Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

MATEMATICA

Ciclo 34

Settore Concorsuale: 01/A5 – ANALISI NUMERICA

Settore Scientifico Disciplinare: MAT/08 – ANALISI NUMERICA

ORDER REDUCTION OF SEMILINEAR DIFFERENTIAL MATRIX AND TENSOR EQUATIONS

Presentata da: Gerhardus Petrus Kirsten

Coordinatore Dottorato

Valeria Simoncini

Supervisore

Valeria Simoncini

Esame finale anno 2021

Abstract

In this thesis, we are interested in approximating, by model order reduction, the solution to large-scale matrix- or tensor-valued semilinear Ordinary Differential Equations (ODEs), as well as coupled systems of such equations. Under specific hypotheses on the linear operators and the considered domain, these types of ODEs often stem from the space discretization on a tensor basis of semilinear Partial Differential Equations (PDEs) with a dimension greater than or equal to two. PDEs of this form and their discrete counterparts play a significant role in the modeling and control of a large number of complex systems of industrial relevance or scientific interest.

The bulk of this thesis is devoted to the case where the discrete system is a matrix equation. We consider separately the cases of general Lipschitz continuous nonlinear functions and the Differential Riccati Equation (DRE) with a quadratic nonlinear term. In both settings, we construct a pair of left-right approximation spaces that leads to a reduced semilinear *matrix* differential equation with the same structure as the original problem, which can be more rapidly integrated with matrix-oriented integrators. The particular structure of the DRE, however, allows several computational advantages. More precisely, under certain assumptions on the data, we show that a reduction process onto rational Krylov subspaces obtains significant computational and memory savings as opposed to current approaches. Moreover, the nonlinear term can be explicitly projected onto the low-dimensional space.

In the more general setting, a challenging difference lies in selecting and constructing the two approximation bases to handle the nonlinear term effectively. In addition, the nonlinear term also needs to be approximated for efficiency. To this end, in the framework of the Proper Orthogonal Decomposition (POD) methodology and the Discrete Empirical Interpolation Method (DEIM), we derive a novel matrix-oriented reduction process leading to a practical, structure-aware low order approximation of the original problem. Furthermore, the reduction of the nonlinear term is also performed utilizing a fully matricial interpolation using left and right projections onto two distinct reduction spaces, giving rise to a new two-sided version of DEIM.

In the final part of the thesis, we consider the multidimensional setting. Here we extend the matrix-oriented POD-DEIM algorithm to the tensor setting and illustrate how we can apply it to systems of such equations. Moreover, we discuss how to integrate the reduced-order model and, in particular, how to solve the tensor-valued linear system arising at each timestep of a semi-implicit time discretization scheme. All proposed methods are supported by numerical experiments on typical benchmark problems, with comparisons to existing state-of-the-art procedures.

Acknowledgements

Writing a thesis of this magnitude, especially in a foreign country, is a daunting task if not surrounded by a beneficial support structure. I was aware of the challenges lying ahead when I first arrived in Italy. Still, I would have never imagined how uncomplicated these three years would end up being. There are many people to thank for this, but none more than my supervisor Prof. Valeria Simoncini. It has been an absolute privilege working under her profound leadership. Not only did she guide me in the best possible academic way to complete my thesis in three years while offering me many networking and traveling opportunities, but she also cared about my personal life and happiness in Italy. For that, I am very grateful.

I have been privileged to have been raised by two academically influential parents. I have never had a fear of studying, thanks to the incredible academic leadership that they have offered me. It hasn't been any different in the past three years, despite living on other ends of the globe. I am also extremely grateful for how they unselfishly motivated me to go and live and study abroad. In his absence, I also need to acknowledge my grandfather. His academic aura also greatly encouraged me to work towards getting a Ph.D. of my own.

Then, I certainly also need to acknowledge the friends I have made in Bologna, as well as my friends back home. I am a big believer in "work hard, play hard", and it has been an absolute pleasure to spend the more relaxed moments in these three years with the most exciting people who are now scattered all over the world.

Finally, I need to acknowledge the institute of advanced studies (ISA). Prof. Braga and his colleagues accepted me as an international ISA fellow at the beginning of the Ph.D. program. This enabled me to live for free in Bologna and offered me exciting networking opportunities.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

2S-DEIM	**T**wo **S**ided **D**iscrete **E**mpirical **I**nterpolation **M**ethod
2S-POD	**T**wo **S**ided **P**roper **O**rthogonal **D**ecomposition
ARE	**A**lgebraic **R**iccati **E**quation
BDF	**B**ackward **D**ifference **F**ormulae
CFL	**C**ourant-**F**riedrichs- **L**ewy
DEIM	**D**iscrete **E**mpirical **I**nterpolation **M**ethod
DRE	**D**ifferential **R**iccati **E**quation
EIM	**E**mpirical **I**nterpolation **M**ethod
EKSM	**E**xtended **K**rylov **S**ubspace **M**ethod
ETD	**E**xponential **T**ime **D**ifferencing
HO-DEIM	**H**igher **O**rder **D**iscrete **E**mpirical **I**nterpolation **M**ethod
HO-POD	**H**igher **O**rder **P**roper **O**rthogonal **D**ecomposition
IMEX	**IM**plicit–**EX**plicit
MGS	**M**odified **G**ram **S**chmidt
ODE	**O**rdinary **D**ifferential **E**quation
PDE	**P**artial **D**ifferential **E**quation
POD	**P**roper **O**rthogonal **D**ecomposition
RKSM	**R**ational **K**rylov **S**ubspace **M**ethod
SI	**S**emi **I**mplicit
STHOSVD	**S**equentially **T**runcated **H**igher **O**rder **S**ingular **V**alue **D**ecomposition
SVD	**S**ingular **V**alue **D**ecomposition

Aan Oupa Grootbeer.

Introduction

Motivation and goal

Besides theoretical analysis and experimentation, computational science and engineering disciplines can be seen as the third pillar of the 21st century. It is mentioned in [45] that

> *computational science is a rapidly growing multidisciplinary field that uses advanced computing capabilities to understand and solve complex problems.*

A common practice in computational science and engineering involves the efficient simulation of dynamical systems; this forms a basis for modeling and control of a large collection of complex systems of industrial importance or scientific interest. A condensed list of examples includes heat transfer, fluid dynamics, electric circuit analysis, biological systems, aeronautical engineering, and chemical reactant flows.

These dynamical systems typically stem from the discretization of an underlying partial differential equation (PDE). With the rapid increase in computational power and industrial demand, two interconnected issues lead to an increase in the complexity of the discretized models. Firstly, the need for improved accuracy in the approximation of the continuous problem results in more detail required in the modeling stage. That is an increase in the dimension of the discrete dynamical system, which results in more extensive memory requirements and higher computational complexity in the simulation of the model. Secondly, the demand for improved system performance results in repetitive simulation of different realizations of the complex dynamical system, which ultimately leads to unmanageably great demands on the available computational resources.

A related issue is that to capture real-life applications' dynamics efficiently, the considered models are often nonlinear and even parameter-dependent. It is well-known that nonlinear models are far more challenging to solve than their linear counterparts, making the simulation of these large-scale, complex models even more cumbersome. As a result, alleviating this computational burden – in terms of memory requirements and computational complexity – is the foundation of *model order reduction*; the derivation of efficient low-dimensional models that can provide similar accuracy to the high-fidelity model, but at a fraction of the computational complexity. The success of model order reduction is due to the observation that the solutions of the

considered systems are typically not scattered all over the high-dimensional solution space. Instead, they generally are attracted to a low-dimensional manifold, which can be well approximated by a low-dimensional (linear) reduced space.

An extensive literature is dedicated towards model order reduction of dynamical systems of the form

$$\dot{\boldsymbol{u}}(t) = \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{f}(\boldsymbol{u}, t) + \boldsymbol{c}, \quad \boldsymbol{u}(0) = \boldsymbol{u}_0, \tag{1}$$

where the solution $\boldsymbol{u}$ is a vector with N entries and $\boldsymbol{f}$ is a nonlinear function; see for example [11] and the recent survey [25] and the references therein. Standard techniques for reducing the dimension of (1) rely on constructing a low-dimensional approximation space based on information obtained from the high-fidelity model. These are typically referred to as *snapshots* of the full order model. Then, once the basis vectors spanning the approximation space have been obtained, the original data is projected onto the low-dimensional space. As a result, a reduced model of the form (1) can be rapidly simulated for different realizations of the problem, with a computational complexity independent of the full dimension N. Nevertheless, despite an extensive effort towards decreasing the complexity of the reduced model, the overwhelming computational cost and memory requirements of constructing the basis vectors from high-fidelity simulations remain a concern.

In this thesis we aim to address precisely this issue. By taking full account of the structure of the original problem under certain assumptions on the underlying model, the dynamical system (1) can be recast into an array (i.e., matrix or tensor) formulation, which may lead to a better structural understanding of the problem and reduced memory requirements. Consequently, we are interested in numerically approximating, by model order reduction, the solution $\boldsymbol{\mathcal{U}} \in S$ of the following semilinear *matrix*- or *tensor*-valued ordinary differential equation (ODE)

$$\dot{\boldsymbol{\mathcal{U}}}(t) = \mathcal{A}(\boldsymbol{\mathcal{U}}(t)) + \mathcal{F}(\boldsymbol{\mathcal{U}}, t) + \boldsymbol{\mathcal{C}}, \qquad \boldsymbol{\mathcal{U}}(0) = \boldsymbol{\mathcal{U}}_0, \quad t \in [0, t_f] = T. \tag{2}$$

Here $\boldsymbol{\mathcal{C}}$ is a constant term, $\mathcal{A} : S \to R$ is a linear operator, the function $\mathcal{F} : S \times T \to R$ is a nonlinear function that can be evaluated elementwise, S is an appropriate functional space containing the sought after solution, and R is the appropriate matrix or tensor space, dependent on the dimensions of the discretization. ODEs of this form typically arise from the discretization of high-dimensional semilinear PDEs (dimension two or higher) on a tensor basis, for certain choices of the physical domain [51, 123, 142].

The bulk of the thesis is devoted to the setting where $\boldsymbol{\mathcal{U}}(t)$ is a matrix $\boldsymbol{U}(t) \in \mathbb{R}^{n_1 \times n_2}$. Accordingly, (2) can be reformulated as the following semilinear matrix differential equation

$$\dot{\boldsymbol{U}}(t) = \boldsymbol{A}_1\boldsymbol{U}(t) + \boldsymbol{U}(t)\boldsymbol{A}_2 + \mathcal{F}(\boldsymbol{U}, t) + \boldsymbol{C}, \quad \boldsymbol{U}(0) = \boldsymbol{U}_0, \tag{3}$$

where $\boldsymbol{A}_1 \in \mathbb{R}^{n_1 \times n_1}, \boldsymbol{A}_2 \in \mathbb{R}^{n_2 \times n_2}, \boldsymbol{C} \in \mathbb{R}^{n_1 \times n_2}$ are *constant* matrices, independent of t. In this thesis, we develop matrix-oriented order reduction strategies for the

problem (3) that leads to a semilinear *matrix* differential equation with the same structure as (3), but of significantly reduced dimension. More precisely, we determine an approximation to $\boldsymbol{U}(t)$ of the type

$$\boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top, \quad t \in T \tag{4}$$

where $\boldsymbol{V}_1 \in \mathbb{R}^{n_1\times k_1}$ and $\boldsymbol{V}_2 \in \mathbb{R}^{n_2\times k_2}$ are matrices to be determined, independent of time, where $k_1, k_2 \ll n_1, n_2$ and $\widehat{\boldsymbol{U}}(t)$ is a far smaller matrix. The function $\widehat{\boldsymbol{U}}(t)$ is determined as the numerical solution to the following *reduced* semilinear matrix differential problem

$$\begin{aligned}\dot{\widehat{\boldsymbol{U}}}(t) &= \widehat{\boldsymbol{A}}_1\widehat{\boldsymbol{U}}(t) + \widehat{\boldsymbol{U}}(t)\widehat{\boldsymbol{A}}_2 + \widehat{\mathcal{F}(\widehat{\boldsymbol{U}}, t)} + \widehat{\boldsymbol{C}}\\ \widehat{\boldsymbol{U}}(0) &= \boldsymbol{V}_1^\top\boldsymbol{U}_0\boldsymbol{V}_2,\end{aligned} \tag{5}$$

with $\widehat{\boldsymbol{A}}_1 = \boldsymbol{V}_1^\top\boldsymbol{A}_1\boldsymbol{V}_1$, $\widehat{\boldsymbol{A}}_2 = \boldsymbol{V}_2^\top\boldsymbol{A}_2\boldsymbol{V}_2$, $\widehat{\boldsymbol{C}} = \boldsymbol{V}_1^\top\boldsymbol{C}\boldsymbol{V}_2$, and $\widehat{\mathcal{F}(\widehat{\boldsymbol{U}}, t)}$ is an approximation to

$$\widehat{\mathcal{F}}(\widehat{\boldsymbol{U}}, t) = \boldsymbol{V}_1^\top\mathcal{F}(\boldsymbol{V}_1\widehat{\boldsymbol{U}}\boldsymbol{V}_2^\top, t)\boldsymbol{V}_2 \tag{6}$$

if the explicit computation of (6) is not feasible in low-dimension.

For $\mathcal{F} \equiv 0$, the equation (3) simplifies to the differential Sylvester equation, for which reduction methods have shown to be competitive; see, e.g., [20] and references therein. If the function $\mathcal{F}$ has a special quadratic structure of the form $\mathcal{F}(\boldsymbol{U}, t) \equiv -\boldsymbol{U}(t)\boldsymbol{B}\boldsymbol{U}(t)$, with $\boldsymbol{B} \in \mathbb{R}^{n_2\times n_1}$, the resulting ODE is known as the Differential Riccati Equation (DRE), which plays a primal role in the optimal control of linear dynamical systems with quadratic cost functions [1, 46, 105]. Approximations of the form (4) are also not unheard of in this quadratic context. Indeed, the authors of [99] and [75] have independently applied an approximation of the form (4) to approximate the solution of the DRE, under certain assumptions on the considered data. Their approaches are realized by applying polynomial and extended Krylov subspaces as approximation spaces; see, e.g., [11] for further details regarding these approximation spaces. Furthermore, a similar approximation is considered in [21], where the approximation space stems from the (low-rank) solution of the related *Algebraic* Riccati Equation (ARE). Consequently, since approximations of the form (4) are not novel for the DRE, our contribution towards this quadratic context, which we treat separately from the more general setting, can rather be formulated as follows:

Contribution 1. *We show that, under certain assumptions on the data, a reduction process onto rational Krylov subspaces obtains great computational and memory savings as opposed to current approaches. This reduction process is realized by specifically addressing the solution of the reduced differential equation and reliable stopping criteria, which allows us to obtain accurate final approximations at low memory and computational requirements.*

The unique quadratic structure of the DRE simplifies the reduction procedure. This is because Krylov subspaces which are typically applied to *linear* equations, approximate the solution to the *quadratic* problem well, as discussed in [19, 21, 141, 144], and the explicit computation of (6) can be done in low dimension. However, in the more general context, where $\mathcal{F}$ is considered as any Lipschitz continuous nonlinear function, the reduction process is undoubtedly more challenging. A significant difference lies in the selection and construction of the two matrices $\boldsymbol{V}_1, \boldsymbol{V}_2$, to effectively handle the nonlinear term in (3). In addition, by definition of (6), the reduction of the nonlinear term requires evaluating $\mathcal{F}$ in high dimension before projection onto the low-dimensional subspace, resulting in a computational cost dependent on the full space dimension. To this end, the nonlinear function also needs to be approximated for efficiency.

By stacking the columns of the matrix $\boldsymbol{U}(t)$ one after the other into a long vector, a collection of existing approaches typically map the matrix-valued problem (3) to the vectorized dynamical system (1), for which order reduction is a well-established procedure. Among various methods, the Proper Orthogonal Decomposition (POD) methodology has been widely employed, as it mainly relies on solution samples, rather than the a-priori generation of an appropriate basis [27],[24],[82],[104]. Other approaches include reduced basis methods, see, e.g., [124], and rational interpolation strategies [10]; see, e.g., [23] for an overview of the most common reduction strategies. The overall effectiveness of the POD procedure is largely influenced by the capability of evaluating the nonlinear term within the reduced space, motivating a considerable amount of work towards this estimation, including quadratic bilinear approximation [25, 73, 101] and trajectory piecewise-linear approximation [162]. Alternatively, several approaches consider interpolating the nonlinear function, such as missing point estimation [15] and the best points interpolation method [120]. One very successful approach is the Discrete Empirical Interpolation Method (DEIM) [41], which is based on the Empirical Interpolation Method (EIM) originally introduced in [17].

As mentioned earlier, a shortcoming of this framework is the massive computational and storage demand when constructing the reduced model. Even when simulating the reduced model, several vectors of length N need to be stored to lift the low-dimensional functions back to the full dimension. In this thesis, we strive to contribute precisely to this shortcoming, focusing on POD for dimension reduction and DEIM to interpolate the nonlinear function.

Contribution 2. *We devise a matrix-oriented POD approach tailored towards the construction of the matrix reduced problem formulation (5); this is realized with the generation of the pair $\boldsymbol{V}_1, \boldsymbol{V}_2$ of left and right space bases, respectively, for the representation in (4). In addition, we develop an adaptive procedure to limit the number of snapshots that contribute to the generation of the approximation spaces. The reduction of the nonlinear term is also performed employing a fully matricial interpolation*

using left and right projections onto two distinct reduction spaces, giving rise to a new two-sided version of DEIM.

The idea of using left and right reduction bases in a matrix-oriented setting is not new in the general context of semilinear differential equations (see Section 4.2.2). Nonetheless, after reduction, these strategies resume the vector form of (5) for integration purposes, thus losing the structural and computational benefits of the matrix formulation. We claim that once (5) is obtained, matrix-oriented integrators should be employed. In other words, **by combining matrix-oriented versions of POD, DEIM, and ODE integrators, we can carry the whole approximation with explicit reference to the two-dimensional computational domain.**

The advantages of the proposed matrix-oriented methodology become even more apparent in the three-dimensional (3D) case. In the final part of the thesis, we explore precisely this extension to the tensor setting as well as the application to coupled systems of equations of the form (2). This extension naturally presents some extra challenges in comparison to the two-dimensional setting. Firstly, the increase in dimension requires unmanageably large demands on the computational resources, especially if the system is vectorized. Furthermore, if the system is not vectorized but instead reduced in such a way that the structure of (2) is preserved, the integration of such a reduced system is not a trivial task without reverting to vectorization. We contribute to this challenging problem as follows:

Contribution 3. *We investigate the possibility of applying a higher-order POD-DEIM algorithm to the system* (2) *to form a tensor-valued reduced model with the same structure as the original problem. Furthermore, we illustrate how standard ODE solvers can be applied to tensor-valued differential equations and how to solve the resulting linear systems. We also discuss how to efficiently treat coupled systems of equations of the form* (2)*, which typically arise in real-life simulations.*

Thesis outline and scope

The thesis is organized as follows:

- Chapter 1 is introductory. We discuss important notation and preliminary definitions in Section 1.1, and illustrate how to derive the problem (2) from the space discretization of semilinear evolutive PDEs in Section 1.2. In Section 1.3 we present some crucial tools for model reduction. More precisely, Krylov subspaces and their related Arnoldi relations are discussed in Section 1.3.1, whereas the standard POD and DEIM methods are respectively presented in Sections 1.3.2 and 1.3.3.

- Chapter 2 is both introductory and motivational. We present matrix-oriented versions of standard integrators, tailored towards ODEs of the form (3). Section 2.1 is devoted to semi-implicit (SI) matrix methods for general semilinear ODEs. In particular, in Section 2.1.1 we present the methods, derive their matrix formulations and discuss their convergence and stability properties. The efficient implementation of the procedures is discussed in Section 2.1.2, and comparisons to vector-based schemes are presented in Section 2.1.3. Section 2.2 presents matrix methods for the quadratic DRE. More precisely, the fully implicit matrix-oriented BDF methods and splitting schemes are respectively presented Sections 2.2.1 and 2.2.2. Our conclusions are formulated in Section 2.3.

- Chapter 3 is dedicated towards semilinear ODEs of the form (3) with a special quadratic nonlinear term, i.e., the DRE. In Section 3.1 we introduce reduction methods for the DRE and discuss the use of Krylov subspace-based strategies. In Section 3.2 we devise a stopping criterion for the order reduction methods and illustrate its vital role in the implementation. Section 3.3 is devoted to the analysis of matrix properties of the solution, as well as the reduced model, from a control theory perspective. Several numerical experiments are reported in Section 3.4, where the new methods are also compared with state-of-the-art procedures. Our conclusions are discussed in Section 3.5.

- Chapter 4 treats the matrix setting with general nonlinear terms. We first review how the standard POD-DEIM algorithm is applied to systems of the form (1), after which our new two-sided POD-DEIM algorithm is presented in Section 4.2. In particular, the two-sided POD is derived in Section 4.2.1, and in Section 4.2.2 we discuss the relation to other matrix-based interpolation strategies. In Section 4.2.3 we present a dynamical procedure for selecting the snapshots. Section 4.2.4 is devoted to the crucial approximation of the nonlinear function by the new two-sided DEIM. The overall new procedure with the numerical treatment of the reduced differential problem is summarized in Section 4.2.5, together with technical implementation details and computational costs. Numerical experiments are reported in Section 4.2.6 to illustrate the effectiveness of the proposed procedure, and our conclusions are presented in Section 4.3.

- In Chapter 5 we extend the procedure from Chapter 4 to three dimensions. More precisely, in Section 5.1 we extend POD-DEIM to the tensor setting and discuss several crucial implementational aspects such as time discretization and the solution of tensor linear systems. We illustrate how the new procedure can be applied to systems of array-valued ODEs in Section 5.2, and the efficiency of the method is illustrated by numerical experiments in Section 5.3. Our conclusions are formalized in Section 5.4.

- The thesis is concluded with a discussion of results, shortcomings and future research lines in Chapter 6.

Since the bulk of the thesis is devoted to the setting where (2) is a matrix equation, we will retain the matrix notation of (3) throughout the thesis and only revert to the tensor notation in Chapter 5. See Section 1.1 for a discussion regarding further notational nuances.

Software

All reported experiments have been performed using MATLAB 9.9 (R2020b) [113] on a MacBook Pro with 8-GB memory and a 2.3-GHz Intel core i5 processor. The codes implementing the main algorithms developed in Chapters 3, 4 and 5 can be downloaded from

`https://github.com/Gerhard-Kirsten`

Each package includes a driver implementing one of the experiments reported in the respective chapters together with the source code.

Chapter 1

Preliminaries

1.1 Notation and preliminary definitions

Here we overview the main notation used in this thesis and recall some essential definitions used in the sequel.

General notation.

Scalar quantities are indicated by lower case letters, and vectors are denoted by boldface lower case letters. Tensors are given by boldface, curly upper case letters, spaces by standard upper case letters and operators by standard curly upper case letters. Matrices are mainly given by boldface upper case letters; however, matrix blocks are denoted by upper case sans serif font, that is $\boldsymbol{M} = [\mathsf{M}_1, \mathsf{M}_2]$. $\boldsymbol{I}_n$ denotes the $n \times n$ identity matrix and $\mathbf{0}_n$ denotes the $n \times n$ matrix of zeros. The subscript n is omitted if the dimension is clear from the context. For a matrix $\boldsymbol{M}$, $\|\boldsymbol{M}\|$ denotes the matrix norm induced by the Euclidean vector norm, and $\|\boldsymbol{M}\|_F$ is the Frobenius norm.

Notation used for model reduction.

All reduced dimensional quantities are emphasized with a '$\widehat{\ }$'. In this thesis, we consider several different subspaces for reduced-order modeling. In Table 1.1 we summarize the notation used for the different basis matrices, that is, the matrices containing the orthonormal basis vectors spanning the respective subspaces. Notice that in order to avoid ambiguity, we use curly letters for the Krylov subspaces instead of boldface.

TABLE 1.1: Summary of the different notation used for basis matrices.

Matrix	Description	Dimension
$\mathcal{V}_k$	Krylov subspace basis matrix obtained after k iterations	n_v
$\boldsymbol{V}_{\text{VEC}}$	POD basis matrix in the vector setting	k
$\boldsymbol{\Phi}_{\text{VEC}}$	DEIM basis matrix in the vector setting	p
$\boldsymbol{V}_m$	POD basis matrix in the mth mode	k_m
$\boldsymbol{\Phi}_m$	DEIM basis matrix in the mth mode	p_m
$\boldsymbol{V}_{m,i}$	POD basis matrix in the mth mode for the ith equation	k_{mi}
$\boldsymbol{\Phi}_{m,i}$	DEIM basis matrix in the mth mode for the ith equation	p_{mi}

For all reduced quantities stemming from projection onto a Krylov subspace, we will emphasize the number of iterations needed to obtain the reduced model with a subscript k, e.g., $\widehat{\boldsymbol{M}}_k = \mathcal{V}_k^\top \boldsymbol{M} \mathcal{V}_k$. In the more general setting (that it, not Krylov subspaces), the subscript m always refers to the *mode* (see below) along which the matrix is acting. Furthermore, to differentiate between time instances used for the full and reduced models, we respectively use the notation t_j and $\mathfrak{t}_j$. Finally, the low-rank decomposition of a rank r symmetric matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ will be written as $\boldsymbol{M} = \boldsymbol{M}^\ell \left(\boldsymbol{M}^\ell\right)^\top$, with $\boldsymbol{M}^\ell \in \mathbb{R}^{n\times r}$, where $r \ll n$.

Tensor notation and definitions.

For a third-order tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n_1\times n_2\times n_3}$, the unfolding along the third mode is given by (see e.g., [97])

$$\boldsymbol{\mathcal{T}}_{(3)} = \left(\boldsymbol{T}_1, \boldsymbol{T}_2, \cdots, \boldsymbol{T}_{n_2}\right),$$

where $\boldsymbol{\mathcal{T}}_{(3)}$ is a matrix in $\mathbb{R}^{n_3\times n_1 n_2}$, and $\boldsymbol{T}_i \in \mathbb{R}^{n_3\times n_1}, i = 1, 2, \ldots, n_2$ is called a *lateral slice*. The multiplication of a tensor by a matrix, along a specific mode is done via the $m-$mode product, which, for a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{n_1\times n_2\times n_3}$ and a matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n_m}$, we express as

$$\boldsymbol{\mathcal{Q}} = \boldsymbol{\mathcal{T}} \times_m \boldsymbol{M} \quad \Longleftrightarrow \quad \boldsymbol{\mathcal{Q}}_{(m)} = \boldsymbol{M}\boldsymbol{\mathcal{T}}_{(m)}.$$

The *Kronecker product* of two matrices $\boldsymbol{M} \in \mathbb{R}^{m_1\times m_2}$ and $\boldsymbol{N} \in \mathbb{R}^{n_1\times n_2}$ is defined as

$$\boldsymbol{M} \otimes \boldsymbol{N} = \begin{pmatrix} M_{1,1}\boldsymbol{N} & \cdots & M_{1,m_2}\boldsymbol{N} \\ \vdots & \ddots & \vdots \\ M_{m_1,1}\boldsymbol{N} & \cdots & M_{m_1,m_2}\boldsymbol{N} \end{pmatrix} \in \mathbb{R}^{m_1 n_1\times m_2 n_2},$$

and the vec$(\cdot)$ operator maps the entries of a matrix, into a long vector, by stacking the columns of the matrix one after the other. The vectorization operator is applied to a third-order tensor via the first mode unfolding. Moreover, we will often make use of the property

$$(\boldsymbol{M} \otimes \boldsymbol{N})\text{vec}(\boldsymbol{X}) = \text{vec}(\boldsymbol{N}\boldsymbol{X}\boldsymbol{M}^\top). \tag{1.1}$$

As a result, if $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1\times n_2\times n_3}$, and $\mathbf{X} = \boldsymbol{\mathcal{X}}_{(3)}^\top$, then

$$(\mathbf{L} \otimes \mathbf{M} \otimes \mathbf{N})\text{vec}\left(\boldsymbol{\mathcal{X}}\right) = \text{vec}\left((\mathbf{M} \otimes \mathbf{N})\mathbf{X}\mathbf{L}^\top\right). \tag{1.2}$$

More properties used in the sequel are (see, e.g., [70]): (i) $(\boldsymbol{M} \otimes \boldsymbol{N})^\top = \boldsymbol{M}^\top \otimes \boldsymbol{N}^\top$; (ii) $(\boldsymbol{M}_1 \otimes \boldsymbol{N}_1)(\boldsymbol{M}_2 \otimes \boldsymbol{N}_2) = (\boldsymbol{M}_1\boldsymbol{M}_2 \otimes \boldsymbol{N}_1\boldsymbol{N}_2)$; (iii) $\|\boldsymbol{M} \otimes \boldsymbol{N}\|_2 = \|\boldsymbol{M}\|_2\|\boldsymbol{N}\|_2$; and (iv) $(\boldsymbol{M} \otimes \boldsymbol{N})^{-1} = \boldsymbol{M}^{-1} \otimes \boldsymbol{N}^{-1}$, where property (iv) holds if and only if both $\boldsymbol{M}$ and $\boldsymbol{N}$ are invertible.

Further preliminary definitions.

A matrix $\boldsymbol{M}$ is *stable* (sometimes also called Hurwitz) if all its *eigenvalues* are contained in the left half-open complex plane. A linear dynamical system, $\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x}$, is

called *dissipative* if the real matrix $\boldsymbol{A}$ has its field of values contained in the left half open complex plane. Furthermore, a pair $(\boldsymbol{A}, \boldsymbol{B}^\ell)$, with $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B}^\ell \in \mathbb{R}^{n\times n_b}$ and $n_b \leq n$, is said to be *stabilizable* (or controllable) if the matrix

$$\left(\boldsymbol{B}^\ell, \boldsymbol{A}\boldsymbol{B}^\ell, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{B}^\ell\right)$$

has full rank, i.e., rank n. A pair $(\boldsymbol{A}, \boldsymbol{C}^\ell)$ is *detectable* (or observable) if $(\boldsymbol{A}^\top, (\boldsymbol{C}^\ell)^\top)$ is stabilizable.

We define $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{n_1\times\cdots\times n_d\times n_s}$ as a snapshot tensor of order $d+1$ containing a collection of n_s tensor snapshots of order d. In the well-known case where the snapshots are vectors, this operation corresponds to collecting the vector snapshots into a *snapshot matrix*. Instead, we will sometimes deal with snapshots of higher dimensions, which results in the definition of a *snapshot tensor*.

Finally, we will often refer to the *offline* and *online* phases of the model reduction procedure. The *offline* phase is a computationally demanding phase, typically performed once, where high-fidelity snapshots are determined and processed to construct the basis vectors spanning the reduced space. Once the reduced model has been determined, it can be rapidly and efficiently simulated for different realizations of the model in the *online* phase, preferably with a computational cost independent of the full space dimension.

1.2 Matrix and tensor-based discretization of semilinear PDEs

Here we illustrate how, under certain assumptions, semilinear evolutive PDEs can be discretized and represented in matrix or tensor form. Consider a semilinear evolutive PDE of the form

$$u_t = \mathcal{L}(u) + f(u,t) + c(\boldsymbol{x}), \quad u = u(\boldsymbol{x},t) \quad \text{with } \boldsymbol{x} \in \Omega \subset \mathbb{R}^d, \ t \in T, \tag{1.3}$$

with suitable boundary consitions. We assume that the differential operator $\mathcal{L}$ is linear in u with seperable coefficients, typically a second order operator in the space variables, while $f : S_{\text{PDE}} \times T \to \mathbb{R}$ is a Lipschitz continuous nonlinear function, where S_{PDE} is an appropriate space with $u \in S_{\text{PDE}}$. In this section we illustrate that under certain hypotheses, the continuous PDE (1.3) can be discretized directly in matrix (tensor) form when $d = 2$ ($d > 2$); see e.g., [51, 123, 142]. In addition to a better structural interpretation of the discrete quantities, this formulation can also lead to reduced memory requirements and computational costs.

Standard procedures employ a vector-oriented approach: semi-discretization of (1.3) in space, such as finite difference and finite element methods, leads to the following

system of ordinary differential equations (ODEs)

$$\dot{\boldsymbol{u}}(t) = \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{f}(\boldsymbol{u}, t) + \boldsymbol{c}, \quad \boldsymbol{u}(0) = \boldsymbol{u}_0. \tag{1.4}$$

For $t > 0$, the vector $\boldsymbol{u}(t)$ contains the representation coefficients of the sought after solution in the chosen discrete space, $\boldsymbol{L} \in \mathbb{R}^{N\times N}$ accounts for the discretization of the linear differential operator $\mathcal{L}$ and $\boldsymbol{f}$ is evaluated componentwise at $\boldsymbol{u}(t)$. Furthermore $\boldsymbol{c} \in \mathbb{R}^N$ is a constant vector. If finite differences are the considered space discretization scheme, then $N = \prod_{m=1}^{d} n_m$, where n_m is the number of discretization nodes in the mth spatial direction.

Instead, if $\mathcal{L}$ is discretized by means of a tensor basis, such as finite differences on parallelepipedal domains and certain spectral methods, then the physical domain can be mapped to a reference hypercubic domain $\Omega = [a_1, b_1] \times \cdots \times [a_d, b_d]$. Hence, it holds that[1] (see e.g., [123])

$$\boldsymbol{L} = \sum_{m=1}^{d} \boldsymbol{I}_{n_d} \otimes \cdots \otimes \overset{m}{\boldsymbol{A}}_m \otimes \cdots \otimes \boldsymbol{I}_{n_1} \in \mathbb{R}^{N\times N}, \tag{1.5}$$

where $\boldsymbol{A}_m \in \mathbb{R}^{n_m \times n_m}$ contains the approximation of the second derivative in the x_m direction. The vector $\boldsymbol{u}(t) \in \mathbb{R}^N$ from (1.4) then represents the vectorization of the elements of a tensor $\boldsymbol{\mathcal{U}}(t) \in \mathbb{R}^{n_1\times\cdots\times n_d}$, such that $\boldsymbol{u}(t) = \operatorname{vec}(\boldsymbol{\mathcal{U}}(t))$, and $\boldsymbol{L}\boldsymbol{u} = \operatorname{vec}(\mathcal{A}(\boldsymbol{\mathcal{U}}))$, where[2]

$$\mathcal{A}(\boldsymbol{\mathcal{U}}) := \sum_{m=1}^{d} \boldsymbol{\mathcal{U}} \times_m \boldsymbol{A}_m. \tag{1.6}$$

Moreover, if the function $\mathcal{F} : S_{\text{TEN}} \times T \to \mathbb{R}^{n_1\times\cdots\times n_d}$ respresents the function f evaluated at the entries of $\boldsymbol{\mathcal{U}}$, then it holds that $\boldsymbol{f}(\boldsymbol{u}, t) = \texttt{vec}(\mathcal{F}(\boldsymbol{\mathcal{U}}, t))$, and (1.4) can be written in the form

$$\dot{\boldsymbol{\mathcal{U}}}(t) = \mathcal{A}(\boldsymbol{\mathcal{U}}(t)) + \mathcal{F}(\boldsymbol{\mathcal{U}}(t), t) + \boldsymbol{\mathcal{C}}, \qquad \boldsymbol{\mathcal{U}}(0) = \boldsymbol{\mathcal{U}}_0. \tag{1.7}$$

The boundary conditions are contained in the matrices $\boldsymbol{A}_m$; see, e.g., [51, 123] for the case where $d = 2$.

In the latter part of the thesis, we will also consider PDEs with nonlinear functions depending on the gradient ∇u. In this setting, the nonlinear function in the vectorized model (1.4) will also depend on the term $\boldsymbol{D}\boldsymbol{u}$, where $\boldsymbol{D} \in \mathbb{R}^{N\times N}$ accounts for the discretization of the gradient operator. Under the same assumptions considered above, it holds that

$$\boldsymbol{D} = \sum_{m=1}^{d} \boldsymbol{I}_{n_d} \otimes \cdots \otimes \overset{m}{\boldsymbol{D}}_m \otimes \cdots \otimes \boldsymbol{I}_{n_1} \in \mathbb{R}^{N\times N},$$

[1] We display the discretized Laplace operator, but more general operators can also be treated; see, e.g. [142, Section 3].

[2] For the case $d = 2$, (1.6) is a Sylvester operator of the form $\boldsymbol{A}_1\boldsymbol{U} + \boldsymbol{U}\boldsymbol{A}_2^\top$ [142].

where $\boldsymbol{D}_m \in \mathbb{R}^{n_m \times n_m}$ contains the approximation of the first derivative in the x_m direction and $\boldsymbol{Du} = \mathrm{vec}\,(\mathcal{D}(\boldsymbol{\mathcal{U}}))$, so that the nonlinear function in (1.7) will additionally depend on the tensor

$$\mathcal{D}(\boldsymbol{\mathcal{U}}) := \sum_{m=1}^{d} \boldsymbol{\mathcal{U}} \times_m \boldsymbol{D}_m.$$

If, instead, the underlying model is a coupled *system* of PDEs, the discretization procedure described above can be applied independently to each equation in the system; see Chapter 5.

This derivation has focused on parallelepipedal computational domains discretized using finite differences, which is the setting we will address for the remainder of this thesis. Achieving a matrix (tensor) formulation of the problem is, however, not restricted to this setting. In fact, several discretization techniques have been used to rewrite the discrete problem in array form, as long as a tensor-based decomposition is used. Apart from finite differences, this includes isogeometric analysis [9, 112, 137], a collection of spectral methods [63] and certain finite elements [65].

The array formulation of spatial methods for PDEs is also not restricted to simple parallelepipedal computational domains. Indeed, a wide range of computational strategies has been derived to map the physical domain to a simpler reference domain. For example, in the case of curved boundaries in two or three dimensions, bilinear projectors have been used for the mapping; see e.g., [61, 96]. Other PDE-based techniques such as elliptic grid generators have also been extensively employed, especially for fluid and air dynamics applications [40, 96, 154]. More recently, conformal mappings have been applied in [78] to address more general polygonal domains in two dimensions, and x-normal domains have been tackled using finite elements in [65]. In [112] more general domains defined through splines or NURBS are considered and the matrix formulation is achieved through a suitable low-rank factorization of the kernels.

More complex differential operators have also been treated in array form. For example, differential operators with separable coefficients and non-constant convection terms have been addressed in [78, 123] using finite differences on rectangular and more general polygonal domains, whereas in [127] elliptic anisotropic PDEs with stochastic terms were approximated in matrix form via a Galerkin method.

The extension to more complex domains, operators, and discretization bases, however, often comes with added difficulties related to time integration and solving the associated linear systems. One of these drawbacks is that that the resulting matrix $\boldsymbol{L}$ typically has a more complex structure than in (1.5). For example, the Kronecker sum may consist of more than d terms and some or all of the identity matrices may be different from the identity. These additional terms typically account for e.g., the discretization weights, the geometric contribution of the domain shape, or the convection terms; see e.g., [65, 78, 123]. A more complex structure in (1.5) leads to

generalized (multiterm) linear matrix or tensor operators when written in array form. Except for some special cases, the direct solution of these multiterm equations in array form without vectorization still remains unexplored; see e.g., [142] and the references therein.

1.3 Tools for model order reduction

This section aims to present three important model reduction tools required in the later stages of the thesis. More precisely, in Section 1.3.1 we present the most common Krylov subspaces, together with their related Arnoldi relations. Krylov subspaces and their properties will play a principal role in Chapter 3 for treating the quadratic DRE. In Sections 1.3.2 and 1.3.3 we respectively present the standard POD and DEIM methods. These methods form the backbone of the latter part of this thesis. In particular, the new algorithms derived in Chapters 4 and 5 are based upon and compared to the POD and DEIM methods presented here.

