h)\left(\begin{pmatrix} \psi^{q-1}_s \\ \phi^{q-1}_s \\ \theta^{q-1}_s + k_1\psi^{q-1}_s \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 3k_1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} \psi^{q-1}_{k_1} \\ \phi^{q-1}_{k_1} \\ \theta^{q-1}_{k_1} \end{pmatrix}\right) + \mathcal{O}(h^4) \\
&= C_s(h)\begin{pmatrix} \psi^{q-1}_s + \psi^{q-1}_{k_1} \\ \phi^{q-1}_s + \phi^{q-1}_{k_1} + 3k_1\psi^{q-1}_{k_1} \\ \theta^{q-1}_s + \theta^{q-1}_{k_1} + k_1\psi^{q-1}_s \end{pmatrix} + \mathcal{O}(h^4)
\end{aligned}$$

and with an analogical result

$$e^{q-1}_{s+k_1+1}(h) = C_s(h)\begin{pmatrix} \psi^{q-1}_s + \psi^{q-1}_{k_1+1} \\ \phi^{q-1}_s + \phi^{q-1}_{k_1+1} + 3(k_1+1)\psi^{q-1}_{k_1+1} \\ \theta^{q-1}_s + \theta^{q-1}_{k_1+1} + (k_1+1)\psi^{q-1}_s \end{pmatrix} + \mathcal{O}(h^4).$$

Again, note that terms of higher order than 3 are ignored. Now, we are able to determine the local error coefficients of one PRK step with step size H at layer q by using equation (3.3) through

$$C_1(H)\begin{pmatrix} \xi^{\mathrm{PRK}} \\ \gamma^{\mathrm{PRK}} \\ \eta^{\mathrm{PRK}} \end{pmatrix} = e^{q-1}_s(h).$$

This leads to

$$\begin{aligned}
\xi^{\mathrm{PRK}} &= \psi_{k+1}^{q-1} + \alpha M(\psi_{k+1}^{q-1} - \psi_k^{q-1}) + (1-\alpha)M(\psi_{k_1+1}^{q-1} - \psi_{k_1}^{q-1}) \\
&\quad +2\alpha M(M + k_1 + 1) - M(M + 2k_1 + 1) \\
\gamma^{\mathrm{PRK}} &= \phi_{k+1}^{q-1} - 3M\psi_{k+1}^{q-1} + \alpha M(\phi_{k+1}^{q-1} - \phi_k^{q-1} - 3M\psi_{k+1}^{q-1} + 3(M+1)\psi_k^{q-1}) \\
&\quad +(1-\alpha)M(\phi_{k_1+1}^{q-1} - \phi_{k_1}^{q-1} + 3(k_1+1)\psi_{k_1+1}^{q-1} - 3k_1\psi_{k_1}^{q-1}) \\
&\quad +3\alpha M(k_1 - M)(M + k_1 + 1) + M(M^2 - 3k_1(k_1+1) - 1) \\
\eta^{\mathrm{PRK}} &= \theta_{k+1}^{q-1} + \alpha M(\theta_{k+1}^{q-1} - \theta_k^{q-1}) + (1-\alpha)M(\theta_{k_1+1}^{q-1} - \theta_{k_1}^{q-1} + \psi_s^{q-1}).
\end{aligned}$$

To gain a second-order accurate integrator, we have to choose an α such that the second error coefficient ξ^{PRK} vanishes. Rewrite the PRK discretization error to

$$d_s^{\mathrm{PRK}}(h,\alpha) = C_s(h) D^{\mathrm{PRK}} \begin{pmatrix} \alpha M \\ 1 \end{pmatrix} + \mathcal{O}(h^4)$$

with

$$D^{\mathrm{PRK}} = \begin{pmatrix} 2(M+1+k_1) & -M(M+1+2k_1) \\ 3(k_1 - M)(M+1+k_1) & M(M^2 - 3k_1(k_1+1) - 1) \\ 0 & 0 \end{pmatrix}.$$

Similar to that, rewrite the remaining error part as follows

$$\begin{aligned}
& e_{k+1}^{q-1}(h) + M\left(\alpha(e_{k+1}^{q-1}(h) - e_k^{q-1}(h)) + (1-\alpha)(e_{s+k_1+1}^{q-1}(h) - e_{s+k_1}^{q-1}(h))\right) \\
= \;& C_s(h) E^{\mathrm{PRK}} \begin{pmatrix} \alpha M \\ 1 \end{pmatrix} + \mathcal{O}(h^4)
\end{aligned}$$

with

$$E^{\mathrm{PRK}} = \begin{pmatrix} \psi_{k+1}^{q-1} - \psi_k^{q-1} - \psi_{k_1+1}^{q-1} + \psi_{k_1}^{q-1}, & \psi_{k+1}^{q-1} + M(\psi_{k_1+1}^{q-1} - \psi_{k_1}^{q-1}) \\ \phi_{k+1}^{q-1} - \phi_k^{q-1} - 3M\psi_{k+1}^{q-1} + 3(M+1)\psi_k^{q-1} & \phi_{k+1}^{q-1} - 3M\psi_{k+1}^{q-1} + M\left(\phi_{k_1+1}^{q-1} - \phi_{k_1}^{q-1}\right. \\ -\phi_{k_1+1}^{q-1} + \phi_{k_1}^{q-1} - 3(k_1+1)\psi_{k_1+1}^{q-1} - 3k_1\psi_{k_1}^{q-1}, & \left. +3(k_1+1)\psi_{k_1+1}^{q-1} - 3k_1\psi_{k_1}^{q-1}\right) \\ \theta_{k+1}^{q-1} - \theta_k^{q-1} - \theta_{k_1+1}^{q-1} + \theta_{k_1}^{q-1} - \psi_s^{q-1}, & \theta_{k+1}^{q-1} + M(\theta_{k_1+1}^{q-1} - \theta_{k_1}^{q-1} + \psi_s^{q-1}) \end{pmatrix}.$$

This allows us to write the error compactly as

$$e_s^{q-1}(h) = C_s(h)(E^{\mathrm{PRK}} + D^{\mathrm{PRK}}) \begin{pmatrix} \alpha M \\ 1 \end{pmatrix} + \mathcal{O}(h^4).$$

In order to get second-order accuracy, we take a look at the following linear equation system

$$C_s(h)(E^{\mathrm{PRK}} + D^{\mathrm{PRK}}) \begin{pmatrix} \alpha M \\ 1 \end{pmatrix} \overset{!}{=} -\xi^{\mathrm{PRK}} \frac{H^2}{2} y_s'' - \gamma^{\mathrm{PRK}} \frac{H^3}{6} y''' - \eta^{\mathrm{PRK}} \frac{H^3}{2} J y''$$

and choose α such that ξ^{PRK} vanishes. By defining

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} := E^{\mathrm{PRK}} + D^{\mathrm{PRK}},$$

it holds

$$\alpha M a_{11} + a_{12} \overset{!}{=} 0 \qquad \Leftrightarrow \qquad \alpha = \frac{-a_{12}}{M a_{11}}.$$

Finally, we obtain

$$\alpha = \frac{-\psi_{k+1}^{q-1} - M(\psi_{k_1+1}^{q-1} - \psi_{k_1}^{q-1}) + M(M+1+2k_1)}{M(\psi_{k+1}^{q-1} - \psi_{k}^{q-1} - \psi_{k_1+1}^{q-1} + \psi_{k_1}^{q-1} + 2(M+1+k_1))}$$

and the third order coefficients

$$\gamma^{\mathrm{PRK}} = \frac{1}{s^3}\left(\frac{-a_{21}}{a_{11}}a_{21} + a_{22}\right), \qquad \eta^{\mathrm{PRK}} = \frac{1}{s^3}\left(\frac{-a_{21}}{a_{11}}a_{31} + a_{32}\right).$$

□

Lemma 3.3.2 (Special Choice of α) *Consider a PRK integrator at the layer q and let $k_1 = k$, $\xi_n^{q-1} = \xi^{q-1}$, $\gamma_n^{q-1} = \gamma^{q-1}$ and $\eta_n^{q-1} = \eta^{q-1}$ for $n = 1, \dots, k$. Then it holds*

$$\alpha = \frac{-(M+k+1)\xi^{q-1} + M(M+1+2k)}{2M(M+1+k)}.$$

Proof. The requirements lead to

$$\psi_{k_1}^{q-1} = \psi_{k}^{q-1}, \quad \psi_{k_1+1}^{q-1} = \psi_{k+1}^{q-1}$$

and

$$\psi_{k+1}^{q-1} = (k+1)\xi^{q-1}.$$

Then, it holds

$$\begin{aligned} \alpha &= \frac{Mk\xi^{q-1} - (M+1)(k+1)\xi^{q-1} + M(M+1+2k)}{2M(M+1+k)} \\ &= \frac{-(M+k+1)\xi^{q-1} + M(M+1+2k)}{2M(M+1+k)}. \end{aligned}$$

□

The implementation of a Projective Runge–Kutta Method in MATLAB is listed in Listing 3.3 using a function `innerInt()` which is already listed in Listing 3.2 representing the inner PFE integrator and a function `getAlpha()` which is listed in Listing 3.4 calculating the real scalar α as proposed in Lemma 3.3.2. Additionally, we implemented a PRK integrator in `C++`, cf. appendix: Section C and D.

Listing 3.3: prk.m

```
function [T,Y] = prk(f,tstart,tend,y0,M,h0,nk,L)

%set problem parameters
alpha = getAlpha(M,nk,L);
```

```
nofelem = ceil(tend/((M+nk+1)^L*h0)) +1;
dim = max(size(y0,1),size(y0,2));
nearEquilibrium = false;
tol = 10e-17;

%allocate memory and set initial values
Y = zeros(nofelem,dim);
if (size(y0,1) ~= 1)
    Y(1,:) = y0';
else
    Y(1,:) = y0;
end
T = zeros(1,nofelem);
T(1) = tstart;

%allocate step
step = zeros(nk+2,dim);
step_pred = zeros(nk+2,dim);

for j=1:nofelem-1
    if (~nearEquilibrium)

    %-- predictor step --%
    t = T(j);
    step(1,:) = Y(j,:);

    %perform nk+1 damping steps
    for i = 1:nk+1
        [t,step(i+1,:)] = innerInt(f,t,step(i,:),M,h0,nk,L
           -1);
    end

    %perform a projective step using chord slope
    t = t + M*(M+nk+1)^(L-1)*h0;
    T(j+1) = t;
    step_pred(1,:) = (1+M)*step(end,:) - M*step(end-1,:);

    %perform nk+1 damping steps starting from y_s
    for i = 1:nk+1
        [t,step_pred(i+1,:)] = innerInt(f,t,step_pred(i,:),M
           ,h0,nk,L-1);
    end
```

```
45
46      %-- corrector step --%
47      Y(j+1,:) =  (1+M*alpha) *step(end,:)- M*alpha*step(end
           -1,:) +...
48                (1-alpha)*M*step_pred(end,:) - ...
49                (1-alpha) * M*step_pred(end-1,:);
50        if( norm(abs(Y(j+1,:)-Y(j,:))) < tol )
51            nearEquilibrium = true;
52        end
53      else
54         Y(j+1,:) = Y(j,:);
55         T(j+1) = T(j) + (nk+1+M)^L*h0;
56      end
57  end
```

Line 1:	Calling the function `prk()` with the same parameters as in function `pfe()`, cf. Section 3.1.
Line 4:	Calculate α to guarantee a second-order accuracy as proved in Lemma 3.3.2.
Line 25,53-56:	If the current point is close to the equilibrium, we do not continue calculating new values.
Line 31-34:	Performing $k+1$ damping steps using an inner integrator `innerInt()` to obtain y_{n+k+1} and y_{n+k}.
Line 37-39:	Performing one projective step to obtain a predicted value $$y^P_{n+s} = y_{n+k+1} + M(y_{n+k+1} - y_{n+k})$$ whereas $s = k+1+M$.
Line 42-44:	Performing another $k+1$ damping steps starting from y^P_{n+s} to gain $y^P_{n+s+k+1}$ and y^P_{n+s+k}.
Line 37-39:	Performing a corrector step with a weighted average of chord slopes $$y_{n+s} = y_{n+k+1} + M\left(\alpha(y_{n+k+1} - y_{n+k}) + (1-\alpha)(y^P_{n+k_1+1} - y^P_{n+k_1})\right).$$

Listing 3.4: getAlpha.m

```
1  function alpha = getAlpha(M,k,L)
2
3  s = M + k + 1;
4  xsi = 1; %xsi_0 if forward euler is used at innermost layer
```

```

for i = 1:L
    xsi = xsi/s + M*(M+1)/s^2;
end

alpha = ((-M-k-1)*xsi + M*(M+1+2*k))/(2*M*s);
```

Chapter 4
Numerical Results and Comparison to Other Methods

In this chapter, we give an overview of several tests and analyze the numerical behavior of the algorithms which are presented in Chapter 3. Furthermore, we compare the projective integrators with a BDF integrator implemented by Dominik Skanda, cf. [29], and in addition, we compare these methods with a BDF integrator for the corresponding reduced model applying a model reduction technique as discussed in Section 2.3 by using the software `MoRe` by Jochen Siehr, cf. [28]. All tests are performed on an Apple MacBook Pro with the following specifications:

Kernel:	Intel Core 2 Duo, 2.26 GHz
RAM:	4GB DDR3 RAM, 1067 MHz
OS:	Mac OS X 10.6.8, Build 10K549
Matlab:	Version 7.9.0 (R2009b)
`g++`:	Version 4.7.1.

4.1 Projective Forward Euler vs. Projective Runge–Kutta

In this section, we compare the PFE Method with the PRK Method. In order to look at the error between a *true* or *correct* solution and the solution calculated by PFE or PRK, i.e.

$$\left|\left|y^{\text{cor}}(t_k) - y_k^{\text{PFE}}\right|\right| \qquad \text{and} \qquad \left|\left|y^{\text{cor}}(t_k) - y_k^{\text{PRK}}\right|\right|,$$

we assume that the result of Matlab's `ode23s` using the smallest possible tolerance is the *correct* solution to have a reference. We consider the Davis–Skodje model, cf. Section 2.4, i.e.

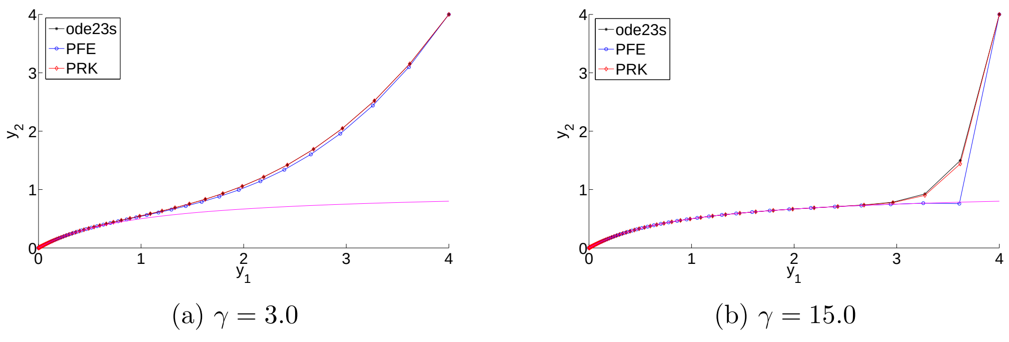
$$\begin{aligned} \dot{y}_1(t) &= -y_1(t) \\ \dot{y}_2(t) &= -\gamma y_2(t) + \frac{(\gamma-1)y_1(t) + \gamma y_1^2(t)}{(1+y_1(t))^2} \end{aligned}$$

with $\gamma \in \{3.0, 15.0\}$, $t \in [0, 10]$ and the following test setups involving various initial values and parameters M and k for the integrators:

Tab. 4.1: Test cases comparing PFE with PRK.

Test case 1	$M = 6,\ k = 3$ and $y_0 = (4,\ 4)^T$
Test case 2	$M = 8,\ k = 3$ and $y_0 = (4,\ 4)^T$
Test case 3	$M = 8,\ k = 4$ and $y_0 = (4,\ 4)^T$
Test case 4	$M = 12,\ k = 4$ and $y_0 = (4,\ 4)^T$
Test case 5	$M = 6,\ k = 3$ and $y_0 = (3,\ 0.2)^T$
Test case 6	$M = 8,\ k = 3$ and $y_0 = (3,\ 0.2)^T$
Test case 7	$M = 8,\ k = 4$ and $y_0 = (3,\ 0.2)^T$
Test case 8	$M = 12,\ k = 4$ and $y_0 = (3,\ 0.2)^T$.

Besides, we choose $L = 2$ and $h_0 = 0.001$ for every test case. Figure 4.1a and 4.1b depict the solution trajectories in test case 1 for $\gamma = 3$ resp. $\gamma = 15$. Note that the SIM is represented by the magenta line. The corresponding error plots are depicted in Figure 4.2a and 4.2b. In the same way, the plots of the remaining test cases are listed in the appendix, cf. Section A. It is obvious, that the error of the PRK Method is always smaller than the error of the PFE, cf. for example Figure 4.2a and 4.2b. Furthermore, in general choosing more damping steps does not lead to a higher accuracy, cf. Figure 4.4, but it allows us to choose a larger projective step which ends up in a better performance, because we need fewer integration steps. Both algorithms react very sensitive on the choice of the parameters M, L, k and h_0. For example Figure 4.3 shows, that for $M = 12$ and $k = 4$ the PFE begins to oscillate and enters negative values, which are prohibited if we consider concentrations of chemical species, while the PRK Method still fits the solution trajectories almost perfect. This demonstrates, that in this case the PFE is not [0,1]-stable anymore, cf. Table 3.1 listing critical values for [0,1]-stable PFE integrators.



(a) $\gamma = 3.0$ (b) $\gamma = 15.0$

Figure 4.1: Plots of the solutions in test case 1.

Table 4.2 provides the runtime for each test case. It is remarkable and expectable that the runtime of PRK, due to the predictor-corrector schema, is slightly higher than the runtime of PFE. The runtime of `ode23s` is very large, because we calculate

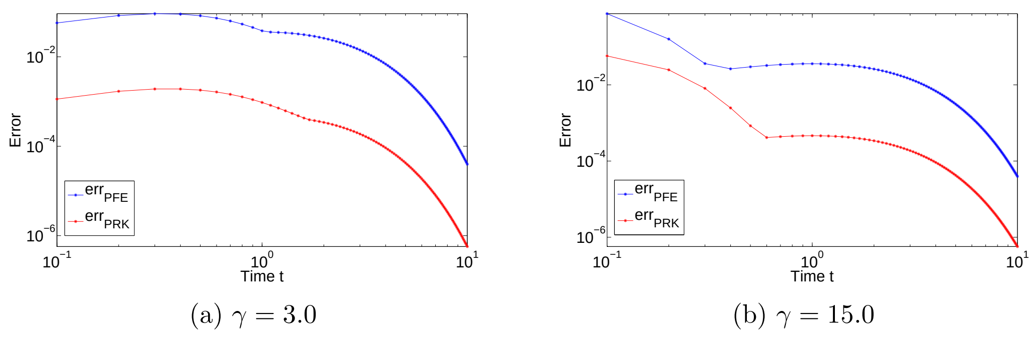


Figure 4.2: Error plots of test case 1.

the solution with the smallest possible tolerance by using the same time discretization as PRK or PFE. According to all mentioned facts, it is worthwhile to perform an integration via PRK, because we gain a better accuracy, although we have more function evaluation and thus, a little higher runtime.

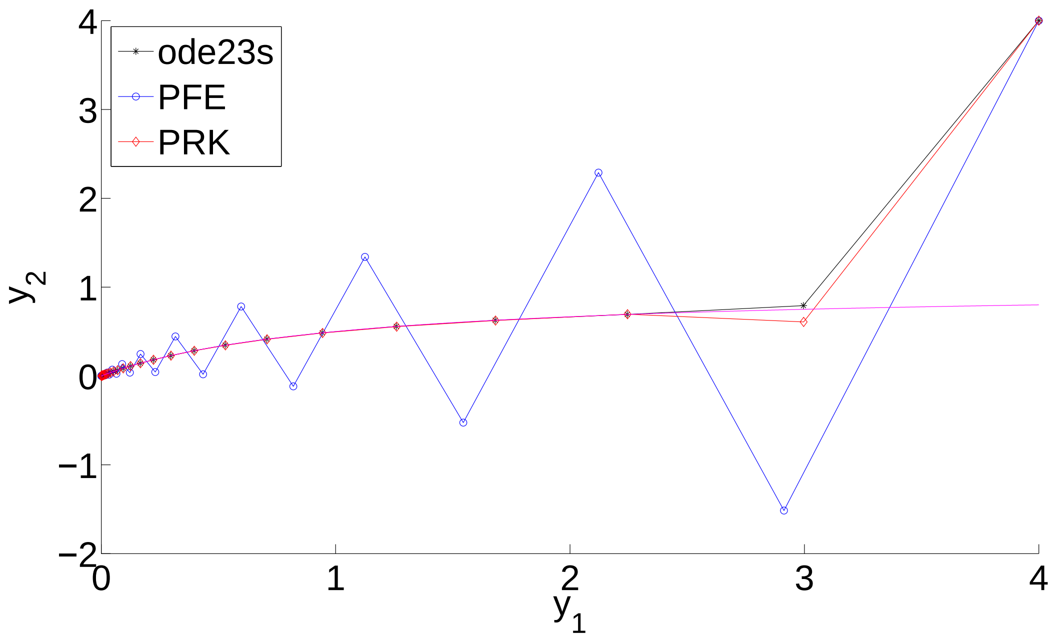


Figure 4.3: Plots of the solutions for $\gamma = 15.0$ in test case 4.