1.3.1 Krylov subspaces and Arnoldi relations

Krylov subspaces (in short generically denoted as K_k) that have been explored in the past years have the form

$$\begin{aligned} PK_k(\boldsymbol{A},\boldsymbol{Z}) &= \text{range}\left\{[\boldsymbol{Z},\boldsymbol{A}\boldsymbol{Z},\boldsymbol{A}^2\boldsymbol{Z},\ldots,\boldsymbol{A}^{k-1}\boldsymbol{Z}]\right\} \qquad \text{polynomial} \\ EK_k(\boldsymbol{A},\boldsymbol{Z}) &= \text{range}\left\{[\boldsymbol{Z},\boldsymbol{A}^{-1}\boldsymbol{Z},\ldots,\boldsymbol{A}^{k-1}\boldsymbol{Z},\boldsymbol{A}^{-k}\boldsymbol{Z}]\right\} \qquad \text{extended} \\ RK_k(\boldsymbol{A},\boldsymbol{Z},\boldsymbol{s}) &= \text{range}\left\{[\boldsymbol{Z},(\boldsymbol{A}-s_2I)^{-1}\boldsymbol{Z},\ldots,\prod_{i=2}^{k}(\boldsymbol{A}-s_iI)^{-1}\boldsymbol{Z}]\right\} \qquad \text{rational,} \end{aligned}$$

where $\boldsymbol{A}\in\mathbb{R}^{n\times n}$ and $\boldsymbol{Z}\in\mathbb{R}^{n\times n_z}$, with $n_z \ll n$, is a tall matrix associated with the given problem. In the rational subspace, $\boldsymbol{s}=\{s_2,\ldots,s_k\}$ is a set of properly chosen real or complex shifts, whose computation can be performed a priori or dynamically during the generation of the subspace; see [59, 142] for more complete descriptions.

A basis for the Krylov subspaces mentioned above can be obtained by the associated block Arnoldi algorithm [133, Section 6.12]. Suppose the orthonormal columns of $\mathcal{V}_k=[\mathsf{V}_1,\ldots,\mathsf{V}_k]\in\mathbb{R}^{n\times n_v}$ span the considered Krylov subspace after k iterations of the respective Arnoldi algorithm. Here $n_v=kqn_z$ with $q=1$ for the polynomial and rational spaces and $q=2$ for the extended space. In Algorithm 1.1 we recall how to obtain V_{k+1} from $\mathcal{V}_k$ by one step of the standard block Arnoldi iteration with Modified Gram-Schmidt (MGS) [133, Section 6.12].

Algorithm 1.1 One step of standard Block Arnoldi with MGS

1: **INPUT:** $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ and $\mathcal{V}_k = [\mathsf{V}_1, \ldots, \mathsf{V}_k] \in \mathbb{R}^{n\times kn_z}$
2: **OUTPUT:** Matrix block $\mathsf{V}_{k+1} \in \mathbb{R}^{n\times n_z}$
3: Determine $\overline{\mathsf{V}} = \boldsymbol{A}\mathsf{V}_k$;
4: **for** $j = 1, \ldots, k$ **do**
5: Update $\overline{\mathsf{V}} = \overline{\mathsf{V}} - \mathsf{V}_j \left(\mathsf{V}_j^\top \overline{\mathsf{V}}\right)$;
6: **end for**
7: Orthogonalize $\overline{\mathsf{V}}$ by a skinny QR factorization to determine V_{k+1};

Once $\mathcal{V}_k$ has been constructed with the appropriate Arnoldi algorithm, the following Arnoldi-type relation holds,

$$\boldsymbol{A}\mathcal{V}_k = \mathcal{V}_k \widehat{\boldsymbol{A}}_k + \boldsymbol{\Upsilon}_{k+1}\boldsymbol{\Gamma}_k^\top, \quad \widehat{\boldsymbol{A}}_k \in \mathbb{R}^{n_v\times n_v}, \quad n_v \ll n, \tag{1.8}$$

where the actual values of the matrix $\boldsymbol{\Gamma}_k^\top \in \mathbb{R}^{qn_z\times n_v}$ and the orthonormal columns of $\boldsymbol{\Upsilon}_{k+1} \in \mathbb{R}^{n\times qn_z}$ depend on the chosen subspace. Moreover, setting $\mathcal{V}_{k+1} = [\mathcal{V}_k, \boldsymbol{\Upsilon}_{k+1}]$ we have that $K_{k+1} = \text{range}(\mathcal{V}_{k+1})$, which shows that Krylov subspaces are nested, that is $K_k \subseteq K_{k+1}$, resulting in a dimension increase after each iteration. In what follows, we recall the matrix relations leading to (1.8) for the extended and rational Krylov subspaces.

Extended Krylov subspace. The extended Krylov subspace $EK_k(\boldsymbol{A}, \mathbf{Z})$ takes the form presented above. The matrix $\mathcal{V}_k \in \mathbb{R}^{n\times 2kn_z}$ whose orthonormal columns spans the subspace is formed using the extended Arnoldi algorithm [58]. Let

$$\widehat{\boldsymbol{A}}_{k+1} = \mathcal{V}_{k+1}^\top \boldsymbol{A}\mathcal{V}_k = \begin{bmatrix} \widehat{\boldsymbol{A}}_k \\ \widehat{\mathsf{A}}_{k+1,k}\mathsf{E}_{2k}^\top \end{bmatrix} \in \mathbb{R}^{2(k+1)n_z\times 2kn_z}, \tag{1.9}$$

where $\mathcal{V}_{k+1} = [\mathcal{V}_k\ \mathsf{V}_{k+1}] \in \mathbb{R}^{n\times 2(k+1)n_z}$, $\widehat{\mathsf{A}}_{k+1,k} \in \mathbb{R}^{2n_z\times 2n_z}$ is the (k+1,k) block of $\widehat{\boldsymbol{A}}_{k+1}$ and E_{2k} is the last $2n_z$ columns of $\boldsymbol{I}_{2kn_z}$. The extended Arnoldi algorithm produces the Arnoldi-type relation

$$\boldsymbol{A}\mathcal{V}_k = \mathcal{V}_{k+1}\widehat{\boldsymbol{A}}_{k+1} = \mathcal{V}_k\widehat{\boldsymbol{A}}_k + \mathsf{V}_{k+1}\widehat{\mathsf{A}}_{k+1,k}\mathsf{E}_{2k}^\top. \tag{1.10}$$

so that $\boldsymbol{\Upsilon}_{k+1} = \mathsf{V}_{k+1}$ and $\boldsymbol{\Gamma}_k^\top = \widehat{\mathsf{A}}_{k+1,k}\mathsf{E}_{2k}^\top$.

Rational Krylov subspace. Given $\boldsymbol{s} = \{s_2, s_3, \ldots\}$, with $s_j \in \mathbb{C}^+$ closed under conjugation, the rational Krylov subspace is given by $RK_k(\boldsymbol{A}, \boldsymbol{Z}, \boldsymbol{s})$ as defined above.

The resulting basis of the rational Krylov subspace after k iterations of the rational Arnoldi algorithm [130] is given by $\mathcal{V}_k = [\mathsf{V}_1, \ldots, \mathsf{V}_k] \in \mathbb{R}^{n\times kn_z}$. We assume here that the matrix $\mathcal{V}_k$ consists of only real numbers; this is further clarified in the latter part of this section. We also define the matrices $\mathcal{V}_{k+1} = [\mathcal{V}_k, \mathsf{V}_{k+1}] \in \mathbb{R}^{n\times(k+1)n_z}$ and the

matrix

$$\boldsymbol{H}_{k+1} = \begin{bmatrix} \boldsymbol{H}_k \\ \mathsf{H}_{k+1,k}\mathsf{E}_k^\top \end{bmatrix} \in \mathbb{R}^{(k+1)n_z \times kn_z}, \tag{1.11}$$

where $\mathsf{H}_{k+1,k} \in \mathbb{R}^{n_z \times n_z}$ and E_k holds the last n_z columns of $\boldsymbol{I}_{kn_z}$. The matrix $\boldsymbol{H}_{k+1}$ contains the orthogonalization coefficients obtained during the rational Arnoldi algorithm.

Let $\widehat{\boldsymbol{A}}_k = \mathcal{V}_k^\top \boldsymbol{A}\mathcal{V}_k \in \mathbb{R}^{kn_z \times kn_z}$. The rational Krylov basis satisfies the Arnoldi-type relation

$$\boldsymbol{A}\mathcal{V}_k = \mathcal{V}_k\widehat{\boldsymbol{A}}_k + \check{\boldsymbol{V}}_{k+1}\check{\boldsymbol{R}}\mathsf{H}_{k+1,k}\mathsf{E}_k^\top \boldsymbol{H}_k^{-1}, \tag{1.12}$$

so that $\boldsymbol{\Upsilon}_{k+1} = \check{\boldsymbol{V}}_{k+1}$ and $\boldsymbol{\Gamma}_k^\top = \check{\boldsymbol{R}}\mathsf{H}_{k+1,k}\mathsf{E}_k^\top \boldsymbol{H}_k^{-1}$ where $\check{\boldsymbol{V}}_{k+1}$ has orthonormal columns such that

$$\check{\boldsymbol{V}}_{k+1}\check{\boldsymbol{R}} = \mathsf{V}_{k+1}s_k - (\boldsymbol{I}_n - \mathcal{V}_k\mathcal{V}_k^\top)\boldsymbol{A}\mathsf{V}_{k+1} \tag{1.13}$$

is the skinny QR decomposition of the matrix on the right (see [59, 108]). The rational Krylov procedure requires as an extra input the (usually real) values $s_0^{(1)}, s_0^{(2)}$, which form a rough approximation of the spectral region used to compute the next shift. The reader is referred to [59, 141] for implementation details.

Construction of an all-real basis. The rational algorithm presented in [59] forms a complex basis, when the shifts are not all real. In short, when $s_j \in \mathbb{C}^+$, the original approach would be to use the shift s_j to form the next block V_j and to then let the following shift be given by $s_{j+1} = \overline{s}_j$, where $\overline{s}_j$ denotes the complex conjugate of s_j. This results in both V_j and V_{j+1} being complex. For real data (see, e.g., Chapter 3), complex arithmetic can slow down the evaluation of the reduced-order model. If so, an all-real basis can be constructed following the method introduced in [131], which works as follows. If the shift s_j is complex then the block $\mathsf{W}_j = (\boldsymbol{A} - s_j\boldsymbol{I})^{-1}\mathsf{V}_{j-1}$ is also complex, hence we split it into its real and complex parts, that is $\mathsf{W}_j = \mathsf{W}_j^{(r)} + \mathsf{W}_j^{(c)}\imath$. The block V_j is then formed by orthogonalizing $\mathsf{W}_j^{(r)}$ with respect to all vectors in the already computed basis, after which V_{j+1} is formed by orthogonalizing $\mathsf{W}_j^{(c)}$ with respect to all previous vectors in in the computed basis, and in V_j. This determines the same space, since span$\{\mathsf{W}_j, \bar{\mathsf{W}}_j\}$ =span$\{\mathsf{V}_j, \mathsf{V}_{j+1}\}$.

1.3.2 The Proper Orthogonal Decomposition (POD)

The POD is a well-known technique for reducing the dimensionality of a given dynamical system by projection onto a space determined in a least-squares optimal sense, based on information from the full order model. More precisely, consider a set of *snapshots* $\boldsymbol{\xi}_j = \boldsymbol{\xi}(t_j)$ at n_s different time instances $(0 \le t_1 < \cdots < t_{n_s} \le t_f)$. Let

$$\boldsymbol{S} = [\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_{n_s}] \in \mathbb{R}^{N \times n_s}, \tag{1.14}$$

and $S = \text{range}(\boldsymbol{S})$ of dimension d_s. A POD basis $\{\boldsymbol{v}_1 \ldots, \boldsymbol{v}_k\} \subset \mathbb{R}^N$ of dimension $k < d_s$ is a set of orthonormal vectors whose linear span gives the best approximation of the space S according to the criterion

$$\min_{\text{rank}\{\boldsymbol{V}_{\text{VEC}}\}=k} \sum_{j=1}^{n_s} \|\boldsymbol{\xi}_j - \boldsymbol{V}_{\text{VEC}} \boldsymbol{V}_{\text{VEC}}^\top \boldsymbol{\xi}_j\|^2 \quad \text{s.t.} \quad \boldsymbol{V}_{\text{VEC}}^\top \boldsymbol{V}_{\text{VEC}} = \boldsymbol{I}_k,$$

where $\boldsymbol{V}_{\text{VEC}} = [\boldsymbol{v}_1 \ldots, \boldsymbol{v}_k] \in \mathbb{R}^{N \times k}$. This basis can be obtained through the singular value decomposition (SVD) of the matrix $\boldsymbol{S}$, which we write as $\boldsymbol{S} = \boldsymbol{V}\boldsymbol{\Sigma}\boldsymbol{W}^\top$, with $\boldsymbol{V}$ and $\boldsymbol{W}$ orthogonal matrices and $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_{n_s})$ diagonal with non-increasing positive diagonal elements. If the diagonal elements of $\boldsymbol{\Sigma}$ have a rapid decay, the first k columns of $\boldsymbol{V}$ (left singular vectors) are the most dominant in the approximation of $\boldsymbol{S}$. Denoting with $\boldsymbol{S}_k = \mathsf{V}_k \Sigma_k \mathsf{W}_k^\top$ the reduced SVD where only the $k \times k$ top left portion of $\boldsymbol{\Sigma}$ is retained and $\boldsymbol{V}, \boldsymbol{W}$ are truncated accordingly, then $\|\boldsymbol{S} - \boldsymbol{S}_k\| = \sigma_{k+1}$ [70]. The resulting POD basis matrix is given by $\boldsymbol{V}_{\text{VEC}} = \mathsf{V}_k$.

For model reduction of semilinear dynamical systems, the POD is often used in conjunction with DEIM. This tool for efficiently interpolating a nonlinear function is presented in the following section.

1.3.3 The Discrete Empirical Interpolation Method (DEIM)

The DEIM procedure was originally introduced in [41] as a discrete variant of the Empirical Interpolation Method (EIM) from [17]. It is utilized to approximate a nonlinear vector function $\boldsymbol{f} : T \to \mathbb{R}^N$ by interpolating it onto an empirical basis, that is,

$$\boldsymbol{f}(t) \approx \widetilde{\boldsymbol{f}}(t) = \boldsymbol{\Phi}_{\text{VEC}} \widehat{\mathbf{f}}(t), \tag{1.15}$$

where $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_p\} \subset \mathbb{R}^N$ is a low dimensional basis, $\boldsymbol{\Phi}_{\text{VEC}} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_p] \in \mathbb{R}^{N \times p}$ and $\widehat{\mathbf{f}}(t) \in \mathbb{R}^p$, with $p \ll n$ is the vector of time-dependent coefficients to be determined. The optimal choice of $\widehat{\mathbf{f}}(t)$ can be obtained by solving the overdetermined linear system (1.15) such that

$$\widehat{\mathbf{f}}(t) = \left(\boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}}\right)^{-1} \boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{f}(t) \quad \text{and} \quad \boldsymbol{f}(t) \approx \boldsymbol{\Phi}_{\text{VEC}} \left(\boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}}\right)^{-1} \boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{f}(t).$$

If the matrix $\boldsymbol{\Phi}_{\text{VEC}}$ has orthonormal columns, we will obtain a projection error of the form

$$\|\boldsymbol{f}(t) - \widetilde{\boldsymbol{f}}(t)\|_2 = \|\boldsymbol{f}(t) - \boldsymbol{\Phi}_{\text{VEC}} \boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{f}(t)\|_2,$$

hence just depending on the quality of the approximation space range($\boldsymbol{\Phi}_{\text{VEC}}$). Nevertheless, this type of approximation would require evaluating $\boldsymbol{f}(t)$ at all N entries for each time t before projection onto range($\boldsymbol{\Phi}_{\text{VEC}}$). The complexity of evaluating N entries at each t results in a bottleneck when evaluating a nonlinear term within

a reduced model. Instead, the DEIM aims to determine $\widehat{\mathbf{f}}(t)$ by interpolating the nonlinear function at $p \ll n$ entries.

Let $\boldsymbol{P}_{\text{VEC}} = [\boldsymbol{e}_{\rho_1}, \ldots, \boldsymbol{e}_{\rho_p}] \in \mathbb{R}^{N\times p}$ be a subset of columns of the identity matrix, named the "selection matrix". The role of $\boldsymbol{P}_{\text{VEC}}$ is to select appropriate rows of the function $\boldsymbol{f}(t)$ at which it will be interpolated. Therefore, if $\boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}}$ is invertible, in [41] the coefficient vector $\widehat{\mathbf{f}}(t)$ is uniquely determined by solving the linear system $\boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}} \widehat{\mathbf{f}}(t) = \boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{f}(t)$, so that

$$\widetilde{\boldsymbol{f}}(t) = \boldsymbol{\Phi}_{\text{VEC}} \widehat{\mathbf{f}}(t) = \boldsymbol{\Phi}_{\text{VEC}} (\boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}})^{-1} \boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{f}(t). \tag{1.16}$$

Since the matrix $\boldsymbol{P}_{\text{VEC}}$ is merely responsible for selecting rows, this type of approximation allows us to evaluate $\boldsymbol{f}$ at only p entries for each t. The row selection is relatively straightforward if $\boldsymbol{f}$ is evaluated elementwise. In the more general setting where $\boldsymbol{f}$ is not necessarily evaluated elementwise, this is still achievable; however, more complex procedures are required; see the discussion in Section 4.2.2 and [41, Section 3.5]. The quality of the approximation (1.16) has been investigated in [41, Lemma 3.2], where it is shown that a projection error of the form

$$\left\| \boldsymbol{f}(t) - \widetilde{\boldsymbol{f}}(t) \right\|_2 = \left\| \left(\boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}} \right)^{-1} \right\|_2 \left\| \boldsymbol{f}(t) - \boldsymbol{\Phi}_{\text{VEC}} \boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{f}(t) \right\|_2 \tag{1.17}$$

is obtained, given that $\boldsymbol{\Phi}_{\text{VEC}}$ has orthonormal columns. The approximation's quality therefore depends on two factors, namely the quality of the approximation space and the quality of the interpolation indices, respectively represented by the terms $\left\| \boldsymbol{f}(t) - \boldsymbol{\Phi}_{\text{VEC}} \boldsymbol{\Phi}_{\text{VEC}}^\top \boldsymbol{f}(t) \right\|_2$ and $\left\| \left(\boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}} \right)^{-1} \right\|_2$.

In most applications the interpolation basis $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_p\}$ is selected as the POD basis of the set of snapshots $\{\boldsymbol{f}(t_1), \ldots, \boldsymbol{f}(t_{n_s})\}$, as described earlier in Section 1.3.2, that is given the matrix

$$\boldsymbol{N} = [\boldsymbol{f}(t_1), \ldots, \boldsymbol{f}(t_{n_s})] \in \mathbb{R}^{N\times n_s}, \tag{1.18}$$

the columns of the matrix $\boldsymbol{\Phi}_{\text{VEC}} = [\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_p]$ are determined as the first $p \leq n_s$ dominant left singular vectors in the SVD of $\boldsymbol{N}$. The matrix $\boldsymbol{P}_{\text{VEC}}$ for DEIM is selected by a greedy algorithm based on the system residual in [41]; see Algorithm 1.2.

Algorithm 1.2 DEIM [41, Algorithm 3.1]

1: **INPUT:** $\{\boldsymbol{\phi}_i\}_{i=1}^p \subset \mathbb{R}^N$
2: **OUTPUT:** Selection matrix $\boldsymbol{P}_{\text{VEC}}$
3: Determine $[\sim, \rho_1] = \texttt{max}(|\boldsymbol{\phi}_1|)$;
4: $\boldsymbol{\Phi}_{\text{VEC}} = [\boldsymbol{\phi}_1], \quad \boldsymbol{P}_{\text{VEC}} = [\boldsymbol{e}_{\rho_1}]$;
5: **for** $i = 2, \ldots, p$ **do**
6: Compute $\boldsymbol{r} = \boldsymbol{\phi}_i - \boldsymbol{\Phi}_{\text{VEC}} (\boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\Phi}_{\text{VEC}})^{-1} \boldsymbol{P}_{\text{VEC}}^\top \boldsymbol{\phi}_i$;
7: Determine $[\sim, \rho_i] = \texttt{max}(|\boldsymbol{r}|)$;
8: $\boldsymbol{\Phi}_{\text{VEC}} \leftarrow [\boldsymbol{\Phi}_{\text{VEC}}, \boldsymbol{\phi}_i], \quad \boldsymbol{P}_{\text{VEC}} \leftarrow [\boldsymbol{P}_{\text{VEC}}, \boldsymbol{e}_{\rho_i}]$;
9: **end for**

In [57] the authors showed that a pivoted QR-factorization of $\boldsymbol{\Phi}_{\text{VEC}}^{\top}$ may lead to better accuracy and stability properties of the computed matrix $\boldsymbol{P}_{\text{VEC}}$ and ultimately lower values of $\left\| \left(\boldsymbol{P}_{\text{VEC}}^{\top} \boldsymbol{\Phi}_{\text{VEC}} \right)^{-1} \right\|_2$. The resulting approach, called `q-deim`, will be used in the sequel and is implemented as presented in Algorithm 1.3.

Algorithm 1.3 `q-deim` [57]

1: **INPUT:** $\boldsymbol{\Phi}_{\text{VEC}} \in \mathbb{R}^{N \times p}, \quad p \leq N$
2: **OUTPUT:** Selection matrix $\boldsymbol{P}_{\text{VEC}}$
3: Perform pivoted QR of $\boldsymbol{\Phi}_{\text{VEC}}^{\top}$ so that $\boldsymbol{\Phi}_{\text{VEC}}^{\top} \boldsymbol{\Pi}_{\text{VEC}} = \boldsymbol{Q}_{\text{VEC}} \boldsymbol{R}_{\text{VEC}}$
4: $\boldsymbol{P}_{\text{VEC}} = \boldsymbol{\Pi}_{\text{VEC}}(:, 1:p)$

Chapter 2

Matrix Methods for Semilinear ODEs

The efficient numerical integration of semilinear ODEs has been widely researched, thanks to the fundamental role that these equations play in studying the time evolution of complex systems. The aim of this chapter is to report on matrix-oriented schemes that can be used to approximate the solution $\boldsymbol{U}(t) \in S_{\text{MAT}}$ to the following semilinear matrix differential equation

$$\dot{\boldsymbol{U}}(t) = \boldsymbol{A}_1\boldsymbol{U}(t) + \boldsymbol{U}(t)\boldsymbol{A}_2 + \mathcal{F}(\boldsymbol{U}, t) + \boldsymbol{C}, \quad \boldsymbol{U}(0) = \boldsymbol{U}_0, \tag{2.1}$$

where $\boldsymbol{A}_1 \in \mathbb{R}^{n_1 \times n_1}, \boldsymbol{A}_2 \in \mathbb{R}^{n_2 \times n_2}, \boldsymbol{C} \in \mathbb{R}^{n_1 \times n_2}$ and $t \in [0, t_f] = T \subset \mathbb{R}$. The function $\mathcal{F} : S_{\text{MAT}} \times T \to \mathbb{R}^{n_1 \times n_2}$ is nonlinear, and S_{MAT} is an appropriate functional space containing the sought after solution.

To integrate (2.1) several alternatives can be considered. The standard approach would be to vectorize the equation and integrate the resulting system of $n_1 n_2$ semilinear ODEs by any suitable ODE solver; see, e.g., [35]. However, this could result in high computational costs and memory requirements, especially when the considered coefficient matrices are dense since the well-established iterative methods for sparse linear systems are ineffective. Instead, in the recent literature, standard ODE solvers have been applied directly to the ODE in matrix form; see, e.g., [51, 52, 55, 114, 115, 149]. To this end, we present a collection of these matrix methods for solving equations of the form (2.1). Semilinear equations are characterized by a stiff linear part and a nonstiff nonlinear part [86, 152]. As a result, explicit time integration schemes are deemed unsuitable for these problems due to unrealistically small timestep requirements to ensure stability, as a result of the Courant-Friedrichs-Lewy (CFL) condition [152]. On the other hand, fully implicit schemes require the solution of a nonlinear equation at each timestep, which generally requires the application of an expensive iterative method.

To this end, Semi-Implicit (SI) schemes have been identified as a good compromise to treat semilinear differential equations of the form (2.1) efficiently [86, chapter IV.3]. In

particular, they are identified as splitting methods for ODE systems, where the linear term is treated implicitly to avoid excessively small timesteps, whereas the nonlinear term is treated explicitly to avoid the application of a nonlinear solver. As a result, only a single *linear* system needs to be solved at each timestep.

In Section 2.1 we present the matrix versions of two classes of the most widely used SI methods. We also discuss the convergence and stability properties of the schemes and efficient algorithmic implementations, and we compare them numerically.

In the second part of this chapter, that is Section 2.2, we consider the quadratic DRE separately. This is because the structure of the DRE allows several computational advantages when it comes to integration [52, 55, 114, 115, 149]. More precisely, we show how matrix versions of fully implicit schemes and splitting methods can be efficiently applied to approximate the time evolution of the DRE. We also discuss some convergence and stability results of the considered methods and illustrate the expected accuracy of the implicit schemes through a numerical experiment.

In the more general setting (i.e., non-quadratic $\mathcal{F}$), fully implicit methods applied to (2.1) will require the solution of a nonlinear algebraic matrix equation at each timestep. To the best of our knowledge, it is unclear how to solve such problems without reverting to vectorization and applying standard nonlinear solvers, e.g., Newton. This is, however, not within the scope of this thesis and will not be further discussed.

2.1 Semi-Implicit (SI) methods for semilinear ODEs

In this section, we present the matrix-oriented versions of some of the most widely used SI schemes, that is, Implicit–Explicit (IMEX) methods [14, 132] and exponential integrators [83, 84]. For the IMEX methods, we consider the single-step IMEX Euler scheme and the multistep schemes separately, as is generally done in the literature, whereas we present only the first-order exponential Euler for the exponential time differencing (ETD) schemes.

2.1.1 Matrix methods, convergence and stability

For all methods, we will first present the scheme in its *vectorized* form (with $N = n_1 n_2$), applied to

$$\dot{\boldsymbol{u}}(t) = \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{f}(\boldsymbol{u}, t) + \boldsymbol{c}, \quad \boldsymbol{u} \in \mathbb{R}^N, \quad \boldsymbol{u}(0) = \boldsymbol{u}_0, \tag{2.2}$$

with $\boldsymbol{L} \in \mathbb{R}^{N \times N}$ defined as in (1.5) for $d = 2$, and stepsize $h = t_f/n_t$, where n_t is the number of timesteps. We will show how the methods can be recast into matrix form, after which we will discuss the convergence and stability properties of the methods.

The stability properties of the matrix and vector forms of the schemes, in this setting, will be equivalent [51]. For the sake of exposition, we let $\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)}, t_j) = \mathcal{F}(\boldsymbol{U}^{(j)}, t_j) + \boldsymbol{C}$ and $\boldsymbol{f}_{\boldsymbol{c}}(\boldsymbol{u}^{(j)}, t_j) = \boldsymbol{f}(\boldsymbol{u}^{(j)}, t_j) + \boldsymbol{c}$ for the remainder of the chapter.

Exponential Time Differencing (ETD)

ETD schemes were originally introduced in the setting of electrodynamics [153], but they have been rederived in several contexts [29, 47, 64, 83, 84, 117]. In [47, 84] it is stated that exponential integrators are more potent than IMEX schemes. This is because the stiff, linear part of the equation is treated exactly, resulting in better stability properties and greater accuracy in the transient solution. Furthermore, the methods are also preferred to others, such as Integrating Factor (see, e.g., [102]) since they treat non-transient solutions better. However, from a computational point of view, this depends on the efficient evaluation of matrix exponentials.

Here we consider the first order exponential Euler scheme, where the approximation $\boldsymbol{u}^{(j+1)} \approx \boldsymbol{u}(t_{j+1})$ to (2.2) is given by the relation

$$\boldsymbol{u}^{(j+1)} = e^{h\boldsymbol{L}}\boldsymbol{u}^{(j)} + h\varphi_1(h\boldsymbol{L})\boldsymbol{f}_{\boldsymbol{c}}(\boldsymbol{u}^{(j)}, t_j), \quad j = 0, 1, \ldots, n_t - 1, \tag{2.3}$$

where $\varphi_1(z) = (e^z - 1)/z$ is the first *phi* function [84] and $e^{h\boldsymbol{L}}$ is the matrix exponential. The relation (2.3) above is more naturally expressed via a two step procedure as

1. Solve $\boldsymbol{L}\boldsymbol{\varphi}^{(j)} = e^{h\boldsymbol{L}}\boldsymbol{f}_{\boldsymbol{c}}(\boldsymbol{u}^{(j)}, t_j) - \boldsymbol{f}_{\boldsymbol{c}}(\boldsymbol{u}^{(j)}, t_j)$;
2. Evaluate $\boldsymbol{u}^{(j+1)} = e^{h\boldsymbol{L}}\boldsymbol{u}^{(j)} + \boldsymbol{\varphi}^{(j)}$.

In our setting, $\boldsymbol{L} = \boldsymbol{A}_2^\top \otimes \boldsymbol{I}_{n_1} + \boldsymbol{I}_{n_2} \otimes \boldsymbol{A}_1$, for which it holds that $e^{h\boldsymbol{L}} = e^{h\boldsymbol{A}_2^\top} \otimes e^{h\boldsymbol{A}_1}$ [81, Th.10.9], so that the computation of the exponential of the large matrix $\boldsymbol{L}$ reduces to

$$e^{h\boldsymbol{L}}\boldsymbol{u} = e^{h\boldsymbol{L}}\mathrm{vec}(\boldsymbol{U}) = \mathrm{vec}(e^{h\boldsymbol{A}_1}\boldsymbol{U}e^{h\boldsymbol{A}_2}). \tag{2.4}$$

The two matrices used in the exponential functions now have dimensions n_1 and n_2 respectively instead of $n_1 n_2$, so that the computation of the matrix exponential is fully affordable for moderate dimensions n_1 and n_2. As a consequence, the integration step can be performed all at the matrix level, without resorting to the vectorized form. More precisely (see also [51]), we can compute $\boldsymbol{U}^{(j+1)}$ as

$$\boldsymbol{U}^{(j+1)} = e^{h\boldsymbol{A}_1}\boldsymbol{U}^{(j)}e^{h\boldsymbol{A}_2} + \boldsymbol{\Phi}^{(j)}, \quad j = 0, 1, \ldots, n_t - 1, \tag{2.5}$$

which corresponds to step (2) above, where the matrix $\boldsymbol{\Phi}^{(j)}$ solves the following linear (Sylvester) matrix equation

$$\boldsymbol{A}_1\boldsymbol{\Phi} + \boldsymbol{\Phi}\boldsymbol{A}_2 = e^{h\boldsymbol{A}_1}\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)}, t_j)e^{h\boldsymbol{A}_2} - \mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)}, t_j), \tag{2.6}$$

corresponding to the linear system solve in step (1).

Convergence and stability properties.

The convergence of the exponential Euler scheme has been discussed in [83, 84]. In our setting, that is, stiff ODEs with smooth solutions, the convergence properties of the exponential Euler method are studied in the classic sense by inserting the exact solution into the scheme and Taylor expanding to obtain the defects. To obtain these bounds, the matrix $\boldsymbol{L}$ has to fulfill certain conditions, see [84, Assumption 2.2, Assumption 2.9]; however, the problems considered in this thesis all fit this framework, since they stem from the space discretization of reaction-diffusion PDEs [84, Example 2.11]. Therefore, with these assumptions satisfied, as well as the crucial assumption of Lipschitz continuity of the nonlinear function, the exponential Euler scheme is shown to have the expected first-order convergence; see [84, Theorem 2.14].

The stability properties of the exponential Euler scheme applied to a scalar test problem have been studied in [51] and compared to other SI schemes. To this end, consider the scalar test problem (with $u_i(t) = (\boldsymbol{u}(t))_i$)

$$\dot{u}_i(t) = \lambda_i u_i(t) + \eta_i u_i(t), \tag{2.7}$$

where λ_i is the ith eigenvalue of $\boldsymbol{L}$ and η_i is the ith component modelling the nonlinear function. If we define $\tilde{\lambda}_i = h\lambda_i$ and $\tilde{\eta}_i = h\eta_i$, then the exponential Euler scheme applied to the test equation yields

$$u_i^{(j+1)} = \left(e^{\tilde{\lambda}_i} + \frac{e^{\tilde{\lambda}_i} - 1}{\tilde{\lambda}_i} \tilde{\eta}_i \right) u_i^{(j)},$$

which yields the stability region

$$\mathcal{D}_{\text{EXP}} = \left\{ (\tilde{\lambda}_i, \tilde{\eta}_i) \in \mathbb{R}^- \times \mathbb{R} : \left| e^{\tilde{\lambda}_i} + \frac{e^{\tilde{\lambda}_i} - 1}{\tilde{\lambda}_i} \tilde{\eta}_i \right| \leq 1 \right\} \quad \Longleftrightarrow \quad \tilde{\lambda}_i \frac{1 + e^{\tilde{\lambda}_i}}{1 - e^{\tilde{\lambda}_i}} \leq \tilde{\eta}_i \leq -\tilde{\lambda}_i.$$

See [51] for a plot of this region and discussions regarding timestep restrictions.

IMEX Methods

IMEX methods consist of using *implicit* methods to advance the *linear* part of the equation and *explicit* methods to advance the *nonlinear* part of the equation. The first works on IMEX methods and their stability dates back to the 1980s, see, e.g., [48, 160]. As a result of the bottleneck related to evaluating the matrix exponential in the ETD schemes, the IMEX schemes were initially introduced as rational approximations to the exponential integrators, yielding an approximation to the linear part of the equation. Consequently, these schemes can potentially be more computationally efficient than their exponential counterparts; however, to the expense of stability and accuracy in the transient solution, see Example 2.1.

IMEX Euler. Consider the first order IMEX Euler scheme [14, 132]. Suppose $\boldsymbol{u}^{(j+1)} \approx \boldsymbol{u}(t_{j+1})$, the IMEX Euler time discretization is given by

$$\boldsymbol{u}^{(j+1)} - \boldsymbol{u}^{(j)} = h\left(\boldsymbol{L}\boldsymbol{u}^{(j+1)} + \boldsymbol{f}_c(\boldsymbol{u}^{(j)}, t_j)\right),$$

so that

$$(\boldsymbol{I}_N - h\boldsymbol{L})\,\boldsymbol{u}^{(j+1)} = \boldsymbol{u}^{(j)} + h\boldsymbol{f}_c(\boldsymbol{u}^{(j)}, t_j), \quad j = 0, 1, \ldots, n_t - 1.$$

where $\boldsymbol{u}^{(0)} = \boldsymbol{u}_0$.

In a similar way we can rewrite the relation in matrix form, using the fact that $\boldsymbol{L} = \boldsymbol{A}_2^\top \otimes \boldsymbol{I}_{n_1} + \boldsymbol{I}_{n_2} \otimes \boldsymbol{A}_1$ together with (1.1). More precisely, the IMEX Euler time discretization to determine $\boldsymbol{U}^{(j+1)}$ can be written as

$$\boldsymbol{U}^{(j+1)} - \boldsymbol{U}^{(j)} = h\left(\boldsymbol{A}_1\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}\boldsymbol{A}_2 + \mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)}, t_j)\right),$$

so that

$$(\boldsymbol{I}_{n_1} - h\boldsymbol{A}_1)\,\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}\,(-h\boldsymbol{A}_2) = \boldsymbol{U}^{(j)} + h\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)}, t_j), \tag{2.8}$$

for $j = 0, 1, \ldots, n_t - 1$, where $\boldsymbol{U}^{(0)} = \boldsymbol{U}_0$.

IMEX Multistep methods. Following [86, Chapter IV.4.2], IMEX multistep SBDF schemes can be derived from fully implicit multistep schemes via an extrapolation formula. More precisely, consider the fully implicit s-step backward difference formulae (BDF) scheme applied to (2.2), given by

$$\sum_{i=0}^{s} \alpha_i \boldsymbol{u}^{(j+1-i)} = h\beta\left(\boldsymbol{L}\boldsymbol{u}^{(j+1)} + \boldsymbol{f}_c\left(\boldsymbol{u}^{(j+1)}, t_{j+1}\right)\right), \quad j = s-1, \ldots, n_t - 1. \tag{2.9}$$

The nonlinear term can then be handled explicitly via the extrapolation formula

$$\boldsymbol{f}_c\left(\boldsymbol{u}^{(j+1)}, t_{j+1}\right) = \sum_{i=1}^{s} \gamma_i \boldsymbol{f}_c\left(\boldsymbol{u}^{(j+1-i)}, t_{j+1-i}\right) + \mathcal{O}(h^q),$$

which yields

$$\sum_{i=0}^{s} \alpha_i \boldsymbol{u}^{(j+1-i)} = h\beta\boldsymbol{L}\boldsymbol{u}^{(j+1)} + h\sum_{i=1}^{s} \beta_i^* \boldsymbol{f}_c\left(\boldsymbol{u}^{(j+1-i)}, t_{j+1-i}\right), \tag{2.10}$$

for $j = s-1, \ldots, n_t - 1$, where $\beta_i^* = \beta(1 + \gamma_i)$. For each j, the unknown term $\boldsymbol{u}^{(j+1)}$ is determined by solving the linear system

$$(\alpha_0\boldsymbol{I}_N - h\beta\boldsymbol{L})\,\boldsymbol{u}^{(j+1)} = -\sum_{i=1}^{s} \alpha_i \boldsymbol{u}^{(j+1-i)} + h\sum_{i=1}^{s} \beta_i^* \boldsymbol{f}_c\left(\boldsymbol{u}^{(j+1-i)}, t_{j+1-i}\right),$$

where $\boldsymbol{u}^{(0)} = \boldsymbol{u}_0$ and the terms $\boldsymbol{u}^{(1)}, \dots, \boldsymbol{u}^{(j)}$ can be determined via a lower-order SBDF scheme.

To derive the matrix formulation of the IMEX SBDF schemes, we once again utilize the Kronecker sum structure of $\boldsymbol{L}$ and (1.1). More precisely, the s-step SBDF time discretization can be written in matrix form as

$$\sum_{i=0}^{s} \alpha_i \boldsymbol{U}^{(j+1-i)} = h\beta \boldsymbol{A}_1 \boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}(h\beta \boldsymbol{A}_2) + h \sum_{i=0}^{s} \beta_i^* \mathcal{F}_{\boldsymbol{C}} \left(\boldsymbol{U}^{(j+1-i)}, t_{j+1-i} \right),$$

so that $\boldsymbol{U}^{(j+1)}$ is obtained by solving the Sylvester equation

$$(\alpha_0 \boldsymbol{I}_{n_1} - h\beta \boldsymbol{A}_1)\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}(-h\beta \boldsymbol{A}_2) = -\sum_{i=1}^{s} \alpha_i \boldsymbol{U}^{(j+1-i)} + h \sum_{i=1}^{s} \beta_i^* \mathcal{F}_{\boldsymbol{C}} \left(\boldsymbol{U}^{(j+1-i)}, t_{j+1-i} \right), \tag{2.11}$$

for $j = s-1, \dots, n_t - 1$. See, e.g., [14] for the derivation and precise values of the constants β, α_i and β_i^* for different s.

Convergence and stability properties.

The convergence and stability properties of the IMEX schemes have been discussed thoroughly in [86, Chapter IV.4.2], as well as [14, 48]. In [86, Chapter IV.4.2], it is shown that the IMEX Euler scheme yields a local truncation error of the form

$$\boldsymbol{e}_{\text{EULER}}(t_{j+1}) = -\frac{h}{2}\ddot{\boldsymbol{u}}(t_j) + h\frac{\partial}{\partial t}\boldsymbol{f}(\boldsymbol{u}(t_j), t_j) + \mathcal{O}(h^2).$$

As a consequence, if the solution $\boldsymbol{u}$ and the nonlinear function $\boldsymbol{f}$ are sufficiently smooth, that is $\boldsymbol{u} \in \mathcal{C}^2$ and $\boldsymbol{f} \in \mathcal{C}^1$ in the second variable, the scheme will admit the desired first-order accuracy.

Furthermore, once again following [86, Chapter IV.4.2], the higher-order SBDF schemes admit a truncation error of the form

$$\boldsymbol{e}_{\text{SBDF}}(t_{j+1}) = c_1 h^p \frac{\partial^{p+1}}{\partial t^{p+1}}\boldsymbol{u}(t_j) + \mathcal{O}(h^{p+1}) + \beta c_2 h^q \frac{\partial^q}{\partial t^q}\boldsymbol{f}(\boldsymbol{u}(t_j), t_j) + \mathcal{O}(h^{q+1}),$$

where p is the order of the BDF scheme and q is the order of the extrapolation scheme, such that the order of convergence is given by $r = \min(p, q)$ [86, Chapter IV.4.2], under the assumption that $\boldsymbol{u}$ and $\boldsymbol{f}$ are sufficiently smooth. Nevertheless, for the SBDF schemes the coefficients α_i, β and β_i^* are chosen so that the method has convergence order $r = s$, for an s-step scheme given that the method is stable [14, Theorem 2.1]. Consequently, the multistep schemes indeed admit a higher convergence order than the one-step scheme, but the regularity requirements to achieve this convergence may be far more strenuous. See [48] for a more detailed discussion regarding the desired regularity of the functions.