Tab. 4.2: Runtime for every test case using the MATLAB routines `pfe()`, `prk()` and `ode23s()`.

Test case	γ	`pfe()`	`prk()`	`ode23s()`
1	15.0	0.1447s	0.2600s	51.4742s

	3.0	0.1435s	0.2597s	29.6462s
2	15.0	0.1094s	0.1870s	51.7240s
	3.0	0.1086s	0.1866s	29.7160s
3	15.0	0.1349s	0.2774s	51.3672s
	3.0	0.1369s	0.2388s	29.7111s
4	15.0	0.0890s	0.1453s	51.5495s
	3.0	0.0883s	0.1466s	29.3708s
5	15.0	0.1451s	0.2603s	43.7229s
	3.0	0.1443s	0.2625s	26.5912s
6	15.0	0.1081s	0.1864s	43.2031s
	3.0	0.1089s	0.1859s	26.6114s
7	15.0	0.1348s	0.2379s	42.8709s
	3.0	0.1348s	0.2388s	26.3386s
8	15.0	0.0876s	0.1504s	43.4573s
	3.0	0.0855s	0.1449s	26.3494s

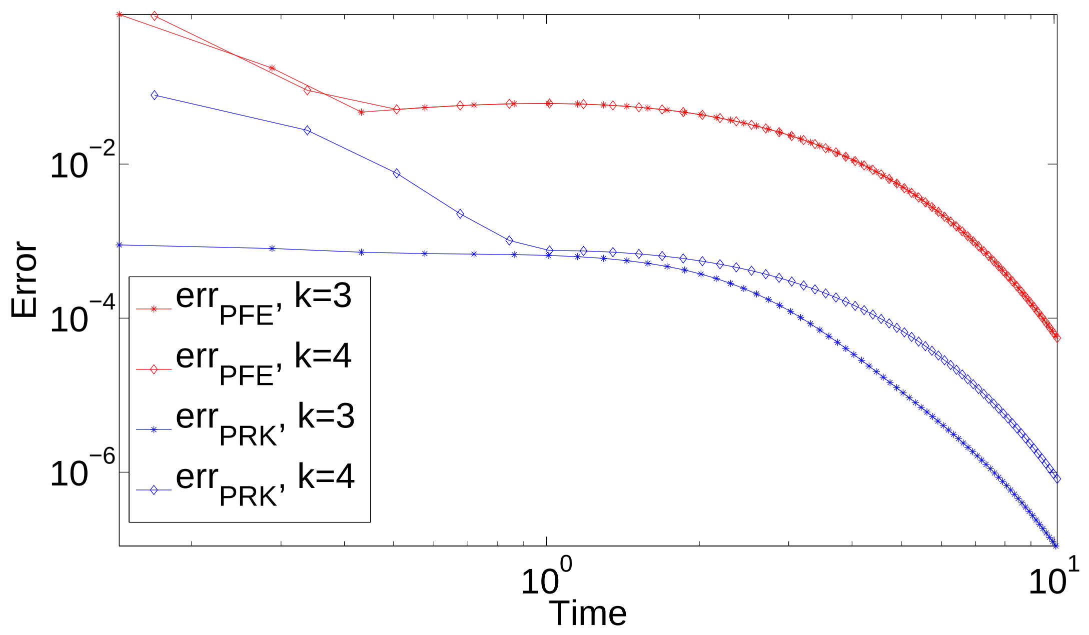


Figure 4.4: Test case 2 vs. 3, $\gamma = 15$: More damping steps do not yield to a higher accuracy.

4.2 Projective Runge–Kutta vs. Backward Differentiation Formulas

In this section, we consider the Simplified Six Species Hydrogen Combustion mechanism, cf. Section 2.5, and compare the PRK Method with BDF Methods integrating on the one hand the full system and on the other hand the corresponding reduced system. We provide a `C++` implementation of a projective Runge–Kutta integrator to compare this method with a BDF integrator implemented by Dominik Skanda in `C++`. Additionally, we compare those methods with an integration using a model reduction technique based on ideas of Lebiedz [19] while making use of the software `MoRe` by Jochen Siehr. In order to use Skandas BDF integrator, we are forced to build a right-hand side using the open source automatic differential package `CppAD`. Listing 4.1 shows the building of such a right-hand side.

Listing 4.1: Building a right-hand side using `CppAD`

```
//-------- build RHS for BDF integrator ---------- //
vector< CppAD::AD<double> >    z(nspec);
CppAD::Independent(z);

vector< CppAD::AD<double> >    c(nspec);
vector< CppAD::AD<double> > zdot(nspec);

//convert z_s to c_s
CppAD::AD<double> sum = 0.0;
for(int i = 0; i < nspec; ++i) {
    sum += z[i];
}
CppAD::AD<double> rho = 101325.0/(8.314472*3000*sum);
for(int i = 0; i < nspec; ++i) {
    c[i] = rho*z[i];
}

// ODE system
CppAD::AD<double> M = (1.0*c[0]+2.5*c[1]+1.0*
                       c[2]+1.0*c[3]+12.0*c[4]+1.0*c[5]);
CppAD::AD<double> q1 =  kf[0]*c[0]*c[1]- kr[0]*c[2]*c[3];
CppAD::AD<double> q2 =  kf[1]*c[1]*c[3]- kr[1]*c[2]*c[4];
CppAD::AD<double> q3 =  kf[2]*c[0]*c[4]- kr[2]*c[3]*c[3];
CppAD::AD<double> q4 = (kf[3]*c[1]      - kr[3]*c[2]*c[2])*M;
CppAD::AD<double> q5 = (kf[4]*c[0]*c[2]- kr[4]*c[3]      )*M;
CppAD::AD<double> q6 = (kf[5]*c[2]*c[3]- kr[5]*c[4]      )*M;

```

```
CppAD::AD<double> fac = 1.0/rho;
zdot[ 0]  = (-q1       -q3       -q5     )*fac;
zdot[ 1]  = (-q1 -q2       - q4          )*fac;
zdot[ 2]  = ( q1 +q2       +2*q4 -q5 -q6 )*fac;
zdot[ 3]  = ( q1 -q2 +2*q3       +q5 -q6 )*fac;
zdot[ 4]  = (     q2 -  q3           +q6 )*fac;
zdot[ 5] = 0;

CppAD::ADFun<double> RHS(z, zdot);
```

In the next step, we build a right-hand side which uses a model reduction method. The software `MoRe` provides a suitable `MoRe-Wrapper`, developed by Marcel Rehberg, such that we only call the following function

    cppadMore(0,xTemp,yTemp)

which represents the map h in Section 2.2. Thus, there it holds `yTemp` $= h($`xTemp`$)$. This leads to a reduced right-hand side as listed in Listing 4.2.

Listing 4.2: Building a reduced right-hand side using `CppAD`

```
/*-------- build RHS for BDF integrator
                using model reduction    ---------- */
vector< CppAD::AD<double> >    z_more(1);

z_more[0]  = y0(5);
CppAD::Independent(z_more);

vector< CppAD::AD<double> > xTemp(1);
vector< CppAD::AD<double> > yTemp(5);

xTemp[0]=z_more[0];
cppadMore(0,xTemp,yTemp);

//calculate constants
CppAD::AD<double> sum_more = z_more[0];
for(int i = 0; i < 5; ++i) {
    sum_more += yTemp[i];
}

//convert z_s to c_s
CppAD::AD<double> rho_m = 101325.0/(8.314472*3000*sum_more);
vector<CppAD::AD<double> > c_m(nspec);
c_m[0] = rho_m*yTemp[0]; c_m[1] = rho_m*yTemp[1];
c_m[2] = rho_m*yTemp[2]; c_m[3] = rho_m*yTemp[3];
```

```
c_m[4] = rho_m*xTemp[0]; c_m[5] = rho_m*yTemp[4];

// ODE system
vector<CppAD::AD<double> > zdot_more(1);
CppAD::AD<double> M_m = (1.0*c_m[0]+2.5*c_m[1]+1.0*c_m[2]
                          +1.0*c_m[3]+12.0*c_m[4]+1.0*c_m[5]);
CppAD::AD<double> q2_m = kf[1]*c_m[1]*c_m[3] - kr[1]*c_m[2]*
   c_m[4];
CppAD::AD<double> q3_m = kf[2]*c_m[0]*c_m[4] - kr[2]*c_m[3]*
   c_m[3];
CppAD::AD<double> q6_m = (kf[5]*c_m[2]*c_m[3] - kr[5]*c_m[4]
     )*M_m;
CppAD::AD<double> fac_m = 1.0/rho_m;
zdot_more[ 0]  = ( q2_m - q3_m + q6_m )*fac_m;

CppAD::ADFun<double> RHS_MORE(z_more, zdot_more);
```

We consider the following test cases

Tab. 4.3: Test cases comparing PRK with BDF.

Test case	Initial Value
1	$y_0 = (0.34563,\ 2.02816,\ 1.51936,\ 0.76437,\ 3.00000,\ 32.90513)^T$
2	$y_0 = (0.75000,\ 0.99002,\ 4.00000,\ 0.36001,\ 3.00000,\ 32.90513)^T$
3	$y_0 = (1.03189,\ 2.02541,\ 3.21111,\ 1.07811,\ 2.00000,\ 32.90513)^T$
4	$y_0 = (1.50000,\ 2.86502,\ 2.00000,\ 0.61001,\ 2.00000,\ 32.90513)^T$
5	$y_0 = (2.03867,\ 1.79891,\ 5.67089,\ 1.07133,\ 1.00000,\ 32.90513)^T$
6	$y_0 = (3.00000,\ 3.11502,\ 4.00000,\ 0.11001,\ 1.00000,\ 32.90513)^T$
7	$y_0 = (3.19024,\ 1.12902,\ 8.91224,\ 0.66976,\ 0.25000,\ 32.90513)^T$
8	$y_0 = (3.50000,\ 3.49002,\ 4.50000,\ 0.36001,\ 0.25000,\ 32.90513)^T$

involving initial values in the near-field (case 1,3,5,7) and far-field (case 2,4,6,8) relative to the equilibrium and to the one dimensional SIM. The initial values in test cases 1,3,5 and 7 are calculated a priori via `MoRe` and all initial values satisfy the mass conservation relation, cf. Section 2.5. Table 4.4 lists various integrators we deal with by comparing PRK with BDF.

Tab. 4.4: Used integrators comparing PRK with BDF.

PRK	**BDF**	**BDF<MoRe>**
PRK(6,3,3)	BDF(10^{-1})	BDF<MoRe>(10^{-1})
PRK(6,4,3)	BDF(10^{-3})	BDF<MoRe>(10^{-3})
PRK(6,3,4)	BDF(10^{-5})	BDF<MoRe>(10^{-5})

PRK(6,4,4)	BDF(10^{-7})	BDF<MoRe>(10^{-7})
PRK(8,4,3)	BDF(10^{-9})	BDF<MoRe>(10^{-9})
PRK(8,4,4)		
PRK(8,4,5)		
PRK(8,5,5)		

Note that we use the syntax `PRK(M,k,L)`, `BDF(tolerance)` and `BDF<MoRe>(tolerance)`. Besides, we choose $t \in [0, 2.5]$ and $h_0 = 0.00000001$.

Figure 4.5 shows the solution trajectories calculated by PRK(6,3,3) and BDF(10^{-7})

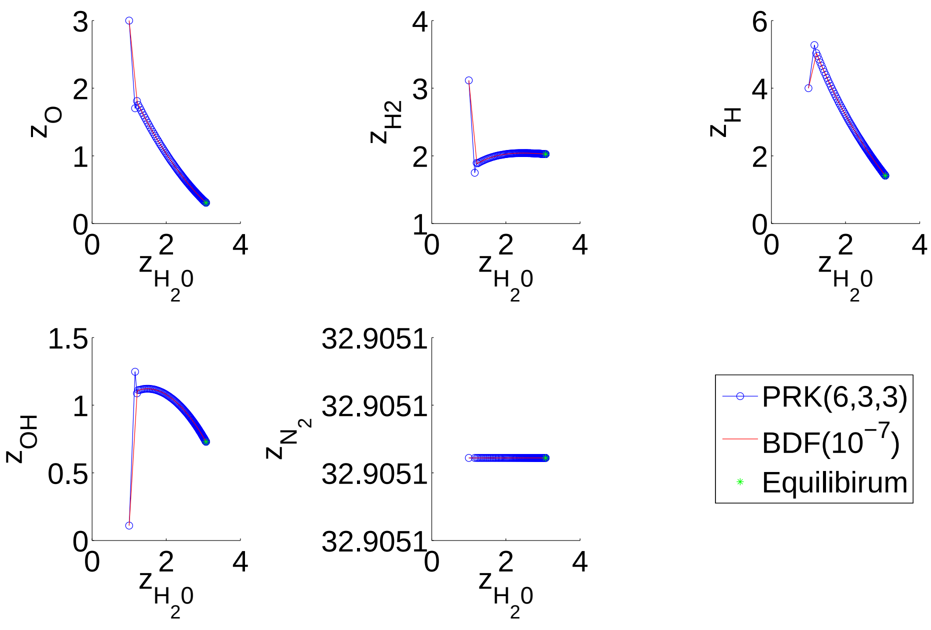


Figure 4.5: Plots of the solutions using PRK(6,3,3) and BDF(10^{-7}) in test case 6.

using the progress variable $z_{\mathrm{H_2O}}$. We notice, that the PRK trajectory fits the BDF solution pretty well. In contrast to this, Figure 4.6 depicts the solution trajectories of of PRK(8,4,4) and BDF(10^{-7}). Besides, we note that a suitable choice of the parameters M, k and L is very important to map the solution trajectory slightly perfect, while keeping in mind to choose them in a way, that does not end up in instability.

In the following, we take a look at the errors of each method, i.e.

$$||z^{\mathrm{cor}}(t_k) - z_k^{\mathrm{PRK}}||, \quad ||z^{\mathrm{cor}}(t_k) - z_k^{\mathrm{BDF}}|| \qquad \text{and} \qquad ||z^{\mathrm{cor}}(t_k) - z_k^{\mathrm{BDF<MoRe>}}||.$$

Again, we calculate a *true* or *correct* solution via MATLAB's `ode23s` using the smallest possible tolerance. We are only interested in the specific moles of the progress variable so that we evaluate those errors only for $z_{\mathrm{H_2O}}$. Figure 4.8 and

4.7 depict the error evolution using the following integrators for test case 5 resp. 6: PRK(6,3,3), PRK(8,4,4), BDF(10^{-7}), BDF(10^{-9}), BDF<MoRe>(10^{-3}) and BDF<MoRe>(10^{-5}). At the beginning, the errors of the projective integrators are significant higher than those of the BDF integrators, because projective integrators need at least a few steps to damp the fast dynamics. But we notice, over the course of time, those errors become smaller than that ones occurring by performing a BDF integration. Obviously, the error of reducing the model only to one species is depicted clearly. It does not matter whether choosing a smaller tolerance or not, we always obtain an error of about 10^{-4}. Those trends of error evolution are observable for all test cases. The entire error plots of all test cases are listed in the appendix, cf. Section B.

Table 4.5 lists the effort of function evaluations and runtime for test case 1, 2, 7 and 8 and each integrator. The effort for each integrator in the other test cases is listed in Table B.2. Although we need more function evaluations by using PRK, we often need fewer runtime, because we are not forced to evaluate the right-hand side with `CppAD`. This is a huge advantage of projective integrators. We only need an efficient vector handling. In our case, we use the `C++`-Library `FLENS` by Michael Lehn et al., cf. [22]. Additionally, we notice that if we start the integration apart of the SIM, the BDF integration of the full system needs more function evaluations than the reduced one, cf. Figure 4.9 and 4.10. The number of steps and function evaluations integrating the reduced model in test case 1 resp. 7 is equal to the number of integration steps of the reduced model in test case 2 resp. 8. This is obvious, because we start from the same initial value $z^0_{H_2O} = 3.0$ resp. $z^0_{H_2O} = 0.25$. Choosing a smaller tolerance for the BDF integrators naturally leads to more integration steps, cf. Figure 4.11. Nevertheless, the integration of a full system starting from a point in the far-field needs overall more steps than integrating the reduced system, cf. Figure 4.11 and 4.12. Moreover, we notice that the runtime of integrating a reduced system is always a little bit higher than integrating the full system. In fact, the model reduction technique is not worth it considering a ODE system involving only six species, but the number of integration steps and function evaluations is almost always less such that if the function evaluation is very expensive, this may be a good way in order to integrate a high-dimensional system.

Tab. 4.5: Effort of several integrators for test case 1,2,7 and 8.

Test case	Integrator	Time	Integration Steps	F-Evals
1	PRK(6,3,3)	0.150629s	250000	99968
	PRK(6,4,3)	0.206526s	187829	144500
	PRK(6,3,4)	0.056418s	25001	39424
	PRK(6,4,4)	0.092976s	17076	67500

	PRK(8,4,3)	0.128372s	113792	89000
	PRK(8,4,4)	0.049617s	8754	36250
	PRK(8,4,5)	0.121431s	674	87500
	PRK(8,5,5)	0.104718s	465	77760
	BDF(10^{-1})	0.217126s	9	54
	BDF(10^{-3})	0.220425s	18	112
	BDF(10^{-5})	0.220265s	26	156
	BDF(10^{-7})	0.222221s	33	198
	BDF(10^{-9})	0.222720s	101	546
	BDF<MoRe>(10^{-1})	0.340569s	9	52
	BDF<MoRe>(10^{-3})	0.357300s	18	96
	BDF<MoRe>(10^{-5})	0.384890s	27	160
	BDF<MoRe>(10^{-7})	0.406148s	39	218
	BDF<MoRe>(10^{-9})	0.503730s	105	500
2	PRK(6,3,3)	0.16768s	250000	108928
	PRK(6,4,3)	0.227961s	187829	159750
	PRK(6,3,4)	0.063001s	25001	44032
	PRK(6,4,4)	0.107439s	17076	77500
	PRK(8,4,3)	0.138569s	113792	96750
	PRK(8,4,4)	0.053056s	8754	38750
	PRK(8,4,5)	0.077521s	674	56250
	PRK(8,5,5)	0.104728s	465	77760
	BDF(10^{-1})	0.222621s	10	152
	BDF(10^{-3})	0.222831s	48	362
	BDF(10^{-5})	0.225845s	67	470
	BDF(10^{-7})	0.223214s	92	574
	BDF(10^{-9})	0.229765s	248	1350
	BDF<MoRe>(10^{-1})	0.344359s	9	52
	BDF<MoRe>(10^{-3})	0.359549s	12	96
	BDF<MoRe>(10^{-5})	0.387263s	27	160
	BDF<MoRe>(10^{-7})	0.402807s	39	218
	BDF<MoRe>(10^{-9})	0.521398s	105	500
7	PRK(6,3,3)	0.175368s	250000	115712
	PRK(6,4,3)	0.243891s	187829	171000
	PRK(6,3,4)	0.067883s	25001	47616
	PRK(6,4,4)	0.109522s	17076	80000
	PRK(8,4,3)	0.149097s	113792	104250
	PRK(8,4,4)	0.059949s	8754	43750
	PRK(8,4,5)	0.112015s	674	81250
	PRK(8,5,5)	0.125351s	465	93312

	BDF(10^{-1})	0.230762s	14	121
	BDF(10^{-3})	0.230521s	30	232
	BDF(10^{-5})	0.229540s	41	284
	BDF(10^{-7})	0.230606s	56	394
	BDF(10^{-9})	0.234747s	162	980
	BDF<MoRe>(10^{-1})	0.388919s	15	124
	BDF<MoRe>(10^{-3})	0.418198s	35	193
	BDF<MoRe>(10^{-5})	0.450390s	52	289
	BDF<MoRe>(10^{-7})	0.485036s	69	369
	BDF<MoRe>(10^{-9})	0.709923s	192	955
8	PRK(6,3,3)	0.171794s	250000	113280
	PRK(6,4,3)	0.238962s	187829	167000
	PRK(6,3,4)	0.066588s	25001	46592
	PRK(6,4,4)	0.112892s	17076	82500
	PRK(8,4,3)	0.148919s	113792	104000
	PRK(8,4,4)	0.056964s	8754	41250
	PRK(8,4,5)	0.077379s	674	56250
	PRK(8,5,5)	0.105041s	465	77760
	BDF(10^{-1})	0.229202s	23	190
	BDF(10^{-3})	0.226813s	51	338
	BDF(10^{-5})	0.228253s	76	476
	BDF(10^{-7})	0.228742s	108	633
	BDF(10^{-9})	0.237838s	296	1654
	BDF<MoRe>(10^{-1})	0.390162s	15	124
	BDF<MoRe>(10^{-3})	0.413756s	35	193
	BDF<MoRe>(10^{-5})	0.453405s	52	289
	BDF<MoRe>(10^{-7})	0.481376s	69	369
	BDF<MoRe>(10^{-9})	0.718570s	192	955

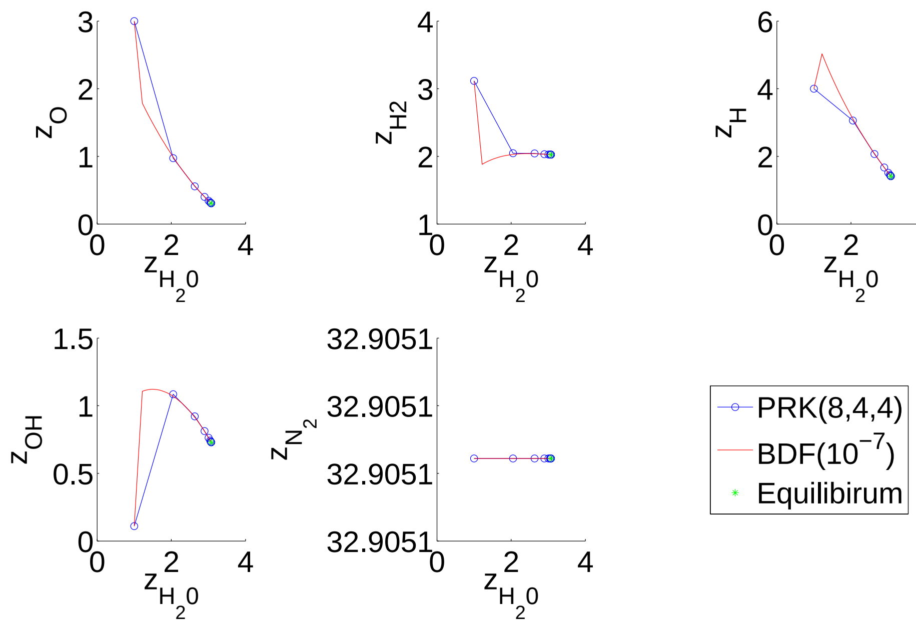


Figure 4.6: Plots of the solutions using PRK(8,4,4) and BDF(10^{-7}) in test case 6.

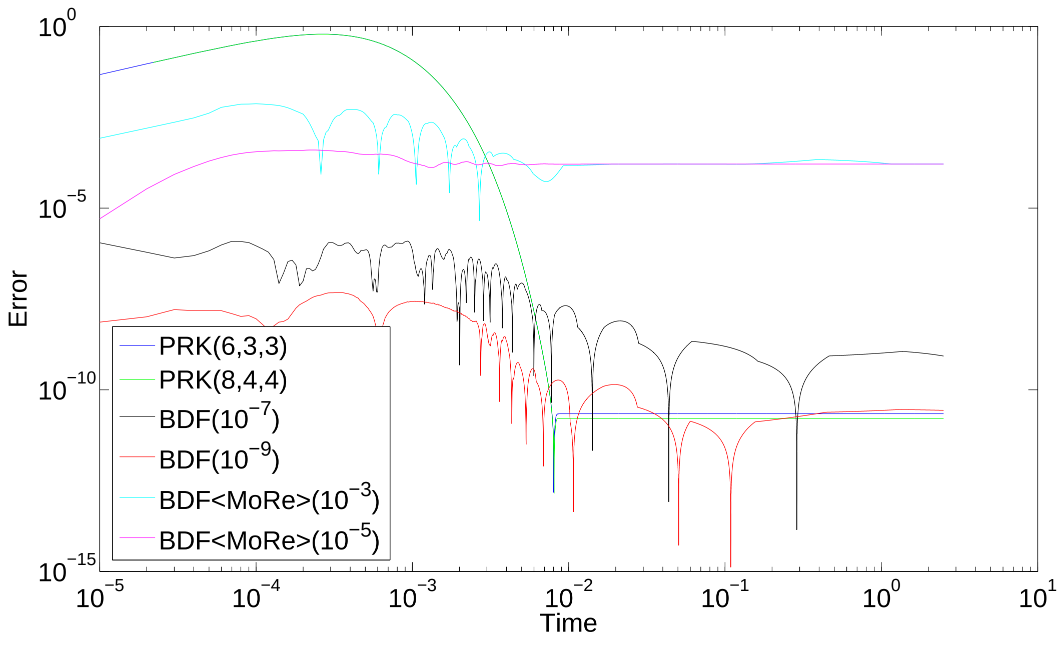


Figure 4.7: Plots of the errors using various integrators for test case 5.

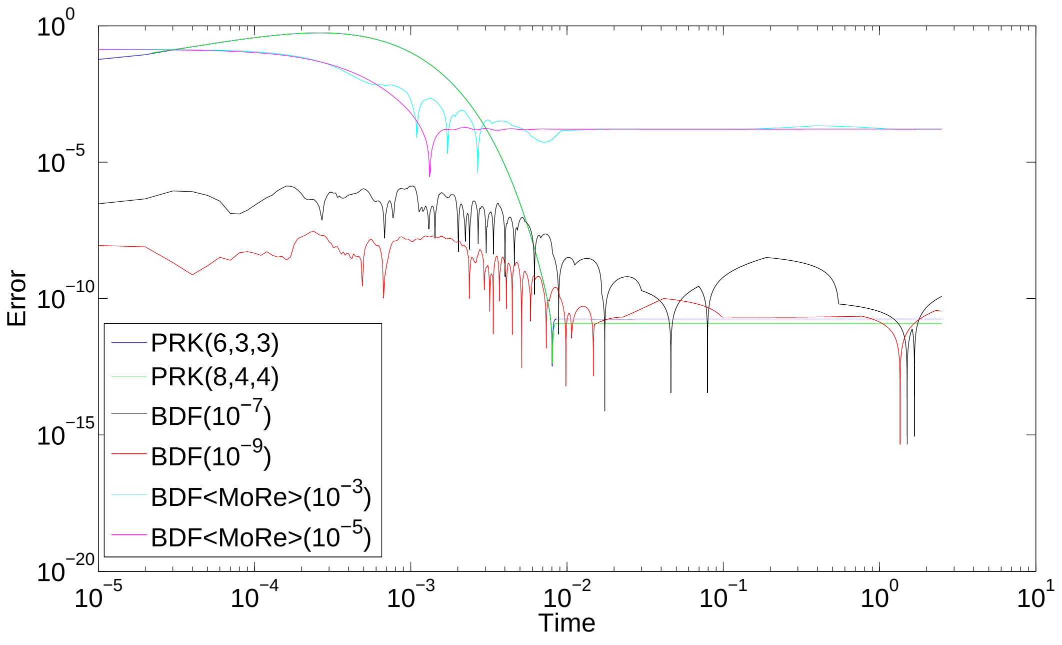


Figure 4.8: Plots of the errors using various integrators for test case 6.

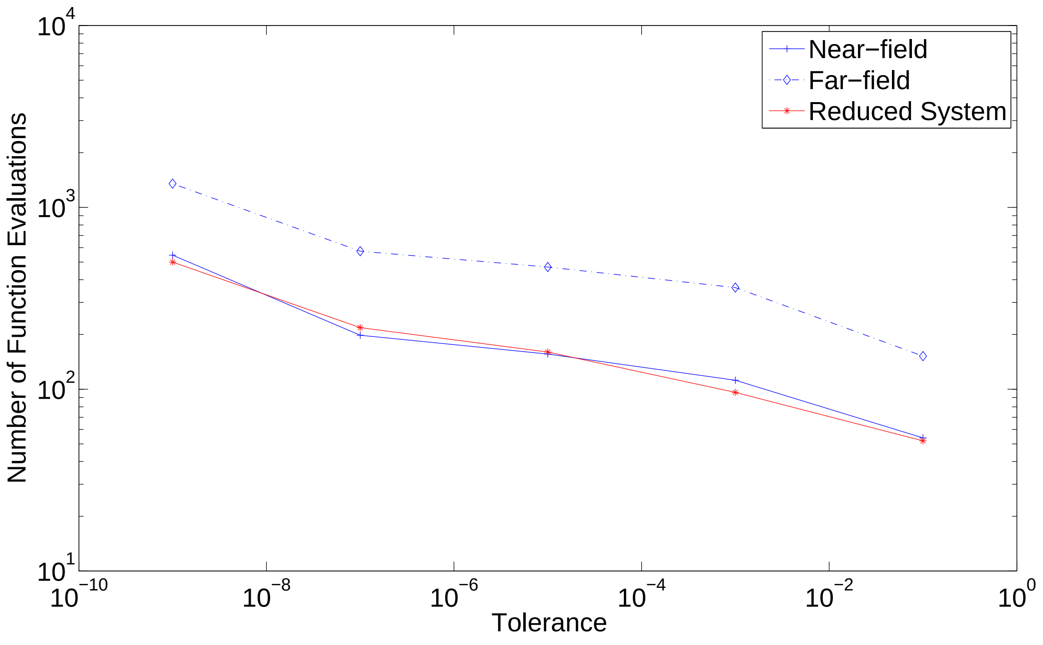


Figure 4.9: Tolerance vs. function evaluations of the BDF integrator for test case 1 and 2.

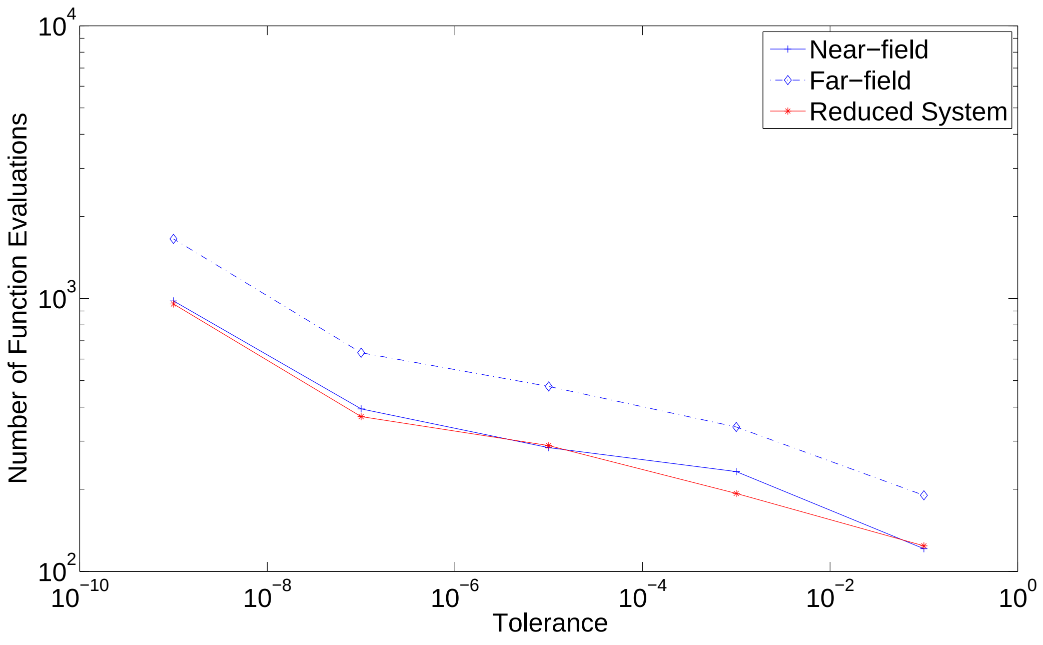


Figure 4.10: Tolerance vs. function evaluations of the BDF integrator for test case 7 and 8.

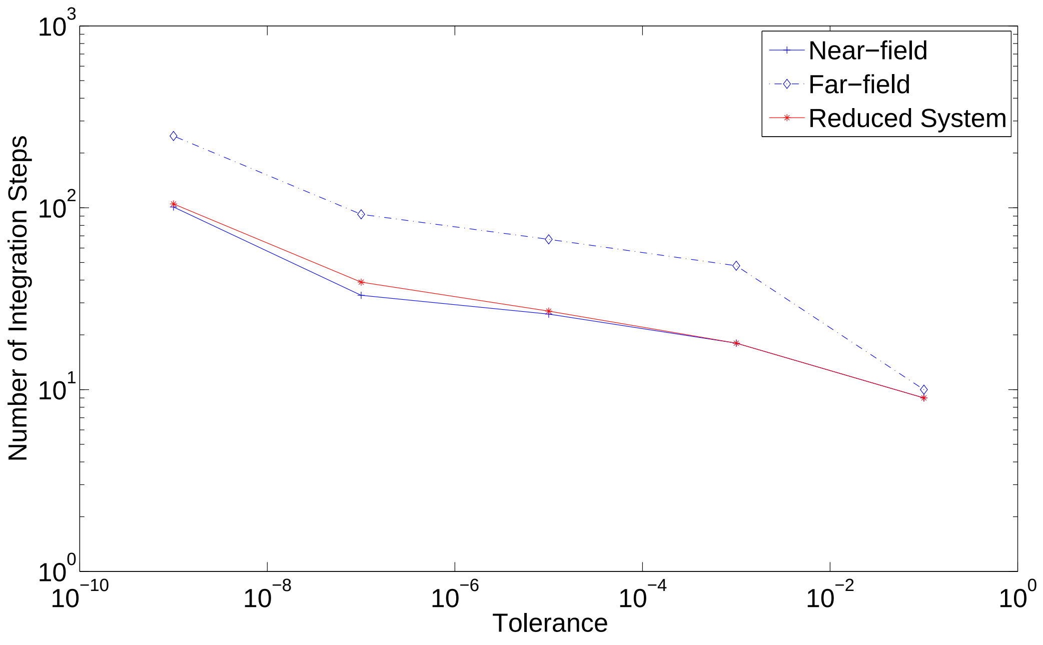


Figure 4.11: Tolerance vs. number of BDF integration steps for test case 1 and 2.

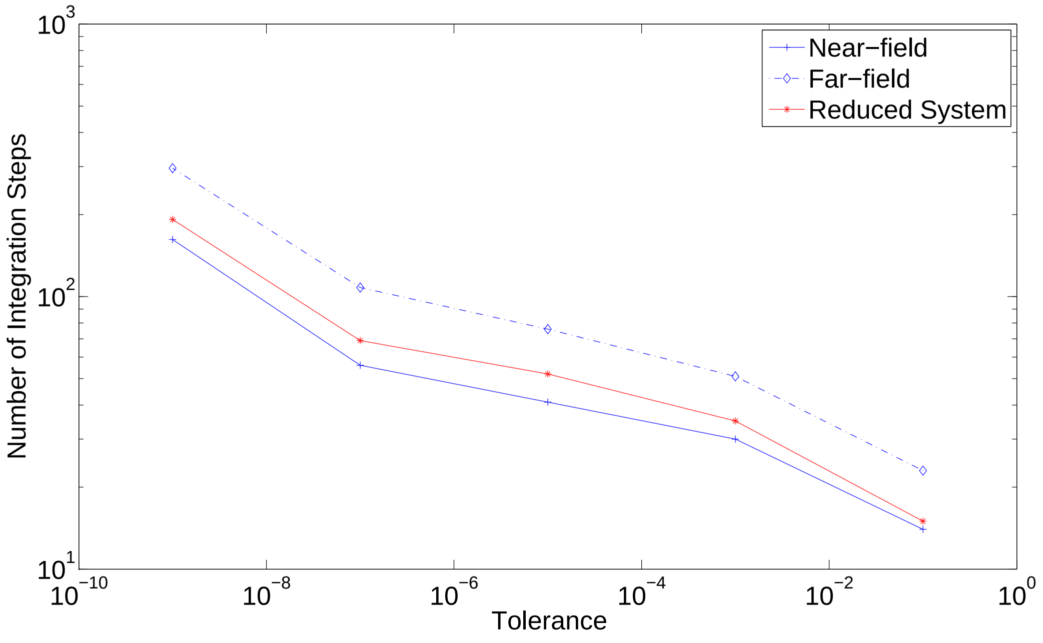


Figure 4.12: Tolerance vs. number of BDF integration steps for test case 7 and 8.

Chapter 5
Conclusion

We give a short overview of the theory of ODE systems and two models in the second chapter as an example of multi-scale problems. Moreover, we treat the theory of projective integrators and give a detailed proof of the second-order accuracy of the Projective Runge–Kutta Method based on ideas of Lee and Gear in [26]. Further, we implement these algorithms in MATLAB and `C++` to compare them with existing integrators, especially with the BDF integrator written by Skanda. Furthermore, we compare projective integrators which integrate the full system with a BDF integrator dealing with a reduced model using the software `MoRe` by Siehr. In general, there is no best choice. The following table illustrates the advantages and disadvantages of the mentioned integration methods:

	PFE	PRK	BDF
explicit	+	+	−
high-order accuracy	−	o	++
fast	+	+	o
simplicity of the implementation	+	+	−
stability	o	o	++

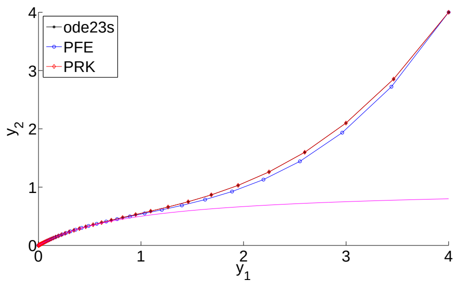
In other words, by choosing a BDF integrator, we achieve a high-order accuracy and we can always apply this method to all problems. However, we have to solve a non-linear equation system in every step. This might need a lot of runtime. To avoid this curse of implicit methods, we can choose an explicit integrator as presented previously. Those explicit methods can be applied to legacy codes without the knowledge of the right-hand side explicitly. This occurs, if the microscopic behavior is represented by a simulation, e.g. a Monte-Carlo simulation. The implementation of projective integrators as against the implementation of implicit methods does not need a non-linear equation solver or methods to compute an approximation of the system Jacobian. Indeed, we only need an efficient vector arithmetic. Nevertheless, we still have to choose the parameters in a suitable way such that the method becomes stable.

Appendix A
Plots of the Test Cases Comparing PFE with PRK

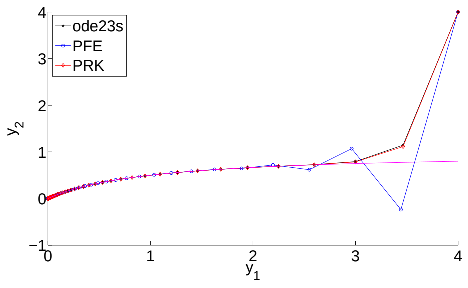
The following table contains the number of each figure belonging to different test cases by comparing PFE with PRK:

Tab. A.1: Overview of the corresponding plots of each test case comparing PFE with PRK.

Test Case	γ	Plot of Solutions	Error Plots
1	3.0	4.1a	4.2a
	15.0	4.1b	4.2b
2	3.0	A.1a	A.2a
	15.0	A.1b	A.2b
3	3.0	A.3a	A.4a
	15.0	A.3b	A.4b
4	3.0	A.5a	A.6a
	15.0	4.3	A.6b
5	3.0	A.7a	A.8a
	15.0	A.7b	A.8b
6	3.0	A.9a	A.10a
	15.0	A.9b	A.10b
7	3.0	A.11a	A.12a
	15.0	A.11b	A.12b
8	3.0	A.13a	A.14a
	15.0	A.13b	A.14b

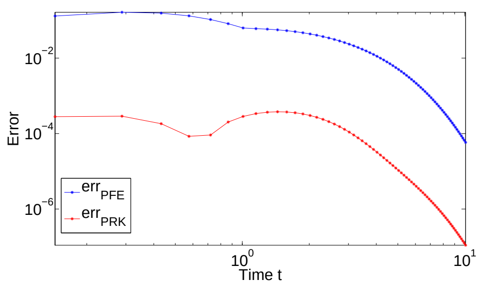


(a) $\gamma = 3.0$

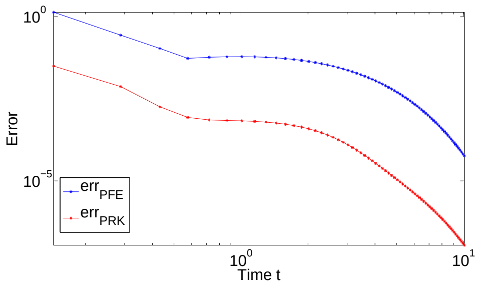


(b) $\gamma = 15.0$

Figure A.1: Plots of the solutions in test case 2.