Regarding stability, for the IMEX Euler scheme ensuring stability in both the implicit scheme and the explicit scheme will be sufficient [86, Chapter IV.4.2]. To see this,

consider once again the scalar test problem (2.7), then the IMEX Euler scheme yields

$$u_i^{(j+1)} = \frac{1+\tilde{\eta}_i}{1-\tilde{\lambda}_i} u_i^{(j)},$$

so that stability is ensured if $\frac{|1+\tilde{\eta}_i|}{|1-\tilde{\lambda}_i|} \leq 1$. If we recall that the implicit part of the scheme is stable for Re $\tilde{\lambda}_i \leq 1$ and fix this quantity, then the stability for the IMEX scheme is ensured for $|1+\tilde{\eta}_i| \leq 1$, which is precisely the stability region for the explicit Euler scheme. On the other hand, if we fix $|1+\tilde{\eta}_i| \leq 1$, it can be shown that stability is ensured for Re $\tilde{\lambda}_i \leq 1$, the stability region of implicit Euler. Therefore, for the IMEX Euler scheme, ensuring the stability of both the implicit and explicit schemes is enough to ensure stability. This condition is quite feasible given that the values of $\tilde{\eta}_i$ are generally much smaller than $\tilde{\lambda}_i$ for the considered class of equations.

For the higher-order schemes, the stability can be studied by the roots $|z_s(\tilde{\lambda}_i, \tilde{\eta}_i)|$ of the order s characteristic polynomial related to (2.10). As an example (see also [14, 51, 138]), for $s = 2$ the second degree characteristic polynomial is given by

$$(3 - 2\tilde{\lambda}_i)z^2 - 4(\tilde{\eta}_i + 1)z + 1 + 2\tilde{\eta}_i = 0,$$

so that the stability region for $s = 2$ is given by

$$\mathcal{D}_{\text{SBDF2}} = \{(\tilde{\lambda}_i, \tilde{\eta}_i) \in \mathbb{R}^- \times \mathbb{R} : |z_2(\tilde{\lambda}_i, \tilde{\eta}_i)| \leq 1\}.$$

We refer the reader to [14] for a discussion regarding the stability regions for $s \geq 2$, and to [14, 51] for the derivation of timestep restrictions related to the stability regions of the IMEX schemes.

2.1.2 Implementation details

All three matrix-oriented schemes discussed in Section 2.1.1 require the solution of a Sylvester equation of the form

$$\mathcal{M}_1(\boldsymbol{A}_1)\boldsymbol{Y} + \boldsymbol{Y}\mathcal{M}_2(\boldsymbol{A}_2) = \widetilde{\boldsymbol{F}}^{(j)}, \qquad \boldsymbol{Y} \in \mathbb{R}^{n_1 \times n_2}, \tag{2.12}$$

at each time iteration, where the linear operators $\mathcal{M}_1 : \mathbb{R}^{n_1 \times n_1} \to \mathbb{R}^{n_1 \times n_1}$ and $\mathcal{M}_2 : \mathbb{R}^{n_2 \times n_2} \to \mathbb{R}^{n_2 \times n_2}$ and the time-dependent matrix $\widetilde{\boldsymbol{F}}^{(j)} \in \mathbb{R}^{n_1 \times n_2}$ are respectively defined in Table 2.1 for the three schemes. Here we present some methods to accelerate the solution of (2.12) at each time instance; see also [51].

An eigenvalue method

Since (2.12) needs to be evaluated for a new right-hand side at each timestep with time-independent coefficient matrices, the procedure can be sufficiently accelerated

TABLE 2.1: Summary of $\mathcal{M}_1(\boldsymbol{A}_1)$, $\mathcal{M}_2(\boldsymbol{A}_2)$ and $\widetilde{\boldsymbol{F}}^{(j)}$ for different schemes.

Scheme	$\mathcal{M}_1(\boldsymbol{A}_1)$	$\mathcal{M}_2(\boldsymbol{A}_2)$	$\widetilde{\boldsymbol{F}}^{(j)}$
Exp. Euler	$\boldsymbol{A}_1$	$\boldsymbol{A}_2$	$e^{h\boldsymbol{A}_1}\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)e^{h\boldsymbol{A}_2}-\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)$
IMEX Euler	$\boldsymbol{I}_{n_1}-h\boldsymbol{A}_1$	$-h\boldsymbol{A}_2$	$\boldsymbol{U}^{(j)}+h\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)$
IMEX SBDF	$\alpha_0\boldsymbol{I}_{n_1}-h\beta\boldsymbol{A}_1$	$-h\beta\boldsymbol{A}_2$	$-\sum_{i=1}^{s}\alpha_i\boldsymbol{U}^{(j+1-i)}+h\sum_{i=1}^{s}\beta_i^*\mathcal{F}_{\boldsymbol{C}}\left(\boldsymbol{U}^{(j+1-i)},t_{j+1-i}\right)$

with an a-priori eigenvalue decomposition of the coefficient matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$, given that they are diagonalizable and that the decompositions are stable. More precisely, assuming that $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are diagonalizable we determine the eigenvalue decompositions $\boldsymbol{A}_k=\boldsymbol{X}_k\boldsymbol{\Lambda}_k\boldsymbol{X}_k^{-1}$, for $k=1,2$, where $\boldsymbol{X}_k$ is invertible and $\boldsymbol{\Lambda}_k=\operatorname{diag}\left(\lambda_1^{(k)},\lambda_2^{(k)},\dots\right)$.

By pre- and post-multiplying (2.12) by $\boldsymbol{X}_1^{-1}$ and $\boldsymbol{X}_2$ respectively we obtain a transformed Sylvester equation with diagonal coefficient matrices of the form

$$\mathcal{M}_1(\boldsymbol{\Lambda}_1)\widetilde{\boldsymbol{Y}}+\widetilde{\boldsymbol{Y}}\mathcal{M}_2(\boldsymbol{\Lambda}_2)=\widetilde{\mathbf{G}}^{(j)},\qquad \widetilde{\boldsymbol{Y}}=\boldsymbol{X}_1^{-1}\boldsymbol{Y}\boldsymbol{X}_2, \tag{2.13}$$

where $\widetilde{\mathbf{G}}^{(j)}=\boldsymbol{X}_1^{-1}\widetilde{\boldsymbol{F}}^{(j)}\boldsymbol{X}_2$. The solution to (2.13) and consequently (2.12) can then be explicitly obtained by the elementwise Hadamard product as

$$\widetilde{\boldsymbol{Y}}=\boldsymbol{\Psi}\circ\widetilde{\mathbf{G}}^{(j)},\qquad \text{s.t.}\qquad \boldsymbol{Y}=\boldsymbol{X}_1\widetilde{\boldsymbol{Y}}\boldsymbol{X}_2^{-1},$$

where the (i,j)th entry of $\boldsymbol{\Psi}\in\mathbb{R}^{n_1\times n_2}$ is defined as the inverted sum of the ith and jth diagonal elements of $\mathcal{M}_1(\boldsymbol{\Lambda}_1)$ and $\mathcal{M}_2(\boldsymbol{\Lambda}_2)$ respectively. More precisely,

$$\begin{aligned}
\boldsymbol{\Psi}_{i,j}&=\left(\lambda_i^{(1)}+\lambda_j^{(2)}\right)^{-1} && \text{for exp. Euler,}\\
\boldsymbol{\Psi}_{i,j}&=\left(1-h\lambda_i^{(1)}-h\lambda_j^{(2)}\right)^{-1} && \text{for IMEX Euler,}\\
\boldsymbol{\Psi}_{i,j}&=\left(\alpha_0-\beta h\lambda_i^{(1)}-\beta h\lambda_j^{(2)}\right)^{-1} && \text{for IMEX SBDF.}
\end{aligned}$$

The IMEX schemes require only the solution of the Sylvester equation (2.12) at each time instance. For the exponential Euler scheme, even further advantages can be drawn from the a-priori eigenvalue decomposition. Indeed, the full computation of (2.5) and (2.6) can be performed in the eigenbases resulting in almost negligible costs for evaluating the matrix exponentials. More precisely, the solution to the Sylvester equation (2.6) can be determined in a similar fashion to (2.13) so that the relation (2.5) (with $\boldsymbol{\Phi}^{(j)}\equiv\boldsymbol{Y}$) can be written as

$$\boldsymbol{U}^{(j+1)}=e^{h\boldsymbol{A}_1}\boldsymbol{U}^{(j)}e^{h\boldsymbol{A}_2}+\boldsymbol{X}_1\widetilde{\boldsymbol{Y}}\boldsymbol{X}_2^{-1}=\boldsymbol{X}_1\left(e^{h\boldsymbol{\Lambda}_1}\boldsymbol{X}_1^{-1}\boldsymbol{U}^{(j)}\boldsymbol{X}_2e^{h\boldsymbol{\Lambda}_2}+\widetilde{\boldsymbol{Y}}\right)\boldsymbol{X}_2^{-1},$$

where we have used the fact that $e^{h\boldsymbol{A}_i}=\boldsymbol{X}_ie^{h\boldsymbol{\Lambda}_i}\boldsymbol{X}_i^{-1}$ for $i=1,2$. Since $\boldsymbol{\Lambda}_1$ and $\boldsymbol{\Lambda}_2$ are diagonal, the products with the matrix exponentials can be more efficiently

determined elementwise as

$$e^{h\boldsymbol{\Lambda}_1}\left(\boldsymbol{X}_1^{-1}\boldsymbol{U}^{(j)}\boldsymbol{X}_2\right)e^{h\boldsymbol{\Lambda}_2} = \boldsymbol{\mathfrak{E}}\circ\left(\boldsymbol{X}_1^{-1}\boldsymbol{U}^{(j)}\boldsymbol{X}_2\right), \qquad \boldsymbol{\mathfrak{E}}_{i,j} = e^{h\lambda_i^{(1)}}e^{h\lambda_j^{(2)}}.$$

The same can be done for the products involving the matrix exponentials in the definition of $\widetilde{\boldsymbol{F}}^{(j)}$, in such a way that the right-hand side of (2.13) can be written as

$$\begin{aligned}\widetilde{\mathbf{G}}^{(j)} &= \boldsymbol{X}_1^{-1}\left(\boldsymbol{X}_1 e^{h\boldsymbol{\Lambda}_1}\boldsymbol{X}_1^{-1}\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)\boldsymbol{X}_2 e^{h\boldsymbol{\Lambda}_2}\boldsymbol{X}_2^{-1} - \mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)\right)\boldsymbol{X}_2 \\ &= \boldsymbol{\mathfrak{E}}\circ\left(\boldsymbol{X}_1^{-1}\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)\boldsymbol{X}_2\right) - \boldsymbol{X}_1^{-1}\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)\boldsymbol{X}_2.\end{aligned} \tag{2.14}$$

The full matrix-oriented exponential Euler scheme, performed with an a-priori eigenvalue decomposition is summarized below in Algorithm 2.1.

Algorithm 2.1 Matrix-Oriented Exponential Euler (Eigenvalue decomposition)

Require: $\boldsymbol{A}_1 \in \mathbb{R}^{n_1\times n_1}$, $\boldsymbol{A}_2 \in \mathbb{R}^{n_2\times n_2}$, $\mathcal{F} : \mathbb{R}^{n_1\times n_2}\times T \to \mathbb{R}^{n_1\times n_2}$, $\boldsymbol{C} \in \mathbb{R}^{n_1\times n_2}$, $\boldsymbol{U}_0 \in \mathbb{R}^{n_1\times n_2}$, final time t_f, number of timesteps n_t.

1: Determine the eigenvalue decompositions $\boldsymbol{A}_k = \boldsymbol{X}_k\boldsymbol{\Lambda}_k\boldsymbol{X}_k^{-1}$ for $k = 1, 2$
2: Compute $\boldsymbol{\mathfrak{E}}_{i,j} = e^{h\lambda_i^{(1)}}e^{h\lambda_j^{(2)}}$ and $\boldsymbol{\Psi}_{i,j} = \left(\lambda_i^{(1)} + \lambda_j^{(2)}\right)^{-1}$ and set $h = t_f/n_t$
3: **for** $j = 0$ **to** $n_t - 1$ **do**
4: Compute $\widetilde{\mathcal{F}}_{\boldsymbol{C}}^{(j)} = \boldsymbol{X}_1^{-1}\mathcal{F}_{\boldsymbol{C}}(\boldsymbol{U}^{(j)},t_j)\boldsymbol{X}_2$ // Project nonlinear term
5: Compute $\widetilde{\mathbf{G}}^{(j)} = \boldsymbol{\mathfrak{E}}\circ\widetilde{\mathcal{F}}_{\boldsymbol{C}}^{(j)} - \widetilde{\mathcal{F}}_{\boldsymbol{C}}^{(j)}$ // Evaluate (2.14)
6: Compute $\widetilde{\boldsymbol{Y}} = \boldsymbol{\Psi}\circ\widetilde{\mathbf{G}}^{(j)}$ // Solve (2.13)
7: Determine $\boldsymbol{U}^{(j+1)} = \boldsymbol{X}_1\left(\boldsymbol{\mathfrak{E}}\circ\left(\boldsymbol{X}_1^{-1}\boldsymbol{U}^{(j)}\boldsymbol{X}_2\right) + \widetilde{\boldsymbol{Y}}\right)\boldsymbol{X}_2^{-1}$ // Evaluate (2.5)
8: **end for**
9: **return** $\boldsymbol{U}^{(j)} \approx \boldsymbol{U}(t_j)$, $t_j = 0, h, \ldots, t_f$

When the coefficient matrices are nonsymmetric, the stability of the eigenvalue decompositions may be a concern. If stability is an issue, one can rather resort to the stable Schur decompositions of the matrices. The solution of the Sylvester equation (2.12) can then be efficiently obtained for all methods using the Bartels–Stewart algorithm [18], which is described below. In this case, the full matrix exponentials will, however, need to be evaluated for the exponential Euler scheme. In our implementations this is done with a scaling and squaring algorithm [4, 80], as implemented in the Matlab function `expm` for moderate n_1 and n_2.

The Bartels–Stewart method

The Bartels–Stewart algorithm relies on a Schur decomposition of the coefficient matrices. Since $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are considered to be real, the real Schur decompositions [70] $\boldsymbol{A}_1 = \boldsymbol{Q}_1\boldsymbol{R}_1\boldsymbol{Q}_1^\top$ and $\boldsymbol{A}_2^\top = \boldsymbol{Q}_2\boldsymbol{R}_2\boldsymbol{Q}_2^\top$ are computed with $\boldsymbol{Q}_1 \in \mathbb{R}^{n_1\times n_1}$ and $\boldsymbol{Q}_2 \in \mathbb{R}^{n_2\times n_2}$ orthogonal. The real Schur decomposition, as opposed to its complex variant, allows us to avoid complex arithmetic, however this might come at the expense of $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ being quasi-triangular instead of triangular. This means that $\boldsymbol{R}_1$

and $\boldsymbol{R}_2$ will have 2×2 matrix blocks on the main diagonal in the case of complex eigenvalues.

By pre- and post-multiplying, the equation by $\boldsymbol{Q}_1^\top$ and $\boldsymbol{Q}_2$ respectively, the transformed Sylvester equation with (quasi-)triangular coefficient matrices is given by

$$\mathcal{M}_1(\boldsymbol{R}_1)\widetilde{\boldsymbol{Y}} + \widetilde{\boldsymbol{Y}}\mathcal{M}_2\left(\boldsymbol{R}_2^\top\right) = \boldsymbol{Q}_1^\top \widetilde{\boldsymbol{F}}^{(j)}\boldsymbol{Q}_2, \qquad \widetilde{\boldsymbol{Y}} = \boldsymbol{Q}_1^\top \boldsymbol{Y}\boldsymbol{Q}_2.$$

This system can be solved entrywise by substitution to determine $\widetilde{\boldsymbol{Y}}$ for triangular $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$. If instead $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are quasi-triangular this is handled by solving the 2×2 Sylvester equations corresponding to the 2×2 blocks on the main diagonal; see, [18] for more details. The solution can be recovered by the transformation $\boldsymbol{Y} = \boldsymbol{Q}_1\widetilde{\boldsymbol{Y}}\boldsymbol{Q}_2^\top$. If required, we use the built-in MATLAB function `lyap` for the implementation of the Bartels–Stewart algorithm.

Further comments

The Sylvester equation (2.12) has a unique solution if and only if the spectra of $\mathcal{M}_1(\boldsymbol{A}_1)$ and $-\mathcal{M}_2(\boldsymbol{A}_2)$ are disjoint; see e.g., [141]. In some settings the spectra of $\mathcal{M}_1(\boldsymbol{A}_1)$ and $-\mathcal{M}_2(\boldsymbol{A}_2)$ may overlap so that (2.12) does not admit a unique solution. In this case, a relaxation parameter can be introduced to avoid the singularity. This will also ultimately affect the stability of the considered scheme; see the discussion in [51, Section 3.2].

Finally, we emphasize that for both the eigenvalue and Bartels–Stewart methods, only one decomposition of both $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ is required for all timesteps, since both matrices do not depend on time. Furthermore, even if adaptive time-stepping is implemented, it would not be necessary to recompute any decompositions when the stepsize is updated. Indeed, for the eigenvalue method, only the matrices $\boldsymbol{\Psi}$ and $\mathfrak{C}$ will need to be updated, and for the Bartels–Stewart method, the matrices $\mathcal{M}_1(\boldsymbol{R}_1)$ and $\mathcal{M}_2\left(\boldsymbol{R}_2^\top\right)$ will need to be multiplied by a constant and the diagonal terms shifted.

2.1.3 Numerical comparison the schemes

In this section, we experimentally investigate the expected accuracy and computational efficiency of the matrix-oriented integrators. The presented matrix and vector schemes have already been compared in [51] for the problem (2.1) with sparse coefficient matrices. In this thesis, we are, however, concerned with the efficient simulation of *reduced models* of the form (2.1), which typically contain dense coefficient matrices after projection. To this end, we compare the presented schemes in precisely this setting.

Example 2.1. We consider an ODE of the form (2.1) that stems from the space discretization of the following semilinear PDE

$$u_t = \delta \Delta u + \frac{\sin(\pi u)^2}{\sqrt{u^2 + 0.1^2}}, \quad \Omega = [-1, 1]^2, \quad t \in [0, 1], \tag{2.15}$$

with zero Dirichlet boundary conditions and initial condition $u_0 = \sin(2\pi x)\cos(2\pi y)$. More precisely, in order to obtain a *dense* ODE of the form (2.1), we consider a pseudospectral space discretization of (2.15); see e.g., [156].

First we investigate the expected accuracy of the schemes. To this end, we discretize (2.15) with $n_1 = n_2 = 30$ nodes in each spatial direction, and we construct a reference solution $\boldsymbol{U}_{\text{ref}}$ by applying the Matlab function `ode23s` with 4000 timesteps to the vectorized model (2.2) and reshaping the solution vectors into matrices. We display the accuracy of three of the presented schemes, namely exponential Euler, IMEX Euler and a 2nd order IMEX SBDF scheme by comparing them to the reference solution via the relative error measure $\|\boldsymbol{U}_{\text{ref}}^{(j)} - \boldsymbol{U}^{(j)}\|/\|\boldsymbol{U}^{(j)}\|$ at 1000 timesteps in the timespan. The results are presented in Figure 2.1 for $\delta = 0.05$ (left) and $\delta = 0.001$ (right).

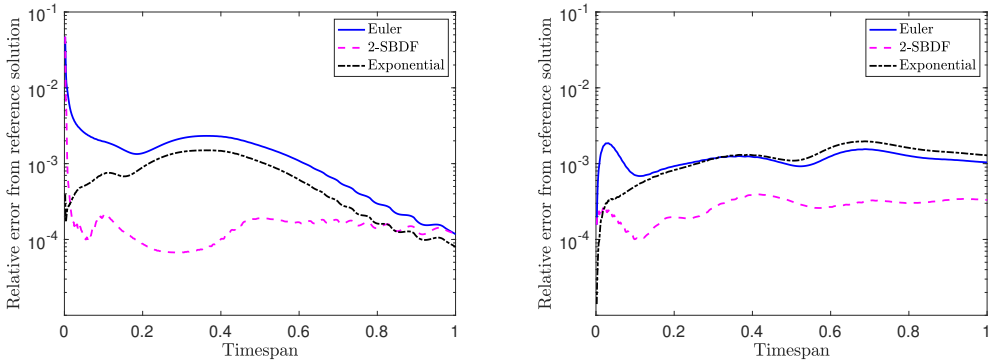

FIGURE 2.1: Expected accuracy of the three presented schemes for $\delta = 0.05$ (left) and $\delta = 0.001$ (right). The solutions are compared to a reference solution obtained by the Matlab function `ode23s`.

The figure on the left indicates that when the stiff linear term is more prominent (that is larger δ), the rational propagators of the two IMEX schemes fail to accurately capture the movement of the function at the initial transient phase. On the other hand, the exponential integrator suffices since the linear term is treated exactly. The figure on the right indicates that decreasing δ improves the IMEX schemes' performance compared to the exponential integrator. This illustrates that exponential integrators are able to handle the stiff linear term in semilinear ODEs efficiently.

To compare the efficiency in terms of computational time, we consider five space discretization refinements, namely $n = \{20, 40, 60, 80, 100\}$, where $n = n_1 = n_2$. We compare the time needed to integrate the discretized model at $n_t = 1000$ equispaced timesteps in the timespan, between the matrix-oriented and vectorized schemes. The results are displayed in Figure 2.2.

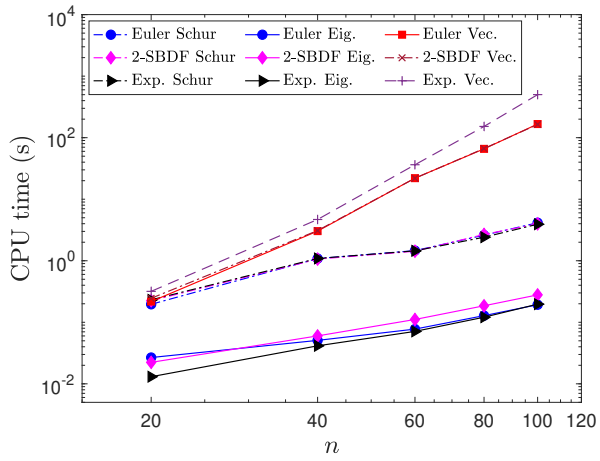

FIGURE 2.2: CPU time comparison between the matrix-oriented and vectorized integrators at $n_t = 1000$ timesteps for increasing dimension n.

Despite the nonsymmetric structure of the dense coefficient matrices, we experienced no stability issues with an a-priori eigenvalue decomposition of the matrices $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$. As a consequence, the results of the matrix-oriented schemes are presented for both the a-priori Schur (Bartels–Stewart) and eigenvalue decompositions, as discussed in Section 2.1.2, to accelerate the solution of the Sylvester equation (2.12) at each timestep. For the vectorized form of the integrators, we perform an a-priori pivoted LU factorization[1] to accelerate the linear system solves at each time iteration.

All timings in Figure 2.2 confirm the power of the matrix-oriented schemes. Even if an a-priori eigenvalue decomposition is not feasible, the matrix-oriented integrators still significantly outperform the vectorized versions. The eigenvalue decompositions, however, allow this gap to increase with a further order of magnitude. Moreover, the bottleneck of evaluating the large matrix exponential in the vectorized exponential Euler scheme is clear from the figure. Nevertheless, thanks to property (2.4), no such bottleneck is observed in the matrix setting. Here the matrix exponential in the vector setting was explicitly calculated using `expm` since the matrix $\boldsymbol{L}$ is nearly dense. Instead, one could also consider one of the algorithms for approximating action of the matrix exponential to a vector; see [37] for a collection and comparison of these methods. □

2.2 Matrix methods for the DRE

In this section we consider the numerical integration of semilinear ODEs of the form (2.1) with a special quadratic nonlinear term. That is, we consider a DRE of the form

[1]For the vector form of the exponential Euler method, we also experimented with an a-priori eigenvalue decomposition, but we did not experience any acceleration.

$$\dot{\boldsymbol{U}}(t) = \boldsymbol{A}_1\boldsymbol{U}(t) + \boldsymbol{U}(t)\boldsymbol{A}_2 - \boldsymbol{U}(t)\boldsymbol{B}\boldsymbol{U}(t) + \boldsymbol{C}, \quad \boldsymbol{U}(0) = \boldsymbol{U}_0, \tag{2.16}$$

where $\boldsymbol{A}_1 \in \mathbb{R}^{n_1\times n_1}$, $\boldsymbol{A}_2 \in \mathbb{R}^{n_2\times n_2}$, $\boldsymbol{B} \in \mathbb{R}^{n_2\times n_1}$ and $\boldsymbol{C} \in \mathbb{R}^{n_1\times n_2}$.

The numerical solution of the DRE (especially the small-scale setting) is a well-studied topic, see, e.g., [52, 114, 115, 149]. Among the explored methods are matrix generalizations of the BDF methods [55, 114], Rosenbrock methods [28] and splitting methods [149, 150]. All three methods are well tailored for the small-scale setting and can be efficiently implemented independent of whether the coefficient matrices are sparse or dense. In the large–scale setting, however, the matrices $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{U}_0$ are often assumed to have a low-rank structure. To this end, low-rank versions of these integrators have been introduced to attack the large-scale setting [107, 115, 150], which are not presented here; comparisons to these schemes are, however, presented in Section 3.4. Instead, in this thesis we aim to deal with ODEs of the form (2.16) by first reducing the dimension and then integrating; see e.g., Chapter 3. Recall that in this setting $\boldsymbol{f}_{\boldsymbol{c}}\left(\boldsymbol{u}^{(j)}, t_j\right) = \left(\boldsymbol{U}^{(j)} \otimes \boldsymbol{U}^{(j)}\right)\operatorname{vec}(\boldsymbol{B}) + \boldsymbol{c}$.

2.2.1 Fully implicit methods

Here we consider fully implicit schemes for solving the DRE. The class of methods we consider are implicit linear multistep schemes, in particular BDF methods. We derive their matrix formulation and report on their stability and convergence properties.

BDF methods

BDF methods are considered to be the most popular class of linear multistep methods for treating stiff ODEs. The s-step BDF scheme applied to the vectorized DRE is given by (2.9), where the α_i's and β are the coefficients of the s-step BDF method (see e.g., [13]) for $s \leq 6$ and are given in Table 2.2 in scaled form so that $\alpha_0 = 1$. Here $\boldsymbol{u}^{(0)} = \boldsymbol{u}_0$, but more care should be taken to determine the remaining $s-1$ initial values required for an s-step method. More precisely, for a method to have order p convergence, the initial values must all be determined with $\mathcal{O}(h^p)$ accuracy, which can be achieved by recursively applying an $s-1$ step method [13].

To derive the matrix-oriented application of BDF schemes to the DRE (2.16) (see also [55, 114]) we define

$$\mathbf{F}\left(\boldsymbol{U}^{(j+1)}\right) = \boldsymbol{A}_1\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}\boldsymbol{A}_2 - \boldsymbol{U}^{(j+1)}\boldsymbol{B}\boldsymbol{U}^{(j+1)} + \boldsymbol{C}. \tag{2.17}$$

Then, the approximation $\boldsymbol{U}^{(j+1)}$ of $\boldsymbol{U}(t_{j+1})$ is given by the implicit relation

$$\boldsymbol{U}^{(j+1)} = \sum_{i=1}^{s} \alpha_i \boldsymbol{U}^{(j+1-i)} + h\beta\mathbf{F}(\boldsymbol{U}^{(j+1)}), \quad j = s-1, \ldots, n_t - 1. \tag{2.18}$$

TABLE 2.2: Coefficients of the s-step BDF method

p	β	α_1	α_2	α_3	α_4	α_5	α_6
1	1	1					
2	2/3	4/3	−1/3				
3	6/11	18/11	−9/11	2/11			
4	12/25	48/25	−36/25	16/25	−3/25		
5	60/137	300/137	−300/137	200/137	−75/137	12/137	
6	60/147	360/147	−450/147	400/147	−225/147	72/147	-10/147

Substituting (2.17) into (2.18) results in the following semilinear matrix equation

$$-\boldsymbol{U}^{(j+1)} + h\beta\left(\boldsymbol{A}_1\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}\boldsymbol{A}_2 - \boldsymbol{U}^{(j+1)}\boldsymbol{B}\boldsymbol{U}^{(j+1)} + \boldsymbol{C}\right) + \sum_{i=1}^{s}\alpha_i\boldsymbol{U}^{(j+1-i)} = 0,$$

which can be reformulated as the following continuous-time ARE

$$\widetilde{\boldsymbol{A}}_1\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}\widetilde{\boldsymbol{A}}_2 - \boldsymbol{U}^{(j+1)}\widetilde{\boldsymbol{B}}\boldsymbol{U}^{(j+1)} + \widetilde{\boldsymbol{C}} = 0, \tag{2.19}$$

which needs to be evaluated at each time instance. The coefficient matrices are given by

$$\widetilde{\boldsymbol{A}}_1 = h\beta\boldsymbol{A}_1 - \frac{1}{2}\boldsymbol{I}_n, \quad \widetilde{\boldsymbol{A}}_2 = h\beta\boldsymbol{A}_2 - \frac{1}{2}\boldsymbol{I}_n, \quad \widetilde{\boldsymbol{B}} = h\beta\boldsymbol{B}, \quad \widetilde{\boldsymbol{C}} = h\beta\boldsymbol{C} + \sum_{i=1}^{s}\alpha_i\boldsymbol{U}^{(j+1-i)}.$$

The Riccati equation (2.19) can be solved using "direct" methods based on Schur decompositions, or the matrix sign function [36, 92]. Otherwise, iterative methods such as the Newton-Kleinmann iteration are more efficient when (2.19) is slightly larger [22, 44, 95, 106]; see also [26, 30]. For low-dimensional systems of the form (2.16), the MATLAB solver `icare` from the control systems toolbox can be used if $\widetilde{\boldsymbol{A}}_1 = \widetilde{\boldsymbol{A}}_2^\top$, which solves the generalized eigenvalue problem associated to the corresponding Hamiltonian formulation of the problem [12]. A brief sketch of the s-step BDF method is reported in Algorithm 2.2. An efficient MATLAB implementation of

Algorithm 2.2 s-step BDF method – BDF(s, n_t)

Require: $\boldsymbol{A}_1 \in \mathbb{R}^{n_1\times n_1}$, $\boldsymbol{A}_2 \in \mathbb{R}^{n_2\times n_2}$, $\boldsymbol{B} \in \mathbb{R}^{n_2\times n_1}$, $\boldsymbol{C} \in \mathbb{R}^{n_1\times n_2}$, $\boldsymbol{U}_0 \in \mathbb{R}^{n_1\times n_2}$, final time t_f, number of timesteps n_t, initial approximations $\boldsymbol{U}^{(0)}, \dots \boldsymbol{U}^{(s-1)}$.

1: $h = t_f/n_t$, $\widetilde{\boldsymbol{B}} = h\beta\boldsymbol{B}$, , $\widetilde{\boldsymbol{A}}_i = h\beta\boldsymbol{A}_i - \frac{1}{2}\boldsymbol{I}_n$ for $i = 1, 2$
2: **for** $j = s-1$ **to** $n_t - 1$ **do**
3: $\quad\widetilde{\boldsymbol{C}} = h\beta\boldsymbol{C} + \sum_{i=1}^{s}\alpha_i\boldsymbol{U}^{(j+1-i)}$
4: $\quad$Solve $\widetilde{\boldsymbol{A}}_1\boldsymbol{U}^{(j+1)} + \boldsymbol{U}^{(j+1)}\widetilde{\boldsymbol{A}}_2 - \boldsymbol{U}^{(j+1)}\widetilde{\boldsymbol{B}}\boldsymbol{U}^{(j+1)} + \widetilde{\boldsymbol{C}} = 0$
5: **end for**
6: **return** $\boldsymbol{U}^{(j)} \approx \boldsymbol{U}(t_j)$, $t_j = 0, h, \dots, t_f$

the s-step BDF scheme for $s \leq 6$ can be found in [134].

Convergence and stability properties.

A thorough discussion of the convergence and stability properties of BDF methods can be found in [13]. The local truncation error of an order p BDF scheme has the form

$$\boldsymbol{e}_{\text{BDF}}(t_{j+1}) = c_1 h^p \frac{\partial^{p+1}}{\partial t^{p+1}} \boldsymbol{u}(t_j) + \mathcal{O}(h^{p+1}),$$

which corresponds to that of the SBDF scheme from Section 2.1 without the extrapolation error, as expected. Moreover, as with the SBDF scheme, a convergent s-step BDF method converges with order $p = s$ given that the solution is sufficiently differentiable.

The consistency and stability required for an s-step BDF method to be convergent can be studied by the roots of the characteristic polynomials

$$\rho(\xi) = \sum_{i=0}^{s} \alpha_i \xi^{j+1-i} \qquad \phi(\xi) = \beta \xi^{j+1}.$$

More precisely, the method is consistent if and only if it has order $s \geq 1$; that is, it is consistent if and only if $\rho(1) = 0$ and $\rho'(1) = \phi(1)$ [13, Chapter 5.2.1]. Finally, the desired 0-stability is obtained if all the roots ξ_i of the polynomail $\rho(\xi)$ satisfy $|\xi_i| \leq 1$ [13, Theorem 5.1].

We conclude the section by depicting the typical convergence behavior of the BDF methods in our context. We consider an example from [115], where the $n \times n$ matrix $\boldsymbol{A} = \boldsymbol{A}_2 = \boldsymbol{A}_1^\top$ stems from the spatial finite difference discretization of the following convection-diffusion equation

$$\partial_t u = \Delta u - 10 x u_x - 100 y u_y, \qquad u|_{\partial\Omega} = 0$$

on $\Omega = (0,1)^2$. The remaining coefficient matrices are given in low-rank form as $\boldsymbol{B} = \boldsymbol{B}^\ell (\boldsymbol{B}^\ell)^\top$ and $\boldsymbol{C} = (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell$, where $\boldsymbol{B}^\ell \in \mathbb{R}^{n \times 1}$ and $\boldsymbol{C}^\ell \in \mathbb{R}^{1 \times n}$ are given as described in [115]. The initial condition is taken to be the zero matrix, that is $\boldsymbol{U}_0 = \boldsymbol{0}_n$. We compare the obtained solution with a "reference" numerical solution $\boldsymbol{U}_{\text{ref}}(t)$ obtained once again by the MATLAB function `ode23s`, with $n = 49$. The convergence behavior for $s = 1, 2, 3$ and n_t timesteps, with $n_t = 10, 100, 1000$ is displayed in Figure 2.3. The left plot shows the error $\|\boldsymbol{U}(t) - \boldsymbol{U}_{\text{ref}}(t)\|$ as a function of t, for different values of n_t. The right plot shows the evolution of the (1,1) component of the solution throughout the time span for the most accurate choice of BDF method, compared with that of the reference solution. The importance of the accuracy of the numerical integrator for solving the DRE is discussed in more detail in Chapter 3.

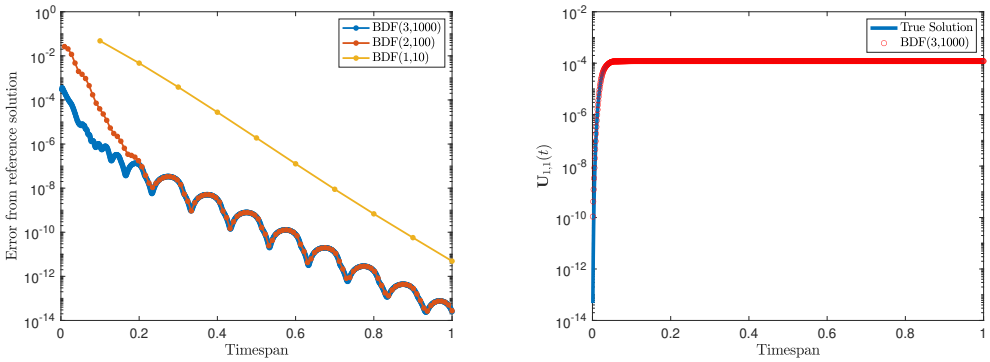

FIGURE 2.3: Typical convergence behavior of BDF methods (left) and evolution of the $\boldsymbol{U}_{1,1}$ component of the reference and BDF$(3, 1000)$ solution (right).

2.2.2 Splitting methods

Matrix-oriented splitting methods give an alternative to the presented BDF schemes for the DRE [149, 150]. These methods are based on splitting the linear and nonlinear parts of (2.16); that is, given the functions

$$\mathbf{F}_1(\boldsymbol{U}) = \boldsymbol{A}_1\boldsymbol{U}(t) + \boldsymbol{U}(t)\boldsymbol{A}_2 + \boldsymbol{C}, \quad \text{and} \quad \mathbf{F}_2(\boldsymbol{U}) = -\boldsymbol{U}(t)\boldsymbol{B}\boldsymbol{U}(t),$$

the subproblems

$$\dot{\boldsymbol{U}} = \mathbf{F}_1(\boldsymbol{U}), \quad \text{and} \quad \dot{\boldsymbol{U}} = \mathbf{F}_2(\boldsymbol{U}), \quad \text{with} \quad \boldsymbol{U}(0) = \boldsymbol{U}_0, \tag{2.20}$$

are considered far easier to solve, since both have closed form solutions and the first subproblem is affine. Denoting the solution operators of (2.20) by $\mathcal{P}_{\mathbf{F}_1}(t)$ and $\mathcal{P}_{\mathbf{F}_2}(t)$ respectively, the solutions at time t are respectively given by (see [149, Section II])

$$\mathcal{P}_{\mathbf{F}_1}(t)\boldsymbol{U}_0 = e^{t\boldsymbol{A}_1}\boldsymbol{U}_0 e^{t\boldsymbol{A}_2} + \int_0^t e^{s\boldsymbol{A}_1}\boldsymbol{C}e^{s\boldsymbol{A}_2}ds \quad \text{and} \quad \mathcal{P}_{\mathbf{F}_2}(t)\boldsymbol{U}_0 = (\boldsymbol{I} + t\boldsymbol{U}_0\boldsymbol{B})^{-1}\boldsymbol{U}_0.$$

Then, the respective first (Lie) and second (Strang) order splitting schemes are given by the time-stepping operators

$$\mathcal{H}_h^{\text{Lie}} = \mathcal{P}_{\mathbf{F}_1}(h)\mathcal{P}_{\mathbf{F}_2}(h), \quad \text{and} \quad \mathcal{H}_h^{\text{Strang}} = \mathcal{P}_{\mathbf{F}_2}\left(\frac{h}{2}\right)\mathcal{P}_{\mathbf{F}_1}(h)\,\mathcal{P}_{\mathbf{F}_2}\left(\frac{h}{2}\right).$$

Notice that these schemes do not require the solution of a nonlinear ARE at each time instance. This, however, comes at the cost of evaluating matrix exponentials and linear system solves. We refer the reader to [149] for further implementation details regarding the Lie and Strang schemes, including efficient methods for treating the matrix exponential and for approximating the integral. For higher-order and adaptive splitting schemes, we refer the reader to [150].

Convergence and stability properties.

In [86, Chapter IV.1] the Lie and Strang splitting schemes are respectively shown to have first and second-order accuracy. The consistency is proven through Taylor expansions, whereas the stability for small enough h is determined through the local Lipschitz continuity of the nonlinear term. Nevertheless, the determined error bounds depend on the matrices involved. Hence, if the coefficient matrices stem from the space discretization of PDE operators, these error bounds may go to infinity as the grid is refined. To avoid this, a stiff error analysis is required, as is done for the Lie splitting scheme in [77] for the case $\boldsymbol{B} \equiv \boldsymbol{I}$.