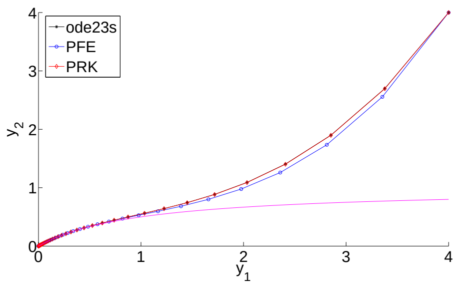


(a) $\gamma = 3.0$

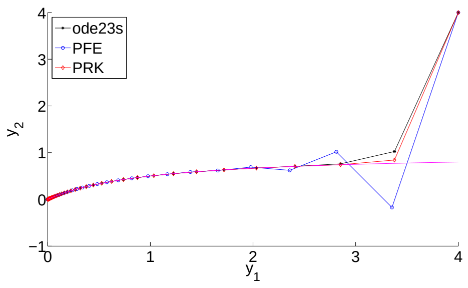


(b) $\gamma = 15.0$

Figure A.2: Error plots of test case 2.

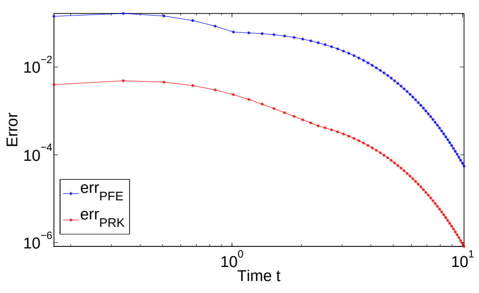


(a) $\gamma = 3.0$

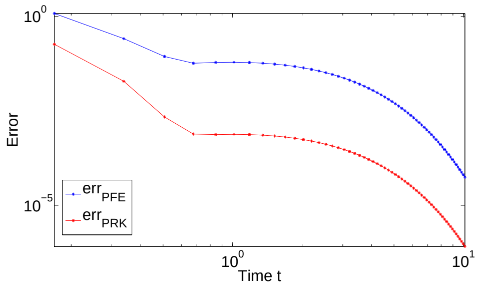


(b) $\gamma = 15.0$

Figure A.3: Plots of the solutions in test case 3.

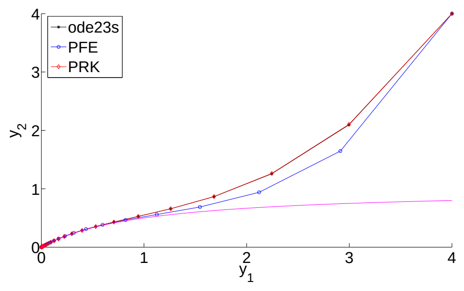


(a) $\gamma = 3.0$



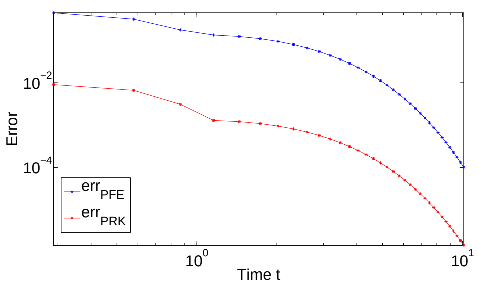
(b) $\gamma = 15.0$

Figure A.4: Error plots of test case 3.

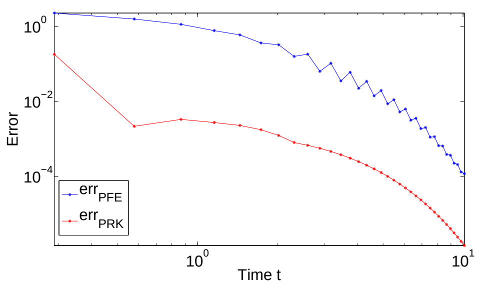


(a) $\gamma = 3.0$

Figure A.5: Plots of the solutions in test case 4

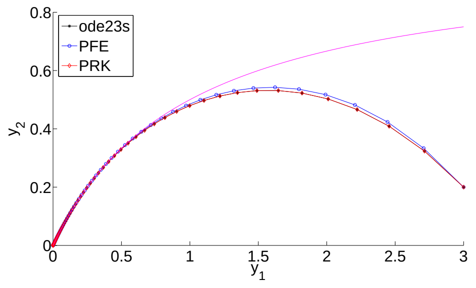


(a) $\gamma = 3.0$

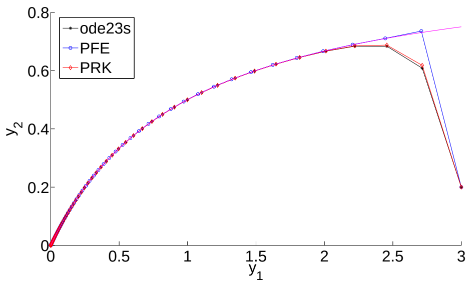


(b) $\gamma = 15.0$

Figure A.6: Error plots of test case 4.

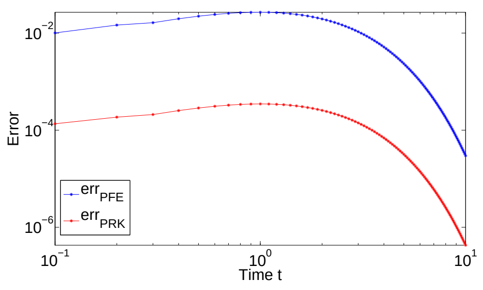


(a) $\gamma = 3.0$

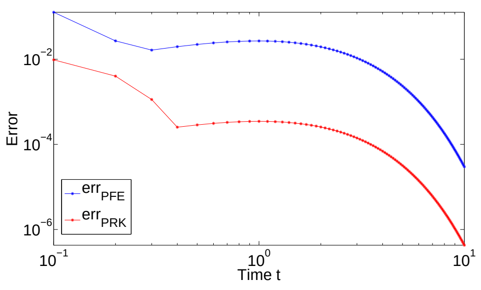


(b) $\gamma = 15.0$

Figure A.7: Plots of the solutions in test case 5.

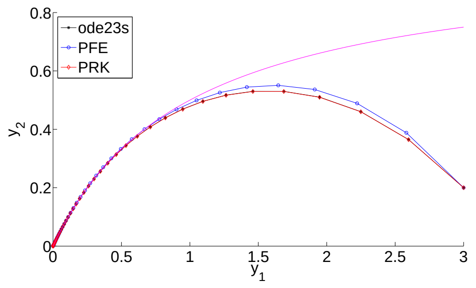


(a) $\gamma = 3.0$

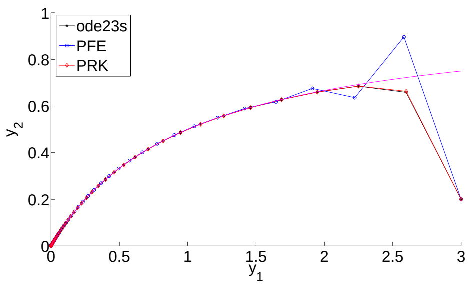


(b) $\gamma = 15.0$

Figure A.8: Error plots of test case 5.

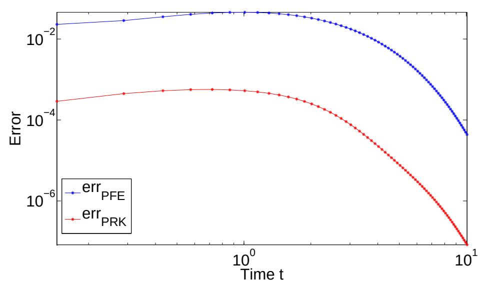


(a) $\gamma = 3.0$

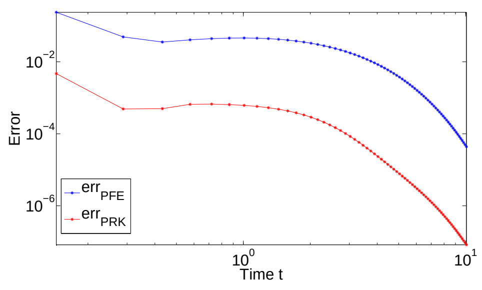


(b) $\gamma = 15.0$

Figure A.9: Plots of the solutions in test case 6.

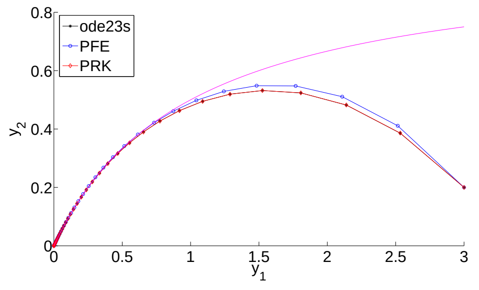


(a) $\gamma = 3.0$

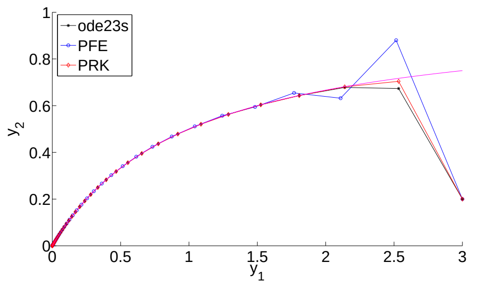


(b) $\gamma = 15.0$

Figure A.10: Error plots of test case 6.

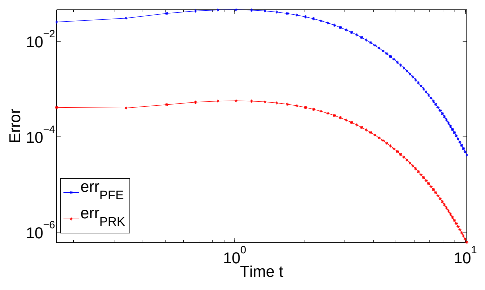


(a) $\gamma = 3.0$

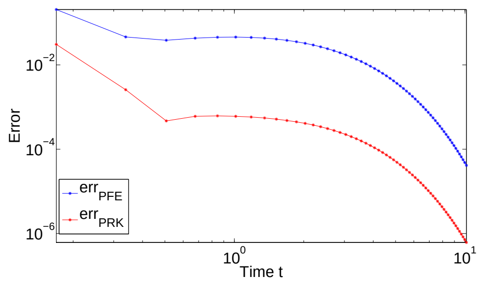


(b) $\gamma = 15.0$

Figure A.11: Plots of the solutions in test case 7.

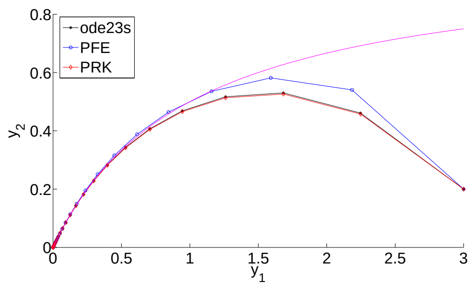


(a) $\gamma = 3.0$

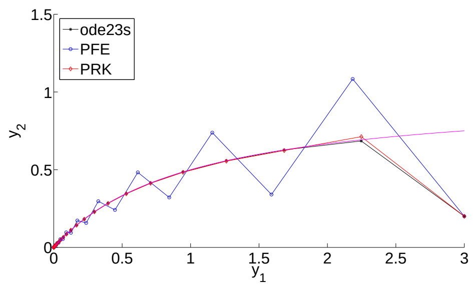


(b) $\gamma = 15.0$

Figure A.12: Error plots of test case 7.

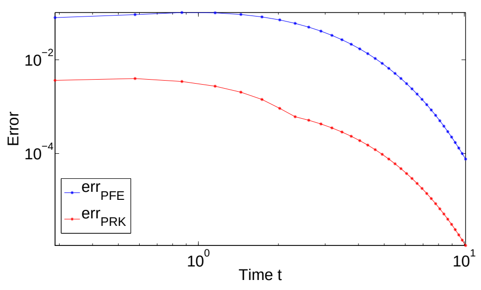


(a) $\gamma = 3.0$

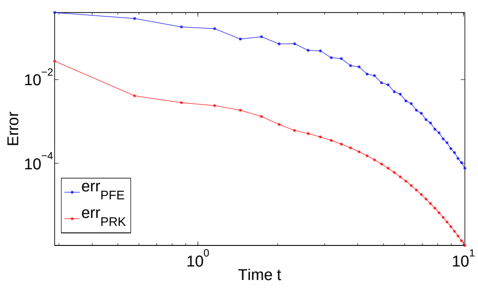


(b) $\gamma = 15.0$

Figure A.13: Plots of the solutions in test case 8.



(a) $\gamma = 3.0$



(b) $\gamma = 15.0$

Figure A.14: Error plots of test case 8.

Appendix B
Plots and Effort of the Test Cases Comparing PRK with BDF

The following table contains the number of each figure belonging to various error plots of different test cases by comparing PRK with BDF.

Tab. B.1: Overview of the corresponding plots of each test case comparing PRK with BDF.

Test Case	Plot of			
	Various Errors	PRK Errors	BDF Errors	BDF<MoRe> Errors
1	B.1	B.2	B.3	B.4
2	B.5	B.6	B.7	B.8
3	B.9	B.10	B.11	B.12
4	B.13	B.14	B.15	B.16
5	4.7	B.17	B.18	B.19
6	4.8	B.20	B.21	B.22
7	B.23	B.24	B.25	B.26
8	B.27	B.28	B.29	B.30

Additionally, the runtime, the number of function evaluations and integration steps of each method for the remaining test cases which are not mentioned in Section 4.2, are listed in the following table:

Tab. B.2: Effort of several integrators for test case 3,4,5 and 6.

Test case	Integrator	Time	Integration Steps	F-Evals
3	PRK(6,3,3)	0.163998s	250000	108288
	PRK(6,4,3)	0.229790s	187829	161500
	PRK(6,3,4)	0.063208s	25001	44032
	PRK(6,4,4)	0.102904s	17076	75000
	PRK(8,4,3)	0.148372s	113792	92012
	PRK(8,4,4)	0.059291s	8754	42500
	PRK(8,4,5)	0.103825s	674	75000
	PRK(8,5,5)	0.104696s	465	77760
	BDF(10^{-1})	0.225583s	12	80

	BDF(10^{-3})	0.228274s	27	203
	BDF(10^{-5})	0.229791s	40	273
	BDF(10^{-7})	0.229404s	53	325
	BDF(10^{-9})	0.230496s	131	745
	BDF<MoRe>(10^{-1})	0.353549s	13	78
	BDF<MoRe>(10^{-3})	0.419275s	30	194
	BDF<MoRe>(10^{-5})	0.437174s	43	264
	BDF<MoRe>(10^{-7})	0.466395s	59	342
	BDF<MoRe>(10^{-9})	0.625311s	157	768
4	PRK(6,3,3)	0.166701s	250000	108800
	PRK(6,4,3)	0.233252s	187829	162250
	PRK(6,3,4)	0.063111s	25001	44032
	PRK(6,4,4)	0.104498s	17076	75000
	PRK(8,4,3)	0.137363s	113792	96000
	PRK(8,4,4)	0.053253s	8754	38750
	PRK(8,4,5)	0.069358s	674	50000
	PRK(8,5,5)	0.125257s	465	93312
	BDF(10^{-1})	0.224013s	20	151
	BDF(10^{-3})	0.224627s	48	317
	BDF(10^{-5})	0.223927s	68	445
	BDF(10^{-7})	0.226419s	101	587
	BDF(10^{-9})	0.229799s	253	1397
	BDF<MoRe>(10^{-1})	0.364595s	13	78
	BDF<MoRe>(10^{-3})	0.414015s	30	194
	BDF<MoRe>(10^{-5})	0.431396s	43	264
	BDF<MoRe>(10^{-7})	0.467953s	59	342
	BDF<MoRe>(10^{-9})	0.626288s	157	768
5	PRK(6,3,3)	0.171182s	250000	111360
	PRK(6,4,3)	0.235033s	187829	166000
	PRK(6,3,4)	0.065821s	25001	46080
	PRK(6,4,4)	0.106241s	17076	77500
	PRK(8,4,3)	0.143065s	113792	100250
	PRK(8,4,4)	0.061662s	8754	45000
	PRK(8,4,5)	0.077508s	674	56250
	PRK(8,5,5)	0.104307s	465	77760
	BDF(10^{-1})	0.232661s	14	94
	BDF(10^{-3})	0.231824s	28	195
	BDF(10^{-5})	0.233146s	40	265
	BDF(10^{-7})	0.232608s	56	349
	BDF(10^{-9})	0.236824s	151	949

	BDF<MoRe>(10^{-1})	0.368798s	13	87
	BDF<MoRe>(10^{-3})	0.445305s	35	246
	BDF<MoRe>(10^{-5})	0.459096s	50	304
	BDF<MoRe>(10^{-7})	0.487362s	67	365
	BDF<MoRe>(10^{-9})	0.652954s	168	809
6	PRK(6,3,3)	0.177366s	250000	117504
	PRK(6,4,3)	0.235281s	187829	165750
	PRK(6,3,4)	0.064396s	25001	45056
	PRK(6,4,4)	0.106607s	17076	77500
	PRK(8,4,3)	0.151894s	113792	106000
	PRK(8,4,4)	0.054663s	8754	40000
	PRK(8,4,5)	0.103510s	674	75000
	PRK(8,5,5)	0.104791s	465	77760
	BDF(10^{-1})	0.225459s	22	175
	BDF(10^{-3})	0.226155s	50	341
	BDF(10^{-5})	0.226669s	73	495
	BDF(10^{-7})	0.227646s	102	628
	BDF(10^{-9})	0.233788s	307	1660
	BDF<MoRe>(10^{-1})	0.361559	13	87
	BDF<MoRe>(10^{-3})	0.437788s	35	246
	BDF<MoRe>(10^{-5})	0.455527s	50	304
	BDF<MoRe>(10^{-7})	0.480850s	67	365
	BDF<MoRe>(10^{-9})	0.647777s	168	809

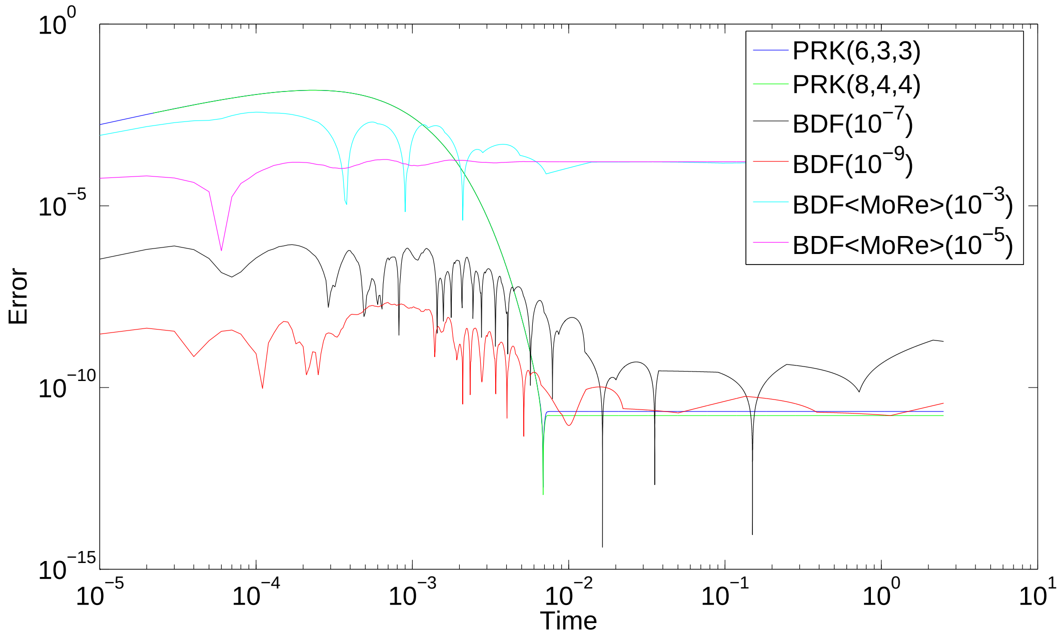


Figure B.1: Plots of the errors using various integrators for test case 1.

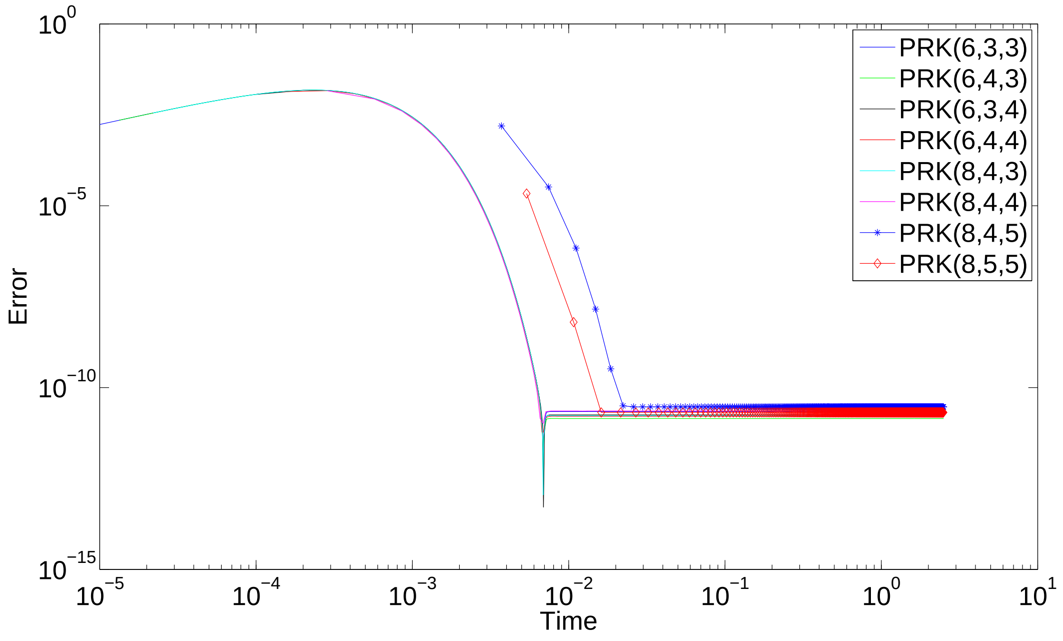


Figure B.2: Plots of the errors using PRK integrators for test case 1.

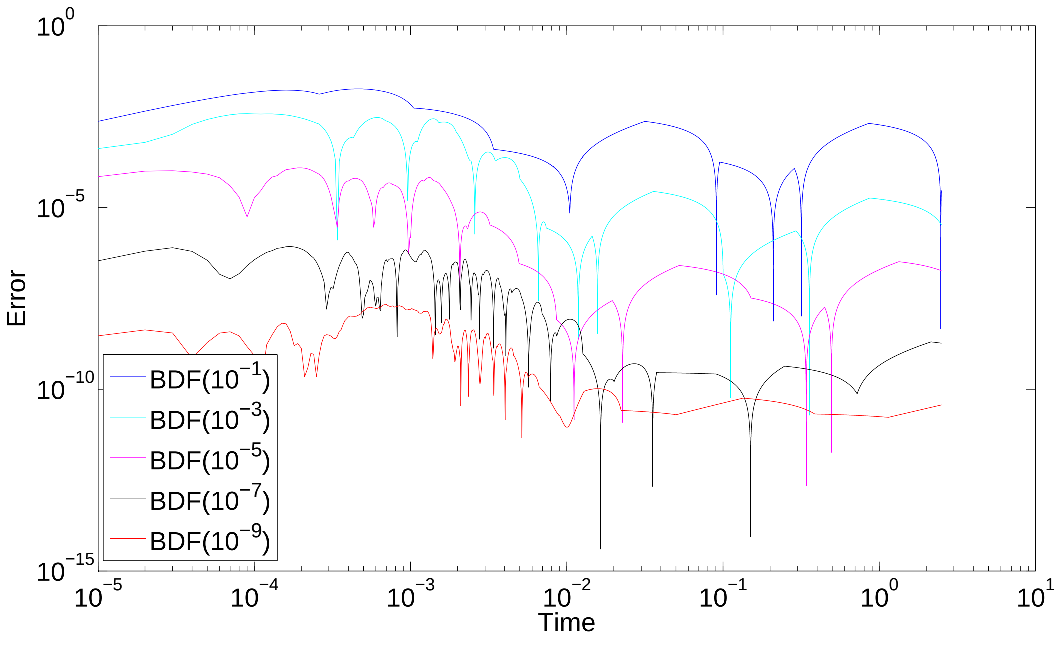


Figure B.3: Plots of the errors using BDF integrators for test case 1.

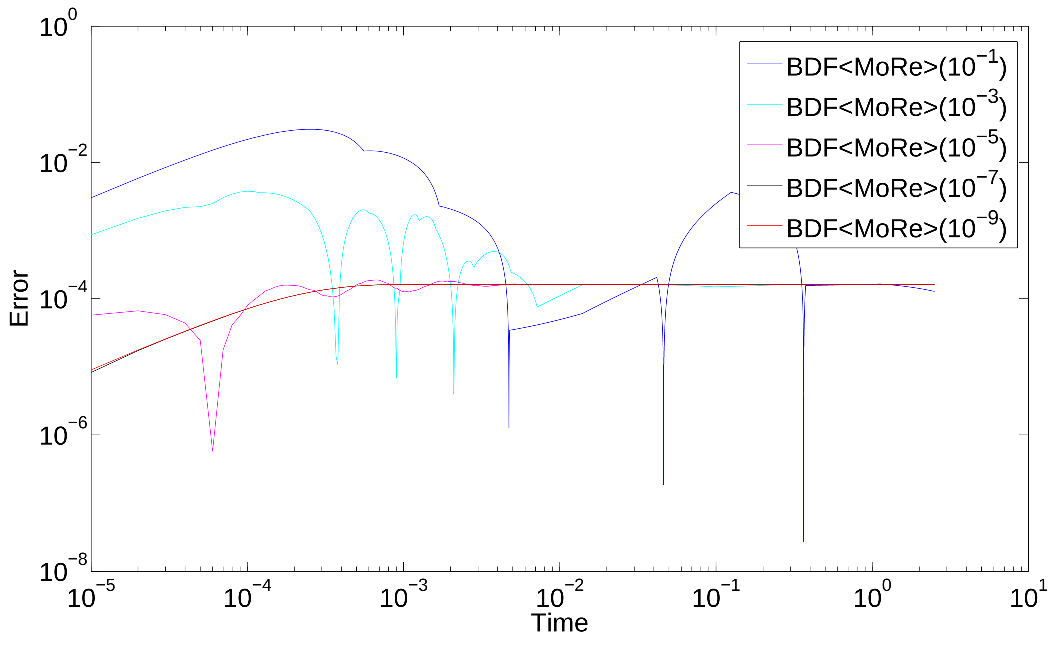


Figure B.4: Plots of the errors using BDF<MoRe> integrators for test case 1.

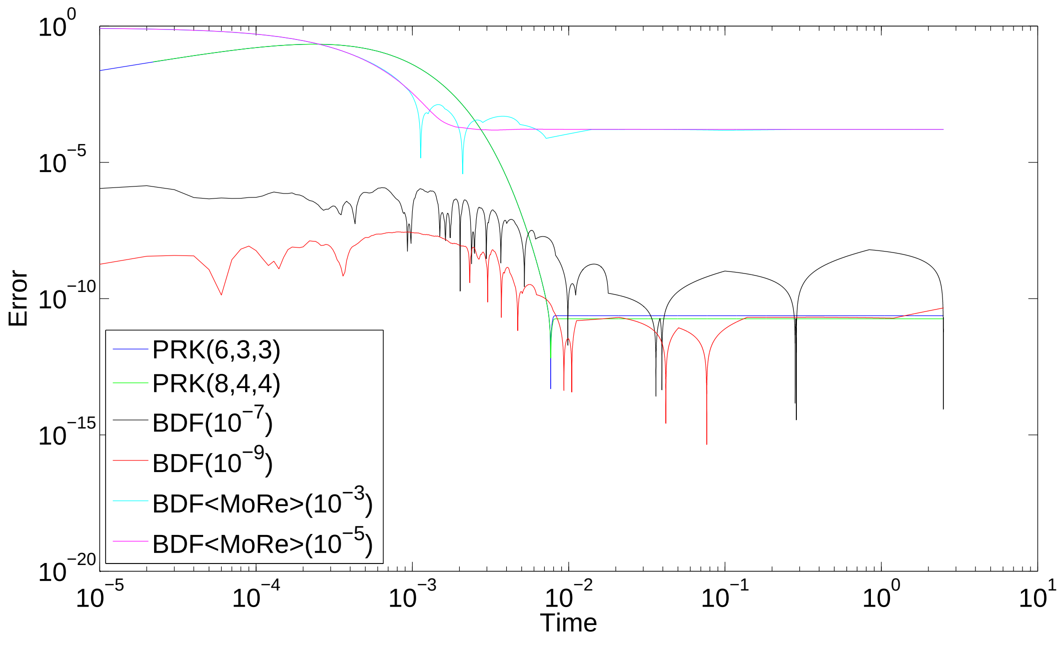


Figure B.5: Plots of the errors using various integrators for test case 2.

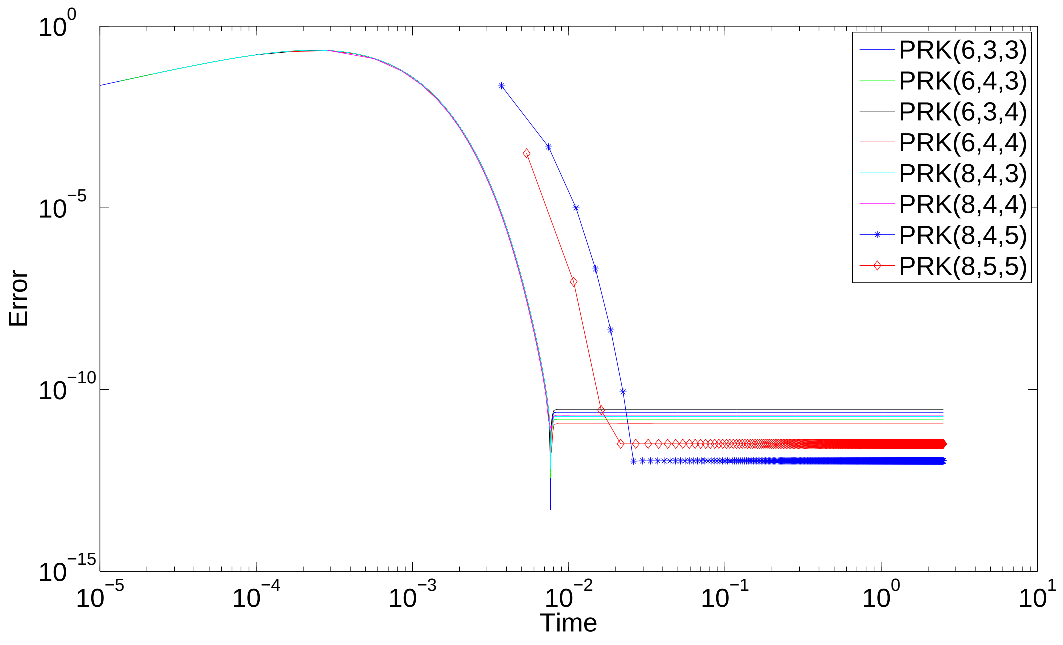


Figure B.6: Plots of the errors using PRK integrators for test case 2.

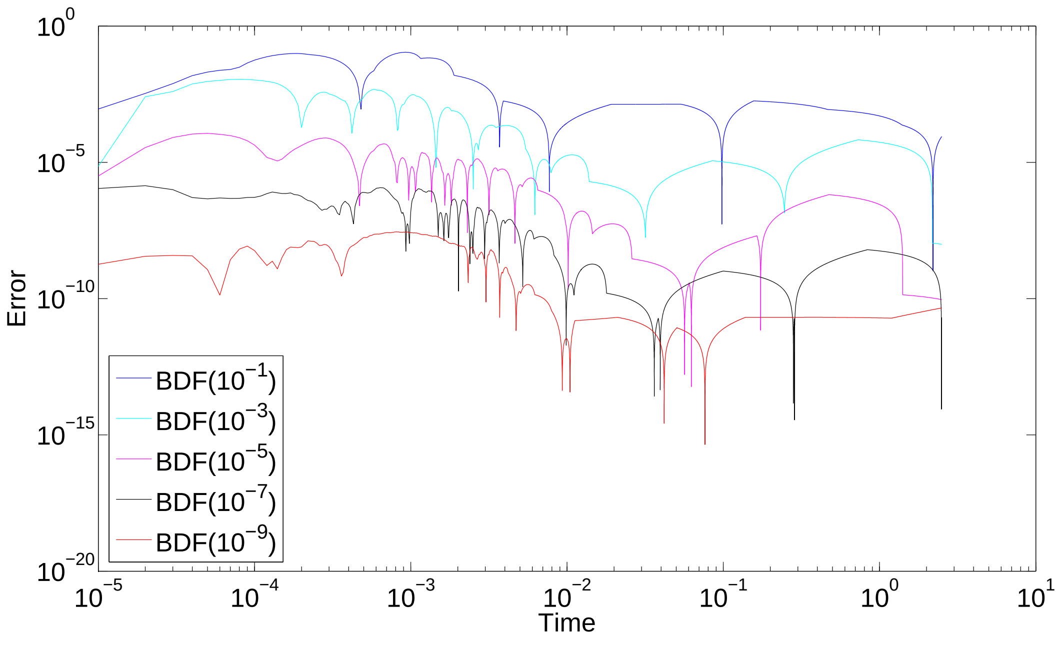


Figure B.7: Plots of the errors using BDF integrators for test case 2.

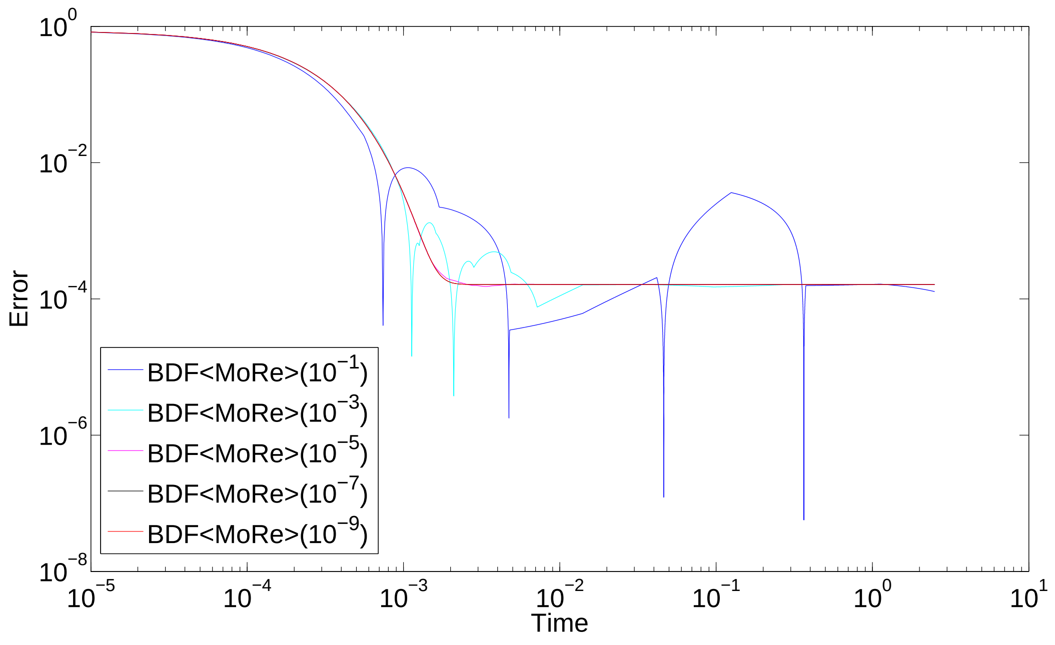


Figure B.8: Plots of the errors using BDF<MoRe> integrators for test case 2.

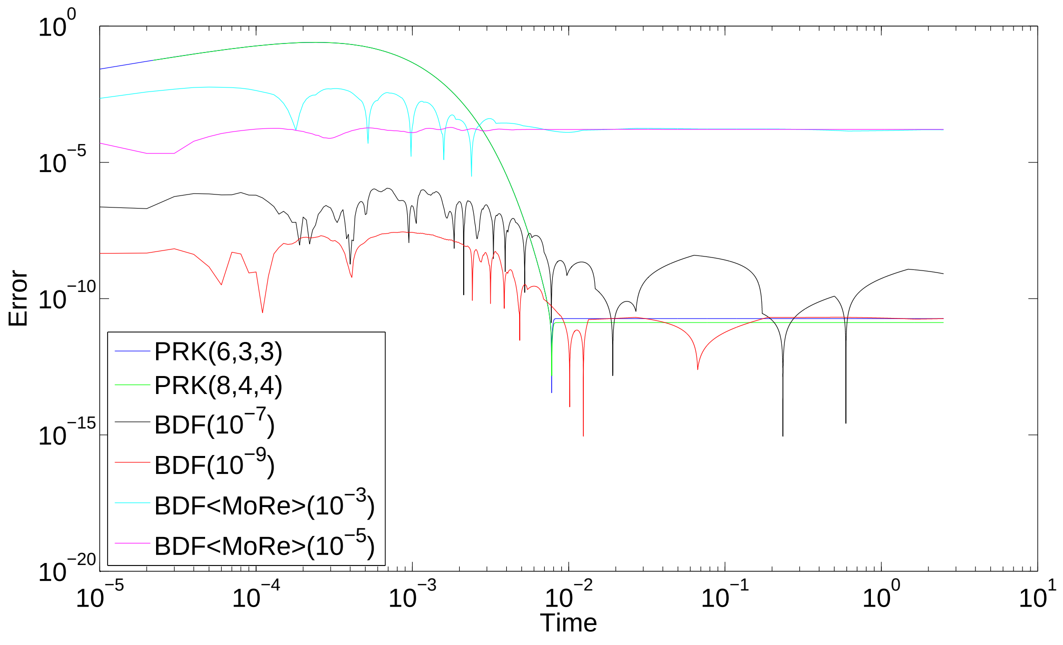


Figure B.9: Plots of the errors using various integrators for test case 3.

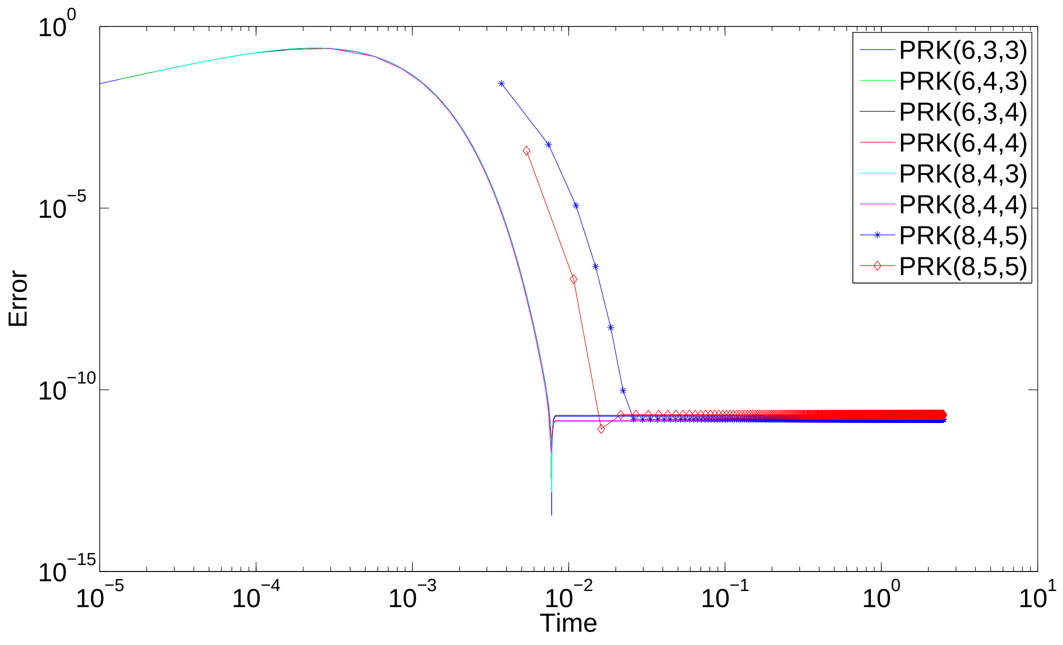


Figure B.10: Plots of the errors using PRK integrators for test case 3.