2.3 Concluding remarks

The efficient simulation of both large-scale and small-scale semilinear ODEs is fundamental for monitoring the time evolution of complex phenomena. This chapter presented matrix-oriented versions of the most widely used methods for treating semilinear ODEs. We have presented SI matrix methods in the first part, where the stiff linear term is treated implicitly (or exactly), and the nonstiff nonlinear term is treated explicitly. We have also discussed the convergence and stability properties of the considered schemes and several essential implementation issues. Among the SI schemes, it is clear from the literature that the ETD schemes are the preferred, more robust integrators, given that the efficient evaluation of the matrix exponential or its action on a vector is feasible. However, this matrix function evaluation can be sufficiently accelerated by taking advantage of the Kronecker sum structure of the coefficient matrix and writing the relation in matrix form. This will be particularly advantageous when dealing with small, dense coefficient matrices; see Chapter 4. Furthermore, the power of the matrix-oriented methods in comparison to their vectorized counterparts has also been illustrated via a numerical experiment.

In the second part of this chapter, we considered the quadratic DRE. The structure of the nonlinear quadratic term allows us to efficiently integrate the DRE with fully implicit schemes and splitting methods. This is because the most common implicit methods applied to the DRE require the solution of an ARE at each time iteration. The efficient solution of both large-scale and small-scale AREs is a well-studied topic (see [26, 30]), and hence solving such a nonlinear system at each timestep is considered feasible, especially when working with low-dimensional matrices. On the other hand, the splitting methods allow an efficient implementation with the quadratic nonlinearity since both the linear and nonlinear subproblems admit exact solution formulas. These two classes of methods for treating the DRE are also compared in Section 3.4.

Chapter 3

The Differential Riccati Equation[1]

This chapter is devoted to the Differential Riccati Equation (DRE), which can be interpreted as a semilinear ODE, where the nonlinear term has a special quadratic structure. More precisely, we aim to approximate the solution of the (low-rank) continuous-time symmetric DRE of the form

$$\dot{\boldsymbol{U}}(t) = \boldsymbol{A}^\top \boldsymbol{U}(t) + \boldsymbol{U}(t)\boldsymbol{A} - \boldsymbol{U}(t)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{U}(t) + (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell, \quad \boldsymbol{U}(0) = \boldsymbol{U}_0, \tag{3.1}$$

in the unknown matrix $\boldsymbol{U}(t) \in \mathbb{R}^{n\times n}$, where $\boldsymbol{U}_0 = \boldsymbol{U}_0^\ell(\boldsymbol{U}_0^\ell)^\top$ and $t \in [0, t_f] = T$. Here, $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B}^\ell \in \mathbb{R}^{n\times n_b}$, $\boldsymbol{C}^\ell \in \mathbb{R}^{n_c\times n}$ and $\boldsymbol{U}_0^\ell \in \mathbb{R}^{n\times n_o}$ are time invariant, and $n_b, n_c, n_o \ll n$. The matrix $\boldsymbol{A}$ is assumed to be very large, sparse, and nonsingular, whereas $\boldsymbol{B}^\ell$, $\boldsymbol{C}^\ell$, and $\boldsymbol{U}_0^\ell$ have maximum rank. Even though the matrix $\boldsymbol{A}$ is sparse, the solution $\boldsymbol{U}(t)$ is typically dense and impossible to store when n is large. Under the considered hypotheses, numerical evidence seems to indicate that $\boldsymbol{U}(t)$ usually has rapidly decaying singular values; hence a low-rank approximation to $\boldsymbol{U}(t)$ may be considered, see e.g., [151]. Results on the existence of low-rank solutions for the (linear) algebraic Sylvester and Lyapunov equations can also be found in [71, 72].

The DRE plays a fundamental role in optimal control theory, filter design theory, model reduction problems, as well as in differential games [1, 23, 32, 46, 129]. Equations of the form (3.1) are crucial in the numerical treatment of the linear quadratic regulator (LQR) problem [1, 46, 105]: given the state equation

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}^\ell \boldsymbol{k}(t), \quad \boldsymbol{y}(t) = \boldsymbol{C}^\ell \boldsymbol{x}(t), \qquad \boldsymbol{x}(0) = \boldsymbol{x}_0 \tag{3.2}$$

consider the finite horizon case, where the finite time cost integral has the form

$$\boldsymbol{J}(\boldsymbol{k}) = \boldsymbol{x}(t_f)^\top \boldsymbol{K}_f \boldsymbol{x}(t_f) + \int_0^{t_f} \Big(\boldsymbol{x}(t)^\top (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell \boldsymbol{x}(t) + \boldsymbol{k}(t)^\top \boldsymbol{k}(t)\Big)\, dt. \tag{3.3}$$

The matrix $\boldsymbol{K}_f$ is assumed to be symmetric and nonnegative definite and $\boldsymbol{k}(t) \in \mathbb{R}^{n_b}$ is a vector of control inputs. Assuming that the pair $(\boldsymbol{A}, \boldsymbol{B}^\ell)$ is stabilizable and the pair $(\boldsymbol{C}^\ell, \boldsymbol{A})$ is detectable, the optimal input $\tilde{\boldsymbol{k}}(t)$, minimizing (3.3), can

[1] An earlier version of this chapter has been published in [94].

be determined as $\tilde{\boldsymbol{k}}(t) = -(\boldsymbol{B}^\ell)^\top \boldsymbol{K}(t)\tilde{\boldsymbol{x}}(t)$, and the optimal trajectory is subject to $\dot{\tilde{\boldsymbol{x}}} = (\boldsymbol{A} - \boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{K}(t))\boldsymbol{x}(t)$. The matrix $\boldsymbol{K}(t)$ is the solution to the DRE

$$\dot{\boldsymbol{K}}(t) = \boldsymbol{A}^\top \boldsymbol{K}(t) + \boldsymbol{K}(t)\boldsymbol{A} - \boldsymbol{K}(t)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{K}(t) + (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell, \quad \boldsymbol{K}(t_f) = \boldsymbol{K}_f. \tag{3.4}$$

Using a common practice, we can transform (3.4) into the initial value problem (3.1) via the change of variables $\boldsymbol{U}(t_f - t) = \boldsymbol{K}(t)$.

Under certain assumptions, the exact solution of (3.1) can be expressed in integral form as (see, e.g., [103, Theorem 8])

$$\boldsymbol{U}(t) = e^{t\boldsymbol{A}^\top} \boldsymbol{U}_0^\ell (\boldsymbol{U}_0^\ell)^\top e^{t\boldsymbol{A}} + \int_0^t e^{(t-s)\boldsymbol{A}^\top} \left((\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell - \boldsymbol{U}(s)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{U}(s) \right) e^{(t-s)\boldsymbol{A}} ds, \tag{3.5}$$

so that when $t \to \infty$ the DRE reaches a steady-state solution satisfying the ARE

$$0 = \boldsymbol{A}^\top \boldsymbol{U}_\infty + \boldsymbol{U}_\infty \boldsymbol{A} - \boldsymbol{U}_\infty \boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{U}_\infty + (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell. \tag{3.6}$$

Due to its semilinear structure, the DRE is classified as a stiff ODE. The stiffness and nonlinearity of the DRE are responsible for the difficulties in its numerical solution even on a small scale ($n < 10^3$); see Section 2.2 for a collection of stiff integrators tailored towards the DRE. In [134], iterative methods are implemented within the matrix-oriented implicit methods discussed in Section 2.2, allowing for the computation of an approximate solution to the DRE when $n \gg 10^3$. These algorithms require the solution of a large *algebraic* Riccati equation at each timestep, which again raises big concerns as of storage and computational efforts.

In this thesis we aim to first reduce the dimension of the problem (3.1) and then integrate a low-dimensional problem with the same structure as the original. A promising idea is to rely on a model order reduction strategy typically used in linear and nonlinear dynamical systems. In this setting, the original system is replaced with

$$\dot{\widehat{\boldsymbol{x}}}(t) = \widehat{\boldsymbol{A}}_k \widehat{\boldsymbol{x}}(t) + \widehat{\boldsymbol{B}}_k^\ell \boldsymbol{k}(t), \quad \boldsymbol{y}(t) = \widehat{\boldsymbol{C}}_k^\ell \widehat{\boldsymbol{x}}(t), \qquad \widehat{\boldsymbol{x}}(0) = \widehat{\boldsymbol{x}}_0 \tag{3.7}$$

where $\widehat{\boldsymbol{A}}_k, \widehat{\boldsymbol{B}}_k^\ell$ and $\widehat{\boldsymbol{C}}_k^\ell$ are projections and restrictions of the original matrices onto a subspace range$(\mathcal{V}_k)$ of small dimension. The DRE associated with this reduced order problem is solved, yielding an optimal corresponding cost function. This strategy allows for a natural low-rank approximation to the sought after DRE solution $\boldsymbol{U}(t)$ of the form $\mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top$, obtained by interpolating the reduced-order solution $\widehat{\boldsymbol{U}}_k(t)$ at selected time instances. One main feature is that a single space is used for all time snapshots so that the approximate solutions can be kept in factored form with few memory allocations. See, e.g., [11] for a general presentation of algebraic reduction methods for linear dynamical systems, and to [141] for a detailed discussion motivating the reduction approach in the context of the ARE.

A key ingredient for the success of the reduction methodology is the choice of the

approximation space onto which the algebraic reduction is performed; [11] presents a comprehensive description of various space selections in the dynamical system setting. Following strategies already successfully adopted for the AREs, the authors of [99] and [75] have independently used polynomial and extended Krylov subspaces as approximation space, respectively, in the differential setting. A significant characteristic of these spaces is that their dimension can be expanded iteratively so that if the determined approximate solution is not sufficiently accurate, the Krylov space can be enlarged, and the process continued. Several questions remain open in the methods proposed in [99],[75]. On the one hand, it is well known that polynomial Krylov subspaces require a large dimension to solve real application problems satisfactorily, thus destroying the reduction advantages. On the other hand, the multiple time stepping method proposed in [75] only provides an accurate approximation at $t = t_f$, except when $\boldsymbol{U}_0 = 0$. For $\boldsymbol{U}_0 = \boldsymbol{U}_0^\ell(\boldsymbol{U}_0^\ell)^\top \neq 0$ of low rank, memory requirements of the extended method grow significantly. These problems can be satisfactorily solved by using a general *rational* Krylov subspace, which is shown in various applications to supply good spectral information on the involved matrices with much smaller dimensions than the polynomial and extended versions. Such gain has been experimentally reported in the literature in the solution of the ARE. We show that great computational and memory savings can be obtained when projecting onto the fully rational Krylov subspace, and that with appropriate implementation, the extended Krylov subspace may also be competitive with certain data.

A related issue that has somehow been overlooked in the available literature is the expected final accuracy and thus the stopping criterion. Time dependence of the DRE makes the reduced problem trickier to handle than in the purely algebraic case; in particular, two intertwined issues arise: i) The accuracy of the approximate solution may vary considerably within the time interval T; ii) Throughout the reduction process the reduced ODE cannot be solved with high accuracy and, quite the opposite, low-order methods should be used to make the overall cost feasible. We analyze these difficulties in detail, and by exploiting the inherent structure of the reduced-order model, we derive a two-phase strategy that first focuses on the reduction, then on the integration, in a way that is efficient for memory and CPU time usage, but also in terms of final expected accuracy.

We also discuss several algebraic properties of the approximate solution and its relation both with the solution $\boldsymbol{U}(t)$ for $t \in T$, and with the steady-state solution $\boldsymbol{U}_\infty$. These results continue a matrix analysis started in [99], where positivity and monotonicity properties of the approximate solution obtained by certain reduction methods are explored.

3.1 Order reduction with Krylov-based subspaces

In this section, we show how the Krylov subspace methods presented in Section 1.3.1 are applied to the DRE. Krylov-based projection methods were first applied to ARE's in [87] (polynomial spaces) and later improved in [79] (extended space) and [144] (rational spaces). The two rational spaces prove to be far superior to the polynomial Krylov space in most reduction strategies where they are applied in the literature, as long as solving linear systems at each iteration is feasible. The DRE has been attacked in [75] with the extended space, and in [99] with the polynomial space; here, we close the gap, as far as Krylov subspaces are concerned. In addition, we address several implementation issues to make the final method computationally reliable and, to the best of our knowledge, a great competitor among the available methods for large-scale DRE problems.

The rational Krylov subspace was proposed initially in the eigenvalue context in [130]. Its use in our context is motivated by [144] and later [141], where its effectiveness in solving the ARE is amply discussed. The approximation effectiveness of this subspace depends on the choice of shifts $\boldsymbol{s}$, and this issue has been investigated in the literature; see, e.g., [125], [59]. The adaptive choice of shifts was tailored to the ARE in [109] by the inclusion of information of the term $\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top$ during the shift selection; see also [141] for a more detailed discussion[2]. We used this last adaptive strategy in our numerical experiments, where the approximate solution at timestep t_f is used. As discussed in Section 1.3.1, this rational Arnoldi algorithm from [59] constructs a complex-valued basis in the presence of complex shifts. Standard ODE solvers do not handle complex arithmetic well, hence we implemented an all-real basis using the method introduced in [131], which has also been presented in Section 1.3.1.

For the DRE, both the rational and extended Krylov subspaces are constructed with the pair $(\boldsymbol{A}^\top, \boldsymbol{Z})$. While for the ARE $\boldsymbol{Z} = (\boldsymbol{C}^\ell)^\top$, in the differential context the starting matrix for generating these spaces is given by $\boldsymbol{Z} = [(\boldsymbol{C}^\ell)^\top, \boldsymbol{U}_0^\ell]$, where $\boldsymbol{U}_0 = \boldsymbol{U}_0^\ell(\boldsymbol{U}_0^\ell)^\top$. Both matrices $\boldsymbol{C}^\ell$ and $\boldsymbol{U}_0^\ell$ play a crucial role in the closed-form DRE solution matrix and are thus included to generate the projection space. The idea of reduction methods is to first, project the large DRE onto the smaller subspace $\mathcal{K}_k$, then solve the projected equation and finally expand the solution back to the original space.

Let the columns of $\mathcal{V}_k \in \mathbb{R}^{n\times n_v}$ span the considered Krylov subspace and assume that $\mathcal{V}_k$ has orthonormal columns. Following similar reduction methods in the dynamical system contexts, see, e.g., [11], the reduction process consists of first projecting and restricting the original data onto the approximation space as

$$\widehat{\boldsymbol{A}}_k = \mathcal{V}_k^\top \boldsymbol{A}\mathcal{V}_k, \quad \widehat{\boldsymbol{B}}_k^\ell = \mathcal{V}_k^\top \boldsymbol{B}^\ell, \quad \widehat{\boldsymbol{U}}_{0,k}^\ell = \mathcal{V}_k^\top \boldsymbol{U}_0^\ell \quad \text{and} \quad \widehat{\boldsymbol{C}}_k^\ell = \boldsymbol{C}^\ell \mathcal{V}_k.$$

[2]The Matlab code of the rational Krylov subspace method for ARE is available at `http://www.dm.unibo.it/~simoncin/software`

Then the following low order DRE needs to be solved,

$$\begin{aligned}\dot{\widehat{\boldsymbol{U}}}_k(t) &= \widehat{\boldsymbol{A}}_k^\top \widehat{\boldsymbol{U}}_k(t) + \widehat{\boldsymbol{U}}_k(t)\widehat{\boldsymbol{A}}_k - \widehat{\boldsymbol{U}}_k(t)\widehat{\boldsymbol{B}}_k^\ell \left(\widehat{\boldsymbol{B}}_k^\ell\right)^\top \widehat{\boldsymbol{U}}_k(t) + \left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top \widehat{\boldsymbol{C}}_k^\ell \\ \widehat{\boldsymbol{U}}_k(0) &= \widehat{\boldsymbol{U}}_{0,k}^\ell \left(\widehat{\boldsymbol{U}}_{0,k}^\ell\right)^\top,\end{aligned} \tag{3.8}$$

for $t \in T$. This low-dimensional DRE admits a unique solution for $t_f < \infty$, see e.g., [103]. Restrictions on the data to allow for positive, stabilizing solutions are discussed in more detail in Section 3.3. An approximation to the sought after solution is then written as

$$\boldsymbol{U}_k(t) = \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top \approx \boldsymbol{U}(t), \qquad t \in T. \tag{3.9}$$

Note that both left and right approximation spaces are given by range$(\mathcal{V}_k)$, thanks to the symmetry of the solution of the DRE (3.1). Furthermore, we stress that $\boldsymbol{U}_k(t)$ is never explicitly computed, but always referred to via the matrix $\mathcal{V}_k$ and the set of matrices $\widehat{\boldsymbol{U}}_k(t)$ at given time instances. In fact, the matrices $\widehat{\boldsymbol{U}}_k(t)$ may also be numerically low rank, so that at the end of the whole process, a further reduction can be performed by truncating the eigendecomposition of $\widehat{\boldsymbol{U}}_k(t)$ for each t.

Remark 3.1. *The approach we have derived is solely based on the order reduction of the dynamical system (3.2). Nonetheless, with some abuse of notation, the reduced DRE could have been formally obtained through a Galerkin condition on the differential equation. For $t \in T$ let*

$$\boldsymbol{R}_k(t) := \dot{\boldsymbol{U}}_k(t) - \boldsymbol{A}^\top \boldsymbol{U}_k(t) - \boldsymbol{U}_k(t)\boldsymbol{A} + \boldsymbol{U}_k(t)\boldsymbol{B}^\ell (\boldsymbol{B}^\ell)^\top \boldsymbol{U}_k(t) - (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell$$

be the residual matrix for $\boldsymbol{U}_k(t) = \mathcal{V}_k \widehat{\boldsymbol{U}}_k \mathcal{V}_k^\top$. The matrix $\widehat{\boldsymbol{U}}_k(t)$ is thus determined by imposing that the residual satisfies the following Galerkin condition

$$\mathcal{V}_k^\top \boldsymbol{R}_k(t)\, \mathcal{V}_k = 0, \quad t \in T, \tag{3.10}$$

that is, $\boldsymbol{R}_k(t) \perp \mathcal{K}_k$ in a matrix sense, so that the residual is forced to belong to a smaller and smaller subspace as $\mathcal{K}_k$ grows. Substituting $\boldsymbol{U}_k(t) = \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top$ into the residual matrix, the application of the Galerkin condition results in the projected system

$$\begin{aligned}\mathcal{V}_k^\top \big(\mathcal{V}_k \dot{\widehat{\boldsymbol{U}}}_k(t) \mathcal{V}_k^\top &- \boldsymbol{A}^\top \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top - \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top \boldsymbol{A} \\ &+ \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top \boldsymbol{B}^\ell (\boldsymbol{B}^\ell)^\top \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top - (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell \big) \mathcal{V}_k = 0,\end{aligned}$$

which corresponds to (3.8). This is rigorous as long as $\dot{\boldsymbol{U}}_k = \mathcal{V}_k \dot{\widehat{\boldsymbol{U}}}_k \mathcal{V}_k^\top$ holds. □

It is crucial to realize that, as opposed to some available methods in the literature (such as, for instance, [134],[150],[34] and the time-invariant algorithms in [107]),

the approximation space is independent of the time-stepping, that is a single space range($\mathcal{V}_k$) is used for all time steps. This provides enormous memory savings whenever the approximate solution is required at different time instances in T. Theoretical motivation for keeping the approximation space independent of the time-stepping is contained in [19, 21], where it is shown that the solution of the DRE lives in an invariant Krylov subspace.

The class of numerical methods we use for solving the reduced DRE (3.8), that is, the matrix-oriented BDF methods, has been described in Section 2.2. In the rest of this paper, we specialize the generic derivation above to the extended and rational Krylov subspaces, which significantly outperformed polynomial spaces both in terms of CPU time and memory requirements. More information on these spaces and their properties are given in Section 1.3.1, where we also discuss the generation of a real rational Krylov basis in the presence of non-real shifts.

3.2 Stopping criterion and the complete algorithm

To complete the reduction algorithm of Section 3.1, we need to introduce a stopping criterion. We found that it is crucial to take into account the accuracy of the numerical method employed to solve the reduced DRE.

To derive our stopping criterion, we were inspired by those in [75, 99], however, we made some important modifications. In both cited references, the authors assume that the inner problem (3.8) is solved exactly, which is not true in general. We thus consider that the numerical method solves the reduced problem with residual matrix $\boldsymbol{R}_k^{(I)}(t) := \dot{\widehat{\boldsymbol{U}}}_k(t) - \mathbf{F}(\widehat{\boldsymbol{U}}_k(t))$, with $\mathbf{F}$ defined as in (2.17), so that the final DRE residual can be split into two components.

Proposition 3.1. *Let $\boldsymbol{U}_k(t) = \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top$ be the Krylov-based approximate solution after k iterations, where $\widehat{\boldsymbol{U}}_k(t)$ approximately solves the reduced problem (3.8). With the previous notation, the residual matrix $\boldsymbol{R}_k(t) = \dot{\boldsymbol{U}}_k(t) - \mathbf{F}(\boldsymbol{U}_k(t))$ satisfies*

$$\|\boldsymbol{R}_k(t)\|_F^2 = \|\boldsymbol{R}_k^{(I)}(t)\|_F^2 + 2\|\boldsymbol{R}_k^{(O)}(t)\|_F^2, \tag{3.11}$$

where $\boldsymbol{R}_k^{(I)}(t) = \dot{\widehat{\boldsymbol{U}}}_k(t) - \mathbf{F}(\widehat{\boldsymbol{U}}_k(t))$ and $\boldsymbol{R}_k^{(O)}(t) = \boldsymbol{\Gamma}_k^\top \widehat{\boldsymbol{U}}_k(t)$ with $\boldsymbol{\Gamma}_k$ as in (1.8).

Proof. Substituting Equation (3.9) into the residual $\boldsymbol{R}_k(t)$ we obtain

$$\begin{aligned} \boldsymbol{R}_k(t) = \mathcal{V}_k \dot{\widehat{\boldsymbol{U}}}_k(t) \mathcal{V}_k^\top - \boldsymbol{A}^\top \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top - \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top \boldsymbol{A} \\ + \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top \boldsymbol{B}^\ell (\boldsymbol{B}^\ell)^\top \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top - (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell. \end{aligned} \tag{3.12}$$

Since $(\boldsymbol{C}^\ell)^\top$ belongs to range$(\mathcal{V}_k)$, we can write $(\boldsymbol{C}^\ell)^\top = \mathcal{V}_k \left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top$. Using (1.8) and keeping in mind that the subspace is constructed with $\boldsymbol{A}^\top$ instead of $\boldsymbol{A}$, we get

$$\begin{aligned}\boldsymbol{R}_k(t) = \mathcal{V}_k\dot{\widehat{\boldsymbol{U}}}_k(t)\mathcal{V}_k^\top - (\mathcal{V}_k\widehat{\boldsymbol{A}}_k^\top + \boldsymbol{\Upsilon}_{k+1}\boldsymbol{\Gamma}_k^\top)\widehat{\boldsymbol{U}}_k(t)\mathcal{V}_k^\top - \mathcal{V}_k\widehat{\boldsymbol{U}}_k(t)(\widehat{\boldsymbol{A}}_k\mathcal{V}_k^\top + \boldsymbol{\Gamma}_k\boldsymbol{\Upsilon}_{k+1}^\top) \\ + \mathcal{V}_k\widehat{\boldsymbol{U}}_k(t)\mathcal{V}_k^\top\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\mathcal{V}_k\widehat{\boldsymbol{U}}_k(t)\mathcal{V}_k^\top - \mathcal{V}_k\left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top\widehat{\boldsymbol{C}}_k^\ell\mathcal{V}_k^\top.\end{aligned}$$

Since $\mathcal{V}_{k+1} = [\mathcal{V}_k, \boldsymbol{\Upsilon}_{k+1}]$, we can write $\boldsymbol{R}_k(t) = \mathcal{V}_{k+1}\boldsymbol{J}_k(t)\mathcal{V}_{k+1}^\top$, where

$$\boldsymbol{J}_k(t) = \left[\begin{array}{c|c} \dot{\widehat{\boldsymbol{U}}}_k(t) - \widehat{\boldsymbol{A}}_k^\top\widehat{\boldsymbol{U}}_k(t) - \widehat{\boldsymbol{U}}_k(t)\widehat{\boldsymbol{A}}_k + \widehat{\boldsymbol{U}}_k(t)\widehat{\boldsymbol{B}}_k^\ell\left(\widehat{\boldsymbol{B}}_k^\ell\right)^\top\widehat{\boldsymbol{U}}_k(t) - \left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top\widehat{\boldsymbol{C}}_k^\ell & \widehat{\boldsymbol{U}}_k(t)\boldsymbol{\Gamma}_k \\ \hline \boldsymbol{\Gamma}_k^\top\widehat{\boldsymbol{U}}_k(t) & \boldsymbol{0}\end{array}\right].$$

Let $\boldsymbol{R}_k^{(I)}(t)$ be the residual of the numerical ODE inner solver. Then

$$\boldsymbol{J}_k(t) = \left[\begin{array}{c|c} \boldsymbol{R}_k^{(I)}(t) & \widehat{\boldsymbol{U}}_k(t)\boldsymbol{\Gamma}_k \\ \hline \boldsymbol{\Gamma}_k^\top\widehat{\boldsymbol{U}}_k(t) & \boldsymbol{0}\end{array}\right].$$

Since the columns of $\mathcal{V}_{k+1}$ are orthonormal,

$$\begin{aligned}\|\boldsymbol{R}_k(t)\|_F^2 &= \|\mathcal{V}_{k+1}\boldsymbol{J}_k(t)\mathcal{V}_{k+1}^\top\|_F^2 = \|\boldsymbol{J}_k(t)\|_F^2 \\ &= \mathrm{Tr}\left(\boldsymbol{R}_k^{(I)}(t)^\top\boldsymbol{R}_k^{(I)}(t) + 2(\widehat{\boldsymbol{U}}_k(t)\boldsymbol{\Gamma}_k)(\boldsymbol{\Gamma}_k^\top\widehat{\boldsymbol{U}}_k(t))\right),\end{aligned}$$

that is, $\|\boldsymbol{R}_k(t)\|_F^2 = \|\boldsymbol{R}_k^{(I)}(t)\|_F^2 + 2\|\boldsymbol{\Gamma}_k^\top\widehat{\boldsymbol{U}}_k(t)\|_F^2$, and the result follows. □

The expression for $\boldsymbol{J}_k(t)$ emphasizes that at each iteration k the matrix $\widehat{\boldsymbol{U}}_k(t)$ is the exact solution of

$$\dot{\widehat{\boldsymbol{U}}}_k(t) - \widehat{\boldsymbol{A}}_k^\top\widehat{\boldsymbol{U}}_k(t) - \widehat{\boldsymbol{U}}_k(t)\widehat{\boldsymbol{A}}_k + \widehat{\boldsymbol{U}}_k(t)\widehat{\boldsymbol{B}}_k^\ell\left(\widehat{\boldsymbol{B}}_k^\ell\right)^\top\widehat{\boldsymbol{U}}_k(t) - \left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top\widehat{\boldsymbol{C}}_k^\ell - \boldsymbol{R}_k^{(I)}(t) = 0.$$

Hence, as long as $\|\boldsymbol{R}_k^{(I)}(t)\|_F$ is not very small, the increase of k aims at more and more accurately approximating a "nearby" differential problem to the truly projected one, with a term $\boldsymbol{R}_k^{(I)}(t)$ that varies with k. Hence, $\boldsymbol{U}_k(t) = \mathcal{V}_k\widehat{\boldsymbol{U}}_k\mathcal{V}_k^\top$ is an approximation not to $\boldsymbol{U}(t)$, but to the solution of a differential problem with an additional term whose projection onto the space is $\boldsymbol{R}_k^{(I)}(t)$.

Proposition 3.1 also implies that we cannot expect an overall small residual norm if either of the two partial residual norms $\|\boldsymbol{R}_k^{(I)}(t)\|_F$, $\|\boldsymbol{R}_k^{(O)}(t)\|_F$ is not small. In particular, we observe that the two residuals can be made small independently. Therefore we propose the following practical strategy:

(i) Run the algorithm as presented, with a low-order cheap ODE inner solver (i.e., BDF(1, $n_\mathfrak{t}$) with $n_\mathfrak{t}$ relatively small; see Algorithm 2.2) and use $\boldsymbol{R}_k^{(O)}(t)$ in the stopping criterion;

(ii) Once completed step (i) after $\widehat{k}$ iterations, use the matrices $\widehat{\boldsymbol{A}}_{\widehat{k}}, \widehat{\boldsymbol{C}}_{\widehat{k}}^\ell, \widehat{\boldsymbol{B}}_{\widehat{k}}^\ell$ and $\widehat{\boldsymbol{U}}_{0,\widehat{k}}^\ell$ to refine the ODE inner solution by using a higher-order ODE solver for the

projected system.

The final matrix $\widehat{\boldsymbol{U}}_{\tilde{k}}(t)$ obtained in step (ii) will provide a more accurate solution matrix than what would have been obtained at the end of step (i). We emphasize that *any* ODE method for small and medium scale DREs could be used at steps (i) and (ii). Our choice of BDF(1, $n_{\mathfrak{t}}$) is due to its good trade-off between accuracy and computational effort; other approaches could be considered.

To complete the description of the stopping criterion, we recall that $\boldsymbol{R}_k^{(O)}(t)$ depends on t, so that we need to estimate the integral over the whole time interval by means of a quadrature formula, that is

$$\left\| \int_0^{t_f} \boldsymbol{R}_k^{(O)}(\gamma) d\gamma \right\|_F = \left\| \boldsymbol{\Gamma}_k^\top \int_0^{t_f} \widehat{\boldsymbol{U}}_k(\gamma) d\gamma \right\|_F \approx \left\| \boldsymbol{\Gamma}_k^\top \sum_{j=1}^{n_{\mathfrak{t}}} \frac{t_f}{n_{\mathfrak{t}}} \widehat{\boldsymbol{U}}_k(t_j) \right\|_F =: \rho_k. \tag{3.13}$$

where the interval T has been divided into $n_{\mathfrak{t}}$ intervals with nodes t_j.

The overall algorithm based on the rational Krylov subspace method (RKSM-DRE) is reported in Algorithm 3.1, whereas the extended Krylov (EKSM-DRE) subspace method is presented in Algorithm 3.2.

Algorithm 3.1 RKSM-DRE

Require: $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B}^\ell \in \mathbb{R}^{n\times n_b}$, $\boldsymbol{C}^\ell \in \mathbb{R}^{n_c\times n}$, $\boldsymbol{U}_0^\ell \in \mathbb{R}^{n\times n_o}$, tol, t_f, $n_{\mathfrak{t}}$, $\mathbf{s_0} = \{s_0^{(1)}, s_0^{(2)}\}$

(i) Perform reduced QR: $[(\boldsymbol{C}^\ell)^\top, \boldsymbol{U}_0^\ell] = \mathsf{V}_1 \boldsymbol{\Lambda}_1$

Set $\mathcal{V}_1 \equiv \mathsf{V}_1$

for $k = 2, 3 \ldots$

Compute the next shift and add it to $\mathbf{s_0}$

Compute the next *real* basis block V_k

Set $\mathcal{V}_k = [\mathcal{V}_{k-1}, \mathsf{V}_k]$

Update $\widehat{\boldsymbol{A}}_k = \mathcal{V}_k^\top \boldsymbol{A} \mathcal{V}_k$ and $\widehat{\boldsymbol{B}}_k^\ell = \mathcal{V}_k^\top \boldsymbol{B}^\ell$, $\widehat{\boldsymbol{U}}_{0,k}^\ell = \mathcal{V}_k^\top \boldsymbol{U}_0^\ell$ and $\widehat{\boldsymbol{C}}_k^\ell = \boldsymbol{C}^\ell \mathcal{V}_k$

Integrate Equation (3.8) from 0 to t_f using BDF$(1, n_{\mathfrak{t}})$

Compute ρ_k using (3.13) with $\boldsymbol{\Gamma}_k^\top$ as in (1.12)

if $\rho_k < tol$

go to (ii)

end if

end for

(ii) Refinement: solve (3.8) with a more accurate integrator

Compute $\widehat{\boldsymbol{U}}_k(\mathfrak{t}_j) = \widehat{\boldsymbol{U}}_k^\ell(\mathfrak{t}_j) \widehat{\boldsymbol{U}}_k^\ell(\mathfrak{t}_j)^\top$, $j = 1, \ldots, n_{\mathfrak{t}}$ using the truncated SVD

return $\mathcal{V}_k \in \mathbb{R}^{n\times k(n_c+n_o)}$ and $n_{\mathfrak{t}}$ factors $\widehat{\boldsymbol{U}}_k^\ell(\mathfrak{t}_j) \in \mathbb{R}^{k(n_c+n_o)\times r}$, $j = 1, \ldots, n_{\mathfrak{t}}$

3.3 Stability analysis and error bounds

This section provides a few results on the spectral and convergence properties of the obtained approximate solution. We first inspect some properties of the asymptotic

Algorithm 3.2 EKSM-DRE

Require: $\boldsymbol{A} \in \mathbb{R}^{n\times n}$, $\boldsymbol{B}^\ell \in \mathbb{R}^{n\times n_b}$, $\boldsymbol{C}^\ell \in \mathbb{R}^{n_c\times n}$, $\boldsymbol{U}_0^\ell \in \mathbb{R}^{n\times n_o}$, tol, t_f, $n_\mathfrak{t}$

(i) Perform reduced QR: $\left([(\boldsymbol{C}^\ell)^\top, \boldsymbol{U}_0^\ell],\ \boldsymbol{A}^{-1}[(\boldsymbol{C}^\ell)^\top, \boldsymbol{U}_0^\ell]\right) = \mathsf{V}_1\boldsymbol{\Lambda}_1$
Set $\mathcal{V}_1 \equiv \mathsf{V}_1$
for $k = 2, 3\ldots$
Compute the next basis block V_k
Set $\mathcal{V}_k = [\mathcal{V}_{k-1}, \mathsf{V}_k]$
Update $\widehat{\boldsymbol{A}}_k$ as in [140] and $\widehat{\boldsymbol{B}}_k^\ell = \mathcal{V}_k^\top \boldsymbol{B}^\ell$, $\widehat{\boldsymbol{U}}_{\ell,0} = \mathcal{V}_k^\top \boldsymbol{U}_0^\ell$ and $\boldsymbol{C}_k^\ell = \boldsymbol{C}^\ell \mathcal{V}_k$
Integrate (3.8) from 0 to t_f using BDF$(1, n_\mathfrak{t})$
Compute ρ_k using (3.13) where $\boldsymbol{\Gamma}_k^\top = \widehat{\mathsf{A}}_{m+1,m}\mathsf{E}_{2m}^\top$
if $\rho_k < tol$
go to (ii)
end if
end for
(ii) Refinement: solve (3.8) with a more accurate integrator
Compute $\widehat{\boldsymbol{U}}_k(\mathfrak{t}_j) = \widehat{\boldsymbol{U}}_k^\ell(\mathfrak{t}_j)\widehat{\boldsymbol{U}}_k^\ell(\mathfrak{t}_j)^\top$, $j = 1, \ldots, n_\mathfrak{t}$ using the truncated SVD
return $\mathcal{V}_k \in \mathbb{R}^{n\times 2k(n_c+n_o)}$ and $n_\mathfrak{t}$ factors $\widehat{\boldsymbol{U}}_k^\ell(\mathfrak{t}_j) \in \mathbb{R}^{2k(n_c+n_o)\times r}$, $j = 1, \ldots, n_\mathfrak{t}$.

matrix solution, which solves the ARE. Then we propose a bound for the error matrix in an appropriate functional norm.

Properties of the (steady state) ARE

Properties associated with the ARE – as the asymptotic solution to the DRE – are well known in linear-quadratic optimal control, see, e.g., [46, 105]. In particular, classical uniqueness and stabilization properties of the solution (see, e.g., [46, Lemma 12.7.2]), can directly be extended to the reduced DRE (3.8).

Corollary 3.1. *Let $(\widehat{\boldsymbol{A}}_k, \widehat{\boldsymbol{B}}_k^\ell, \widehat{\boldsymbol{C}}_k^\ell)$ be a stabilizable and detectable system. Let $\widehat{\boldsymbol{U}}_k(t)$ be the solution of Equation (3.8) at time t and let $\widehat{\boldsymbol{U}}_k^\infty = \lim_{t\to\infty} \widehat{\boldsymbol{U}}_k(t)$. Then $\widehat{\boldsymbol{U}}_k^\infty$ is the unique symmetric nonnegative definite solution and the only stabilizing solution to the (reduced) ARE*

$$0 = \widehat{\boldsymbol{A}}_k^\top \widehat{\boldsymbol{U}}_k^\infty + \widehat{\boldsymbol{U}}_k^\infty \widehat{\boldsymbol{A}}_k - \widehat{\boldsymbol{U}}_k^\infty \widehat{\boldsymbol{B}}_k^\ell \left(\widehat{\boldsymbol{B}}_k^\ell\right)^\top \widehat{\boldsymbol{U}}_k^\infty + \left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top \widehat{\boldsymbol{C}}_k^\ell. \tag{3.14}$$

Moreover, if the pair $(\widehat{\boldsymbol{C}}_k^\ell, \widehat{\boldsymbol{A}}_k)$ is observable, $\widehat{\boldsymbol{U}}_k^\infty$ is strictly positive definite.

We notice that the stabilizability and detectability properties of $(\widehat{\boldsymbol{A}}_k, \widehat{\boldsymbol{B}}_k^\ell, \widehat{\boldsymbol{C}}_k^\ell)$ are not necessarily implied by those on $(\boldsymbol{A}, \boldsymbol{B}^\ell, \boldsymbol{C}^\ell)$. Nevertheless, it is shown in [141] that if there exists a feedback matrix $\mathfrak{K}$, such that the linear dynamical system $\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}^\ell\mathfrak{K})x$ is dissipative, then the pair $(\widehat{\boldsymbol{A}}_k, \widehat{\boldsymbol{B}}_k^\ell)$ is stabilizable. A similar result can be formulated for the detectability of $(\widehat{\boldsymbol{C}}_k^\ell, \widehat{\boldsymbol{A}}_k)$, since by duality reasoning, $(\widehat{\boldsymbol{C}}_k^\ell, \widehat{\boldsymbol{A}}_k)$ is detectable if $\left(\widehat{\boldsymbol{A}}_k^\top, \left(\widehat{\boldsymbol{C}}_k^\ell\right)^\top\right)$ is stabilizable. The question regarding the existence of

such a feedback matrix, with respect to $\boldsymbol{A}$ and $\boldsymbol{B}^\ell$ ($\boldsymbol{A}^\top$ and $(\boldsymbol{C}^\ell)^\top$), is addressed in [74].

With these results, we can relate the asymptotic solution of the original and projected problems. Let $\boldsymbol{U}_k(t) = \mathcal{V}_k\widehat{\boldsymbol{U}}_k(t)\mathcal{V}_k^\top$ and $\boldsymbol{U}_k^a = \mathcal{V}_k\widehat{\boldsymbol{U}}_k^\infty\mathcal{V}_k^\top$ respectively be approximate solutions to (3.1) and (3.6) by a projection onto range($\mathcal{V}_k$). If there exist matrices $\mathfrak{K}_1$ and $\mathfrak{K}_2$ such that the systems $\dot{\boldsymbol{x}} = (\boldsymbol{A} - \boldsymbol{B}^\ell\mathfrak{K}_1)\boldsymbol{x}$ and $\dot{\boldsymbol{x}} = (\boldsymbol{A}^\top - (\boldsymbol{C}^\ell)^\top\mathfrak{K}_2)\boldsymbol{x}$ are dissipative, then

$$\lim_{t\to\infty}\boldsymbol{U}_k(t) = \mathcal{V}_k\lim_{t\to\infty}\widehat{\boldsymbol{U}}_k(t)\mathcal{V}_k^\top = \mathcal{V}_k\widehat{\boldsymbol{U}}_k^\infty\mathcal{V}_k^\top = \boldsymbol{U}_k^a, \tag{3.15}$$

that is, $\boldsymbol{U}_k^a$ is the steady state solution of $\boldsymbol{U}_k(t)$ when projected onto the same basis.

Under the hypotheses that $(\boldsymbol{A}, \boldsymbol{B}^\ell, \boldsymbol{C}^\ell)$ is a stabilizable and detectable system, there exists a unique non-negative and stabilizing solution $\boldsymbol{U}_\infty$ to Equation (3.6) (see, e.g., [103, Theorem 5]). In [141] a bound was derived for the error $\boldsymbol{U}_\infty - \boldsymbol{U}_k^a$ in terms of the matrix residual norm. Here we complete the argument by stating that in exact arithmetic and if the whole space can be spanned, the obtained approximate solution equals $\boldsymbol{U}_\infty$.