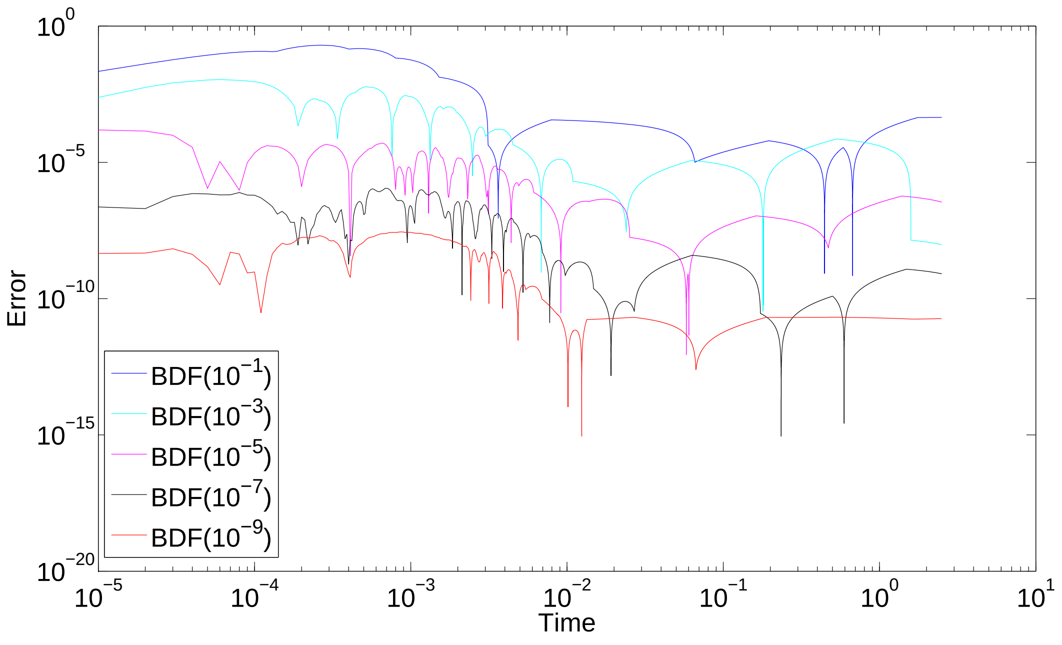


Figure B.11: Plots of the errors using BDF integrators for test case 3.

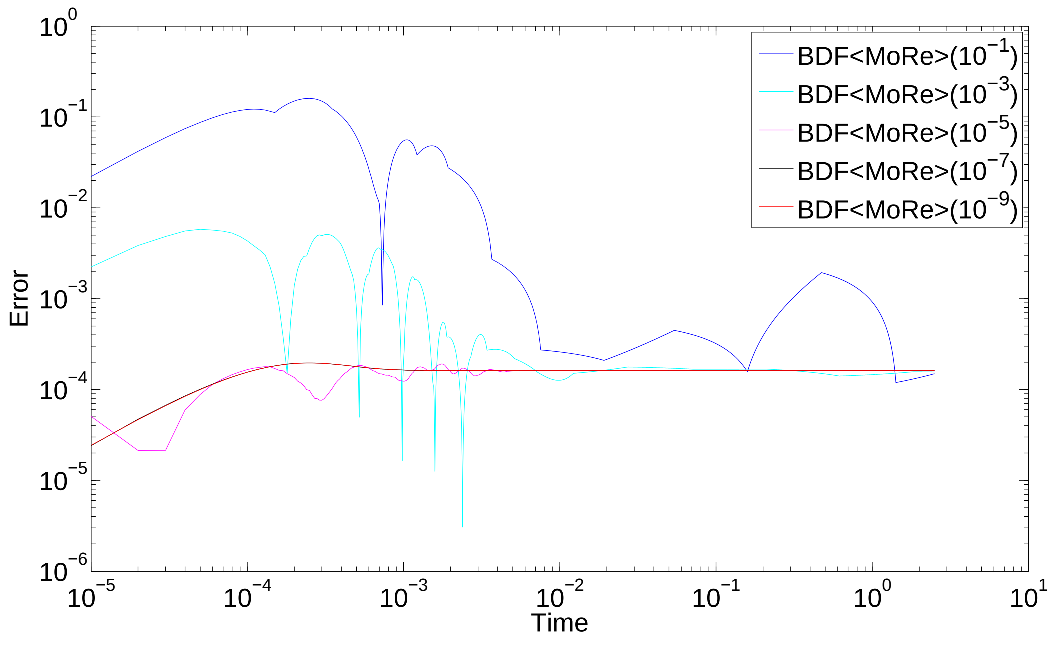


Figure B.12: Plots of the errors using BDF<MoRe> integrators for test case 3.

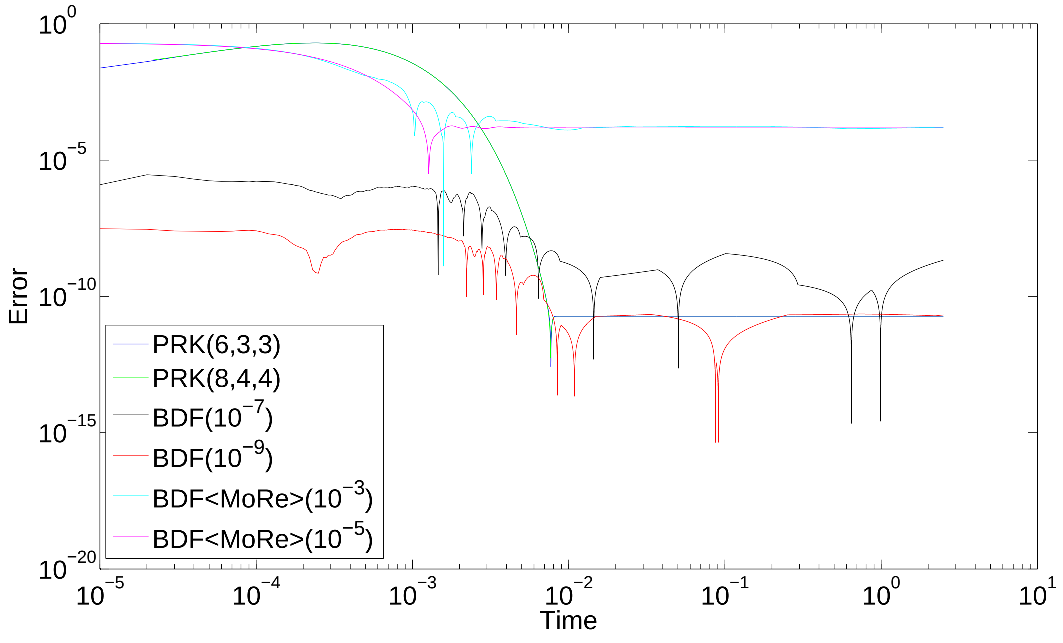


Figure B.13: Plots of the errors using various integrators for test case 4.

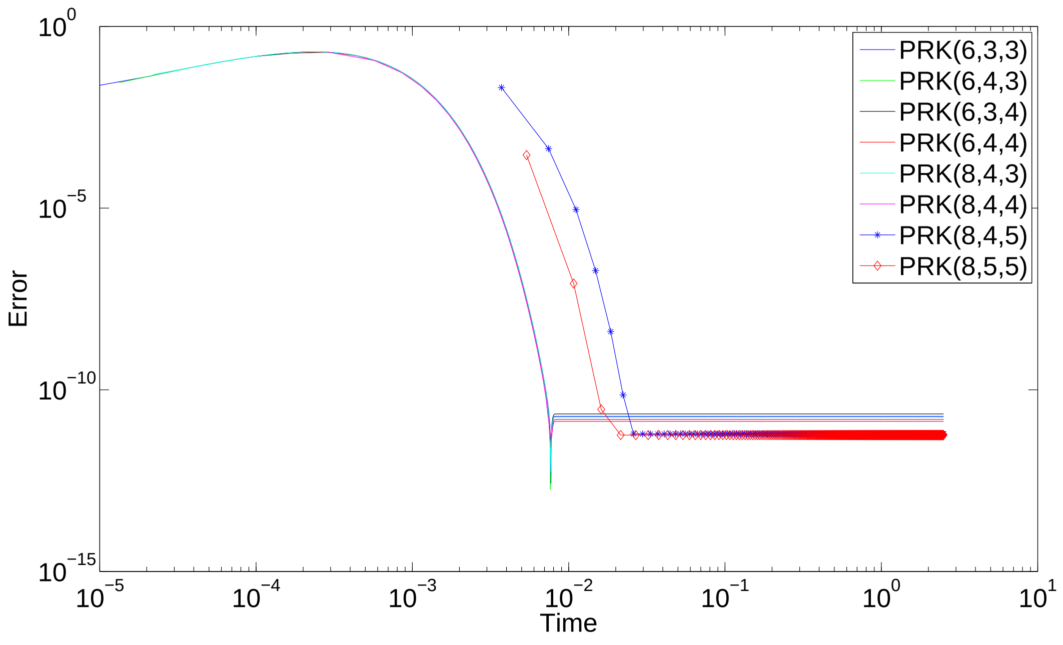


Figure B.14: Plots of the errors using PRK integrators for test case 4.

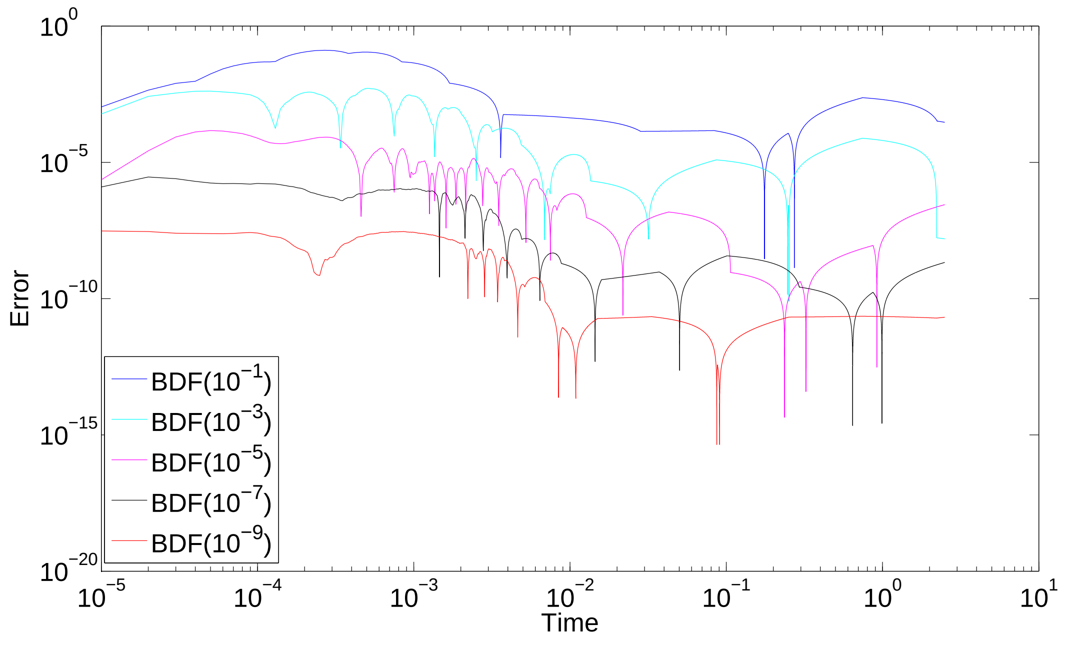


Figure B.15: Plots of the errors using BDF integrators for test case 4.

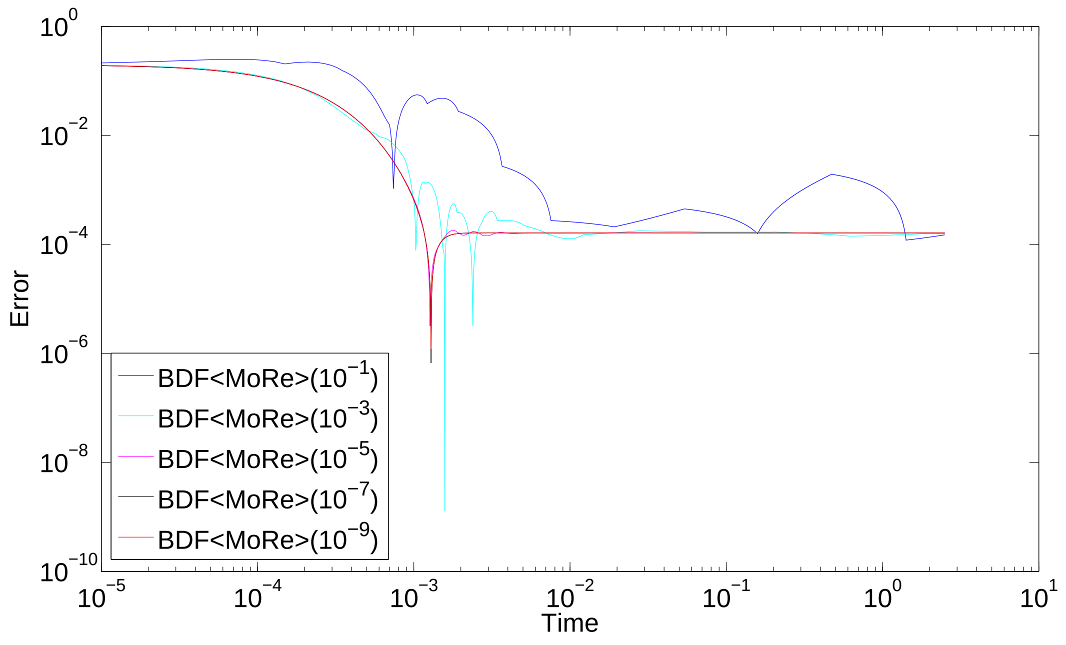


Figure B.16: Plots of the errors using BDF<MoRe> integrators for test case 4.

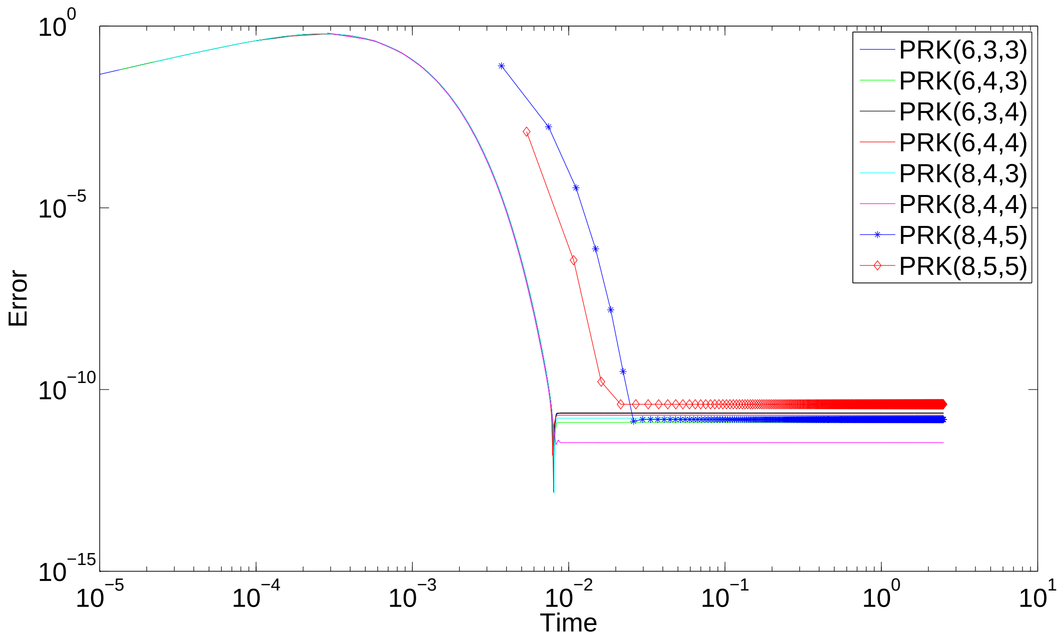


Figure B.17: Plots of the errors using PRK integrators for test case 5.

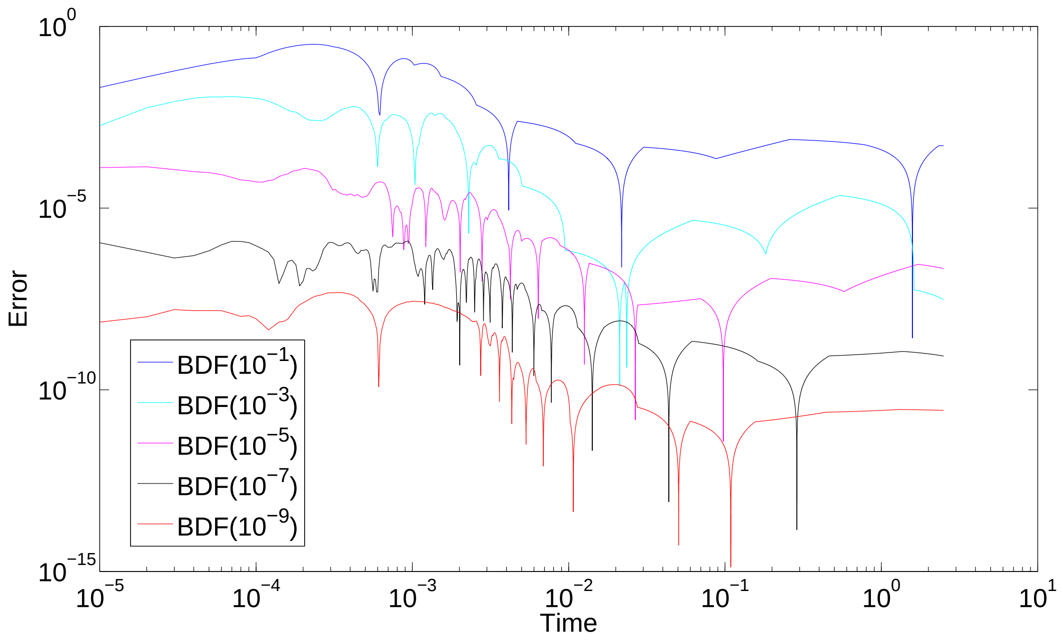


Figure B.18: Plots of the errors using BDF integrators for test case 5.

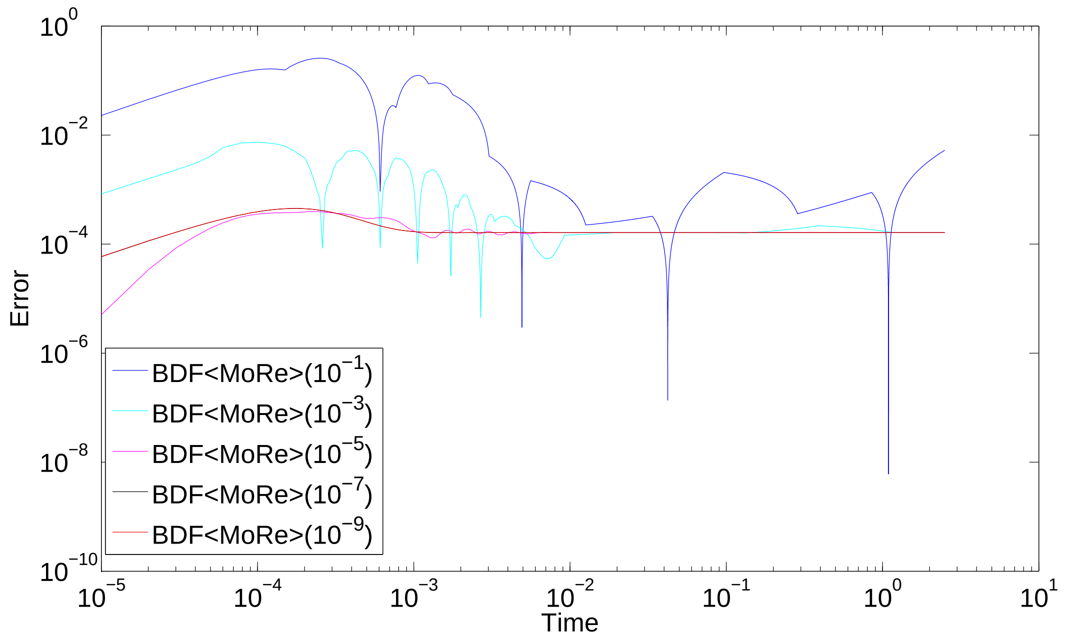


Figure B.19: Plots of the errors using BDF<MoRe> integrators for test case 5.