Proposition 3.2. *Suppose $(\boldsymbol{A}, \boldsymbol{B}^\ell, \boldsymbol{C}^\ell)$ is stabilizable and detectable. Assume it is possible to determine k_* such that dim(range($\mathcal{V}_{k_*}$)) $= n$, and let $\boldsymbol{U}_{k_*}^a = \mathcal{V}_{k_*}\widehat{\boldsymbol{U}}_{k_*}^\infty\mathcal{V}_{k_*}^\top$ be the obtained approximate solution of (3.6) after k_* iterations. Then, $\boldsymbol{U}_{k_*}^a = \boldsymbol{U}_\infty$.*

Proof. Since $\mathcal{V}_{k_*}$ is square and orthogonal the projected ARE is given by

$$0 = \mathcal{V}_{k_*}^\top\boldsymbol{A}^\top\mathcal{V}_{k_*}\widehat{\boldsymbol{U}}_{k_*}^\infty + \widehat{\boldsymbol{U}}_{k_*}^\infty\mathcal{V}_{k_*}^\top\boldsymbol{A}\mathcal{V}_{k_*} - \widehat{\boldsymbol{U}}_{k_*}^\infty\mathcal{V}_{k_*}^\top\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\mathcal{V}_{k_*}\widehat{\boldsymbol{U}}_{k_*}^\infty + \mathcal{V}_{k_*}^\top(\boldsymbol{C}^\ell)^\top\boldsymbol{C}^\ell\mathcal{V}_{k_*}.$$

From $(\boldsymbol{A}, \boldsymbol{B}^\ell, \boldsymbol{C}^\ell)$ stabilizable and detectable it follows that $(\mathcal{V}_{k_*}^\top\boldsymbol{A}\mathcal{V}_{k_*}, \mathcal{V}_{k_*}^\top\boldsymbol{B}^\ell, \boldsymbol{C}^\ell\mathcal{V}_{k_*})$ is also stabilizable and detectable, so that $\widehat{\boldsymbol{U}}_{k_*}^\infty \geq 0$ and stabilizing. Multiplying by $\mathcal{V}_{k_*}$ (by $\mathcal{V}_{k_*}^\top$) from the left (right), we obtain $0 = \boldsymbol{A}^\top\boldsymbol{U}_{k_*}^a + \boldsymbol{U}_{k_*}^a\boldsymbol{A} - \boldsymbol{U}_{k_*}^a\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\boldsymbol{U}_{k_*}^a + (\boldsymbol{C}^\ell)^\top\boldsymbol{C}^\ell$, that is, $\boldsymbol{U}_{k_*}^a \geq 0$ is a solution to the original ARE. Since $\boldsymbol{U}_\infty$ is the unique nonnegative definite solution, it must be $\boldsymbol{U}_{k_*}^a = \boldsymbol{U}_\infty$. □

Error bound for the DRE

In this section we derive a bound for the maximum error obtained by the reduction process, in terms of the residual

$$\boldsymbol{R}_k(t) = \boldsymbol{A}^\top\boldsymbol{U}_k(t) + \boldsymbol{U}_k(t)\boldsymbol{A} - \boldsymbol{U}_k(t)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\boldsymbol{U}_k(t) + (\boldsymbol{C}^\ell)^\top\boldsymbol{C}^\ell - \dot{\boldsymbol{U}}_k(t). \tag{3.16}$$

Note that $\boldsymbol{R}_k(t)$ is the residual matrix with respect to the exact solution of the reduced differential problem; that is, it also includes the discretization error. A similar bound on the error has been derived for the nonsymmetric DRE in [7], which used matrix perturbation techniques from [98].

Proposition 3.3. *For $t \in T$ let $\boldsymbol{E}_k(t) = \boldsymbol{U}(t) - \boldsymbol{U}_k(t)$ and assume that $\breve{\boldsymbol{A}}(t) := \boldsymbol{A} - \boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{U}(t)$ is stable for all $t \in T$. Denote*

$$\nu := \max_{t\in T} \left\{ \int_0^t \|\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}^\top}(t,s)\| \, \|\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}(t,s)\| ds \right\},$$

with $\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}$ the state-transition matrix satisfying

$$\frac{\partial \boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}(t,s)}{\partial t} = \breve{\boldsymbol{A}}(t)\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}(t,s), \quad \boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}(s,s) \;=\; \boldsymbol{I}_n.$$

If $4\nu^2\|\boldsymbol{B}^\ell\|^2\|\boldsymbol{R}_k\|_{\infty_t} < 1$, then

$$\|\boldsymbol{E}_k\|_{\infty_t} \leq 2\nu\|\boldsymbol{R}_k\|_{\infty_t},$$

where $\|\boldsymbol{\mathfrak{L}}\|_{\infty_t} = \max_{t\in T}\|\boldsymbol{\mathfrak{L}}(t)\|$ for any continuous matrix function $\boldsymbol{\mathfrak{L}}(t)$.

Proof. By subtracting Equation (3.16) from (3.1) and manipulating terms we obtain

$$\dot{\boldsymbol{E}}_k(t) = (\boldsymbol{A}-\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\boldsymbol{U}(t))^\top\boldsymbol{E}_k(t)+\boldsymbol{E}_k(t)(\boldsymbol{A}-\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\boldsymbol{U}(t))+\boldsymbol{E}_k(t)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\boldsymbol{E}_k(t)+\boldsymbol{R}_k(t),$$

with $\boldsymbol{E}_k(0) = 0$. Therefore, by the variation of constants formula (see, e.g., [103])

$$\boldsymbol{E}_k(t) = \int_0^t \boldsymbol{\Psi}_{\breve{\boldsymbol{A}}^\top}(t,s)\left(\boldsymbol{R}_k(s) + \boldsymbol{E}_k(s)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\boldsymbol{E}_k(s)\right)\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}(t,s)ds.$$

Taking norms yields

$$\|\boldsymbol{E}_k(t)\|_{\infty_t} \leq \max_{t\in T}\int_0^t \|\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}^\top}(t,s)\|\,\|\boldsymbol{\Psi}_{\breve{\boldsymbol{A}}}(t,s)\|\left(\|\boldsymbol{R}_k(s)\| + \|\boldsymbol{E}_k(s)\|^2\|\boldsymbol{B}^\ell\|^2\right)ds,$$

so that $\|\boldsymbol{E}_k(t)\|_{\infty_t} \leq \nu\left(\|\boldsymbol{R}_k(t)\|_{\infty_t} + \|\boldsymbol{E}_k(t)\|_{\infty_t}^2\|\boldsymbol{B}^\ell\|^2\right)$. Solving this quadratic inequality yields

$$\|\boldsymbol{E}_k(t)\|_{\infty_t} \leq \frac{1-\sqrt{1-4\nu^2\|\boldsymbol{B}^\ell\|^2\|\boldsymbol{R}_k\|_{\infty_t}}}{2\nu\|\boldsymbol{B}^\ell\|^2}.$$

The result follows from multiplying and dividing by $(1+\sqrt{1-4\nu^2\|\boldsymbol{B}^\ell\|^2\|\boldsymbol{R}_k\|_{\infty_t}})$ and noticing that at the denominator this quantity can be bounded from below by 1. □

We conclude with a remark on the intuitive fact that if the approximation space spans the whole space, the obtained solution by projection necessarily coincides with the sought-after solution of the DRE.

Remark 3.2. *If it is possible to determine k_* such that $dim(\mathcal{V}_{k_*}) = n$, then the approximate solution $\boldsymbol{U}_{k_*}(t)$ coincides with $\boldsymbol{U}(t)$ for all $t \geq 0$. Indeed, let us write $\boldsymbol{U}_{k_*}(t) = \mathcal{V}_{k_*}\widehat{\boldsymbol{U}}_{k_*}(t)\mathcal{V}_{k_*}^\top$, where $\mathcal{V}_{k_*}$ is square and orthogonal. The reduced DRE is given by*

$$\dot{\widehat{\boldsymbol{U}}}_{k_*} = \mathcal{V}_{k_*}^\top\boldsymbol{A}^\top\mathcal{V}_{k_*}\widehat{\boldsymbol{U}}_{k_*} + \widehat{\boldsymbol{U}}_{k_*}\mathcal{V}_{k_*}^\top\boldsymbol{A}\mathcal{V}_{k_*} - \widehat{\boldsymbol{U}}_{k_*}\mathcal{V}_{k_*}^\top\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top\mathcal{V}_{k_*}\widehat{\boldsymbol{U}}_{k_*} + \mathcal{V}_{k_*}^\top(\boldsymbol{C}^\ell)^\top\boldsymbol{C}^\ell\mathcal{V}_{k_*}$$

with $\widehat{\boldsymbol{U}}_{k_*} = \widehat{\boldsymbol{U}}_{k_*}(t)$. *Multiplying by* $\mathcal{V}_{k_*}$ *(by* $\mathcal{V}_{k_*}^\top$*) from the left (right), we obtain*

$$\dot{\boldsymbol{U}}_{k_*}(t) = \boldsymbol{A}^\top \boldsymbol{U}_{k_*}(t) + \boldsymbol{U}_{k_*}(t)\boldsymbol{A} - \boldsymbol{U}_{k_*}(t)\boldsymbol{B}^\ell(\boldsymbol{B}^\ell)^\top \boldsymbol{U}_{k_*}(t) + (\boldsymbol{C}^\ell)^\top \boldsymbol{C}^\ell,$$

hence, $\boldsymbol{U}_{k_*}(t) \geq 0$ *is a solution of* (3.1). *Since* $\boldsymbol{U}(t)$ *is the unique nonnegative definite solution of* (3.1) *for any* $\boldsymbol{U}_0 \geq 0$ *(see, e.g., [103]), then* $\boldsymbol{U}_{k_*}(t) = \boldsymbol{U}(t)$ *for* $t \geq 0$. □

3.4 Numerical experiments

In this section, we report on our numerical experience with the developed techniques. We consider two artificial symmetric and nonsymmetric model problems, as well as three (of which two are nonsymmetric) standard benchmark problems. Information about the considered data is contained in Table 3.1. For the first two datasets displayed in Table 3.1, the matrix $\boldsymbol{A}$ stems from the finite-difference discretization with homogenous Dirichlet boundary conditions on the unit square and unit cube, respectively. The first matrix (SYM2D) comes from the finite difference discretization of the two-dimensional Laplace operator in the unit square with homogeneous boundary conditions, while the second matrix (NSYM3D) stems from the finite difference discretization of the three-dimensional differential operator

$$\mathcal{L}(u) = e^{xy}(u_x)_x + e^{xy}(u_y)_y + (u_z)_z + (1+x)e^{-x}u_x + y^2 u_y + 10(x+y)u_z,$$

in the unit cube, with homogeneous boundary conditions. For both datasets, the matrices $\boldsymbol{B}^\ell, \boldsymbol{C}^\ell$ and $\boldsymbol{U}_0^\ell$ are selected randomly with normally distributed entries. The realizations of the random matrices are fixed for both examples using the MATLAB command `rng`: for $\boldsymbol{B}^\ell, \boldsymbol{C}^\ell$ and $\boldsymbol{U}_0^\ell$ we use `rng(7)`, `rng(2)` and `rng(3)`, respectively. The following two datasets (CHIP and FLOW) are taken from [121], and all coefficient matrices ($\check{\boldsymbol{A}}$, $\check{\boldsymbol{B}}^\ell$, $\check{\boldsymbol{C}}^\ell$ and $\check{\boldsymbol{M}}$) are contained in the datasets, which stem from the dynamical system

$$\check{\boldsymbol{M}}\dot{\check{\boldsymbol{x}}} = \check{\boldsymbol{A}}\check{\boldsymbol{x}} + \check{\boldsymbol{B}}^\ell \boldsymbol{k}, \qquad \check{\boldsymbol{y}} = \check{\boldsymbol{C}}^\ell \check{\boldsymbol{x}}.$$

Since $\check{\boldsymbol{M}}$ is diagonal and nonsingular, it is incorporated as $\boldsymbol{A} = \check{\boldsymbol{M}}^{-\frac{1}{2}} \check{\boldsymbol{A}}\, \check{\boldsymbol{M}}^{-\frac{1}{2}}$, while $\check{\boldsymbol{B}}^\ell$ and $\check{\boldsymbol{C}}^\ell$ are updated accordingly to form $\boldsymbol{B}^\ell$ and $\boldsymbol{C}^\ell$.

The final considered dataset (RAIL) stems from a semi-discretized heat transfer problem for optimal cooling of steel profiles[3] [24]. We consider the largest of the four available discretizations (file `rail_79841_c60` containing $\check{\boldsymbol{A}}, \check{\boldsymbol{B}}^\ell, \check{\boldsymbol{C}}^\ell$ and $\check{\boldsymbol{M}}$) with $n = 79841$. The symmetric and positive definite mass matrix $\check{\boldsymbol{M}}$ has a sparsity pattern very similar to $\check{\boldsymbol{A}}$. Both matrices are therefore reordered by the same approximate minimum degree (RKSM-DRE) or reverse Cuthill-McKee (EKSM-DRE) permutation to limit fill-in. The state-space transformation is done using the Cholesky factorization

[3] Data available at `http://modelreduction.org/index.php/Steel_Profile`

of $\check{\boldsymbol{M}}$. More precisely, let $\check{\boldsymbol{M}} = \check{\boldsymbol{L}}_M \check{\boldsymbol{L}}_M^\top$ with $\check{\boldsymbol{L}}_M$ lower triangular, and consider the transformed state $\boldsymbol{x} = \check{\boldsymbol{L}}_M^\top \check{\boldsymbol{x}}$. Then

$$\dot{\boldsymbol{x}} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}^\ell \boldsymbol{k}, \qquad \boldsymbol{y} = \boldsymbol{C}^\ell \boldsymbol{x},$$

with $\boldsymbol{A} = \check{\boldsymbol{L}}_M^{-1} \check{\boldsymbol{A}} \check{\boldsymbol{L}}_M^{-\top}$, $\boldsymbol{B}^\ell = \check{\boldsymbol{L}}_M^{-1} \check{\boldsymbol{B}}^\ell$ and $\boldsymbol{C}^\ell = \check{\boldsymbol{C}}^\ell \check{\boldsymbol{L}}_M^{-\top}$. These matrices are *never* explicitly formed, rather they are commonly applied implicitly by solves with the factor $\check{\boldsymbol{L}}_M$ at each iteration; see, e.g., [59, 140].

The initial low-rank factors are selected as the zero vector for FLOW, $\boldsymbol{U}_0^\ell = \sin \boldsymbol{a}$ for CHIP and $\boldsymbol{U}_0^\ell = \cos \boldsymbol{a}$ for RAIL, where $\boldsymbol{a} \in \mathbb{R}^n$ is a vector with entries in $[0, 2\pi]$. Other sufficiently general choices were tried during our numerical investigation however results did not significantly differ from the ones we report.

TABLE 3.1: Relevant information concerning the experimental data

Name	n	$p/s/q$	$\|\|\boldsymbol{A}\|\|_F$	$\|\|\boldsymbol{B}^\ell\|\|_F$	$\|\|\boldsymbol{C}^\ell\|\|_F$	$\|\|\boldsymbol{U}_0^\ell\|\|_F$	$\|\|\boldsymbol{M}\|\|_F$
SYM2D	640000	5/1/1	$3.6 \cdot 10^3$	$8.0 \cdot 10^2$	$1.8 \cdot 10^3$	$8.0 \cdot 10^2$	$8 \cdot 10^2$
NSYM3D	64000	6/1/3	$2.0 \cdot 10^3$	$2.5 \cdot 10^2$	$6.2 \cdot 10^2$	$2.8 \cdot 10^2$	$2.5 \cdot 10^2$
Name	n	$p/s/q$	$\|\|\check{\boldsymbol{A}}\|\|_F$	$\|\|\check{\boldsymbol{B}}^\ell\|\|_F$	$\|\|\check{\boldsymbol{C}}^\ell\|\|_F$	$\|\|\boldsymbol{U}_0^\ell\|\|_F$	$\|\|\check{\boldsymbol{M}}\|\|_F$
CHIP	20082	5/1/1	$2.2 \cdot 10^6$	$1.7 \cdot 10^2$	$3.3 \cdot 10^4$	$1.0 \cdot 10^2$	$2 \cdot 10^{-4}$
FLOW	9669	5/1/1	$4.5 \cdot 10^6$	$2.0 \cdot 10^4$	$1.2 \cdot 10^3$	–	$6.8 \cdot 10^0$
RAIL	79841	7/6/1	$7 \cdot 10^{-3}$	$1 \cdot 10^{-7}$	$6.2 \cdot 10^0$	$1.9 \cdot 10^2$	$8 \cdot 10^{-4}$

Performance of the projection methods. We first investigate the convergence behavior of the outer solver. The quantity we monitor in our stopping criterion is the backward error in an integral norm given by

$$\frac{\rho_k}{t_f \|\boldsymbol{C}^\ell\|_F^2 + 2\xi_k + \psi_k}, \tag{3.17}$$

with ρ_k as in (3.13) and

$$\xi_k = \left\| \boldsymbol{A}^\top \mathcal{V}_k \int_0^{t_f} \widehat{\boldsymbol{U}}_k(\gamma)\, d\gamma \right\|_F \quad \text{and} \quad \psi_k = \left\| \int_0^{t_f} \widehat{\boldsymbol{U}}_k(\gamma) \mathcal{V}_k^\top \boldsymbol{B}^\ell (\boldsymbol{B}^\ell)^\top \mathcal{V}_k \widehat{\boldsymbol{U}}_k(\gamma) d\gamma \right\|_F.$$

The integrals are approximated by a quadrature formula similar to (3.13), and we note that ξ_k can be cheaply computed by using the Arnoldi-type relation (1.8).

For all datasets, the stopping tolerance was chosen as 10^{-7}. For the first four datasets, $t_f = 1$ and BDF(1,10) is used as inner solver. For RAIL, $t_f = 4500$ (see e.g., [24] for further details about the setting) and BDF(1,45) is used as inner solver. Figures 3.1 to 3.5 display the convergence of the rational Krylov subspace method (Algorithm 3.1, RKSM-DRE) and of the extended Krylov subspace method (Algorithm 3.2, EKSM-DRE). The left plots report the history of the backward error as the approximation space dimension increases, while the right plots display the same history versus the total computational time (in seconds) as the iterations proceed. We notice that the cost of the refinement step is not taken into account in these first tests.

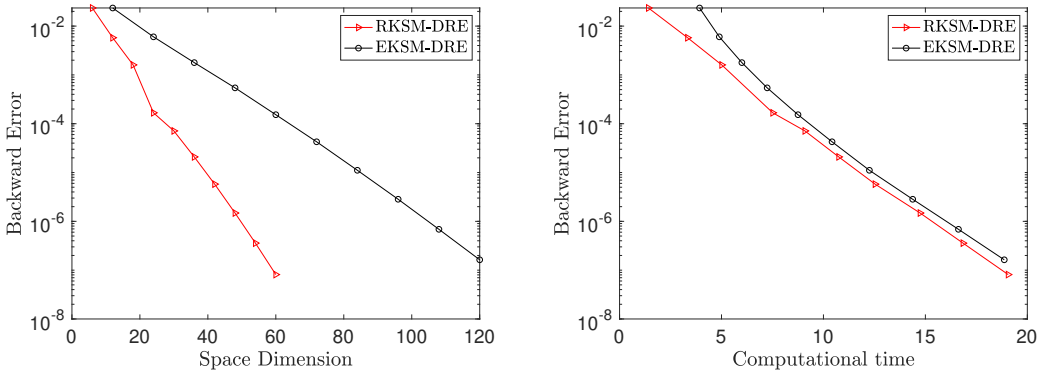

FIGURE 3.1: SYM2D: Convergence history for EKSM-DRE and RKSM-DRE. Left: backward error versus space dimension. Right: backward error versus computational time.

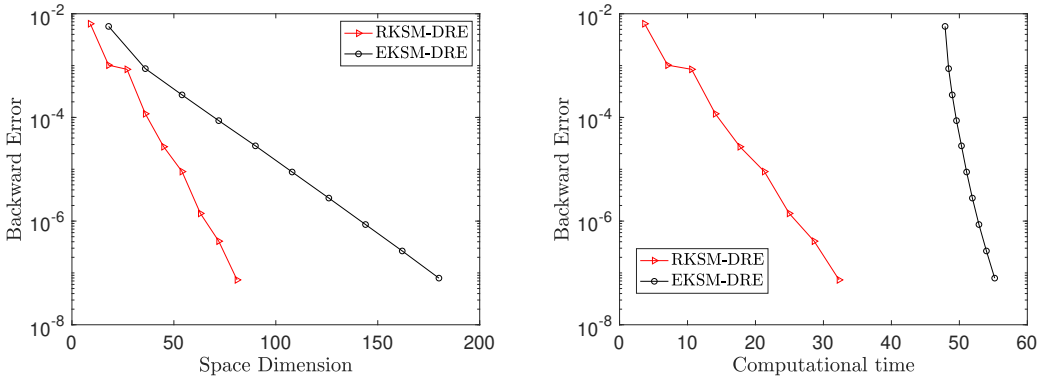

FIGURE 3.2: NSYM3D: Convergence history for EKSM-DRE and RKSM-DRE. Left: backward error versus space dimension. Right: backward error versus computational time.

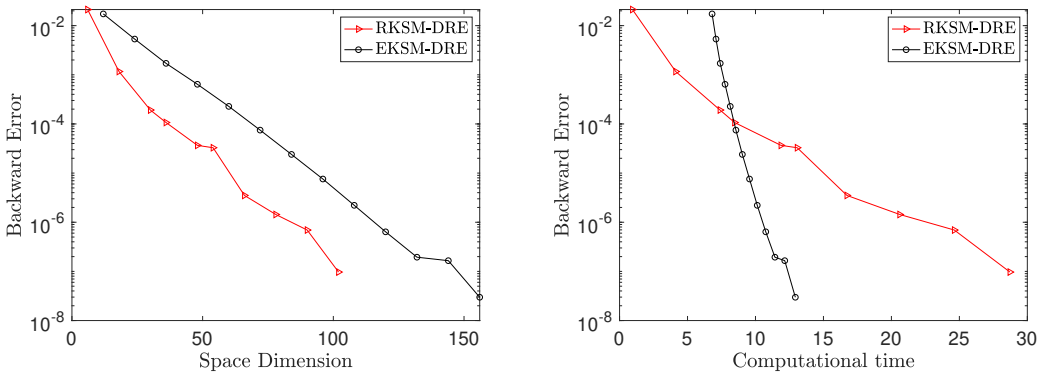

FIGURE 3.3: CHIP: Convergence history for EKSM-DRE and RKSM-DRE. Left: backward error versus space dimension. Right: backward error versus computational time.

For the dataset SYM2D, the large algebraic linear system in RKSM-DRE was iteratively solved by implementing a block conjugate gradient algorithm, with an inner tolerance of 10^{-10}, preconditioned with an incomplete Cholesky factorization with drop tolerance 10^{-4}. For all other datasets, the MATLAB built-in backslash operator was used.

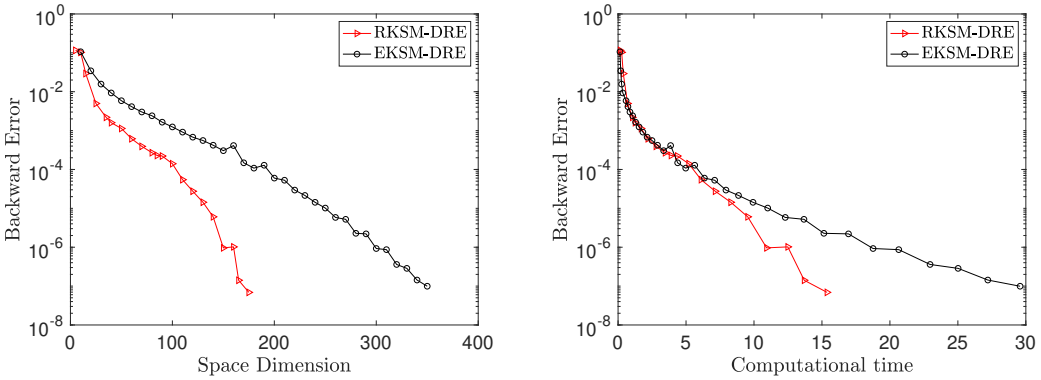

FIGURE 3.4: FLOW: Convergence history for EKSM-DRE and RKSM-DRE. Left: backward error versus space dimension. Right: backward error versus computational time.

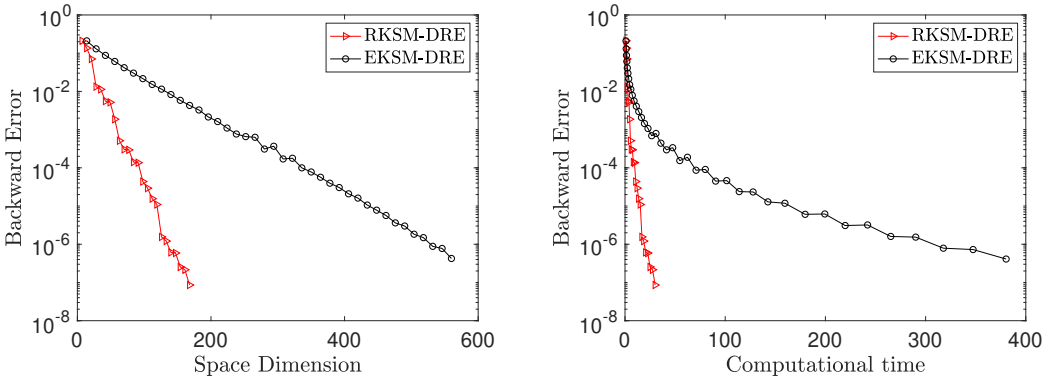

FIGURE 3.5: RAIL: Convergence history for EKSM-DRE and RKSM-DRE. Left: backward error versus space dimension. Right: backward error versus computational time.

For EKSM-DRE the coefficient matrix $\boldsymbol{A}$ used to generate the Krylov space remains constant; hence a sparse reordered Cholesky (for SYM2D and RAIL) or LU (for all other datasets) factorization was performed once for all at the start of the algorithm. Therefore, only sparse triangular solves are required at each iteration. Clearly, the cost of the initial factorization depends on the size and density of the coefficient matrix. These two cost stages are particularly noticeable in the right plots of Figure 3.2 and Figure 3.3, where the EKSM-DRE curve starts towards the right of the plot, while the rest of the computation throughout the iterations is significantly faster.

In the implementation of RKSM-DRE it is possible to decide a priori whether to use only real or generically complex shifts. Our experiments showed that complex shifts were unnecessary for SYM2D and NSYM3D and, in fact, slowed down convergence when used. On the other hand, the use of general complex shifts proved to be crucial for the efficient convergence of RKSM-DRE for CHIP and FLOW. For the symmetric data in RAIL no complex shifts were used. We mention in passing that both algorithms are implemented so that the inner solves of (3.8) and the residual computations are performed at each iteration; for more demanding data, we would advise a user to

perform these computations only periodically to save on computational time.

Comparing performance, we observe that the two algorithms have alternating leadership in terms of computational time, but that RKSM-DRE almost consistently requires half the space dimension of EKSM-DRE. This behavior is expected as the space dimension of EKSM-DRE increases with twice the number of columns per iteration, in comparison to RKSM-DRE. This observation is crucial at the refinement step, where it could be considerably more expensive to integrate a DRE of dimension $2k(n_c + n_o)$ accurately in comparison to a DRE with approximately half the dimension.

To have a clearer picture of how the various steps influence the performance of the methods, Table 3.2 depicts the overall computational time for the system solves, the orthogonalization steps, and the integration of the reduced systems for each algorithm. For EKSM-DRE the CPU time required for the Cholesky and *LU* factorizations are included in the solving time, but indicated in brackets as well. It is particularly interesting to notice the small percentage of time required by RKSM-DRE in comparison to EKSM-DRE for integrating the reduced system, confirming the comment made in the previous paragraph.

TABLE 3.2: A breakdown of the computational time for the considered methods for the first two datasets.

Data	Method	System solves (s)	Orthogonalisation steps (s)	Integration steps (s)
SYM2D	RKSM-DRE	6.1	6.9	0.4
	EKSM-DRE	8.6 (2.7)	12.1	1.3
NSYM3D	RKSM-DRE	38.3	0.9	0.8
	EKSM-DRE	48.6 (43.5)	1.6	4.0

Comparisons with other BDF based methods. We compare the two projection methods RKSM-DRE and EKSM-DRE with low-rank methods that have been developed following different strategies. The package M.E.S.S. [134], for instance, can solve Lyapunov and Riccati equations, and perform model reduction of systems in state space and structured differential-algebraic form, with time-variant and time-invariant data. For our purposes, the solvers in M.E.S.S. first discretize the time interval and then solve the ARE resulting from the ODE solver at each time step. Therefore, the approximation strategy employed at each time iteration to solve the algebraic problem is entirely independent, and the obtained low-rank numerical solution needs to be stored separately. More precisely, if n_t timesteps are performed, the procedure requires solving at least n_t AREs of large dimensions, delivering the corresponding low-rank approximate solutions. Moreover, the rank of the constant term in the ARE increases with the time step due to how the ODE solver is structured, further increasing the complexity of the ARE numerical treatment. In our experiments with M.E.S.S., we only requested

the approximate solution at the final stage. If the whole approximate solution matrix is requested at different times, the memory requirements will grow linearly. The overall strategy appears to be memory and computational time consuming; therefore, we considered datasets of reduced size for our comparisons, as displayed in Table 3.3. The considered timespans were left unchanged.

TABLE 3.3: Data information for comparisons between projection-based methods and M.E.S.S.

Name	n	$p/s/q$	$\|\|\boldsymbol{A}\|\|_F$	$\|\|\boldsymbol{B}^\ell\|\|_F$	$\|\|\boldsymbol{C}^\ell\|\|_F$	$\|\|\boldsymbol{U}_0^\ell\|\|_F$	$\|\|\boldsymbol{M}\|\|_F$
SYM2D	40000	5/1/1	$1.3 \cdot 10^3$	$3.0 \cdot 10^2$	$6.7 \cdot 10^2$	$3.0 \cdot 10^2$	$2 \cdot 10^2$
NSYM3D	8000	6/1/3	$6.1 \cdot 10^2$	$7.7 \cdot 10^1$	$1.9 \cdot 10^2$	$8.3 \cdot 10^1$	$2.8 \cdot 10^2$
Name	n	$p/s/q$	$\|\|\check{\boldsymbol{A}}\|\|_F$	$\|\|\check{\boldsymbol{B}}^\ell\|\|_F$	$\|\|\check{\boldsymbol{C}}^\ell\|\|_F$	$\|\|\check{\boldsymbol{U}}_0^\ell\|\|_F$	$\|\|\check{\boldsymbol{M}}\|\|_F$
FLOW	9669	5/1/1	$4.5 \cdot 10^6$	$2.0 \cdot 10^4$	$1.2 \cdot 10^3$	0	$6.8 \cdot 10^0$
RAIL	20209	7/6/1	$4 \cdot 10^{-3}$	$2.1 \cdot 10^{-7}$	$6.2 \cdot 10^0$	$1.9 \cdot 10^2$	$2 \cdot 10^{-4}$

Our experimental results are displayed in Tables 3.4 to 3.7; we remark that now also the refinement cost is taken into account in the projection methods. In all tables, the code BDF(b, n_t) refers to the BDF method implemented in the refinement procedure of the reduction methods and in the time discretization procedure of M.E.S.S.

TABLE 3.4: SYM2D: Storage and computational time comparison of RKSM-DRE, EKSM-DRE and M.E.S.S.. Reduction phase performed with BDF(1,10), refinement phase with BDF(2,100). In M.E.S.S. only the approximate solution at the final time is stored, with no solutions at intermediate time instances returned.

Method	# n-long Vecs	Min/Max rank	Reduction phase(s)	Refine phase(s)	Tot CPU time(s)
RKSM-DRE	54	23/43	1.4	0.15	1.6
EKSM-DRE	120	23/43	1.7	1.9	3.6
M.E.S.S.-BDF(1,10)	988	58/75			319.9
M.E.S.S.-BDF(2,100)	1032	58/86			4005.4

The tables show the storage requirements in terms of n-length vectors, the minimum and maximum approximate solution rank (with a truncation tolerance 10^{-8} for the projection methods) within the set of solutions, the CPU time break out of projection and refinement phases for the two projected methods, and finally, the total CPU time. The stopping tolerance for all algebraic methods – that is, the two projection methods and the Newton–Kleinmann-type method used in M.E.S.S. to solve each ARE – is set to 10^{-7}.

In the M.E.S.S. software, the user can either select a stopping tolerance (to be used for all solvers within the Newton–Kleinmann strategy) or a maximum number of iterations. We have experimented with both cases, where the maximum number of iterations was detected (a-posteriori) as the maximum number of iterations required within M.E.S.S to reach the tolerance of 10^{-7}. Furthermore, in most cases, it was

observed that avoiding the residual computation may slow down the computational procedure. This is due to the possibility of performing several unnecessary iterations at some timesteps after the desired accuracy has, in fact, been reached. We, therefore, only report the results of the more realistic, reliable case where a stopping tolerance is selected beforehand. Galerkin acceleration is used to boost the performance of Newton–Kleinmann.

TABLE 3.5: NSYM3D: Storage and computational time comparison of RKSM-DRE, EKSM-DRE and M.E.S.S.. Reduction phase performed with BDF(1,10), refinement phase with BDF(2,100). In M.E.S.S. only the approximate solution at the final time is stored, with no solutions at intermediate time instances returned.

Method	# n-long Vecs	Min/Max rank	Reduction phase(s)	Refine phase(s)	Tot CPU time(s)
RKSM-DRE	90	36/66	2.4	2.8	5.2
EKSM-DRE	180	36/66	2.6	5.4	8.0
M.E.S.S.-BDF(1,10)	1116	71/90			431.0
M.E.S.S.-BDF(2,100)	1152	67/94			4965.0

All numbers in the tables illustrate the large computational costs of M.E.S.S., as expected by the strategy "first time-discretize, then solve", whereas both projection methods require just a few seconds of CPU in most cases.

The storage requirements of both reduction methods are independent of the number of timesteps where the solution is required. This is because only a few n-long basis vectors need to be generated and stored, while only the reduced problem solution $\widehat{\boldsymbol{U}}_k(t)$ changes at the timesteps t. The memory requirements of M.E.S.S. are measured as the dimensions of the low-rank factor returned by the Newton-Kleinmann procedure, before column compression, at the final timestep. The dimension decreases significantly with the column compression. In our experiments, we only stored the approximate solution at the last time step; however, memory will be correspondingly higher if the whole approximation matrix is required at more instances (memory will thus grow linearly with the number of time instances to be monitored).

We observe that the extended space yields a significantly larger basis than the approximate solution rank it produces. This means that the approximate solution belongs to a much smaller space than the one constructed by EKSM-DRE. This is far less so with RKSM-DRE. The different behavior confirms what has been already observed for the two projection methods in the ARE case [144].

TABLE 3.6: FLOW: Storage and computational time comparison of RKSM-DRE, EKSM-DRE and M.E.S.S.. Reduction phase performed with BDF(1,10), refinement phase with BDF(2,100). In M.E.S.S. only the approximate solution at the final time is stored, with no solutions at intermediate time instances returned.

Method	# n-long Vecs	Min/Max rank	Reduction phase(s)	Refine phase(s)	Tot CPU time(s)
RKSM-DRE	175	95/100	11.8	4.5	16.3
EKSM-DRE	350	95/100	27.4	23.5	50.9
M.E.S.S.-BDF(1,10)	1280	87/106			431.7

TABLE 3.7: RAIL: Storage and computational time comparison of RKSM-DRE, EKSM-DRE and M.E.S.S.. Reduction phase performed with BDF(1,10), refinement phase with BDF(2,100). In M.E.S.S. only the approximate solution at the final time is stored, with no solutions at intermediate time instances returned.

Method	# n-long Vecs	Min/Max rank	Reduction phase(s)	Refine phase(s)	Tot CPU time(s)
RKSM-DRE	168	153/160	6.4	3.3	9.7
EKSM-DRE	462	153/160	39.2	5.7	44.9
M.E.S.S.-BDF(1,10)	6345	151/158			705.3
M.E.S.S.-BDF(2,100)	4023	124/158			3396.5

Comparisons with splitting methods. We next compare RKSM-DRE with the fourth-order additive splitting method (SPLIT-ADD4(n_t)) developed in [150]; see also Section 2.2.2 for a discussion on lower-order splitting methods. The main computational effort in the splitting methods is due to the repeated evaluation of matrix exponentials, which has been resolved by using a Krylov-based matrix exponential approximation. Similar to the issue discussed with M.E.S.S. in the previous section, the n_t (factored) solution matrices are independently calculated at each timestep, leading to significant memory requirements.

To ensure that we are comparing methods with similar approximation accuracies, we generate reference solutions $\boldsymbol{U}_{\text{ref}}(\mathfrak{t}_j)$ for the selected time instances $\mathfrak{t}_j$. This is done by using RKSM-DRE with a stopping tolerance of 10^{-10}, plus a refinement process with BDF$(4, 10^4)$ from [134]. To allow for such accurate approximations, we consider slightly smaller problem dimensions for the first two datasets, and we set $p = s = 1$ and $\boldsymbol{U}_0 = \boldsymbol{0}_n$.

The input parameters are tailored so that the approximate solutions from different methods have relatable accuracies. In particular, RKSM-DRE is solved with an outer stopping tolerance of 10^{-6} and with BDF(3,1000) in the refinement process. The number of timesteps utilized in SPLIT-ADD4 is selected as $n_t = 500$. The expected approximation errors relative to the reference solution, measured as

$$\frac{\|\boldsymbol{U}_{\text{approx}}(t) - \boldsymbol{U}_{\text{ref}}(t)\|_F}{\|\boldsymbol{U}_{\text{ref}}(t)\|_F},$$

are illustrated in Figure 3.6 (dataset SYM2D in the left plot, dataset NSYM3D in the right plot). The figures indicate that we compare methods having approximation

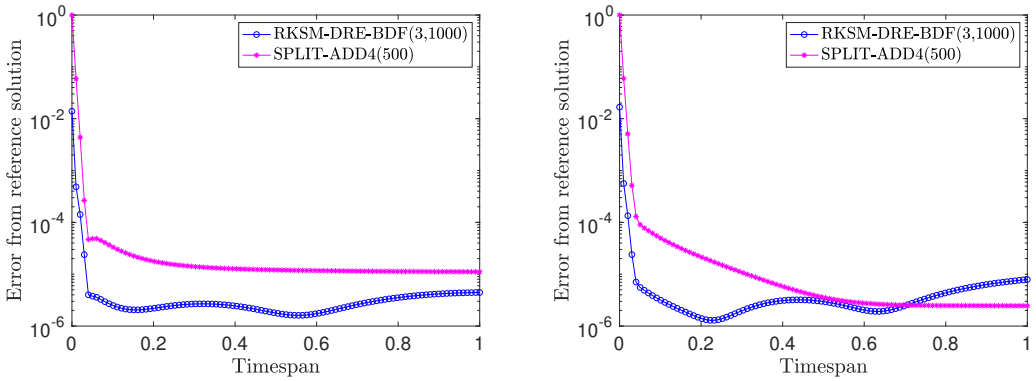

FIGURE 3.6: Expected approximation error for RKSM-DRE and SPLIT-ADD4(500). Left: Dataset SYM2D. Right: Dataset NSYM3D.

errors of a similar order. The performance results are contained in Table 3.8 for two different discretizations of SYM2D and NSYM3D.

TABLE 3.8: Storage and computational time comparison of RKSM-DRE and SPLIT-ADD4(500). Reduction phase performed with BDF(1,10), refinement phase with BDF(3,1000). In SPLIT-ADD4 only the approximate solution at the final time is stored.

Data (n)	Method	# n-long Vecs	Min/Max rank	Tot CPU time (s)
SYM2D (10^4)	RKSM-DRE	8	3/6	0.6
	SPLIT-ADD4(500)	28	3/7	34.9
NSYM3D ($8 \cdot 10^3$)	RKSM-DRE	10	4/7	2.2
	SPLIT-ADD4(500)	36	3/9	37.9
SYM2D ($9 \cdot 10^4$)	RKSM-DRE	6	3/4	1.2
	SPLIT-ADD4(500)	28	3/7	330.0
NSYM3D ($2.7 \cdot 10^4$)	RKSM-DRE	10	4/8	10.1
	SPLIT-ADD4(500)	36	3/9	127.8

All numbers indicate the competitiveness of RKSM-DRE in terms of storage and computational time. The memory requirements for SPLIT-ADD4 is measured as the dimension of the solution factor at the final timestep, before column compression. If the solution is required at more time instances, then these memory requirements will increase accordingly.

We also mention that we have experimented with the dynamic splitting methods introduced in [115], however, the algorithms proposed by the authors[4] in [115] appeared to be better suited for small to medium size problems.

[4]We thank Chiara Piazzola for providing us with her Matlab implementation of the method.

TABLE 3.9: SYM2D (of size 10^6): Results with RKSM-DRE, using different refinement strategies. Reduction phase performed with BDF(1,10) and tolerance 10^{-8}.

Refinement Method	# n-long Vecs	Soln. rank	Reduction phase(s)	Refinement phase(s)	Tot CPU time(s)
BDF(2,100)	72	55	37.7	1.1	38.8
BDF(3,1000)	72	55	37.7	9.6	47.3
BDF(4,10000)	72	55	37.7	95.5	133.2
SPLIT-ADD4(500)	72	55	37.7	1.9	39.6
SPLIT-ADD8(500)	72	55	37.7	5.1	42.8
SPLIT-ADAPT8	72	55	37.7	23.1	60.8

Discussion on the refinement step. In previous sections, we have stressed that the two approximation stages of the projection method are independent, and we have focused on determining an adequate approximation space. Here we linger over the accuracy of the second stage, the refinement step. By exploiting the far smaller problem size of the reduced problem, it is possible to allow for a much more accurate integration phase than what was done during the iteration of the reduction step. This crucial fact is already illustrated in the time breakdown of Tables 3.4 to 3.6, where especially for RKSM-DRE the refinement phase employs a fraction of the overall computational time while still allowing for a somewhat accurate final solution.

We next explore in more detail these advantages with RKSM-DRE on SYM2D, where the discretization was further refined to get a coefficient matrix of dimension 10^6. The dimensions of the other corresponding matrices remain as presented in Table 3.1. We investigate the time taken by DRE solvers with different accuracies to emphasize the advantages and flexibility of the refinement procedure. Table 3.9 reports the timings for a refinement step performed by three different BDF methods and three splitting methods. The 8th order adaptive splitting method (SPLIT-ADAPT8) also comes from [150] and is performed with a tolerance of 10^{-7}. We emphasize that in the refinement phase, we have utilized some of the most accurate integrators available, and nevertheless, the high-dimensional ($n = 10^6$) problem is approximated in less than 150 seconds for all integrators.

3.5 Concluding remarks

In this chapter, we have devised a rational Krylov subspace-based order reduction method for solving the symmetric DRE, providing a low-rank approximate solution matrix at selected time steps. A single projection space is generated for all time instances, and the space is expanded until the solution is sufficiently accurate. We stress that our approach is very general and that it could be applied to subspaces other than Krylov-based ones, as long as the spaces are nested, so that they keep growing as

the iterations proceed. This methodology could then be employed for more complex settings, such as parameter dependent problems, where the involved approximation space may require the inclusion of some parameter sampling.

Like in typical model order reduction strategies, in our methodology time stepping is only performed at the reduced level so that the integration cost is drastically lower than what one would have by applying the time stepping on the original large dimensional problem. We have derived a new stopping criterion that takes into account the different approximation behavior of the algebraic and differential portions of the problem, together with a refinement procedure that can improve the final approximate solution by using a high-order integrator. These enhancement strategies have also been applied to the extended Krylov subspace approach. We have analyzed the asymptotic behavior of the reduced-order solution to ensure that the generated approximation behaves like the sought-after time-dependent solution.

Although our numerical results are promising, there are still several open issues associated with the reduced-order solution of the DRE. In particular, while stability and other matrix properties associated with the solutions $\boldsymbol{U}(t)$ have been thoroughly studied [31, 56, 69, 126], the analysis of corresponding properties for the approximate solution $\boldsymbol{U}_k(t) = \mathcal{V}_k \widehat{\boldsymbol{U}}_k(t) \mathcal{V}_k^\top$ for $t \in T$ is still a largely open problem. In [99] some interesting monotonicity properties have been shown when the polynomial Krylov subspace is used together with particular ODE solvers; a complete analysis for $\boldsymbol{U}_k(t)$ in a more general setting would be desirable.

Chapter 4

Semilinear ODEs with General Nonlinear Term

This chapter focuses on the problem where the nonlinear function is no longer quadratic but rather more general. As demonstrated in the previous chapter, the quadratic structure of the DRE is beneficial in the sense that Krylov subspace methods can be used to generate the approximation space and that the nonlinear term can be explicitly projected onto the reduced space. However, these two issues are no longer trivial in the present case and need to be carefully investigated.

To this end, we are interested in numerically approximating the solution $\boldsymbol{U}(t) \in S_{\text{MAT}}$ to the semilinear matrix differential equation

$$\dot{\boldsymbol{U}}(t) = \boldsymbol{A}_1\boldsymbol{U}(t) + \boldsymbol{U}(t)\boldsymbol{A}_2 + \mathcal{F}(\boldsymbol{U}, t) + \boldsymbol{C}, \quad \boldsymbol{U}(0) = \boldsymbol{U}_0, \tag{4.1}$$

with similar dimensions and constraints as for (2.1), which arises for instance in the discretization of two-dimensional partial differential equations of the form

$$u_t = \mathcal{L}(u) + f(u, t) + c(x, y), \quad u = u(x, y, t) \quad \text{with } (x, y) \in \Omega \subset \mathbb{R}^2,\, t \in T, \tag{4.2}$$

and given initial condition $u(x, y, 0) = u_0(x, y)$, for certain choices of the physical domain Ω, as has been demonstrated in Section 1.2.

We aim to approximate the solution of (4.1) by constructing a reduced order model with the same structure as (4.1) but of much smaller dimension. In this setting, one challenge is to construct a pair of left-right approximation spaces that capture the behavior of the nonlinear function well. Furthermore, by definition, general nonlinear terms can not be explicitly projected onto the low-dimensional space and must first be evaluated in full dimension before projection. Consequently, it is crucial for the success of the reduction procedure that the nonlinear term is also approximated for efficiency.

As discussed in Section 1.2, a large collection of procedures for (4.1) employ a vector-oriented approach. That is, a vector $\boldsymbol{u}(t) \in \mathbb{R}^N$ contains the representation coefficients

of the sought after solution to (4.2) in the discrete space, where $\boldsymbol{u}(t)$ is the solution to the following system of ODEs:

$$\dot{\boldsymbol{u}}(t) = \boldsymbol{L}\boldsymbol{u}(t) + \boldsymbol{f}(\boldsymbol{u}, t) + \boldsymbol{c}, \quad \boldsymbol{u}(0) = \boldsymbol{u}_0. \tag{4.3}$$

See Section 1.2 for further details regarding the origin of (4.3). To reduce the dimension and complexity of the problem (4.3) several alternatives can be considered, as already discussed in the introductory part of the thesis. Nevertheless, in this thesis, we are particularly interested in POD-DEIM type order reduction strategies for semilinear ordinary differential equations [41]. Therefore, in Section 4.1 we review the standard POD-DEIM method as it is applied to (4.3). Then, in Section 4.2, we develop a matrix-oriented POD-DEIM order reduction strategy for the problem (4.1) that leads to a semilinear *matrix* differential equation with the same structure as (4.1), but of significantly reduced dimension. This construction is realized with the generation of a pair of left and right space bases via a novel Two-Sided POD (2S-POD). Furthermore, the reduction of the nonlinear term is also performed employing a fully matricial interpolation using left and right projections onto two distinct reduction spaces, giving rise to a new two-sided version of DEIM. As a result, a fast (offline) reduction phase where a significant decrease in the problem size is carried out, is followed by a light (online) phase where the reduced matrix ODE is integrated over time with the preferred *matrix-oriented* method. In particular, we will illustrate that the structure of the reduced

4.1 Review of POD-DEIM

Consider the system of ODEs (4.3). In Section 1.3.2 we have illustrated how a POD basis can be constructed from a given set of snapshots. Furthermore, we have illustrated in Section 1.3.3 how the DEIM can be used to interpolate a nonlinear function to avoid evaluating the function in full dimension.

To this end, let $\boldsymbol{u}_j = \boldsymbol{u}(t_j)$ and consider a set of snapshots $\{\boldsymbol{u}_j\}_{j=1}^{n_s}$ of the solution of (4.3). Then the orthoormal columns of $\boldsymbol{V}_{\text{VEC}} \in \mathbb{R}^{N\times k}$, $k \ll N$, form a POD basis of the set of snapshots $\{\boldsymbol{u}_j\}_{j=1}^{n_s}$, as discussed in Section 1.3.2. As a result, it is possible to approximate, for $t \in [0, t_f] = T$,the vector $\boldsymbol{u}(t) \in \mathbb{R}^N$ as $\boldsymbol{u}(t) \approx \boldsymbol{V}_{\text{VEC}}\widehat{\boldsymbol{u}}(t)$, where the vector $\widehat{\boldsymbol{u}}(t) \in \mathbb{R}^k$ solves the *reduced* problem

$$\dot{\widehat{\boldsymbol{u}}}(t) = \widehat{\boldsymbol{L}}\widehat{\boldsymbol{u}}(t) + \widehat{\boldsymbol{f}}(\widehat{\boldsymbol{u}}, t) + \widehat{\boldsymbol{c}}, \quad \widehat{\boldsymbol{u}}(0) = \boldsymbol{V}_{\text{VEC}}^\top \boldsymbol{u}_0. \tag{4.4}$$

Here $\widehat{\boldsymbol{L}} = \boldsymbol{V}_{\text{VEC}}^\top \boldsymbol{L}\boldsymbol{V}_{\text{VEC}}$, $\widehat{\boldsymbol{f}}(\widehat{\boldsymbol{u}}, t) = \boldsymbol{V}_{\text{VEC}}^\top \boldsymbol{f}(\boldsymbol{V}_{\text{VEC}}\widehat{\boldsymbol{u}}, t)$ and $\widehat{\boldsymbol{c}} = \boldsymbol{V}_{\text{VEC}}^\top \boldsymbol{c}$.

Although for $k \ll N$ problem (4.4) is cheaper to solve than the original one, the definition of $\widehat{\boldsymbol{f}}$ above requires the evaluation of $\boldsymbol{f}(\boldsymbol{V}_{\text{VEC}}\widehat{\boldsymbol{u}}, t)$ at each timestep and at all N entries, thus still depending on the original system size. This evaluation needs be

performed at each timestep and consequently creates a major bottleneck in solving the reduced order model. For a more detailed discussion regarding this complexity bottleneck see [41, Section 2.2]. One way to overcome this problem is by means of DEIM, as discussed in Section 1.3.3. As a result, the nonlinear term in the reduced model (4.4) is then approximated by

$$\widehat{\boldsymbol{f}}(\widehat{\boldsymbol{u}},t) \approx \boldsymbol{V}_{\text{VEC}}^{\top}\boldsymbol{\Phi}_{\text{VEC}}(\boldsymbol{P}_{\text{VEC}}^{\top}\boldsymbol{\Phi}_{\text{VEC}})^{-1}\boldsymbol{P}_{\text{VEC}}^{\top}\boldsymbol{f}(\boldsymbol{V}_{\text{VEC}}\widehat{\boldsymbol{u}},t). \tag{4.5}$$

DEIM is particularly advantageous when the function $\boldsymbol{f}$ is evaluated componentwise, in which case it holds that $\boldsymbol{P}_{\text{VEC}}^{\top}\boldsymbol{f}(\boldsymbol{V}_{\text{VEC}}\widehat{\boldsymbol{u}},t) = \boldsymbol{f}(\boldsymbol{P}^{\top}\boldsymbol{V}_{\text{VEC}}\widehat{\boldsymbol{u}},t)$, so that the nonlinear function is evaluated only at $p \ll N$ components. In general, this occurs whenever finite differences are the underlying discretization method. If more versatile discretizations are required, then different procedures need to be devised. A discussion of these approaches is deferred to Section 4.2.2.

4.2 A matrix-oriented POD-DEIM algorithm

In this section, we introduce our novel matrix-oriented POD-DEIM algorithm for ODEs of the form (4.1), that fully lives in the matrix setting, without requiring a mapping from $\mathbb{R}^{n_1\times n_2}$ to $\mathbb{R}^N$, so that no vectors of length $N = n_1n_2$ need to be processed or stored.

More precisely, we determine an approximation to $\boldsymbol{U}(t)$ of the type

$$\boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^{\top}, \quad t \in T \tag{4.6}$$

where $\boldsymbol{V}_1 \in \mathbb{R}^{n_1\times k_1}$ and $\boldsymbol{V}_2 \in \mathbb{R}^{n_2\times k_2}$ are matrices to be determined, independent of time. Here $k_1, k_2 \ll n_1, n_2$. The function $\widehat{\boldsymbol{U}}(t)$ is determined as the numerical solution to the following *reduced* semilinear matrix differential problem

$$\begin{aligned}\dot{\widehat{\boldsymbol{U}}}(t) &= \widehat{\boldsymbol{A}}_1\widehat{\boldsymbol{U}}(t) + \widehat{\boldsymbol{U}}(t)\widehat{\boldsymbol{A}}_2 + \widehat{\mathcal{F}(\widehat{\boldsymbol{U}},t)} + \widehat{\boldsymbol{C}}\\ \widehat{\boldsymbol{U}}(0) &= \widehat{\boldsymbol{U}}^{(0)} := \boldsymbol{V}_1^{\top}\boldsymbol{U}_0\boldsymbol{V}_2,\end{aligned} \tag{4.7}$$

with $\widehat{\boldsymbol{A}}_1 = \boldsymbol{V}_1^{\top}\boldsymbol{A}_1\boldsymbol{V}_1$, $\widehat{\boldsymbol{A}}_2 = \boldsymbol{V}_2^{\top}\boldsymbol{A}_2\boldsymbol{V}_2$, $\widehat{\boldsymbol{C}} = \boldsymbol{V}_1^{\top}\boldsymbol{C}\boldsymbol{V}_2$, and $\widehat{\mathcal{F}(\widehat{\boldsymbol{U}},t)}$ is a matrix-oriented DEIM approximation to

$$\widehat{\mathcal{F}}(\widehat{\boldsymbol{U}},t) = \boldsymbol{V}_1^{\top}\mathcal{F}(\boldsymbol{V}_1\widehat{\boldsymbol{U}}\boldsymbol{V}_2^{\top},t)\boldsymbol{V}_2. \tag{4.8}$$

We also discuss several other implementation issues; in particular a dynamic algorithm is developed to minimize the number of high-dimensional snapshots contributing to the approximation space.

4.2.1 A new two-sided POD

Here we discuss the first crucial step towards the approximation (4.6). That is, we introduce a novel algebraic formulation that performs a POD at the matrix level, taking account of the spatial properties of the functions to be approximated, and of the function changes when time snapshots are captured. The algorithm is tailored towards dealing with sets of matrices, that is, sets of function in two variables, rather than vectors in one long variable.

We determine the left and right reduced space bases that approximate the space of the given snapshots $\boldsymbol{\Xi}(t)$ (either the nonlinear functions or the approximation solutions), so that $\boldsymbol{\Xi}(t) \approx \boldsymbol{V}_1 \widehat{\boldsymbol{\Xi}}(t) \boldsymbol{V}_2^\top$, where $\boldsymbol{V}_1, \boldsymbol{V}_2$ have ν_1 and ν_2 orthonormal columns, respectively. Their ranges approximate the span of the rows (left) and columns (right) spaces of the function $\boldsymbol{\Xi}(t)$, independently of the time t. In practice, we wish to have $\nu_1 \ll n_1, \nu_2 \ll n_2$ so that $\widehat{\boldsymbol{\Xi}}(t)$ will have a reduced dimension. A simple way to proceed would be to collect all snapshot matrices in a single large (or tall) matrix and generate the two orthonormal bases corresponding to the rows and columns spaces. Unfortunately, this is way too expensive, both in terms of computational costs and memory requirements. Instead, we present a two-step procedure that avoids explicit computations with all snapshots simultaneously. The first step sequentially selects the most important information of each snapshot matrix relative to the other snapshots, while the second step prunes the generated spaces by building the corresponding orthonormal bases. These two steps can be summarized as follows:

1. *Selective collection.* Assume i snapshots have been processed and dominant SVD information retained. For the next snapshot $\boldsymbol{\Xi}(t_{i+1})$ perform a reduced SVD and retain the leading singular triplets in a way that the retained singular values are at least as large as those already kept from previous iterations. Make sure that at most κ SVD components are retained overall, with κ selected a-priori;

2. *Bases pruning.* Ensure that the vectors spanning the reduced right and left spaces have orthonormal columns. Reduce the space dimension if needed.

In the following, we provide the details for this two-step procedure. The strategy that leads to the selection of the actual time instances used for this construction will be discussed in Section 4.2.3. To simplify the presentation, and without loss of generality, we assume $n_1 = n_2 \equiv n$.

First step. Let $\boldsymbol{\Xi}_i = \boldsymbol{\Xi}(t_i)$. For the collection of snapshots, we wish to determine a (left) reduced basis for the range of the matrix $\boldsymbol{S}_1 \equiv \boldsymbol{\mathcal{S}}_{(1)} := (\boldsymbol{\Xi}_1, \ldots, \boldsymbol{\Xi}_{n_s}) \in \mathbb{R}^{n \times (n \cdot n_s)}$ and a (right) reduced basis for the range of the matrix $\boldsymbol{S}_2 \equiv \boldsymbol{\mathcal{S}}_{(2)}^\top := (\boldsymbol{\Xi}_1^\top, \ldots, \boldsymbol{\Xi}_{n_s}^\top)^\top = (\boldsymbol{\Xi}_1; \ldots; \boldsymbol{\Xi}_{n_s})$, where $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{n \times n \times n_s}$ is a snapshot tensor of order three, as defined in Section 1.1. This is performed by incrementally including leading components of the

snapshots $\boldsymbol{\Xi}_i$, so as to determine the approximations

$$\begin{aligned} \boldsymbol{S}_1 &\approx \widetilde{\boldsymbol{S}}_1 := \widetilde{\boldsymbol{V}}_{n_s}\widetilde{\boldsymbol{\Sigma}}_{n_s}\widetilde{\boldsymbol{W}}_{n_s}^\top, \qquad \widetilde{\boldsymbol{V}}_{n_s} \in \mathbb{R}^{n\times\kappa}, \widetilde{\boldsymbol{W}}_{n_s} \in \mathbb{R}^{n\cdot n_s\times\kappa} \\ \boldsymbol{S}_2 &\approx \check{\boldsymbol{S}}_2 := \check{\boldsymbol{V}}_{n_s}\widetilde{\boldsymbol{\Sigma}}_{n_s}\check{\boldsymbol{W}}_{n_s}^\top, \qquad \check{\boldsymbol{V}}_{n_s} \in \mathbb{R}^{n\cdot n_s\times\kappa}, \check{\boldsymbol{W}}_{n_s} \in \mathbb{R}^{n\times\kappa}. \end{aligned}$$

A rank reduction of the matrices $\widetilde{\boldsymbol{V}}_{n_s}$ and $\check{\boldsymbol{W}}_{n_s}$ will provide the sought-after bases to be used for time instances other than those of the snapshots. Let $\kappa \leq n$ be the chosen maximum admissible dimension for the reduced left and right spaces. For $i \in \{1, \ldots, n_s\}$, let[1]

$$\boldsymbol{\Xi}_i \approx \mathsf{V}_i\boldsymbol{\Sigma}_i\mathsf{W}_i^\top, \quad \mathsf{V}_i, \mathsf{W}_i \in \mathbb{R}^{n\times\kappa}, \quad \boldsymbol{\Sigma}_i = \operatorname{diag}(\sigma_1^{(i)}, \ldots, \sigma_\kappa^{(i)}) \tag{4.9}$$

be the reduced SVD of $\boldsymbol{\Xi}_i$ corresponding to its dominant κ singular triplets, with singular values sorted decreasingly. Let $\widetilde{\boldsymbol{V}}_1 = \mathsf{V}_1$, $\widetilde{\boldsymbol{\Sigma}}_1 = \boldsymbol{\Sigma}_1$ and $\widetilde{\boldsymbol{W}}_1 = \mathsf{W}_1$. For each subsequent $i = 2, \ldots, n_s$, the leading singular triplets of $\boldsymbol{\Xi}_i$ are *appended* to the previous matrices $\widetilde{\boldsymbol{\Sigma}}_{i-1}$, $\widetilde{\boldsymbol{V}}_{i-1}$ and $\widetilde{\boldsymbol{W}}_{i-1}$, that is

$$\widetilde{\boldsymbol{S}}_{1,i} = (\widetilde{\boldsymbol{V}}_{i-1}, \mathsf{V}_i) \begin{pmatrix} \widetilde{\boldsymbol{\Sigma}}_{i-1} & \\ & \boldsymbol{\Sigma}_i \end{pmatrix} \begin{pmatrix} \widetilde{\boldsymbol{W}}_{i-1}^\top & \\ & \mathsf{W}_i^\top \end{pmatrix} \equiv \widetilde{\boldsymbol{V}}_i\widetilde{\boldsymbol{\Sigma}}_i\widetilde{\boldsymbol{W}}_i^\top. \tag{4.10}$$

After at most n_s iterations we will have the final $\widetilde{\boldsymbol{S}}_1$, with no extra subscript. The expansion is performed ensuring that the appended singular values are at least as large as those already present, so that the retained directions are the leading ones among all snapshots. Then the three matrices $\widetilde{\boldsymbol{V}}_i$, $\widetilde{\boldsymbol{W}}_i$ and $\widetilde{\boldsymbol{\Sigma}}_i$ are truncated so that the retained diagonal entries of $\widetilde{\boldsymbol{\Sigma}}_i$ are its κ largest diagonal elements. The diagonal elements of $\widetilde{\boldsymbol{\Sigma}}_i$ are not the singular values of $\widetilde{\boldsymbol{S}}_{1,i}$, however the adopted truncation strategy ensures that the error committed is not larger than $\sigma_{\kappa+1}$, which we define as the largest singular value discarded during the whole accumulation process. Since each column of $\widetilde{\boldsymbol{V}}_{n_s}$ has unit norm, it follows that $\|\widetilde{\boldsymbol{V}}_{n_s}\| \leq \kappa$. Moreover, the columns of $\widetilde{\boldsymbol{W}}_{n_s}$ are orthonormal. Hence $\|\boldsymbol{S}_1 - \widetilde{\boldsymbol{S}}_1\| \leq \kappa\sigma_{\kappa+1}$.

In particular, this procedure is not the same as taking the leading singular triplets of each snapshot per se: this first step allows us to retain the leading triplets of each snapshot *when compared with all snapshots*, using the magnitude of all singular values as a quality measure.

A similar strategy is adopted to construct the right basis. Formally,

$$\check{\boldsymbol{S}}_{2,i} = \begin{pmatrix} \check{\boldsymbol{V}}_{i-1} & \\ & \mathsf{V}_i \end{pmatrix} \begin{pmatrix} \widetilde{\boldsymbol{\Sigma}}_{i-1} & \\ & \boldsymbol{\Sigma}_i \end{pmatrix} \begin{pmatrix} \check{\boldsymbol{W}}_{i-1}^\top \\ \mathsf{W}_i^\top \end{pmatrix} = \check{\boldsymbol{V}}_i\widetilde{\boldsymbol{\Sigma}}_i\check{\boldsymbol{W}}_i^\top; \tag{4.11}$$

notice that $\widetilde{\boldsymbol{\Sigma}}_i$ is the same for both $\widetilde{\boldsymbol{S}}_1$ and $\check{\boldsymbol{S}}_2$, and that the large matrices $\widetilde{\boldsymbol{W}}_i$ and $\check{\boldsymbol{V}}_i$ are not stored explicitly in the actual implementation. At completion, the following

[1] Here we drop the bold-face, since these matrices will serve as blocks in the construction of $\widetilde{\boldsymbol{S}}_1$ and $\check{\boldsymbol{S}}_2$.

two matrices are bases candidates for the left and right spaces:

$$\text{Left:} \quad \widetilde{\boldsymbol{V}}_{n_s}\widetilde{\boldsymbol{\Sigma}}_{n_s}^{\frac{1}{2}}, \qquad \text{Right:} \quad \widetilde{\boldsymbol{\Sigma}}_{n_s}^{\frac{1}{2}}\check{\boldsymbol{W}}_{n_s}^{\top}, \tag{4.12}$$

where the singular value matrices keep track of the relevance of each collected singular vector, and the square root allows us to maintain the order of magnitude of the snapshot matrices, when the product of the two left and right matrices is carried out. Here n_s is the total number of snapshots included in the whole procedure, using the dynamic procedure discussed in Section 4.2.3.

The procedure is described in Algorithm 4.1.

Algorithm 4.1 DYNAMIC SELECTION PROCEDURE

1: **INPUT:** $\boldsymbol{\Xi}_i$, $\widetilde{\boldsymbol{V}}_{i-1} \in \mathbb{R}^{n\times\kappa}$, $\widetilde{\boldsymbol{\Sigma}}_{i-1} \in \mathbb{R}^{\kappa\times\kappa}$, $\check{\boldsymbol{W}}_{i-1} \in \mathbb{R}^{n\times\kappa}$, κ
2: **OUTPUT:** $\widetilde{\boldsymbol{V}}_i \in \mathbb{R}^{n\times\kappa}$, $\widetilde{\boldsymbol{\Sigma}}_i \in \mathbb{R}^{\kappa\times\kappa}$, $\check{\boldsymbol{W}}_i \in \mathbb{R}^{n\times\kappa}$.
3: Compute $[\mathsf{V}_i, \Sigma_i, \mathsf{W}_i] = \texttt{svds}\,(\boldsymbol{\Xi}_i, \kappa)$;
4: Append $\widetilde{\boldsymbol{V}}_i \leftarrow (\widetilde{\boldsymbol{V}}_{i-1}, \mathsf{V}_i)$, $\check{\boldsymbol{W}}_i \leftarrow (\check{\boldsymbol{W}}_{i-1}, \mathsf{W}_i)$, $\widetilde{\boldsymbol{\Sigma}}_i \leftarrow \texttt{blkdiag}(\widetilde{\boldsymbol{\Sigma}}_{i-1}, \Sigma_i)$;
5: Decreasingly order the entries of (diagonal) $\widetilde{\boldsymbol{\Sigma}}_i$ and keep the first κ;
6: Order $\widetilde{\boldsymbol{V}}_i$ and $\check{\boldsymbol{W}}_i$ accordingly and keep the first κ vectors of each;

Second step. We complete the two-sided approximation of the snapshot functions by pruning the two orthonormal bases associated with the representation (4.12). Let

$$\widetilde{\boldsymbol{V}}_{n_s}\widetilde{\boldsymbol{\Sigma}}_{n_s}^{\frac{1}{2}} = \overline{\boldsymbol{V}}\,\overline{\boldsymbol{\Sigma}}\,\overline{\boldsymbol{W}}^{\top} \quad \text{and} \quad \widetilde{\boldsymbol{\Sigma}}_{n_s}^{\frac{1}{2}}\check{\boldsymbol{W}}_{n_s}^{\top} = \check{\boldsymbol{V}}\check{\boldsymbol{\Sigma}}\check{\boldsymbol{W}}^{\top} \tag{4.13}$$

be the SVDs of the given matrices. If the matrices $\overline{\boldsymbol{\Sigma}}$ and $\check{\boldsymbol{\Sigma}}$ have rapidly decaying singular values, we can further reduce the low rank approximation of each $\boldsymbol{\Xi}_i$. More precisely, let

$$\overline{\boldsymbol{V}}\,\overline{\boldsymbol{\Sigma}} = [\overline{\mathsf{V}}_1, \overline{\mathsf{V}}_{\mathcal{E}}]\begin{pmatrix}\overline{\Sigma}_1 & \\ & \overline{\Sigma}_{\mathcal{E}}\end{pmatrix}, \quad \text{and} \quad \check{\boldsymbol{\Sigma}}\check{\boldsymbol{W}}^{\top} = \begin{pmatrix}\check{\Sigma}_2 & \\ & \check{\Sigma}_{\mathcal{E}}\end{pmatrix}\begin{bmatrix}\check{\mathsf{W}}_2^{\top} \\ \check{\mathsf{W}}_{\mathcal{E}}^{\top}\end{bmatrix}, \tag{4.14}$$

and let $\boldsymbol{V}_1 = \overline{\mathsf{V}}_1 \in \mathbb{R}^{n\times\nu_1}$, and $\boldsymbol{V}_2 = \check{\mathsf{W}}_2 \in \mathbb{R}^{n\times\nu_2}$. The final reduced dimensions ν_1 and ν_2, that is the number of columns to be retained in the matrices $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$, respectively, are determined by the following criterion:

$$\|\overline{\Sigma}_{\mathcal{E}}\|_F \le \frac{\tau}{\sqrt{n_{\max}}}\|\overline{\boldsymbol{\Sigma}}\|_F \quad \text{and} \quad \|\check{\Sigma}_{\mathcal{E}}\|_F \le \frac{\tau}{\sqrt{n_{\max}}}\|\check{\boldsymbol{\Sigma}}\|_F \tag{4.15}$$

for some chosen tolerance $\tau \in (0,1)$ and $n_{\max}$ is the maximum number of available snapshots of the considered function.

We have assumed so far that at least one singular triplet is retained for all $\boldsymbol{\Xi}$'s. In practice, it might happen that for some i, none of the singular values is large enough to be retained. In this case, $\boldsymbol{\Xi}_i$ does not contribute to the two-sided basis.

Remark 4.1. *If $\boldsymbol{\Xi}_i = \boldsymbol{\Xi}(t_i)$ is symmetric for $t_i \in T$, the reduction process can preserve this structure. Indeed, since $\boldsymbol{\Xi}_i$ is symmetric it holds that $\boldsymbol{\Xi}_i = \mathsf{V}_i\Sigma_i{\mathsf{W}_i}^{\top} =$*

$\mathsf{V}_i\mathsf{\Sigma}_i\mathsf{D}_i\mathsf{V}_i^\top$, *with* D_i *diagonal of ones and minus ones. As a consequence,* $\boldsymbol{V}_2 = \boldsymbol{V}_1$. *Positive definiteness can also be preserved, with* D_i *the identity matrix.*

In the following we use the pair $(\boldsymbol{V}_1, \boldsymbol{V}_2)$, hereafter denoted as *two-sided POD* (2S-POD), to approximate the function $\boldsymbol{\Xi}(t)$ for some $t \neq t_i$:

$$\boldsymbol{\Xi}(t) \approx \boldsymbol{V}_1 \widehat{\boldsymbol{\Xi}}(t) \boldsymbol{V}_2^\top \tag{4.16}$$

with $\widehat{\boldsymbol{\Xi}}$ depending on t. In the setting where $\boldsymbol{\Xi}(t)$ represents the solution functions, that is $\boldsymbol{U}(t)$, we will retain the notation $(\boldsymbol{V}_1, \boldsymbol{V}_2)$. However, when $\boldsymbol{\Xi}(t)$ represents the nonlinear function $\mathcal{F}(\boldsymbol{U}(t), t)$, the pair $(\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2)$ will play the role of $(\boldsymbol{V}_1, \boldsymbol{V}_2)$ in the approximation (4.16); see Section 4.2.4.

4.2.2 Connections to other matrix-based interpolation POD strategies

The approximation discussed in the previous section is not restricted to problems of the form (4.1), but rather it can be employed to any POD function approximation where the snapshot vectors are transformed into matrices, giving rise to a matrix DEIM methodology. This class of approximation has been explored in the recent literature, where different approaches have been discussed, especially in connection with parameter-dependent problems and Jacobian matrix approximation; see, e.g., [163],[39], and the thorough discussion in [27]. In the former case, the setting is particularly appealing whenever the operator has a parameter-based affine function formulation, while in the Jacobian case, the problem is naturally stated in matrix terms, possibly with a sparse structure [33],[147]. In the nonaffine case, in [33],[119] an affine matrix approximation (MDEIM) was proposed by writing appropriate (local) sparse representations of the POD basis, as is the case for finite element methods. As an alternative in this context, it was shown in [54] that DEIM can be applied locally to functions defined on the unassembled finite element mesh (UDEIM); we refer the reader to [155] for more details and to [8] for a detailed experimental analysis.

In our approach we consider the approximation in (4.16). If $\widehat{\boldsymbol{\Xi}}(t)$ were diagonal, then this approximation could be thought of as an MDEIM approach, since then $\boldsymbol{\Xi}(t)$ would be approximated by a sum of rank-one matrices with the time-dependent diagonal elements of $\widehat{\boldsymbol{\Xi}}(t)$ as coefficients. Instead, in our setting $\widehat{\boldsymbol{\Xi}}(t)$ is far from diagonal, hence our approximation determines a more general memory saving approximation based on the low-rank representation given by $\boldsymbol{V}_1, \boldsymbol{V}_2$.

Another crucial novel fact of our approach is the following. While methods such as MDEIM aim at creating a linear combination of matrices, they still rely on the vector POD for computing these matrices, thus only detecting the leading portion of the left range space. In our construction, the left and right approximation spaces spanned by $\boldsymbol{V}_1, \boldsymbol{V}_2$, respectively, stem from a subspace selection of the range spaces of the whole

snapshot matrix $\boldsymbol{S}_1$ (left space) and $\boldsymbol{S}_2$ (right space); here both spaces are *matrix* spaces. In this way, the leading components of both spaces can be captured. In particular, specific space directions followed by the approximated function during the time evolution can be more easily tracked (see Example 4.2 in Section 4.2.6) while maintaining possible symmetries (see Corollary 4.1).

In light of the discussion above, our approach might also be interpreted in terms of the "local basis" POD framework, see, e.g., [6], where the generality of the bases is ensured by interpolation onto matrix manifolds. For a presentation of this methodology, we also refer the reader to the insightful survey [27, section 4.2]. In this context, the matrices $\boldsymbol{V}_1, \boldsymbol{V}_2$ may represent a new *truncated* interpolation of the matrices in $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$, in a completely algebraic setting.

4.2.3 A dynamic implementation

We describe an adaptive procedure for selecting the time instances employed in the first step of the basis construction of Section 4.2.1. This procedure will be used for the selection of both the solution and the nonlinear function snapshots. The dynamic procedure starts with a coarse discretization of the time interval (using one-fourth of the available nodes) and then continues with two successive refinements if needed.

Let $n_{\max}$ be the maximum number of available snapshots of the considered function $\boldsymbol{\Xi}(t)$, $t \in [t_0, t_f]$, with $t_0 = 0$. A first set I_1 of $n_{\max}/4$ equispaced time instances in T is considered (symbol '$\star$' in Figure 4.1). If needed, a second set I_2 of $n_{\max}/4$ equispaced time instances are considered (symbol '$\times$'), whereas the remaining $n_{\max}/2$ time instances (symbol '$\square$' and set I_3) are considered in the third phase, if needed at all.

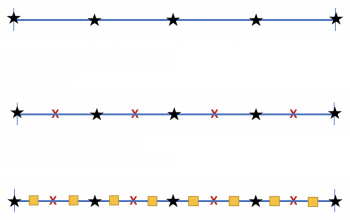

FIGURE 4.1: The three evaluation phases of the refinement procedure.

The initial 2S-POD basis matrices of dimension κ, i.e. $\widetilde{\boldsymbol{V}}_1$ and $\check{\boldsymbol{W}}_1$ from Equation (4.12), are constructed by processing the snapshot $\boldsymbol{\Xi}(t_0)$. For all other time instances t_i in each phase, we use the following inclusion criterion

$$\text{if} \quad \epsilon_i := \frac{\|\boldsymbol{\Xi}(t_i) - \boldsymbol{\Pi}_1 \boldsymbol{\Xi}(t_i) \boldsymbol{\Pi}_2\|}{\|\boldsymbol{\Xi}(t_i)\|} > \texttt{tol} \quad \text{then} \quad \text{include} \tag{4.17}$$

where $\mathbf{\Pi}_1$ and $\mathbf{\Pi}_2$ are orthogonal projections onto the left and right spaces (these are implicitly constructed by performing a reduced QR decomposition of the current matrices $\widetilde{\boldsymbol{V}}_i$ and $\check{\boldsymbol{W}}_i$ on the fly). If a snapshot is selected for inclusion, the leading singular triplets of $\mathbf{\Xi}(t_i)$ are appended to the current bases, and the leading κ components are retained as in Algorithm 4.1; then the next time instance in the phase is investigated. If, by the end of the phase, the arithmetic mean of the errors ϵ_i in (4.17) is above `tol`, it means that the bases are not sufficiently good and we move on to the next refinement phase. Otherwise, the snapshot selection procedure is ended and the matrices $\overline{\boldsymbol{V}}$ and $\check{\boldsymbol{W}}$ are computed and pruned by the second step in Section 4.2.1 to form the final 2S-POD basis matrices $\boldsymbol{V}_1 \in \mathbb{R}^{n\times\nu_1}$ and $\boldsymbol{V}_2 \in \mathbb{R}^{n\times\nu_2}$. The full DYNAMIC procedure to create the 2S-POD approximation space is presented in Algorithm 4.2.

Algorithm 4.2 DYNAMIC 2S-POD

INPUT: Function $\mathbf{\Xi} : T \mapsto \mathbb{R}^{n\times n}$, $n_{\max}$, `tol`, κ, phase sets $I_{1,2,3}$
OUTPUT: $\boldsymbol{V}_1 \in \mathbb{R}^{n\times\nu_1}$ and $\boldsymbol{V}_2 \in \mathbb{R}^{n\times\nu_2}$
Compute $[\mathsf{V}_1, \mathsf{\Sigma}_1, \mathsf{W}_1] = \texttt{svds}\,(\mathbf{\Xi}(t_0), \kappa)$;
Let $\widetilde{\boldsymbol{V}}_1 = \mathsf{V}_1$, $\check{\boldsymbol{W}}_1 = \mathsf{W}_1, \widetilde{\mathbf{\Sigma}}_1 = \mathsf{\Sigma}_1$;
Dynamic selection step :
for PHASE $= 1, 2, 3$ **do**
  for all $t_i \in I_{\text{PHASE}}$ **do**
    if (4.17) satisfied **then**
      Process snapshot $\mathbf{\Xi}(t_i)$ using Dynamic selection (Algorithm 4.1);
      Update $\widetilde{\boldsymbol{V}}_i$ and $\check{\boldsymbol{W}}_i$;
    end if
  end for
  if $\sum_{t_i\in I_{\text{PHASE}}} \epsilon_i \le \texttt{tol}|I_{\text{PHASE}}|$ **then**
    break and go to 17;
  end if
end for
Bases Pruning:
Determine the reduced SVD of $\widetilde{\boldsymbol{V}}_{n_s}\widetilde{\mathbf{\Sigma}}_{n_s}^{\frac{1}{2}}$ and $\widetilde{\mathbf{\Sigma}}_{n_s}^{\frac{1}{2}}\check{\boldsymbol{W}}_{n_s}^{\top}$ in (4.13);
Determine the final reduced $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ as in (4.14) using the criterion (4.15);
Stop

4.2.4 Approximation of the nonlinear function

To complete the reduction of the original problem to the small size problem (4.7), we need to discuss the derivation of the approximation $\widehat{\mathcal{F}(\widehat{\boldsymbol{U}}, t)}$. Let $\{\mathcal{F}(t_j)\}_{j=1}^{n_s}$ be a set of snapshots of the nonlinear function $\mathcal{F}$ at times t_j, $j = 1, \ldots, n_s$. Using Algorithm 4.2 we compute the two matrices $\mathbf{\Phi}_1 \in \mathbb{R}^{n\times p_1}$, $\mathbf{\Phi}_2 \in \mathbb{R}^{n\times p_2}$, which play the role of $(\boldsymbol{V}_1, \boldsymbol{V}_2)$ in the general approximation (4.16), so as to approximate $\mathcal{F}(t)$ as

$$\mathcal{F}(t) \approx \mathbf{\Phi}_1 \widehat{\boldsymbol{F}}(t) \mathbf{\Phi}_2^{\top}, \tag{4.18}$$

with $\widehat{\boldsymbol{F}}(t)$ to be determined. Here p_1, p_2 play the role of ν_1, ν_2 in the general description, and they will be used throughout as basis truncation parameters for the nonlinear snapshots. By adapting the DEIM idea to a two-sided perspective, the coefficient matrix $\widehat{\boldsymbol{F}}(t)$ is determined by selecting independent rows from the matrices $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$, so that

$$\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1 \widehat{\boldsymbol{F}}(t) \boldsymbol{\Phi}_2^\top \boldsymbol{P}_2 = \boldsymbol{P}_1^\top \mathcal{F}(t) \boldsymbol{P}_2,$$

where $\boldsymbol{P}_1 = [e_{\pi_1}, \cdots, e_{\pi_{p_1}}] \in \mathbb{R}^{n \times p_1}$ and $\boldsymbol{P}_2 = [e_{\gamma_1}, \cdots, e_{\gamma_{p_2}}] \in \mathbb{R}^{n \times p_2}$ are columns of the identity matrix of size n. Both matrices are defined similarly to the selection matrix $\boldsymbol{P}_{\text{vec}}$ from Section 1.3.3, and they act on $\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2$, respectively. If $\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1$ and $\boldsymbol{P}_2^\top \boldsymbol{\Phi}_2$ are nonsingular, then the coefficient matrix $\widehat{\boldsymbol{F}}(t)$ is determined by

$$\widehat{\boldsymbol{F}}(t) = (\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1} \boldsymbol{P}_1^\top \mathcal{F}(t) \boldsymbol{P}_2 (\boldsymbol{\Phi}_2^\top \boldsymbol{P}_2)^{-1}.$$

With this coefficient matrix $\widehat{\boldsymbol{F}}(t)$, the final approximation (4.18) becomes[2]

$$\widetilde{\mathcal{F}}(t) = \boldsymbol{\Phi}_1 (\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1} \boldsymbol{P}_1^\top \mathcal{F}(t) \boldsymbol{P}_2 (\boldsymbol{\Phi}_2^\top \boldsymbol{P}_2)^{-1} \boldsymbol{\Phi}_2^\top =: \boldsymbol{\mathfrak{D}}_1 \mathcal{F}(t) \boldsymbol{\mathfrak{D}}_2^\top. \tag{4.19}$$

Note that $\boldsymbol{\mathfrak{D}}_*$ are oblique projectors. A similar approximation can be found in [146]. In addition to that of the two spaces, an important role is played by the choice of the interpolation indices contained in $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. We suggest determining these indices for the matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ as the output of `q-deim` [57] (see also Section 1.3.3) with inputs $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$, respectively.