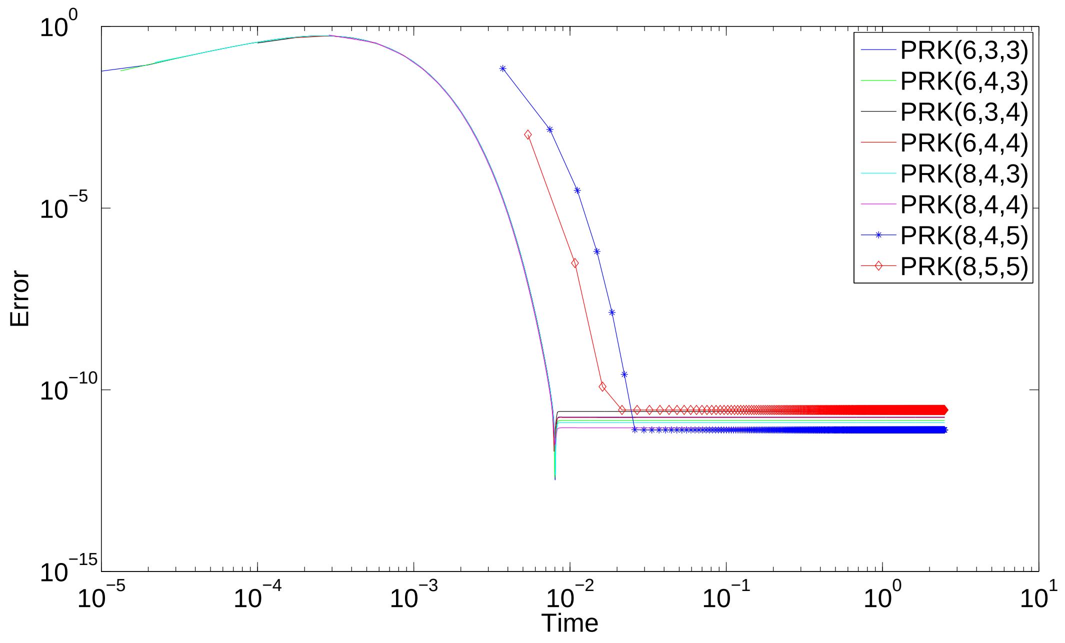


Figure B.20: Plots of the errors using PRK integrators for test case 6.

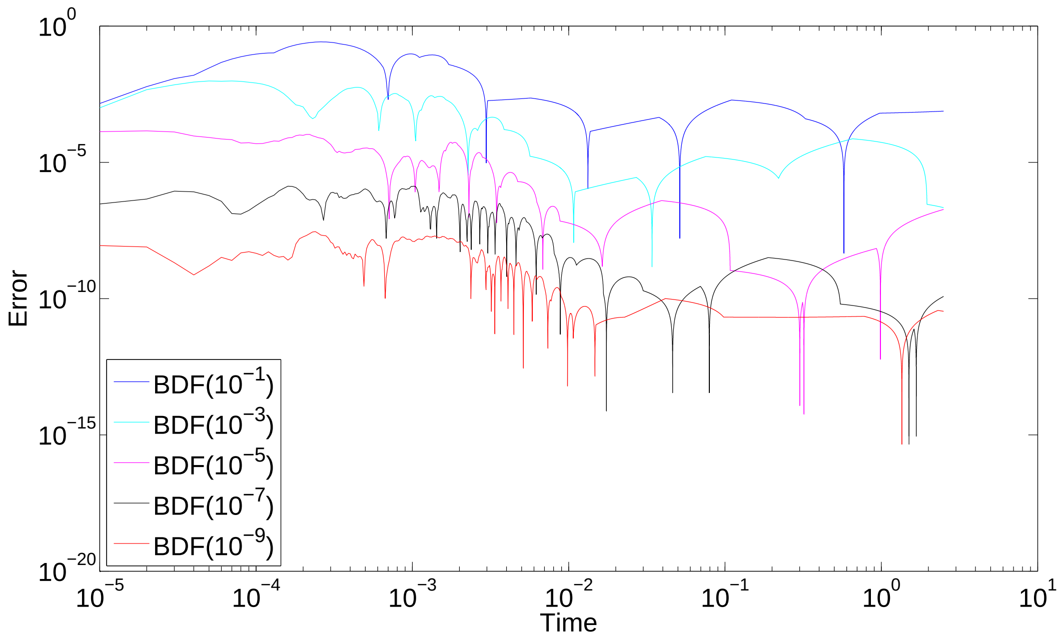


Figure B.21: Plots of the errors using BDF integrators for test case 6.

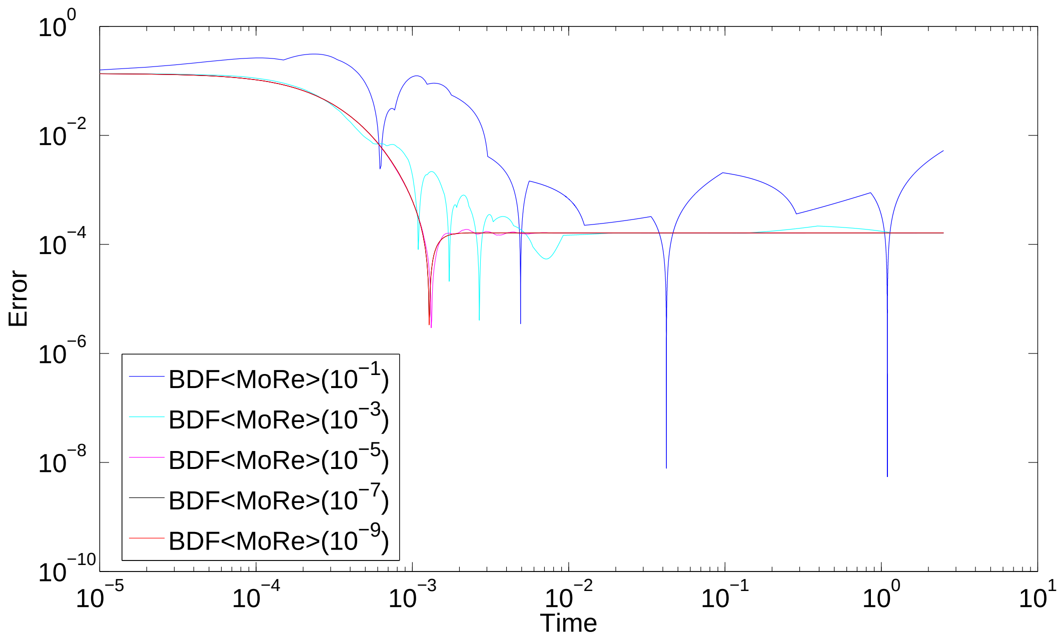


Figure B.22: Plots of the errors using BDF<MoRe> integrators for test case 6.

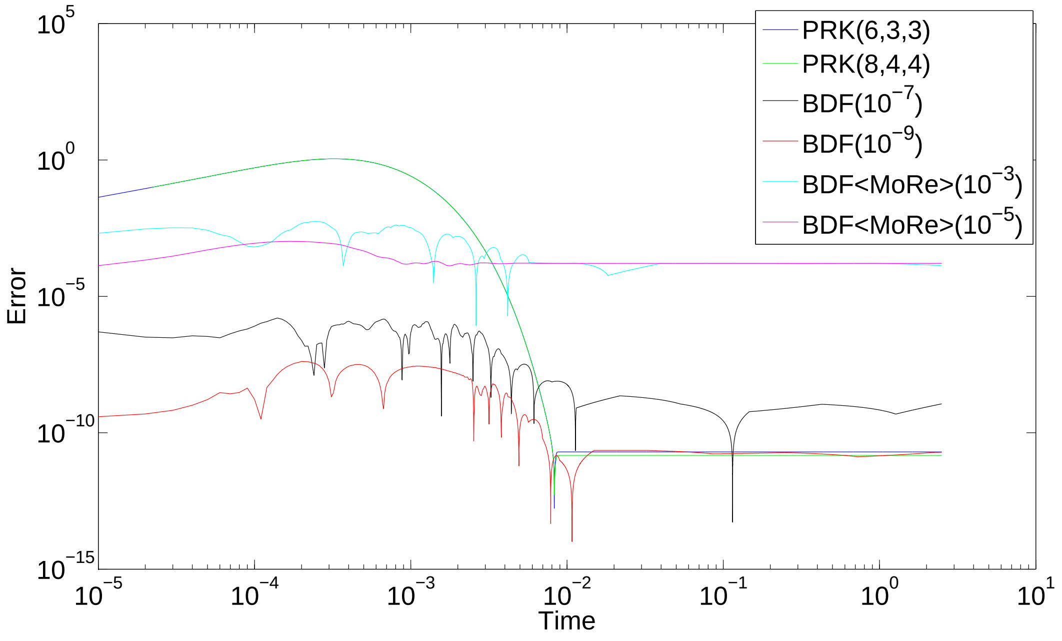


Figure B.23: Plots of the errors using various integrators for test case 7.

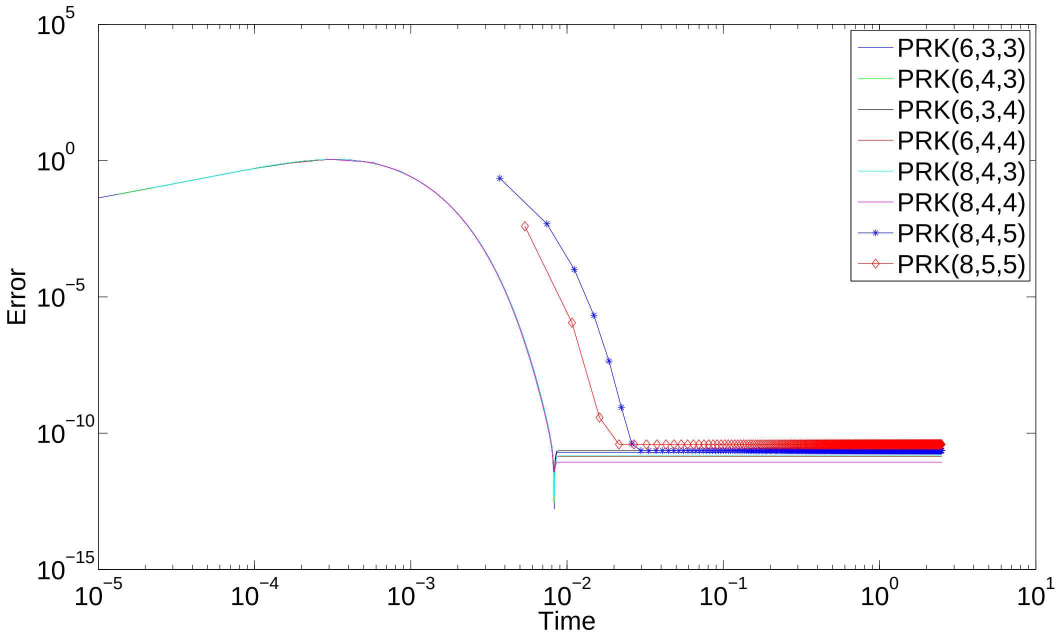


Figure B.24: Plots of the errors using PRK integrators for test case 7.

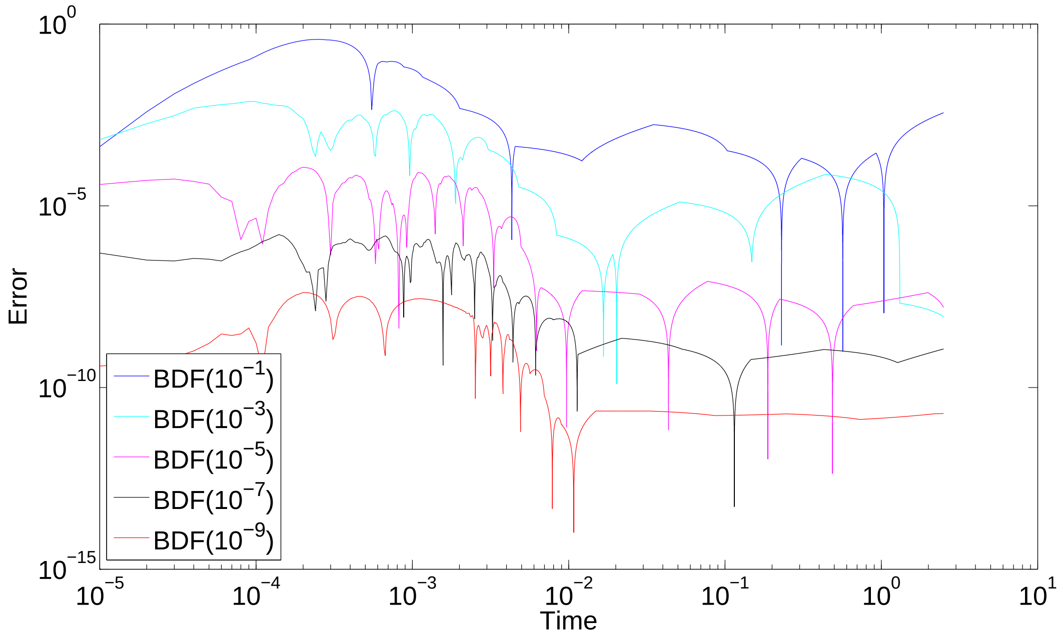


Figure B.25: Plots of the errors using BDF integrators for test case 7.

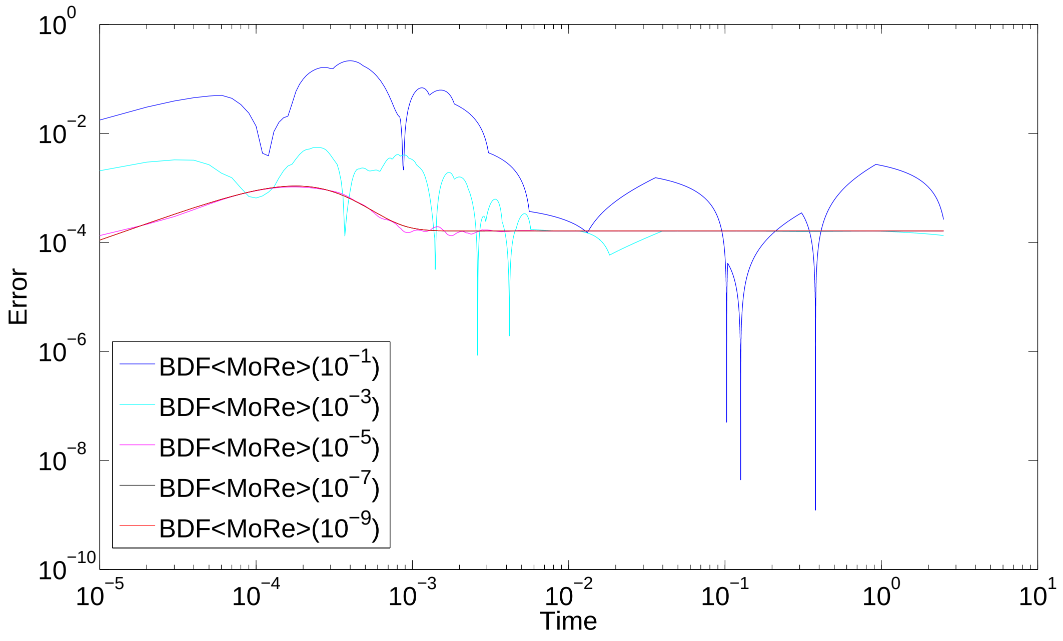


Figure B.26: Plots of the errors using BDF<MoRe> integrators for test case 7.

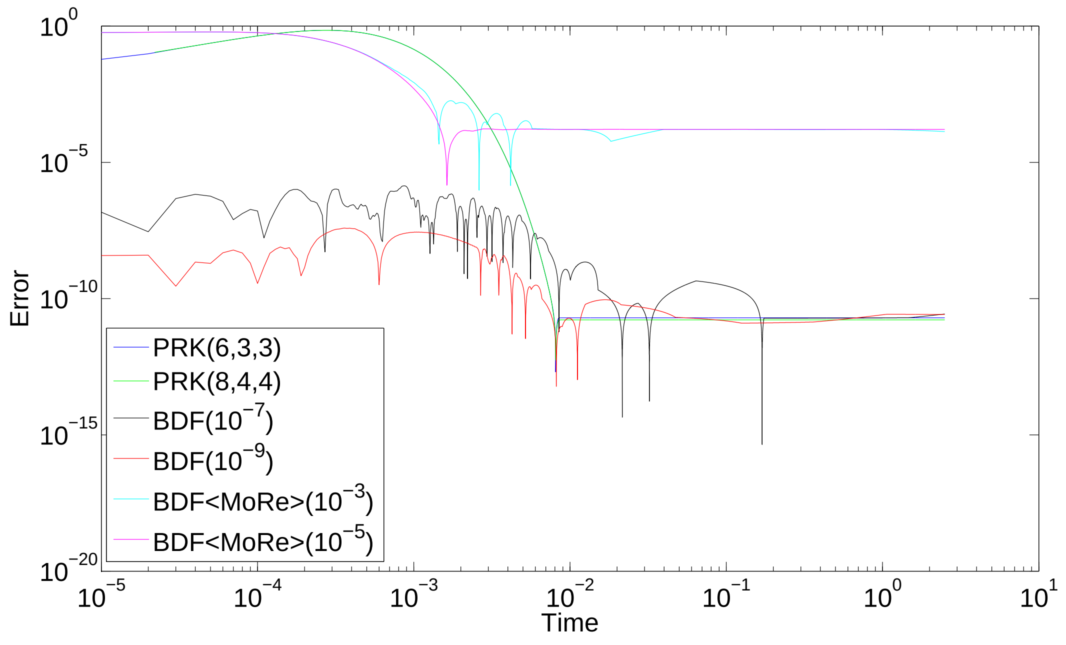


Figure B.27: Plots of the errors using various integrators for test case 8.

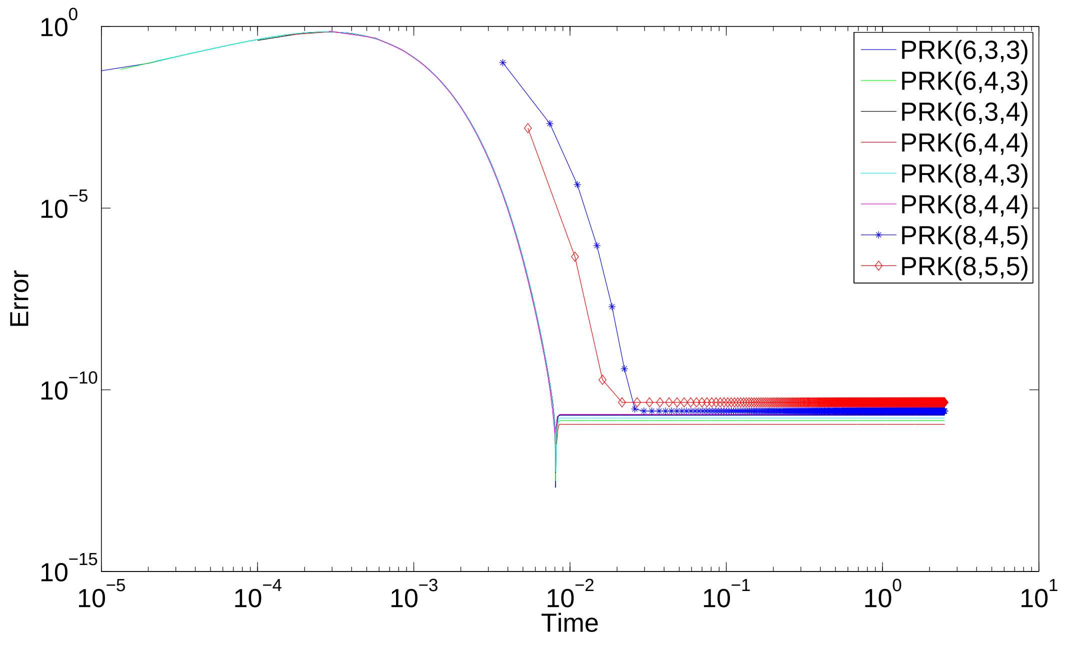


Figure B.28: Plots of the errors using PRK integrators for test case 8.