We next provide a bound measuring the distance between the error obtained with the proposed oblique projection (4.19) and the best approximation error of $\mathcal{F}$ in the same range spaces, where we recall that $\boldsymbol{\Phi}_1$ and $\boldsymbol{\Phi}_2$ have orthonormal columns. This bound is a direct extension to the matrix setting of [41, Lemma 3.2].

Proposition 4.1. *Let $\mathcal{F} \in \mathbb{R}^{n \times n}$ be an arbitrary matrix, and let $\widetilde{\mathcal{F}} = \boldsymbol{\mathfrak{D}}_1 \mathcal{F} \boldsymbol{\mathfrak{D}}_2^\top$, as in Equation* (4.19)*. Then*

$$\|\mathcal{F} - \widetilde{\mathcal{F}}\|_F \leq c_1 c_2 \, \|\mathcal{F} - \boldsymbol{\Phi}_1 \boldsymbol{\Phi}_1^\top \mathcal{F} \boldsymbol{\Phi}_2 \boldsymbol{\Phi}_2^\top\|_F \tag{4.20}$$

where $c_1 = \left\|(\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1}\right\|_2$ and $c_2 = \left\|(\boldsymbol{P}_2^\top \boldsymbol{\Phi}_2)^{-1}\right\|_2$.

Proof. Recall that $\boldsymbol{f} = \text{vec}(\mathcal{F})$. Then, by the properties of the Kronecker product

$$\begin{aligned} \|\mathcal{F} - \widetilde{\mathcal{F}}\|_F &= \|\text{vec}(\mathcal{F}) - \text{vec}(\widetilde{\mathcal{F}})\|_2 = \|\boldsymbol{f} - (\boldsymbol{\mathfrak{D}}_2 \otimes \boldsymbol{\mathfrak{D}}_1)\boldsymbol{f}\|_2 \\ &= \left\| \boldsymbol{f} - (\boldsymbol{\Phi}_2 \otimes \boldsymbol{\Phi}_1) \left((\boldsymbol{P}_2 \otimes \boldsymbol{P}_1)^\top (\boldsymbol{\Phi}_2 \otimes \boldsymbol{\Phi}_1) \right)^{-1} (\boldsymbol{P}_2 \otimes \boldsymbol{P}_1)^\top \boldsymbol{f} \right\|_2 \end{aligned}$$

Therefore, by [41, Lemma 3.2],

$$\begin{aligned} \|\mathcal{F} - \widetilde{\mathcal{F}}\|_F &\leq \left\| \left((\boldsymbol{P}_2 \otimes \boldsymbol{P}_1)^\top (\boldsymbol{\Phi}_2 \otimes \boldsymbol{\Phi}_1) \right)^{-1} \right\|_2 \left\| \boldsymbol{f} - (\boldsymbol{\Phi}_2 \otimes \boldsymbol{\Phi}_1)(\boldsymbol{\Phi}_2 \otimes \boldsymbol{\Phi}_1)^\top \boldsymbol{f} \right\|_2 \\ &= \left\|(\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1}\right\|_2 \left\|(\boldsymbol{P}_2^\top \boldsymbol{\Phi}_2)^{-1}\right\|_2 \left\| \mathcal{F} - \boldsymbol{\Phi}_1 \boldsymbol{\Phi}_1^\top \mathcal{F} \boldsymbol{\Phi}_2 \boldsymbol{\Phi}_2^\top \right\|_F . \ \square \end{aligned}$$

[2] If the nonlinear function $\mathcal{F}(t)$ is symmetric for all $t \in T$, thanks to Remark 4.1 this approximation will preserve the symmetry of the nonlinear function.

We emphasize that c_1, c_2 do not depend on time, in case $\mathcal{F}$ does. As has been discussed in [41],[57], it is clear from (4.20) that minimizing these amplification factors will minimize the error norm with respect to the best approximation onto the spaces range($\boldsymbol{\Phi}_1$) and range($\boldsymbol{\Phi}_2$). The quantities c_1, c_2 depend on the interpolation indices. If the indices are selected greedily, as in [41], then

$$c_1 \leq \frac{(1+\sqrt{2n})^{p_1-1}}{\|\boldsymbol{e}_1^\top \boldsymbol{\Phi}_1\|_\infty}, \quad c_2 \leq \frac{(1+\sqrt{2n})^{p_2-1}}{\|\boldsymbol{e}_1^\top \boldsymbol{\Phi}_2\|_\infty}. \tag{4.21}$$

If the indices are selected by a pivoted QR factorization as in `q-deim`, then

$$c_1 \leq \sqrt{n-p_1+1}\frac{\sqrt{4^{p_1}+6p_1-1}}{3}, \quad c_2 \leq \sqrt{n-p_2+1}\frac{\sqrt{4^{p_2}+6p_2-1}}{3},$$

which are better bounds than those in (4.21), though still rather pessimistic; see [57].

To complete the efficient derivation of the reduced model in (4.7) we consider that $\mathcal{F}$ is evaluated componentwise, as we assume throughout[3], then

$$\boldsymbol{P}_1^\top \mathcal{F}(\boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top, t)\boldsymbol{P}_2 = \mathcal{F}(\boldsymbol{P}_1^\top \boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top \boldsymbol{P}_2, t),$$

so that

$$\begin{aligned}\widehat{\mathcal{F}}(\widehat{\boldsymbol{U}}, t) &\approx \boldsymbol{V}_1^\top \boldsymbol{\Phi}_1(\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1}\boldsymbol{P}_1^\top \mathcal{F}(\boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top, t)\boldsymbol{P}_2(\boldsymbol{\Phi}_2^\top \boldsymbol{P}_2)^{-1}\boldsymbol{\Phi}_2^\top \boldsymbol{V}_2 \\ &= \boldsymbol{V}_1^\top \boldsymbol{\Phi}_1(\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1}\mathcal{F}(\boldsymbol{P}_1^\top \boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top \boldsymbol{P}_2, t)(\boldsymbol{\Phi}_2^\top \boldsymbol{P}_2)^{-1}\boldsymbol{\Phi}_2^\top \boldsymbol{V}_2 \\ &=: \widehat{\mathcal{F}(\widehat{\boldsymbol{U}}, t)}.\end{aligned} \tag{4.22}$$

The matrices $\boldsymbol{V}_1^\top \boldsymbol{\Phi}_1(\boldsymbol{P}_1^\top \boldsymbol{\Phi}_1)^{-1} \in \mathbb{R}^{k_1 \times p_1}$ and $(\boldsymbol{\Phi}_2^\top \boldsymbol{P}_2)^{-1}\boldsymbol{\Phi}_2^\top \boldsymbol{V}_2 \in \mathbb{R}^{p_2 \times k_2}$ are independent of t, therefore they can be precomputed and stored once for all. Similarly for the products $\boldsymbol{P}_1^\top \boldsymbol{V}_1 \in \mathbb{R}^{p_1 \times k_1}$ and $\boldsymbol{V}_2^\top \boldsymbol{P}_2 \in \mathbb{R}^{k_2 \times p_2}$. Note that products involving the selection matrices $\boldsymbol{P}$'s are not explicitly carried out: the operation simply requires selecting corresponding rows or columns in the other factor matrix.

Finally, we remark that in some cases the full space approximation matrix may not be involved. For instance, if $\mathcal{F}$ is a matrix function [81] and $\boldsymbol{U}(t)$ is symmetric for all $t \in [0, t_f]$, so that $\boldsymbol{U}(t) \approx \boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_1^\top$, then, recalling (4.8), it holds that

$$\widehat{\mathcal{F}}(\widehat{\boldsymbol{U}}, t) = \boldsymbol{V}_1^\top \mathcal{F}(\boldsymbol{V}_1\widehat{\boldsymbol{U}}\boldsymbol{V}_1^\top, t)\boldsymbol{V}_1 \overset{\star}{=} \mathcal{F}(\boldsymbol{V}_1^\top \boldsymbol{V}_1\widehat{\boldsymbol{U}}, t) = \mathcal{F}(\widehat{\boldsymbol{U}}, t),$$

where the equality $\overset{\star}{=}$ is due to [81, Corollary 1.34].

[3] For general nonlinear functions the theory from [41, Section 3.5] can be extended to both matrices $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$.

4.2.5 Efficient treatment of the reduced semilinear ODE

To complete the derivation of the numerical method, we need to determine the time-dependent matrix $\widehat{\boldsymbol{U}}(t)$, $t \in [0, t_f]$ in the approximation $\boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top \approx \boldsymbol{U}(t)$, where $\boldsymbol{V}_1 \in \mathbb{R}^{n\times k_1}$ and $\boldsymbol{V}_2 \in \mathbb{R}^{n\times k_2}$, $k_1, k_2 \ll n$. The function $\widehat{\boldsymbol{U}}(t)$ is computed as the numerical solution to the reduced problem (4.7) with $\widehat{\mathcal{F}(\widehat{\boldsymbol{U}}, t)}$ defined in (4.22). To integrate the reduced-order model (4.7) as time t varies, several alternatives can be considered. The vectorized form of the semilinear problem can be treated with classical first or second-order semi-implicit methods such as IMEX methods (see, e.g.,[14]) that appropriately handle the stiff and non-stiff parts of the equation. Nevertheless, as has been illustrated in Section 2.1, great savings could be obtained by sticking to the matrix formulation of the reduced problem for the integration phase. As a result, any of the semi-implicit integrators presented in Section 2.1 can be applied to (4.7), given that the nonlinear function is sufficiently regular. However, due to the ability to efficiently treat the matrix exponentials in the reduced model, the ETD scheme was the most powerful in our experiments. Consequently, the matrix-oriented exponential Euler scheme (2.5) will be applied in the sequel to determine the approximations $\widehat{\boldsymbol{U}}^{(i)} \approx \widehat{\boldsymbol{U}}(\mathrm{t}_i)$, for $i = 1, 2, \ldots, n_\mathrm{t}$. See Section 2.1 and [51] for details regarding the efficient implementation of this integrator in our setting.

Concerning the quality of our approximation, error estimates for the full POD-DEIM approximation of systems of the form (4.3) have been derived in [43, 163]. A crucial hypotheses in the available literature is that $\boldsymbol{f}(\boldsymbol{u}, t) = \mathrm{vec}(\mathcal{F}(\boldsymbol{U}, t))$ be Lipschitz continuous with respect to the first argument, and this is also required for exponential integrators. This condition is satisfied for the nonlinear function of our reduced problem. Indeed, consider the vectorized approximation space $\boldsymbol{V}_\otimes = \boldsymbol{V}_2 \otimes \boldsymbol{V}_1$ and the oblique projector $\boldsymbol{\mathfrak{D}}_\otimes = \mathfrak{D}_2 \otimes \mathfrak{D}_1$ from (4.19). If we denote by $\widehat{\boldsymbol{U}}(t)$ the approximate solution of (4.7), then

$$\|\boldsymbol{U}(t) - \boldsymbol{V}_1\widehat{\boldsymbol{U}}(t)\boldsymbol{V}_2^\top\|_F = \|\boldsymbol{u}(t) - \boldsymbol{V}_\otimes\widehat{\boldsymbol{u}}_\otimes(t)\|_2, \tag{4.23}$$

where $\widehat{\boldsymbol{u}}_\otimes(t) = \mathrm{vec}(\widehat{\boldsymbol{U}}(t))$ solves the reduced problem

$$\dot{\widehat{\boldsymbol{u}}}_\otimes(t) = \boldsymbol{V}_\otimes^\top \boldsymbol{L}\boldsymbol{V}_\otimes\widehat{\boldsymbol{u}}_\otimes(t) + \boldsymbol{V}_\otimes^\top\boldsymbol{\mathfrak{D}}_\otimes\boldsymbol{f}(\boldsymbol{V}_\otimes\widehat{\boldsymbol{u}}_\otimes, t) + \boldsymbol{V}_\otimes^\top\boldsymbol{c}.$$

The error in Equation (4.23) can therefore be approximated by applying the a-priori [43] or a posteriori [163] error estimate to the vectorized system associated with $\widehat{\boldsymbol{u}}$. Moreover, if $\boldsymbol{f}(\boldsymbol{u}, t)$ is Lipschitz continuous, this property is preserved by the reduced order vector model, since $\boldsymbol{V}_\otimes$ has orthonormal columns and $\|\boldsymbol{\mathfrak{D}}_\otimes\|$ is a bounded constant, as shown in Proposition 4.1; see e.g., [43].

The complete 2S-POD-DEIM method for the semilinear differential problem (4.1) is presented in Algorithm 2S-POD-DEIM. In Table 4.1 we summarize the key dimensions

and parameters of the whole procedure. Next follows a technical discussion of the algorithm and its computational complexity.

Algorithm 2S-POD-DEIM

INPUT: Coefficient matrices of (4.1), $\mathcal{F} : \mathbb{R}^{n\times n} \times [0, t_f] \to \mathbb{R}^{n\times n}$, (or its snapshots), $n_{\max}$, κ, and τ, n_{t}, $\{\mathrm{t}\}_{i=0,\ldots,n_{\mathrm{t}}}$.
OUTPUT: $\boldsymbol{V}_1, \boldsymbol{V}_2$ and $\widehat{\boldsymbol{U}}^{(i)}$, $i = 0, \ldots, n_{\mathrm{t}}$ to implicitly form the approximation $\boldsymbol{V}_1\widehat{\boldsymbol{U}}^{(i)}\boldsymbol{V}_2^\top \approx \boldsymbol{U}(\mathrm{t}_i)$

Offline:

1. Determine $\boldsymbol{V}_1, \boldsymbol{V}_2$ for $\{\boldsymbol{U}\}_{i=1}^{n_{\max}}$ and $\boldsymbol{\Phi}_1, \boldsymbol{\Phi}_2$ for $\{\mathcal{F}\}_{i=1}^{n_{\max}}$ via Algorithm 4.2 (DYNAMIC 2S-POD) using at most n_s of the $n_{\max}$ time instances (if not available, this includes approximating the snapshots $\{\mathcal{F}(t_i)\}_{i=1}^{n_{\max}}$, $\{\boldsymbol{U}(t_i)\}_{i=1}^{n_{\max}}$ as the time interval is spanned);
2. Compute $\widehat{\boldsymbol{U}}^{(0)} = \boldsymbol{V}_1^\top\boldsymbol{U}_0\boldsymbol{V}_2$, $\widehat{\boldsymbol{A}}_1 = \boldsymbol{V}_1^\top\boldsymbol{A}\boldsymbol{V}_1$, $\widehat{\boldsymbol{A}}_2 = \boldsymbol{V}_2^\top\boldsymbol{A}_2\boldsymbol{V}_2$, $\widehat{\boldsymbol{C}} = \boldsymbol{V}_1^\top\boldsymbol{C}\boldsymbol{V}_2$;
3. Determine $\boldsymbol{P}_1, \boldsymbol{P}_2$ using `q-deim`(2S-DEIM);
4. Compute $\boldsymbol{V}_1^\top\boldsymbol{\Phi}_1(\boldsymbol{P}_1^\top\boldsymbol{\Phi}_1)^{-1}$, $(\boldsymbol{\Phi}_2^\top\boldsymbol{P}_2)^{-1}\boldsymbol{\Phi}_2^\top\boldsymbol{V}_2$, $\boldsymbol{P}_1^\top\boldsymbol{V}_1$ and $\boldsymbol{V}_2^\top\boldsymbol{P}_2$;

Online:

5. For each $j = 1, \ldots, n_{\mathrm{t}}$
   (i) Evaluate $\widehat{\mathcal{F}(\widehat{\boldsymbol{U}}^{(j-1)}, \mathrm{t}_{j-1})}$ as in (4.22) using the matrices computed above;
   (ii) Numerically solve the matrix equation (2.6) and compute

$$\widehat{\boldsymbol{U}}^{(j)} = e^{h\widehat{\boldsymbol{A}}_1}\widehat{\boldsymbol{U}}^{(j-1)}e^{h\widehat{\boldsymbol{A}}_2} + \boldsymbol{\Phi}^{(j-1)};$$

TABLE 4.1: Summary of leading dimensions and parameters of Algorithm 2S-POD-DEIM.

Par.	Description
n_s	Employed number of snapshots
k	Dimension of vector POD subspace
p	Dimension of vector DEIM approx. space
N	Length of $\boldsymbol{u}(t)$, $N = n^2$.
κ	Dimension of the snapshot space approximation
k_i	Dimension of left ($i = 1$) and right ($i = 2$) 2S-POD subspaces
p_i	Dimension of left ($i = 1$) and right ($i = 2$) 2S-DEIM subspaces
n	Dimension of square $\boldsymbol{U}(t)$, for $n = n_1 = n_2$

Discussion of the algorithm and computational complexity

Here, we compare the computational complexity of the new 2S-POD-DEIM method applied to (4.1) with that of standard POD-DEIM. All discussions are related to Algorithm 2S-POD-DEIM above.

The offline phase. The first four steps of the presented algorithm define the offline phase. For the $\boldsymbol{U}$-set, we considered the matrix-oriented IMEX Euler scheme to integrate the full order model; see, e.g., [51] and Section 2.1, whereas the snapshot

selection is done via the adaptive procedure discussed in Section 4.2.3. Notice that moving from one phase to the next in Algorithm 4.2 does not require recomputing any quantities. Indeed, if h^* is the stepsize of the new phase, we determine $\boldsymbol{U}(h^*)$ from $\boldsymbol{U}(0)$ and initialize the semi-implicit Euler scheme from there; see also Figure 4.1.

The computational complexity of approximating the κ leading singular triplets with `svds`, as required by Algorithm 4.1 is given by the implicitly restarted Lanczos bidiagonalization, as implemented in the MATLAB function `svds`. For each i this cost is mainly given by matrix vector multiplications with the dense $n \times n$ matrix; one Arnoldi cycle involves at most 2κ such products, together with 2κ basis orthogonalizations, leading to $\mathcal{O}(n^2\kappa + n\kappa)$ operations per cycle [16]. The final SVDs for *bases pruning* at the end of Algorithm 4.2 are performed with a dense solver, and each has complexity $\mathcal{O}(11\kappa^3)$ [70, p.493]. Furthermore, each skinny QR-factorization required for the projections $\boldsymbol{\Pi}_1$ and $\boldsymbol{\Pi}_2$ in (4.17) has complexity $\mathcal{O}(2n\kappa^2)$ [70, p.255]. For the standard POD-DEIM algorithm, the reported SVD complexity is the total cost of orthogonalizing all selected snapshots by means of Gram-Schmidt (see the discussion in Section 4.2.6), which is $\mathcal{O}(Nn_s^2)$.

The projected coefficient matrices $\widehat{\boldsymbol{A}}_1$, $\widehat{\boldsymbol{A}}_2$, $\widehat{\boldsymbol{C}}$ and $\widehat{\boldsymbol{U}}_0$ are computed once for all and stored in step 2 of Algorithm 2S-POD-DEIM, with a total complexity of approximately $\mathcal{O}(n^2(k_1 + k_2) + n(k_1 + k_2 + k_1^2 + k_2^2))$, assuming that $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are sparse and $\boldsymbol{C}$ and $\boldsymbol{U}_0$ are dense. This step is called POD projection in Table 4.2.

Step 3 in the Algorithm 2S-POD-DEIM has a computational complexity of $\mathcal{O}(n(p_1^2+p_2^2))$ [57], while the matrices in step 4 need to be computed and stored with computational complexity $\mathcal{O}(n(k_1 + k_2)(p_1 + p_2) + (p_1^2 + p_2^2)n + p_1^3 + p_2^3)$ in total [41]. This step is called DEIM projection in Table 4.2. We recall that the products involving the selection matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$ do not entail any computation.

Finally, for the ETD procedure, an eigenvalue decomposition of each of the reduced matrices $\widehat{\boldsymbol{A}}_1$ and $\widehat{\boldsymbol{A}}_2$ is done once for all, which allows for a significant speedup in the online integration phase (see Section 2.1), and it has complexity $\mathcal{O}(9k_1^3+9k_2^3)$ for dense symmetric[4] matrices [70]. This makes the cost of evaluating the matrix exponentials negligible since all the computations at each iteration online will be performed within the eigenbases. Furthermore, thanks to the small dimension of the matrices, we also explicitly compute the inverse of the eigenvector matrices with a cost of $\mathcal{O}(k_1^3 + k_2^3)$, as required by the online computation.

All these costs are summarized in Table 4.2 and compared with those of the standard POD-DEIM offline phase applied to (4.3), as indicated in [41], with dimension $N = n^2$. All coefficient matrices are assumed to be sparse, and it is assumed that both methods select n_s snapshots via the adaptive procedure. In practice, however, it appears that the two-sided procedure requires far fewer snapshots than the vectorization procedure;

[4] If the coefficient matrices were nonsymmetric, this would be more expensive (still of cubic order), but determining the exact cost is, however still an open problem [70].

see, e.g., Table 4.4. The table also includes the memory requirements for the snapshots and the basis matrices.

TABLE 4.2: Offline phase: Computational costs (flops) for standard POD-DEIM applied to (4.3), and 2S-POD-DEIM applied to (4.1), and principal memory requirements. Here $N = n^2$.

Procedure	POD-DEIM	DYNAMIC 2S-POD-DEIM
SVD	$\mathcal{O}(Nn_s^2)$	$\mathcal{O}(n^2\kappa n_s + n\kappa n_s + 6n\kappa^2 + 11\kappa^3)$
QR	–	$\mathcal{O}(n\kappa^2 n_s)$
DEIM	$\mathcal{O}(Np^2)$	$\mathcal{O}(n(p_1^2 + p_2^2))$
POD projection	$\mathcal{O}(Nk + Nk^2)$	$\mathcal{O}(n^2(k_1 + k_2) + n(k_1 + k_2 + k_1^2 + k_2^2))$
DEIM projection	$\mathcal{O}(Nkp + p^2N + p^3)$	$\mathcal{O}(n(k_1 + k_2)(p_1 + p_2) + (p_1^2 + p_2^2)n + p_1^3 + p_2^3)$
Snapshot Storage	$\mathcal{O}(Nn_s)$	$\mathcal{O}(n\kappa)$
Basis Storage	$\mathcal{O}(N(k + p))$	$\mathcal{O}(n(k_1 + k_2 + p_1 + p_2))$

The online phase. The total cost of performing step 6.(i) is $\mathcal{O}(\omega(p_1p_2) + k_1p_1p_2 + k_1k_2p_2)$, where $\omega(p_1p_2)$ is the cost of evaluating the nonlinear function at p_1p_2 entries. Step 6.(ii) requires a matrix–matrix product and the solution of the Sylvester equation (2.6) in the eigenspace; see Algorithm 2.1. The latter demands only matrix–matrix products, which come at a cost of $\mathcal{O}(k_1^2k_2 + k_1k_2^2)$ and two Hadamard products with complexity $\mathcal{O}(k_1k_2)$; see Algorithm 2.1 for more details. This brings the total complexity of one time iteration online to $\mathcal{O}(\omega(p_1p_2)+k_1p_1p_2+k_1k_2p_2+k_1^2k_2+k_1k_2^2+k_1k_2)$, which is independent of the original problem size n.

4.2.6 Numerical experiments

In this section, we will illustrate the performance of our matrix-oriented 2S-POD-DEIM integrator. First, we analyze the quality of the approximation space created by the DYNAMIC algorithm on three nonlinear functions with different characteristics, after which we illustrate the ability of the procedure to capture the underlying geometric properties of the considered function. Thereafter, we focus on the ODE setting by comparing the new 2S-POD-DEIM procedure to the standard POD-DEIM.

Approximation of a nonlinear function $\mathcal{F}$

We investigate the effectiveness of the proposed DYNAMIC 2S-POD procedure for determining the two-sided approximation space of a nonlinear function. As a reference comparison, we consider the vector form of the DEIM approximation (hereafter VECTOR) in Section 1.3.3.

We also include comparisons with a simple two-sided matrix reduction strategy that uses a sequential evaluation of all available snapshots and the updating of the bases $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ during the snapshot processing. In particular, if $[\mathsf{V}_i, \mathbf{\Sigma}_i, \mathsf{W}_i] = \texttt{svds}\,(\mathcal{F}_i, \kappa)$ is the singular value decomposition of $\mathcal{F}_i$ limited to the leading κ singular triplets,

then in this simple approach the basis matrices $\mathbf{\Phi}_1$ and $\mathbf{\Phi}_2$ are directly updated by orthogonal reduction of the augmented matrices

$$\left(\mathbf{\Phi}_1, \mathsf{V}_i \Sigma_i^{\frac{1}{2}}\right) \in \mathbb{R}^{n\times\kappa_1} \quad \text{and} \quad \left(\mathbf{\Phi}_2, \mathsf{W}_i \Sigma_i^{\frac{1}{2}}\right) \in \mathbb{R}^{n\times\kappa_2} \tag{4.24}$$

respectively, where $\kappa_1, \kappa_2 \geq \kappa$. To make this procedure comparable in terms of memory to Algorithm 4.1, we enforce that the final dimension ν_i of each basis satisfies $\nu_i \leq \kappa$, for $i = \ell, r$. We will refer to this as the VANILLA procedure for adding a snapshot to the approximation space; see, for instance, [122] for a similar procedure in the vector setting.

Example 4.1. Consider the nonlinear functions $\phi_i : \Omega \times [0, t_f] \to \mathbb{R}$, $\Omega \subset \mathbb{R}^2$, $i = 1, 2, 3$ defined as

$$\begin{cases} \phi_1(x_1, x_2, t) &= \frac{x_2}{\sqrt{(x_1+x_2-t)^2+(2x_1-3t)^2+0.01^2}}, \ \Omega = [0,2]\times[0,2], t_f = 2, \\ \phi_2(x_1, x_2, t) &= \frac{x_1 x_2}{(x_2 t+0.1)^2} + \frac{2^{(x_1+x_2)}}{\sqrt{(x_1+x_2-t)^2+(x_2^2+x_1^2-t^2)^2+0.01^2}}, \ \Omega = [0,1]\times[0,1.5], t_f = 3, \\ \phi_3(x_1, x_2, t) &= \frac{x_1(0.1+t)}{(x_2 t+0.1)^2} + \frac{t2^{(x_1+x_2)}}{\sqrt{(x_1+x_2-t)^2+(x_2^2+x_1^2-3t)^2+0.01^2}}, \ \Omega = [0,3]\times[0,3], t_f = 5. \end{cases}$$

Each function is discretized with $n = 2000$ nodes in each spatial direction to form three matrix valued functions $\mathcal{F}^{(i)} : T \to \mathbb{R}^{n\times n}$, for $i = 1, \ldots, 3$, respectively. In the truncation criterion (4.15), for all functions we set $n_{\max} = 60$ and $\tau = 10^{-3}$. The function ϕ_1 shows significant variations at the beginning of the time window, ϕ_3 varies more towards the right-hand of the time span, while ϕ_2 is somewhere in between.

The approximations obtained with the considered methods are reported in Table 4.3 for $\kappa = 50$ and $\kappa = 70$, with the following information: For the adaptive snapshot selection procedure, we indicate the required number of PHASES and the final total number n_s of snapshot used, the CPU time to construct the basis vectors (time for Algorithm 4.1 for DYNAMIC and the time for the SVDs (4.24) for VANILLA), the final dimensions ν_ℓ and ν_r and finally, the arithmetic mean of the errors

$$\frac{\|\mathcal{F}(\mathsf{t}_j) - \mathbf{\Phi}_1 \mathbf{\Phi}_1^\top \mathcal{F}(\mathsf{t}_j) \mathbf{\Phi}_2 \mathbf{\Phi}_2^\top\|}{\|\mathcal{F}(\mathsf{t}_j)\|} \tag{4.25}$$

over 300 equispaced timesteps t_j, for each $\mathcal{F} = \mathcal{F}^{(i)}$, $i = 1, 2, 3$. For the vector approach, where we used $n_s = \kappa$, the reported time consists of the CPU time needed to perform the SVD of the long snapshots, while the error is measured using the vector form corresponding to the formula above; see the description at the beginning of Section 4.2.6.

Between the matrix-oriented procedures, the dynamic procedure outperforms the simplified one in terms of the space dimensions ν_1 and ν_2, the number of utilized snapshots n_s, and in terms of CPU time, especially when all of the time selection phases are not required. The error is comparable for the two methods. Not unexpectedly, increasing κ allows one to save on the number of snapshots n_s, though a slightly larger reduced dimension ν_ℓ/ν_r may occur. The vector method is not competitive for any of the

		$\kappa = 50$				$\kappa = 70$			
$\mathcal{F}$	alg.	phases (n_s)	time sec.	ν_1/ν_2	error	phases (n_s)	time sec.	ν_1/ν_2	error
ϕ_1	DYNAMIC	2 (9)	3.5	33/39	$6 \cdot 10^{-4}$	1(7)	4.7	40/50	$3 \cdot 10^{-4}$
	VANILLA	-(60)	27.6	42/50	$6 \cdot 10^{-4}$	-(60)	38.8	42/60	$3 \cdot 10^{-4}$
	VECTOR	-(50)	35.9	41	$1 \cdot 10^{-3}$	-(70)	77.3	56	$3 \cdot 10^{-4}$
ϕ_2	DYNAMIC	3(21)	8.6	45/26	$8 \cdot 10^{-4}$	2(10)	6.1	48/30	$6 \cdot 10^{-4}$
	VANILLA	-(60)	25.6	50/37	$4 \cdot 10^{-4}$	-(60)	39.6	58/37	$2 \cdot 10^{-4}$
	VECTOR	-(50)	42.7	36	$6 \cdot 10^{-3}$	-(70)	91.8	47	$2 \cdot 10^{-3}$
ϕ_3	DYNAMIC	2(11)	4.4	34/33	$1 \cdot 10^{-3}$	1(10)	5.9	39/39	$3 \cdot 10^{-4}$
	VANILLA	-(60)	25.4	46/46	$2 \cdot 10^{-4}$	-(60)	38.6	46/46	$2 \cdot 10^{-4}$
	VECTOR	-(50)	47.1	50	$2 \cdot 10^{-3}$	-(70)	92.5	64	$8 \cdot 10^{-4}$

TABLE 4.3: Example 4.1. Performance of DYNAMIC, VANILLA and VECTOR algorithms for $n = 2000$.

observed parameters, taking into account that vectors of length n^2 need to be stored. □

Example 4.2. Here we analyze how the proposed 2S-DEIM strategy is able to capture the underlying geometric properties of the considered nonlinear function. To this end, we consider the nonlinear function

$$f(u) = \frac{1}{\sqrt{u + 0.1^2}}, \tag{4.26}$$

for two functions u_1 and u_2 with different structure, namely

$$u_1(x, y, t) = (x - t)^2 + (y - t)^2 \tag{4.27}$$

and

$$u_2(x, y, t) = 10^{-3}x + (y - t)^2 \tag{4.28}$$

with $(x, y) \in [0.1, 0.9]^2$ and $t \in [0, 0.5]$. For this example, $f(u_1)$ and $f(u_2)$ are discretized with $n = 300$ equispaced nodes in each of the space directions. In the figures below we report the initial state (left), the final state (middle), and the route that the peak followed from initial state to final state (right), for both discretized functions $\mathcal{F}(\boldsymbol{U}_1, t)$ (Figure 4.2) and $\mathcal{F}(\boldsymbol{U}_2, t)$ (Figure 4.3).

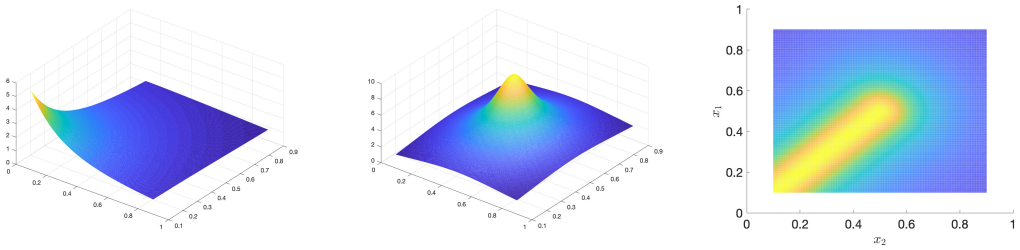

FIGURE 4.2: The initial state (left), final state (middle) and route (right) of the function $\mathcal{F}(\boldsymbol{U}_1, t)$

The geometrical difference between the two functions is clear from the figures. The peak of the first function moves diagonally across the state space, where almost all

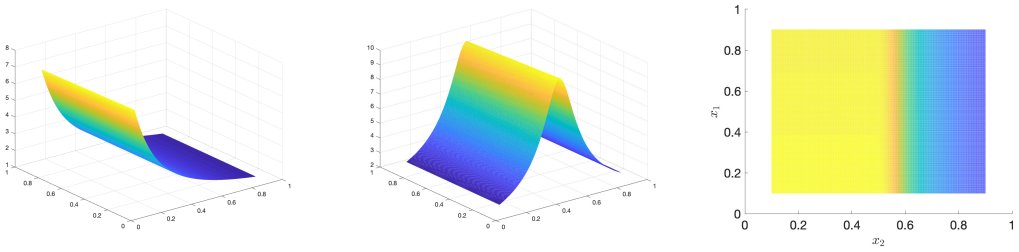

FIGURE 4.3: The initial state (left), final state (middle) and route (right) of the function $\mathcal{F}(\boldsymbol{U}_2, t)$

movement of the second function is horizontal. We analyze how the difference in geometrical behavior of the two functions is captured by the 2S-DEIM basis vectors. We first consider the singular value decay of the left and right approximate snapshot matrices, that is the decay of the diagonal elements of $\overline{\boldsymbol{\Sigma}}$ and $\breve{\boldsymbol{\Sigma}}$ from (4.13), implemented here with $\kappa = 70$. The singular value decays for both functions appear in Figure 4.4. The figure indicates that the left and right matrices for $\mathcal{F}(\boldsymbol{U}_1, t)$ have the same decay

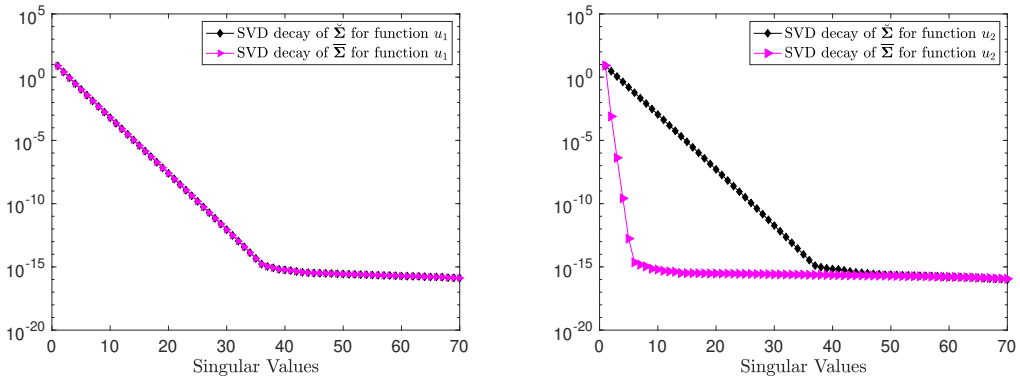

FIGURE 4.4: The decay in the diagonal elements of $\overline{\boldsymbol{\Sigma}}$ and $\breve{\boldsymbol{\Sigma}}$ for both functions $\mathcal{F}(\boldsymbol{U}_1, t)$ (left) and $\mathcal{F}(\boldsymbol{U}_2, t)$ (right) *edit legends*

in their singular values, which is expected since the function is symmetric. A more interesting observation is that for $\mathcal{F}(\boldsymbol{U}_2, t)$, the decay of the right-hand side snapshot vectors remain unchanged, but that the singular values of the left-hand side vectors now decay far more rapidly since there is almost no movement of the function in the vertical direction within the timespan. These results are mirrored in the error plots of Figure 4.5, where we plot the average relative error (4.25) for varying values of p_1 and p_2 on the generic grid $[1, 40] \times [1, 40]$ for both functions $\mathcal{F}(\boldsymbol{U}_1, t)$ (left) and $\mathcal{F}(\boldsymbol{U}_2, t)$ (right).

Consider first the figure on the left. It can be deduced that the fastest route to obtain a low error is to select $p_1 = p_2$ and move down the diagonal of the plot. Notice, however, that for the second function $\mathcal{F}(\boldsymbol{U}_2, t)$, i.e., the plot on the right, effectively the same values of p_2 are required for the error decay as on the left, but notice how rapidly the error decays on the p_1 axis since there is practically no movement of the function in the vertical. This plot indicates that when selecting the 2S-DEIM indices

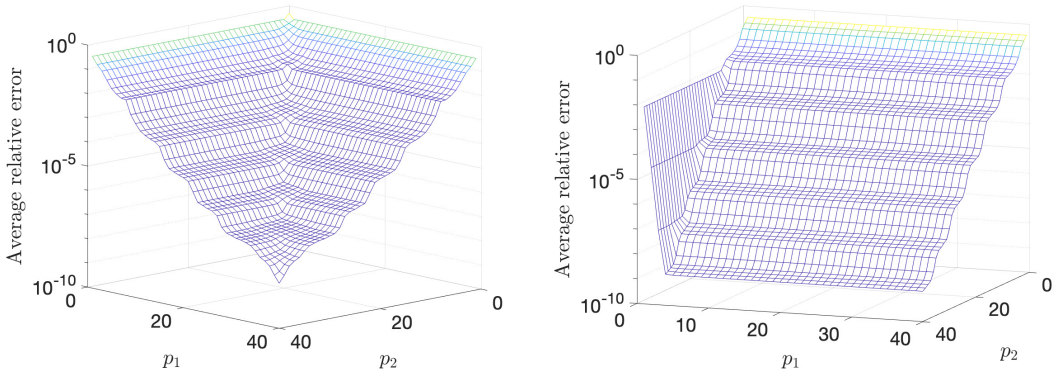

FIGURE 4.5: An average relative error mesh for p_1, p_2 on the generic grid $[1, 40] \times [1, 40]$ for functions (4.27) (left) and (4.28) (right)

we can consider a mere $p_1 = 2$ or $p_1 = 3$ rows of the function, of which p_2 column entries are selected of each for evaluation in the online phase.