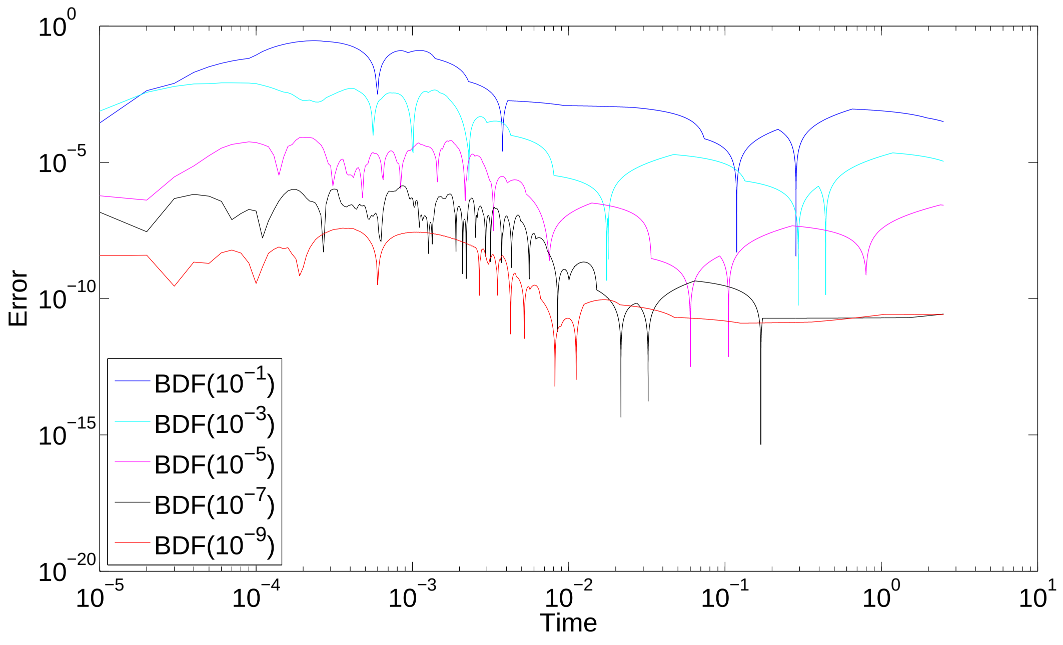


Figure B.29: Plots of the errors using BDF integrators for test case 8.

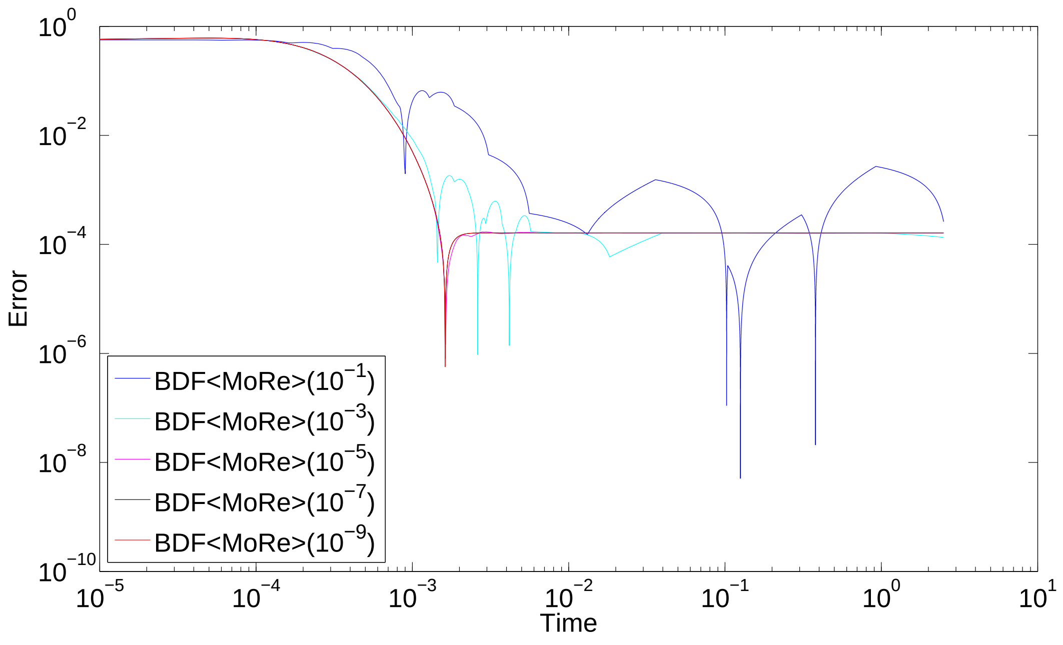


Figure B.30: Plots of the errors using BDF<MoRe> integrators for test case 8.

Appendix C

File prk_integrator.hpp

Listing C.1: The file prk_integrator.cpp

```
#ifndef __PRK_INTEGRATOR_HPP__
#define __PRK_INTEGRATOR_HPP__

#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <math.h>
#include <fstream>
#include <assert.h>
#include <flens/flens.cxx>

using namespace flens;

class PRK_Integrator {

    typedef flens::DenseVector<Array<double> > Vector;
    typedef flens::DenseVector<Array<double> >::IndexType
       IndexType;
    typedef flens::GeMatrix<FullStorage<double, ColMajor> >
       GeMatrix;
    const Underscore<IndexType> _;

    public:
    PRK_Integrator(void (*f)(double t, Vector x, Vector kf,
       Vector kr, Vector &fx),double tstart,double tend,
       unsigned int M,double h,unsigned int k,unsigned int L
       ,unsigned int dim, unsigned int type, std::string
       _prefix);
    ~PRK_Integrator();
    void getSettings();
    bool resetIntegration();
    bool setInitialValue(Vector y0);
    bool setFunction(Vector kf, Vector kr);
    bool performIntegration();
    bool getSolution(Vector &_t);
    void printStatistics();

    private:
    unsigned int M,k,L,dim, sizeV, type, fevals;
    double alpha,tstart,tend,h,tol;
    Vector y0,kf,kr;
    GeMatrix result;
    Vector time;
    std::string prefix = "";
    void (*f)(double t, Vector x, Vector kf, Vector kr,
       Vector &fx);
    double getAlpha();
    bool innerIntegrator(double t0,Vector y0,unsigned int q,
        double &t, Vector::View y);
    bool writeToFile();
    double norm2(Vector x);
};

#endif
```

Appendix D

File prk_integrator.cpp

Listing D.1: The file prk_integrator.cpp

```cpp
#include <prk_integrator.hpp>

//constructor
PRK_Integrator::PRK_Integrator(void (*_f)(double t, Vector x
    , Vector kf, Vector kr, Vector &fx), double _tstart,
    double _tend, unsigned int _M, double _h, unsigned int _k
    ,unsigned int _L, unsigned int _dim, unsigned int _type,
    std::string _prefix) {
     //set parameters
     f = _f;
     tstart = _tstart;
     tend = _tend;
     M = _M;
     h = _h;
     k = _k;
     L = _L;
     dim = _dim;
     type = _type;
     tol = 1e-16;
     prefix = _prefix;
     alpha = getAlpha();
     y0.resize(dim);
     kf.resize(dim);
     kr.resize(dim);
     //calculate number of steps
     sizeV = (unsigned int) (tend/(pow(M+k+1,L)*h) + 1);
     fevals = 0;
     //allocate memory for result and time
     result.resize(dim,sizeV);
     time.resize(sizeV);
};

//destructor
```

```
PRK_Integrator::~PRK_Integrator() {};

bool PRK_Integrator::setFunction(Vector _kf, Vector _kr) {
    kf = _kf;
    kr = _kr;
    return true;
};

bool PRK_Integrator::setInitialValue(Vector _y0) {
    y0 = _y0;
    return true;
};

void PRK_Integrator::getSettings() {
        std::cout << "Settings: " << std::endl;
        if (type == 0) {
            std::cout << "\ttype of integration =
               teleprojective forward euler (tpfe)" << std::
               endl;
        } else {
            std::cout << "\ttype of integration = projective
                runge kutta (prk)" << std::endl;
        }
        std::cout << "\tt in ["<<tstart<<","<<tend<<"] " <<
           std::endl;
        std::cout << "\tM = " << M << std::endl;
        std::cout << "\tk = " << k << std::endl;
        std::cout << "\tL = " << L << std::endl;
        std::cout << "\th = " << h << std::endl;
        std::cout << "\talpha = " << std::setprecision( 20 )
             << alpha << std::endl;
        std::cout << "\ty0 = [";
        for (unsigned int i = 1; i < dim; ++i) {
            std::cout << y0(i) << " ";
        }
        std::cout << y0(dim) << "]" << std::endl << std::
           endl;
};

bool PRK_Integrator::resetIntegration() {
    result.resize(0,0);
    y0.resize(0);
    return true;
};

bool PRK_Integrator::performIntegration() {
    bool isNearEquilibrium = false;
    result(_(1,dim),1) = y0;
    double told = tstart, tnew;

    /* perform teleprojective forward euler integration */
    if (type == 0) {
        for (unsigned int i = 1; i <= sizeV-1; ++i) {
            innerIntegrator(told,result(_(1,dim),i),L,tnew,
               result(_(1,dim),i+1));
            told = tnew;
            time(i+1) = tnew;
        }
    /* perform projective runge kutta integration */
    } else {
        for (unsigned int i = 1; i <= sizeV-1; ++i) {
            if (!isNearEquilibrium) {
                //set initial value
                GeMatrix step(dim,k+2);
                step(_(1,dim),1) = result(_(1,dim),i);
                told = time(i);
                for (unsigned int i = 1; i <= k+1; ++i) {
                    innerIntegrator(told, step(_(1,dim),i),
                       L-1, tnew, step(_(1,dim),i+1));
                    told = tnew;
                }
                /* set new time */
                double t = told + M*pow(k+1+ M,L-1)*h;
                time(i+1) = t;

                /* set initial value for y_{s} */
                GeMatrix step_pred(dim,k+2);
                Vector yk = step(_(1,dim),k+1), ykp1 = step(
                   _(1,dim),k+2);
                blas::scal((int)M*(-1.0),yk); // = -M*y_{k}
                blas::scal(M+1,ykp1); // (M+1)*y_{k+1}
                step_pred(_(1,dim),1) = yk + ykp1; // y = y_
                   {k+1} + M*( y_{k+1} - y_{k} )
                /* perform k+1 daming steps */
                told = t;
                for (unsigned int i = 1; i <= k+1; ++i) {
                    innerIntegrator(told, step_pred(_(1,dim),i), L-1, tnew, step_pred(_(1,dim),i+1));
                    told = tnew;
                }

                /* calculate a correted y_s */
                blas::scal((1+alpha*(int)M),step(_(1,dim),k+2));                // (1+alpha*M)*y_{k+1}
                blas::scal((int)M*(-1.0)*alpha,step(_(1,dim),k+1));             // -alpha*M*y_k
                blas::scal((int)M*(-1.0)*(1-alpha),step_pred(_(1,dim),k+1));    // -(1-alpha)*M*y_{n+k}
                blas::scal((int)M*(1-alpha),step_pred(_(1,dim),k+2));           // (1-alpha)*M*y_{n+k+1}
                result(_(1,dim),i+1) = step(_(1,dim),k+1) + step(_(1,dim),k+2) + step_pred(_(1,dim),k+1) + step_pred(_(1,dim),k+2);

                /* check if result is close to equilibrium */
                Vector err(dim);
                err = result(_(1,dim),i+1) - result(_(1,dim),i);
                if ( norm2(err) < tol ) {
                    isNearEquilibrium = true;
                }
            } else {
                //case: near equilibirum: do not calculate new values
                result(_(1,dim),i+1) = result(_(1,dim),i);
                time(i+1) = time(i) + pow(k+1+M,L)*h;
            }//end nearEquilibirum
        }
    }//end performing runge kutta
    return true;
};
```

```

bool PRK_Integrator::getSolution(Vector &_t) {
    _t = time;

    //print solution vector
    std::cout << "result_cpp = [" << std::endl;
    for( unsigned int i = 1; i <= sizeV; ++i) {
        for (unsigned l = 1; l <= dim; ++l) {
            std::cout << std::setw( 30 ) << std::
                setprecision( 20 )  << result(l,i) << " ";
        }
        std::cout << ";" << std::endl;
    }
    std::cout << "];" << std::endl;
    std::cout << "t = [" << std::setprecision( 5 )  << time
       << "];" << std::endl;

    //write also to file
    writeToFile();

    return true;
};
void PRK_Integrator::printStatistics() {
    std::cout << std::endl << "INTEGRATION STATISTIC:" <<
       std::endl;
    std::cout << "STEPS: " << sizeV << std::endl;
    std::cout << "F-EVAL: " << fevals << std::endl;
};

//calculate alpha, such that the algorithm is 2nd order
double PRK_Integrator::getAlpha() {
    int s = (int) M + k + 1;
    // xsi_0 if forward euler is used at innermost layer
    double xsi = 1.0;
    for (unsigned int i = 1; i <= L; ++i) {
        xsi = xsi/((double) s) + M*(M+1)/((double)(s*s));
    }
    //casting to integer, because dealing with -M
    return ((-(int)M*(int)k-(int)M-1)*xsi + M*(M+1+2*k))/((
       double)(2*M*s));
};

```

```
bool PRK_Integrator::innerIntegrator(double t0,Vector y0,
    unsigned int q, double &t, Vector::View y) {

    /* innermost layer: perform forward euler step */
    if ( q == 0 ) {
        /* evaluate function f */
        Vector fx(dim);
        f(t0,y0,kf,kr,fx);
        fevals++;
        /* calculate new value */
        blas::scal(h,fx); // h*fx
        y = y0 + fx;
        t = t0 + h;
    /* higer layer: perform a projective forward euler step
       depending on M, q, k */
    } else {
        /* set initial value */
        GeMatrix step(dim,k+2);
        step(_(1,dim),1) = y0;
        /* perform k+1 damping steps */
        double told = t0, tnew;
        for (unsigned int i = 1; i <= k+1; ++i) {
            innerIntegrator(told, step(_(1,dim),i), q-1,
                tnew, step(_(1,dim),i+1));
            told = tnew;
        }
        /* calculate y */
        t = told + M*pow(k+1+M,q-1)*h;
        blas::scal((int)M*(-1.0),step(_(1,dim),k+1)); // = -
            M*y_{k}
        blas::scal(M+1,step(_(1,dim),k+2)); // (M+1)*y_{k+1}
        y = step(_(1,dim),k+2) + step(_(1,dim),k+1); // y =
            y_{k+1} + M*( y_{k+1} - y_{k} )
    }

    return true;
};

bool PRK_Integrator::writeToFile() {
    std::fstream file, file_time;
    file.open(prefix+"result.dat", std::ios::out);
    for( unsigned int i = 1; i <= sizeV; ++i) {
        for (unsigned l = 1; l <= dim; ++l) {
            file << std::setw( 30 ) << std::setprecision( 20
                ) << result(l,i) << " ";
        }
        file << std::endl;
    }
    file.close();
    file_time.open(prefix+"time.dat", std::ios::out);
    file_time << std::setw( 30 ) << std::setprecision( 20 )
          << time;
    file_time.close();

    std::cout << "Wrote data to file result.dat and time.dat
       ." << std::endl;
    return true;
};

double PRK_Integrator::norm2(Vector x) {
    double norm = 0.0;
    for(int i = 1; i <= x.length(); ++i) {
        norm += x(i)*x(i);
    }
    norm = sqrt(norm);
    return norm;
};
```

List of Figures

List of Tables

Bibliography

[1] A. Zagaris, C. W. Gear, T. J. Kaper and I. G. Kevrekidis. Analysis of the Accuracy and Convergence of Equation-Free Projection to a Slow Manifold. *ESAIM: Mathematical Modelling and Numerical Aspects*, 43:757–784, 2009.

[2] A. N. S. Al-Khateeb. *Fine Scale Phenomea in Reacting Systems: Identification and Analysis for their Reduction.* PhD thesis, University of Notre Dame, 2010.

[3] M. Bodenstein. Eine Theorie der photochemischen Reaktionsgeschwindigkeiten. *Zeitschrift für Physikalische Chemie*, 85:329–397, 1913.

[4] C. F. Curtiss and J. O. Hirschfelder. Integration of stiff equations. *Proc Natl Acad Sci*, 38:235–243, 1952.

[5] C. W. Gear, T. J. Kaper, I. G. Kevrekidis and A. Zagaris. Projecting to a Slow Manifold: Singularly Perturbated Systems and Legacy Codes. *SIAM Journal of Applied Dynamical Systems*, 4:711–731, 2005.

[6] D. L. Chapman and L.K. Underhill. The Interaction of Chlorine and Hydrogen. The Influence of Mass. *Journal of Chemical Society, Transaction*, 103:496–508, 1913.

[7] D. Lebiedz, D. Skanda and M. Fein. Automatic Complexity Analysis and Model Reduction of Nonlinear Biochemical Systems. *Computational Methods in System Biology*, (123-140), Springer, 2008.

[8] D. Lebiedz, V. Reinhardt and J. Siehr. Minimal Curvature Trajectories: Riemannian Geometry Concepts for Slow Manifold Computation in Chemical Kinetics. *Journal of Computational Physics*, 229:6512–6533, 2010.

[9] E. Hairer and G.Wanner. *Solving Ordinary Differential Equations II.* Springer, 1991.

[10] N. Fenichel. Persistence and smoothness of invariant manifolds for flows. *Indiana University Mathematics Journal*, 21:192–226, 1972.

[11] N. Fenichel. Asymptotic stability with rate conditions. *Indiana University Mathematics Journal*, 23:1109–1137, 1974.

[12] N. Fenichel. Asymptotic stability with rate conditions ii. *Indiana University Mathematics Journal*, 26:81–93, 1977.

[13] N. Fenichel. Geometric singular perturbation theory for ordinary differential equations. *Journal of Differential Equations*, 31:53–98, 1979.

[14] C. W. Gear and I. G. Kevrekidis. Projective methods for stiff differential euqations: Problems with gaps in their eigenvalue spectrum. *SIAM Journal on Scientific Computing*, 24(4):1091–1106, 2002.

[15] C. W. Gear and I. G. Kevrekidis. Telescopic projective methods for parabolic differential equations. *Journal of Computational Physics*, 187(1):95–109, 2003.

[16] J. Li, Z. Zhao, A. Kazakov and F.L. Dryer. An updated comprehensive kinetic model of hydrogen combustion. *International Journal of Chemical Kinetics*, 36:566–575, 2004.

[17] C.K.R.T. Jones. *Geometric Singuar Perturbation Theory.* Number 1609 in Lecture Notes in Mathematics in R. Johnson "Dynamical Systems", chap. 2, pp. 44-118. Springer, 1995.

[18] L. Michaelis und M. L. Menten. Die Kinetik der Invertinwirkung. *Biochemische Zeitschrift*, 49:333–369, 1913.

[19] D. Lebiedz. Computing Minimal Entropy Production Trajectories: An Approach to Model Reduction in Chemical Kinetics. *Journal of Chemical Physics*, 120:6890–6897, 2004.

[20] D. Lebiedz. Optimal Control, Model- and Complexity-Reduction of Self-Organized Chemical and Biochemical Systems: A Scientific Computing Approach. *Habilitation thesis, University of Heidelberg*, 2006.

[21] D. Lebiedz. Entropy-Related Extremum Principles for Model Reduction of Dynamical Systems. *Entropy*, 12:706–719, 2010.

[22] M. Lehn. FLENS - Flexible Library for Efficient Numerical Solutions. Available on www.flens.sourceforge.net.

[23] M. Bodenstein and H. Lütkemeyer. Die photochemische Bildung von Bromwasserstoff und die Bildungsgeschwindigkeit der Brommolekel aus den Atomen. *Zeitschrift für Physikalische Chemie*, 114:208–236, 1924.

[24] L. Perko. *Differential Equations and Dynamical Systems.* Springer, 3. edition, 2001.

[25] V. Reinhardt. *On the Application of Trajectory-Based Optimization for Nonlinear Kinetic Model Reduction.* PhD thesis, University of Heidelberg, 2008.

[26] S. L. Lee and C. W. Gear. On-the-fly local error estimation for projective integrators. 2006.

[27] S. L. Lee and C. W. Gear. Second-order accurate projective integrators for multiscale problems. *Journal of Computational and Applied Mathematics*, 201(1):258–274, 2007.

[28] Jochen Siehr. *Numerical Optimization Methods within a Continuation Strategy for the Reduction of Chemical Combustion Models.* PhD thesis, University of Heidelberg, 2012. Submitted.

[29] D. Skanda. *Robust Optimal Experimental Design for Model Discrimination of Kinetic ODE Systems.* PhD thesis, University of Freiburg, 2012.

[30] J. Unger. *On the Analysis of an Optimazation Approach to Slow Manifold Computation in Chemical Kinetics*, Diplomarbeit, University of Freiburg, Diploma thesis. 2010.

[31] V. Reinhardt, M. Winckler and D. Lebiedz. Approximation of the Slow Attracting Manifolds in Chemical Kinetics by Trajectory-Based Optimization Approach. *Journal of Physical Chemistry A*, 112:1712–1718, 2008.

[32] M. Winckler. *Towards Optimal Criteria for Trajectory-Based Model Reduction in Chemical Kinetics via Numerical Optimization*, University of Heidelberg, Diploma thesis. 2007.

[33] Z. Ren, S.B. Pope, A.Vladimirsky and J.M. Guckenheimer. The invariant constrained equilibrium edge preimage curve method for the dimension reduction of chemical kinetics. *Journal of Chemical Physics*, 124:111–114, 2006.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben wird.

Ulm, den 26. November 2012

(Unterschrift)