We consider the same experiment for the classic vector DEIM approach. In Figure 4.6 we plot the singular values of the snapshot matrix $\boldsymbol{N}$ from (1.18) (left) and average relative error for increasing p from 1 to 60 (right) for both functions. We notice a

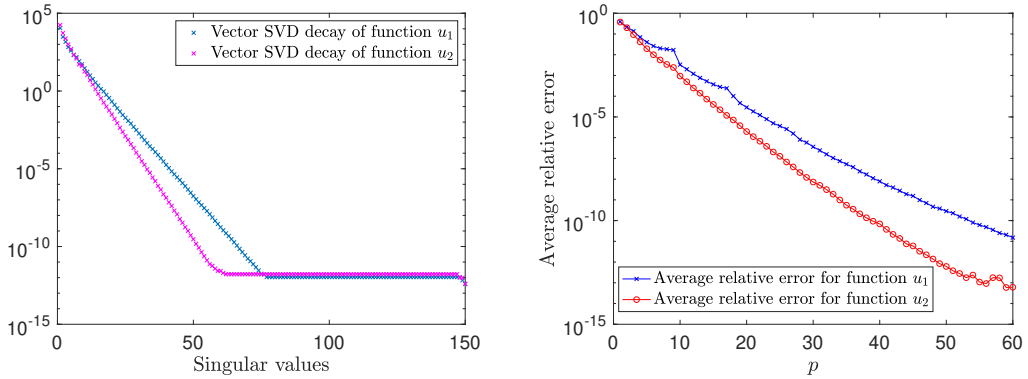

FIGURE 4.6: The singular value decays of the matrix $\boldsymbol{N}$ from (1.18) (left) and the average relative error of the approximation (1.16) for increasing p (right) for both functions $\mathbf{f}(\boldsymbol{u}_1, t) = \mathtt{vec}(\mathcal{F}(\boldsymbol{U}_1, t))$ and $\mathbf{f}(\boldsymbol{u}_2, t) = \mathtt{vec}(\mathcal{F}(\boldsymbol{U}_2, t))$

slightly accelerated decay of the singular values and the relative error for the second function, although the change is not nearly as emphasized as in the matrix case, which indicates how the matrix setting can exploit the geometric structure of the problem.□

Solution approximation of the full problem

Here we report on a selection of numerical experiments with the DYNAMIC 2S-POD-DEIM procedure on semilinnear matrix ODEs of the form (4.1). Once again, we compare the results with a standard vector procedure that applies standard POD-DEIM to the vectorized solution and nonlinear function snapshots. In particular, we

apply the (vectorized) adaptive procedure from Section 4.2.3 using the error $\|\boldsymbol{\xi} - \boldsymbol{V}_{\text{VEC}}\boldsymbol{V}_{\text{VEC}}^{\top}\boldsymbol{\xi}\|/\|\boldsymbol{\xi}\|$ for the selection criterion, where $\boldsymbol{\xi}$ is the vectorized snapshot, and $\boldsymbol{V}_{\text{VEC}}$ is the existing POD basis (either for the solution or nonlinear function snapshots). Notice that in the vector setting, we need to process as many nodes as the final space dimension, which depends on the tolerance τ. This is because no reduction takes place.

In all experiments CPU times are in seconds, and all bases are truncated using the criterion in (4.15). To illustrate the quality of the obtained numerical solution, we also report on the evaluation of the following average relative error norm

$$\bar{\mathcal{E}}(\boldsymbol{U}) = \frac{1}{n_{\mathrm{t}}} \sum_{\gamma=1}^{n_{\mathrm{t}}} \frac{\|\boldsymbol{U}^{(j)} - \widetilde{\boldsymbol{U}}^{(j)}\|_F}{\|\boldsymbol{U}^{(j)}\|_F}, \tag{4.29}$$

where $\widetilde{\boldsymbol{U}}^{(j)}$ represents either the DYNAMIC approximation, or the matricization of the VECTOR approximation and $\boldsymbol{U}^{(j)}$ is determined by means of the exponential Euler method applied to the original problem.

Table 4.4 shows the key numbers for the construction of the bases for both methods. For either $\boldsymbol{U}$ or $\mathcal{F}$ we report the number of PHASES, the number n_s of included snapshots, and the final space dimensions after the reduction procedure. We first describe the considered model problem and then comment on the numbers in Table 4.4 and on the detailed analysis illustrated in Table 4.5. We stress that these may be considered typical benchmark problems for our problem formulation. We also remark that for some of the experimental settings, the problem becomes symmetric, so that thanks to Remark 4.1 the two bases could be taken to be the same with further computational and memory savings. Nonetheless, we do not exploit this extra feature in the reported experiments.

For all examples the full matrix problem has dimension $n = 1000$, while the selected values for κ, $n_{\max}$ and n_{t} are displayed in Table 4.4 and Table 4.5 respectively.

Example 4.3. *The 2D Allen-Cahn equation [5].* Consider the equation[5]

$$u_t = \epsilon_1 \Delta u - \frac{1}{\epsilon_2^2}\left(u^3 - u\right), \quad \Omega = [a,b]^2, \quad t \in [0, t_f], \tag{4.30}$$

with initial condition $u(x, y, 0) = u_0$. The first example is referred to as AC 1. Following [145], we use the following problem parameters: $\epsilon_1 = 10^{-2}$, $\epsilon_2 = 1$, $a = 0$, $b = 2\pi$ and $t_f = 5$. Finally, we set $u_0 = 0.05 \sin x \cos y$ and impose homogeneous Dirichlet boundary conditions. The second problem (hereafter AC 2) is a mean curvature flow problem [60] of the form (4.30) with data as suggested e.g., in [88, Section 5.2.2], that is periodic boundary conditions, $\epsilon_1 = 1$, $a = -0.5$, $b = 0.5$, $t_f = 0.075$, with $u_0 = \tanh\left(\frac{0.4 - \sqrt{x^2+y^2}}{\sqrt{2}\epsilon_2}\right)$. Following [88, Section 5.2.2] we consider $\epsilon_2 \in \{0.01, 0.02, 0.04\}$. As ϵ_2 decreases stability reasons enforce the use of finer time

[5] Note that the linear term $-u$ is kept in the nonlinear part of the operator.

discretizations n_t, also leading to larger values of $n_{\max}$ and κ, as indicated in Table 4.4 and Table 4.5, where we report our experimental results. □

Example 4.4. *Reaction-convection-diffusion equation.* We consider the following reaction-convection-diffusion (hereafter RCD) problem, also presented in [38],

$$u_t = \epsilon_1 \Delta u + (u_x + u_y) + u(u - 0.5)(1 - u), \quad \Omega = [0, 1]^2, \quad t \in [0, 0.3]. \tag{4.31}$$

The initial solution is given by $u_0 = 0.3 + 256\,(x(1-x)y(1-y))^2$, while zero Neumann boundary conditions are imposed. In Table 4.4 and Table 4.5 we present results for $\epsilon_1 \in \{0.5, 0.05\}$. □

TABLE 4.4: Example 4.3: Performance of DYNAMIC and VECTOR algorithms for $n = 1000$.

PB.	$n_{\max}/\kappa$	Ξ	ALGORITHM	PHASES	n_s	ν_1/ν_2
AC 1	40/50	$\boldsymbol{U}$	DYNAMIC	1	8	9/2
			VECTOR	2	9	9
		$\mathcal{F}$	DYNAMIC	1	7	10/3
			VECTOR	2	9	9
AC 2 $\epsilon_2 = 0.04$	400/50	$\boldsymbol{U}$	DYNAMIC	1	2	15/15
			VECTOR	2	25	25
		$\mathcal{F}$	DYNAMIC	1	3	27/27
			VECTOR	2	40	40
AC 2 $\epsilon_2 = 0.02$	1200/70	$\boldsymbol{U}$	DYNAMIC	1	3	30/30
			VECTOR	1	28	28
		$\mathcal{F}$	DYNAMIC	1	4	39/39
			VECTOR	2	53	53
AC 2 $\epsilon_2 = 0.01$	5000/150	$\boldsymbol{U}$	DYNAMIC	1	3	62/62
			VECTOR	1	43	43
		$\mathcal{F}$	DYNAMIC	1	5	73/73
			VECTOR	2	92	92
RDC $\epsilon_1 = 0.5$	60/50	$\boldsymbol{U}$	DYNAMIC	1	3	10/10
			VECTOR	1	7	7
		$\mathcal{F}$	DYNAMIC	1	3	13/13
			VECTOR	2	11	11
RDC $\epsilon_1 = 0.05$	60/50	$\boldsymbol{U}$	DYNAMIC	1	4	14/14
			VECTOR	3	14	14
		$\mathcal{F}$	DYNAMIC	1	3	17/17
			VECTOR	3	34	34

Table 4.4 shows that for both the solution and the nonlinear function snapshots, the DYNAMIC procedure requires merely one phase and it retains snapshots at only a few of the time instances. On the other hand, the vector approach typically requires two or even three phases to complete the procedure. The dimension of the bases is not comparable for the matrix and vector approaches since these are subspaces of spaces of significantly different dimensions, namely $\mathbb{R}^n$ and $\mathbb{R}^{n^2}$ in the matrix and vector cases, respectively. Nonetheless, the memory requirements are largely in favor of the matrix approach, as shown in Table 4.5, where computational and memory details are reported. In particular, in Table 4.5 the offline timings are broken down into two main parts. The column BASIS TIME collects the cost of the Gram-Schmidt orthogonalization in the VECTOR setting, and the cumulative cost of Algorithm 4.1 (for both the solution and nonlinear function snapshots) for the DYNAMIC procedure.

TABLE 4.5: Computational time and storage requirements of 2S-POD-DEIM and standard vector POD-DEIM. CPU times are in seconds and $n = 1000$.

		OFFLINE			ONLINE		
PB.	METHOD	BASIS TIME	DEIM TIME	MEMORY	TIME (n_t)	MEMORY	REL. ERROR
AC 1	DYNAMIC	1.8	0.001	$200n$	0.009 (300)	$24n$	$1 \cdot 10^{-4}$
	VECTOR	0.6	0.228	$18n^2$	0.010 (300)	$18n^2$	$1 \cdot 10^{-4}$
AC 2 0.04	DYNAMIC	0.8	0.005	$200n$	0.010 (300)	$84n$	$3 \cdot 10^{-4}$
	VECTOR	8.4	3.745	$65n^2$	0.020 (300)	$65n^2$	$2 \cdot 10^{-4}$
AC 2 0.02	DYNAMIC	1.8	0.004	$280n$	0.140 (1000)	$138n$	$2 \cdot 10^{-4}$
	VECTOR	14.56	5.273	$81n^2$	0.120 (1000)	$81n^2$	$3 \cdot 10^{-5}$
AC 2 0.01	DYNAMIC	5.3	0.008	$600n$	0.820 (2000)	$270n$	$5 \cdot 10^{-4}$
	VECTOR	46.2	13.820	$135n^2$	0.420 (2000)	$135n^2$	$2 \cdot 10^{-4}$
RDC 0.5	DYNAMIC	0.8	0.001	$200n$	0.008 (300)	$46n$	$2 \cdot 10^{-4}$
	VECTOR	0.6	0.277	$18n^2$	0.010 (300)	$18n^2$	$1 \cdot 10^{-4}$
RDC 0.05	DYNAMIC	0.9	0.001	$200n$	0.010 (300)	$62n$	$2 \cdot 10^{-4}$
	VECTOR	4.1	2.297	$48n^2$	0.010 (300)	$48n^2$	$1 \cdot 10^{-4}$

Column DEIM TIME reports the time required to determine the interpolation indices by `q-deim`. The "online times" report the cost to simulate both reduced order models at n_t timesteps with the relevant exponential Euler scheme, that is (2.3) for VECTOR and (2.5) for DYNAMIC. The relative approximation error in (4.29) is also displayed, together with the total memory requirements for the offline and online parts. The memory requirements include the storage of all processed snapshots for VECTOR, whereas for 2S-POD-DEIM the storage of $\widetilde{\boldsymbol{V}}_i$ and $\check{\boldsymbol{W}}_i$ from Algorithm 4.1, for both the solution and nonlinear function snapshots. This quantity is always equal to $4\kappa \cdot n$ for the DYNAMIC setting.

We point out the significant gain in basis construction time for the DYNAMIC procedure, mainly related to the low number of snapshots employed (cf. Table 4.4). Furthermore, the DYNAMIC procedure enjoys a massive gain in memory requirements, for very comparable online time and final average errors.

For the reaction-convection-diffusion example we also analyze the dependence of the number n_s of retained snapshots on the threshold τ of the two procedures, having fixed $n_{\max} = 60$. For the vector procedure n_s increases as the tolerance τ decreases, whereas for the dynamic procedure n_s remains nearly constant for changing τ. This ultimately indicates that the offline cost will increase for the VECTOR procedure if a richer basis is required. □

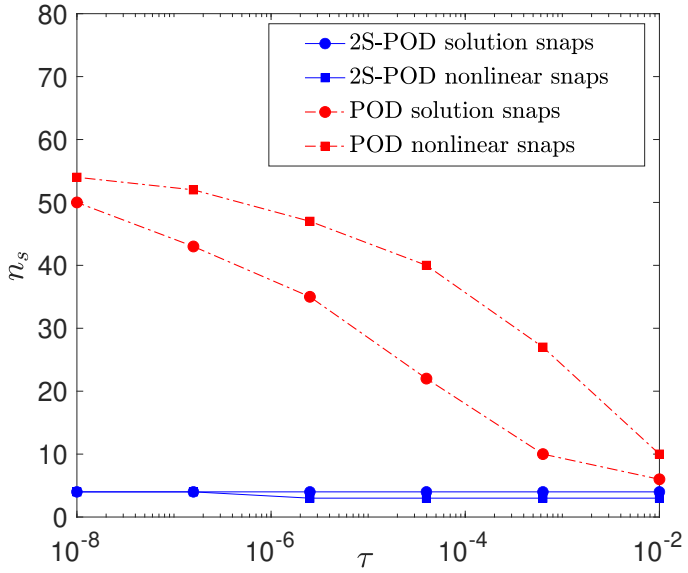

FIGURE 4.7: Example 4.4. $n = 1000$. Number of retained snapshots with respect to τ, for $\epsilon_1 = 0.05$.

4.3 Concluding remarks

In this chapter, we have proposed a matrix-oriented POD-DEIM type order reduction strategy to efficiently handle the numerical solution of semilinear matrix differential equations in two space variables. By introducing a novel interpretation of the POD when applied to functions in two variables, we devised a new two-sided discrete interpolation strategy that can preserve the symmetric structure in the original nonlinear function and approximate solution. The numerical treatment of the matrix reduced-order differential problem can take full advantage of both the small dimension and the matrix framework by exploiting effective exponential integrators. Our very encouraging numerical experiments show that the new procedure can dramatically decrease memory and CPU time requirements for the function reduction procedure in the so-called offline phase. Moreover, we illustrated that the reduced low-dimensional matrix differential equation can be numerically solved in a rapid online phase without sacrificing too much accuracy.

In the following chapter, we explore the advantages of the new matrix-oriented procedure in the 3D setting. Furthermore, we also extend the procedure to treat coupled systems of equations to tackle problems stemming from real-life simulations.

Chapter 5

The Multidimensional Setting[1]

Several complex real-life phenomena are modeled by semilinear PDEs in high dimension ($d \geq 2$), and also by coupled systems of these PDEs of the form

$$\begin{cases} \dot{u}_1 &= \mathcal{L}_1(u_1) + f_1(\nabla u_1, u_1, \dots, u_g, t) \\ &\vdots \\ \dot{u}_g &= \mathcal{L}_g(u_g) + f_g(\nabla u_g, u_1, \dots, u_g, t). \end{cases} \tag{5.1}$$

Here $u_i = u_i(\boldsymbol{x}, t) \in S_{\text{PDE}}$, where S_{PDE} is an appropriate space, with $\boldsymbol{x} \in \Omega \subset \mathbb{R}^d$, $t \in [0, t_f] = T$ and suitable initial and boundary conditions, for all $i = 1, 2, \dots, g$. We restrict our attention to the two-dimensional and three-dimensional cases, that is $d = 2, 3$. In this setting we assume that $\mathcal{L}_i : S_{\text{PDE}} \to \mathbb{R}$ is linear in u_i, typically a diffusion operator, whereas $f_i : S_{\text{PDE}} \times T \to \mathbb{R}$ is assumed to be nonlinear in $(\nabla u_i, u_1, \dots u_g)$ and t. Here we consider, without loss of generality, that any forcing terms of the form $c_i(\boldsymbol{x})$ are either absent or incorporated into the nonlinear function. PDEs of the form (5.1) describe mathematical models in several scientific fields, such as chemistry [53, 158], biology [68, 85, 118] and medicine [139]. For further applications see [110], [111],[128],[157], and references therein.

The method of lines (MOL) based on space discretizations, such as finite differences and spectral methods rewrites the system (5.1) as a system of ODEs of the form

$$\begin{cases} \dot{\boldsymbol{u}}_1(t) &= \boldsymbol{L}_1 \boldsymbol{u}_1(t) + \boldsymbol{f}_1(\boldsymbol{D}_1 \boldsymbol{u}_1, \boldsymbol{u}_1, \dots, \boldsymbol{u}_g, t), \\ &\vdots \\ \dot{\boldsymbol{u}}_g(t) &= \boldsymbol{L}_g \boldsymbol{u}_g(t) + \boldsymbol{f}_g(\boldsymbol{D}_g \boldsymbol{u}_g, \boldsymbol{u}_1, \dots, \boldsymbol{u}_g, t), \end{cases} \tag{5.2}$$

where $\boldsymbol{u}_i \in S_{\text{VEC}}$ and each function $\boldsymbol{f}_i : S_{\text{VEC}} \times T \to \mathbb{R}^N$ represents the function f_i evaluated at the entries of the set of vectors $\{\boldsymbol{u}_i\}_{i=1}^g$ and $\boldsymbol{D}_i \boldsymbol{u}_i$. In this setting, $\boldsymbol{L}_i \in \mathbb{R}^{N \times N}$ accounts for the discretization of the linear operator $\mathcal{L}_i$ on the selected basis and $\boldsymbol{D}_i \in \mathbb{R}^{N \times N}$ for that of the gradient. In the recent literature several model order reduction techniques have been applied to reduce the dimension and the complexity

[1] An earlier version of this chapter is set to appear in J. Comput. Dyn. [93].

of the system (5.2); see e.g., [42, 89–91, 100, 135, 161]. This includes methods such as POD-DEIM, reduced basis methods, as well as reduction methods based on lifting transformations; see e.g., [73, 101].

Instead, following the procedure described in Section 1.2 for each of the g coupled equations, if the system (5.1) is discretized on a tensor basis for certain choices of the physical domain Ω, the discrete system of ODEs (5.2) can be written directly in matrix or tensor form, so that

$$\begin{cases} \dot{\boldsymbol{\mathcal{U}}}_1 &= \mathcal{A}_1(\boldsymbol{\mathcal{U}}_1) + \mathcal{F}_1\left(\mathcal{D}_1(\boldsymbol{\mathcal{U}}_1), \boldsymbol{\mathcal{U}}_1, \boldsymbol{\mathcal{U}}_2, \cdots, \boldsymbol{\mathcal{U}}_g, t\right) \\ & \vdots \\ \dot{\boldsymbol{\mathcal{U}}}_g &= \mathcal{A}_g(\boldsymbol{\mathcal{U}}_g) + \mathcal{F}_g\left(\mathcal{D}_g(\boldsymbol{\mathcal{U}}_g), \boldsymbol{\mathcal{U}}_1, \boldsymbol{\mathcal{U}}_2, \cdots, \boldsymbol{\mathcal{U}}_g, t\right), \end{cases} \tag{5.3}$$

with suitable initial conditions. In this setting, as shown in Section 1.2,

$$\mathcal{A}_i(\boldsymbol{\mathcal{U}}_i) := \sum_{m=1}^{d} \boldsymbol{\mathcal{U}}_i \times_m \boldsymbol{A}_{mi} \quad \text{and} \quad \mathcal{D}_i(\boldsymbol{\mathcal{U}}_i) := \sum_{m=1}^{d} \boldsymbol{\mathcal{U}}_i \times_m \boldsymbol{D}_{mi}. \tag{5.4}$$

where $\boldsymbol{A}_{mi} \in \mathbb{R}^{n_m \times n_m}$ and $\boldsymbol{D}_{mi} \in \mathbb{R}^{n_m \times n_m}$ contain the approximation of the second and first derivatives respectively in the x_m direction, for $i = 1, 2, \ldots, g$, as well as the boundary conditions. Moreover, the function $\mathcal{F}_i : S_{\text{TEN}} \times T \to \mathbb{R}^{n_1 \times \cdots \times n_d}$ respresents the function f_i evaluated at the entries of the arrays $\{\boldsymbol{\mathcal{U}}_i\}_{i=1}^{g}$ and $\mathcal{D}_i(\boldsymbol{\mathcal{U}}_i)$, where $\boldsymbol{\mathcal{U}}_i \in S_{\text{TEN}}$.

This chapter aims to experimentally explore the extension of the 2S-POD-DEIM algorithm from Section 4.2 to the tensor setting and to illustrate the applicability of the new method to systems of ODEs to find approximate solutions to the discrete system of ODEs (5.3). To illustrate this, we do not consider the complete DYNAMIC algorithm from Section 4.2.3 for constructing the approximation space. Instead, we consider a simplified version of the algorithm, that is, the VANILLA algorithm discussed in Section 4.2.6.

The idea of constructing low-rank tensor decompositions is certainly not new in the general context of model reduction of multidimensional or multiparameter problems; see, e.g., [2, 3, 136]. However, after reduction, the vector form of the reduced problem is typically retained for the online phase, therefore losing the structural and computational benefits offered by the tensor form of the problem. Here we aim to retain the tensor structure of the model for both the *reduction* and *integration* phases when $d = 3$, resulting in a significant gain in both CPU time and memory requirements for both the offline and online phases of the procedure (see Example 5.3). We also tailor the basis construction to the semilinear problem at hand so that only one snapshot is treated at a time and discarded, rather than performing computations on all snapshots simultaneously. See also the discussion in Section 5.1 regarding the relation of our procedure to the Tucker decomposition.

To simplify the presentation, we will consider the case where $n_1 = \cdots = n_d = n$ for the rest of the chapter, so that $N = n^d$. The extension to the more general case where $n_1 \neq \cdots \neq n_d$ is, however, possible.

5.1 POD-DEIM in the Multidimensional Setting

In this section we extend POD-DEIM to the tensor setting. We illustrate the procedure for systems of the form (5.3) with $g = 1$ and a gradient-independent nonlinearity. The extension to the case of general g is presented later in the chapter. In particular, we want to approximate the solution $\boldsymbol{\mathcal{U}}(t) \in \mathbb{R}^{n\times\cdots\times n}$, for $t \in T$, of the equation

$$\dot{\boldsymbol{\mathcal{U}}} = \mathcal{A}(\boldsymbol{\mathcal{U}}) + \mathcal{F}(\boldsymbol{\mathcal{U}}, t), \qquad \boldsymbol{\mathcal{U}}(0) = \boldsymbol{\mathcal{U}}_0, \tag{5.5}$$

by constructing d low-dimensional basis matrices (one for each spatial mode $m = 1, \ldots, d$) $\boldsymbol{V}_m \in \mathbb{R}^{n\times k_m}$, with $k_m \ll n$, to approximate $\boldsymbol{\mathcal{U}}(t)$ in low dimension, for all $t \in T$. To this end, given a set of snapshots $\{\boldsymbol{\mathcal{U}}(t_j)\}_{j=1}^{n_s}$ with $\boldsymbol{\mathcal{U}}(t_j) \in \mathbb{R}^{n\times\cdots\times n}$, each matrix $\boldsymbol{V}_m$ is constructed in order to approximate the left range space of the matrix

$$\boldsymbol{\mathcal{S}}_{(m)} = \left(\boldsymbol{\mathcal{U}}_{(m)}(t_1), \ldots, \boldsymbol{\mathcal{U}}_{(m)}(t_{n_s})\right) \in \mathbb{R}^{n\times n_s n^{d-1}}, \quad \text{for} \quad m = 1, \ldots, d,$$

where $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{n\times\cdots\times n\times n_s}$ is as a snapshot tensor[2] of order $d+1$, as defined in Section 1.1, containing a collection of all the snapshots and m represents the mode along which the tensor is unfolded. Forming or storing the matrix $\boldsymbol{\mathcal{S}}_{(m)}$ is too computationally demanding, even for moderate n and n_s. Instead, the approximation spaces are updated one snapshot at a time.

To this end, we determine the sequentially truncated higher order SVD[3] (STHOSVD) [159] of each snapshot $\boldsymbol{\mathcal{U}}(t_j)$, so that $\boldsymbol{\mathcal{U}}(t_j) \approx \boldsymbol{\mathcal{\Sigma}}(t_j) \times_{m=1}^{d} \mathsf{V}_m^{(j)}$, where $\mathsf{V}_m^{(j)}$ contains the dominant left singular vectors of $\boldsymbol{\mathcal{U}}_{(m)}(t_j)$, truncated with respect to a tolerance τ chosen a-priori. Furthermore, $\boldsymbol{\mathcal{\Sigma}}(t_j)$ is the core tensor related to the STHOSVD of $\boldsymbol{\mathcal{U}}(t_j)$, and is generally defined as $\boldsymbol{\mathcal{\Sigma}}(t_j) = \boldsymbol{\mathcal{U}}(t_j) \times_{m=1}^{d} (\mathsf{V}_m^{(j)})^\top$. Note, however, that in our procedure it is not necessary to explicitly compute the core tensors $\boldsymbol{\mathcal{\Sigma}}(t_j)$, since only the matrices $\mathsf{V}_m^{(j)}$ are required. More precisely, the approximation space range$(\boldsymbol{V}_m)$ in each mode is updated by orthogonal reduction of the augmented matrices

$$\left(\boldsymbol{V}_m, \mathsf{V}_m^{(j)}\right), \qquad m = 1, 2, \ldots, d,$$

with respect to τ. If desired, the matrix $\mathsf{V}_m^{(j)}$ can also be weighted by the square root of the singular values in that mode, as is done in (4.24), to keep track of the relevance of each of the spatial directions. Through this procedure, each snapshot can be discarded after it has been processed. We will refer to this as the higher-order

[2] We emphasize that this large dimensional tensor will never be explicitly formed or stored.
[3] For the case $d = 2$, however, we just use the standard MATLAB `SVD` function.

POD (HO-POD) approximation. We call this a higher-order approximation since the standard POD projection is only a one-sided approximation, whereas this procedure is two-sided or three-sided for $d = 2$ and $d = 3$, respectively.

This type of approximation can also be interpreted as a Tucker$_d$ decomposition (see e.g., [97, Section 4] and [136]) of the snapshot tensor $\boldsymbol{\mathcal{S}} \in \mathbb{R}^{n\times\cdots\times n\times n_s}$ of the form

$$\boldsymbol{\mathcal{S}} \approx \widehat{\boldsymbol{\mathcal{S}}} \times_1 \boldsymbol{V}_1 \times_2 \cdots \times_d \boldsymbol{V}_d \times_{d+1} \boldsymbol{I}_{n_s}, \quad \widehat{\boldsymbol{\mathcal{S}}} \in \mathbb{R}^{k_1\times\cdots\times k_d\times n_s},$$

since the principal components of each of the first d modes are analyzed. Instead of determining the core tensor $\widehat{\boldsymbol{\mathcal{S}}}$, which contains a collection of low-dimensional approximations to all the given snapshots, we aim to use the basis matrices to approximate the solution of (5.5) at time instances other than the ones considered for the snapshots.

More precisely, we look for an approximation to the solution of (5.5) of the form $\boldsymbol{\mathcal{U}}(t) \approx \widetilde{\boldsymbol{\mathcal{U}}}(t) := \widehat{\boldsymbol{\mathcal{U}}}(t) \times_{m=1}^{d} \boldsymbol{V}_m$, where $\widehat{\boldsymbol{\mathcal{U}}}(t) \in \mathbb{R}^{k_1\times\cdots\times k_d}$ ($k_m \ll n$) satisfies the low-dimensional equation

$$\dot{\widehat{\boldsymbol{\mathcal{U}}}} = \widehat{\mathcal{A}}(\widehat{\boldsymbol{\mathcal{U}}}) + \widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}, t), \qquad \widehat{\boldsymbol{\mathcal{U}}}(0) = \widehat{\boldsymbol{\mathcal{U}}}_0, \tag{5.6}$$

where

$$\widehat{\mathcal{A}}(\widehat{\boldsymbol{\mathcal{U}}}) := \sum_{m=1}^{d} \widehat{\boldsymbol{\mathcal{U}}} \times_m \widehat{\boldsymbol{A}}_m, \quad \widehat{\boldsymbol{A}}_m = \boldsymbol{V}_m^\top \boldsymbol{A}_m \boldsymbol{V}_m, \quad \widehat{\boldsymbol{\mathcal{U}}}(0) = \boldsymbol{\mathcal{U}}_0 \times_{m=1}^{d} \boldsymbol{V}_m^\top \tag{5.7}$$

and

$$\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}, t) = \mathcal{F}(\widetilde{\boldsymbol{\mathcal{U}}}(t), t) \times_{m=1}^{d} \boldsymbol{V}_m^\top. \tag{5.8}$$

In a similar fashion to the matrix methods discussed in Chapter 2, the time discretization of (5.6) can be applied directly to the equation in the matrix or tensor form. To this end, thanks to the semilinear nature of the equation, we consider the IMEX 2–SBDF scheme presented in Section 2.1; see e.g., [14, 51]. Any of the other schemes presented in Section 2.1 can also be considered. To this end, if $\widehat{\boldsymbol{\mathcal{U}}}^{(j)}$ is an approximation of $\widehat{\boldsymbol{\mathcal{U}}}(\mathrm{t}_j)$, then the linear system

$$(3\widehat{\mathcal{I}} - 2h\widehat{\mathcal{A}})(\widehat{\boldsymbol{\mathcal{U}}}^{(j)}) = \widehat{\mathcal{G}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)}, \widehat{\boldsymbol{\mathcal{U}}}^{(j-2)}) \tag{5.9}$$

needs to be solved for each t_j, where

$$\widehat{\mathcal{G}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)}, \widehat{\boldsymbol{\mathcal{U}}}^{(j-2)}) = 4\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)} - \widehat{\boldsymbol{\mathcal{U}}}^{(j-2)} + 2h\left(2\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)}, \mathrm{t}_{j-1}) - \widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-2)}, \mathrm{t}_{j-2})\right),$$

and $\widehat{\mathcal{I}} : \mathbb{R}^{k_1\times\cdots\times k_d} \to \mathbb{R}^{k_1\times\cdots\times k_d}$ is the identity operator in the reduced dimension. To initiate the procedure, $\widehat{\boldsymbol{\mathcal{U}}}^{(1)}$ can be determined by an IMEX Euler scheme from the known array $\widehat{\boldsymbol{\mathcal{U}}}^{(0)}$. For the case $d = 2$, the linear system (5.9) corresponds precisely to (2.11) with $s = 2$ and $\boldsymbol{C} \equiv 0$. In what follows we discuss how (5.9) is solved for $d = 3$.

To solve (5.9) in tensor form for the case $d > 3$, without reverting to vectorization is still an open problem to the best of our knowledge.

The solution of the linear system (5.9) in Tensor Form

The solution of (5.9) is not trivial when $d \geq 3$, given that the matrices $\widehat{\boldsymbol{A}}_m$, for $m = 1, \ldots, d$ are necessarily dense due to the projection. Nevertheless, a direct method specifically designed for dense third-order (i.e., $d = 3$) tensor linear systems has recently been introduced in [143] for tensors with a rank-one right-hand side. Here we illustrate how this method can be applied to solve the linear system (5.9), accounting for a right-hand side with a rank greater than one. To ease the readability, we drop the superscript (j) for the description of the inner solver when it is clear from the context.

By definition, when $d = 3$, the left hand side of (5.9) can be vectorized as

$$\left(\boldsymbol{I}_{k_3} \otimes 3\boldsymbol{I}_{k_2} \otimes \boldsymbol{I}_{k_1} - \boldsymbol{I}_{k_3} \otimes \boldsymbol{I}_{k_2} \otimes 2h\widehat{\boldsymbol{A}}_1 - \boldsymbol{I}_{k_3} \otimes 2h\widehat{\boldsymbol{A}}_2 \otimes \boldsymbol{I}_{k_1} - 2h\widehat{\boldsymbol{A}}_3 \otimes \boldsymbol{I}_{k_2} \otimes \boldsymbol{I}_{k_1}\right) \operatorname{vec}\left(\widehat{\boldsymbol{U}}\right).$$

Therefore, if we let $\widehat{\boldsymbol{X}} = \widehat{\boldsymbol{U}}_{(3)}^{\top} \in \mathbb{R}^{k_1 k_2 \times k_3}$, then by the use of property (1.2), (5.9) can be recast into the Sylvester equation

$$\left(3\,\boldsymbol{I}_{k_2} \otimes \boldsymbol{I}_{k_1} - \boldsymbol{I}_{k_2} \otimes 2\,h\widehat{\boldsymbol{A}}_1 - 2\,h\widehat{\boldsymbol{A}}_2 \otimes \boldsymbol{I}_{k_1}\right)\widehat{\boldsymbol{X}} + \widehat{\boldsymbol{X}}\left(-2\,h\widehat{\boldsymbol{A}}_3^{\top}\right) = \widehat{\boldsymbol{G}}. \tag{5.10}$$

Here $\widehat{\boldsymbol{G}} = \widehat{\mathcal{G}}(\widehat{\boldsymbol{X}}^{(j-1)}, \widehat{\boldsymbol{X}}^{(j-2)}) \in \mathbb{R}^{k_1 k_2 \times k_3}$. Due to the large left dimension of this Sylvester equation, solving this directly is still not feasible. Instead, as is shown in [143], it is possible to solve a sequence of much smaller Sylvester equations. To this end, let $\widehat{\boldsymbol{A}}_3^{\top} = \boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^{\top}$ be the real Schur decomposition of $\widehat{\boldsymbol{A}}_3^{\top}$, with $\boldsymbol{Q} \in \mathbb{R}^{k_3 \times k_3}$ orthogonal. Then, if $\widehat{\boldsymbol{Y}} = \widehat{\boldsymbol{X}}\boldsymbol{Q}$, it holds that

$$\left(3\,\boldsymbol{I}_{k_2} \otimes \boldsymbol{I}_{k_1} - \boldsymbol{I}_{k_2} \otimes 2\,h\widehat{\boldsymbol{A}}_1 - 2\,h\widehat{\boldsymbol{A}}_2 \otimes \boldsymbol{I}_{k_1}\right)\widehat{\boldsymbol{Y}} + \widehat{\boldsymbol{Y}}\,(-2\,h\boldsymbol{R}) = \widehat{\boldsymbol{G}}\boldsymbol{Q}, \tag{5.11}$$

where $\boldsymbol{R} \in \mathbb{R}^{k_3 \times k_3}$ is upper-triangular[4].

To solve (5.11), let $\widehat{\boldsymbol{Y}} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{k_3}) \in \mathbb{R}^{k_1 k_2 \times k_3}$ and $\widehat{\boldsymbol{G}}\boldsymbol{Q} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_{k_3}) \in \mathbb{R}^{k_1 k_2 \times k_3}$, where $\boldsymbol{z}_\ell, \boldsymbol{h}_\ell \in \mathbb{R}^{k_1 k_2}$. The aim is to determine $\widehat{\boldsymbol{Y}}$ by solving for each column $\boldsymbol{z}_\ell$ separately. Since $\boldsymbol{R}$ is upper-triangular, each column $\boldsymbol{z}_\ell$, for $\ell = 1, \ldots, k_3$ is determined by solving

$$\left(3\,\boldsymbol{I}_{k_2} \otimes \boldsymbol{I}_{k_1} - \boldsymbol{I}_{k_2} \otimes 2\,h\widehat{\boldsymbol{A}}_1 - 2\,h\widehat{\boldsymbol{A}}_2 \otimes \boldsymbol{I}_{k_1}\right)\boldsymbol{z}_\ell + \boldsymbol{z}_\ell\,(-2\,h\boldsymbol{R}_{\ell,\ell}) = \boldsymbol{h}_\ell + 2h\boldsymbol{j}_{\ell-1}, \tag{5.12}$$

where

$$\boldsymbol{j}_{\ell-1} = (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_{\ell-1})\,\boldsymbol{R}_{\ell,1:\ell-1}, \quad \text{with} \quad \boldsymbol{j}_0 \equiv 0. \tag{5.13}$$

[4]We assume that $\boldsymbol{A}_3$ has real eigenvalues so that $\boldsymbol{R}$ is upper-triangular. If $\boldsymbol{A}_3$ had complex eigenvalues we would rather consider the complex Schur decomposition, in which case $\boldsymbol{R} \in \mathbb{C}^{k_3 \times k_3}$ would retain the upper-triangular structure.

Instead of solving the large linear system (5.12) for each $\boldsymbol{z}_\ell$ we reshape the vectors in such a way that $\boldsymbol{z}_\ell = \mathrm{vec}\,(\boldsymbol{Z}_\ell)$, $\boldsymbol{h}_\ell = \mathrm{vec}\,(\boldsymbol{H}_\ell)$ and $\boldsymbol{j}_\ell = \mathrm{vec}\,(\boldsymbol{J}_\ell)$ with $\boldsymbol{Z}_\ell, \boldsymbol{H}_\ell, \boldsymbol{J}_\ell \in \mathbb{R}^{k_1 \times k_2}$. Then, as a result of property (1.1), the matrix $\boldsymbol{Z}_\ell$ is the solution of the smaller Sylvester equation

$$\left((3 - 2\,h\boldsymbol{R}_{\ell,\ell})\boldsymbol{I}_{k_1} - 2\,h\widehat{\boldsymbol{A}}_1\right)\boldsymbol{Z}_\ell + \boldsymbol{Z}_\ell\left(-2\,h\widehat{\boldsymbol{A}}_2^\top\right) = \boldsymbol{H}_\ell + 2\,h\boldsymbol{J}_{\ell-1}, \tag{5.14}$$

for $\ell = 1, 2, \ldots, k_3$, which can be efficiently solved by one of the direct methods discussed in Section 2.1.2.

Finally, recalling that $\widehat{\boldsymbol{Y}} = \widehat{\boldsymbol{X}}\boldsymbol{Q}$, the solution of (5.9), unfolded in the third mode, is then given by the transformation

$$\widehat{\boldsymbol{\mathcal{U}}}_{(3)} = (\widehat{\boldsymbol{Y}}\boldsymbol{Q}^\top)^\top = \boldsymbol{Q}\left(\boldsymbol{z}_1^\top; \ldots; \boldsymbol{z}_{k_3}^\top\right) \in \mathbb{R}^{k_3 \times k_1 k_2},$$

where each $\boldsymbol{z}_\ell$ for $\ell = 1, \ldots, k_3$ is determined by solving (5.14) and setting $\boldsymbol{z}_\ell = \mathrm{vec}\,(\boldsymbol{Z}_\ell)$.

In the particular case where all coefficient matrices are symmetric and positive definite, the procedure can be even further accelerated; see [143] for further details. We refer to this inner solver as the T3-SYLV solver.

Interpolation of the nonlinear function by HO-DEIM

To determine the right-hand side $\widehat{\mathcal{G}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)}, \widehat{\boldsymbol{\mathcal{U}}}^{(j-2)})$ at each $\mathfrak{t}_j$, it is required to evaluate the nonlinear function in full dimension, as per the definition of $\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}, t)$. Instead, we interpolate the nonlinear function through a higher order version of DEIM. Consider the d low-dimensional orthonormal matrices $\boldsymbol{\Phi}_m \in \mathbb{R}^{n \times p_m}$, with $p_m \ll n$, determined as the output of HO-POD of the set of nonlinear snapshots $\{\mathcal{F}(\boldsymbol{\mathcal{U}}(t_j), t_j)\}_{j=1}^{n_s}$, for $m = 1, 2, \ldots, d$. Furthermore, consider the d selection matrices $\boldsymbol{P}_m \in \mathbb{R}^{n \times p_m}$, given as the output of `q-deim` with input $\boldsymbol{\Phi}_m^\top$, for $m = 1, 2, \ldots, d$. The HO-DEIM approximation of (5.8) is then given by

$$\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}, t) \approx \mathcal{F}(\widetilde{\boldsymbol{\mathcal{U}}}, t) \bigtimes_{m=1}^{d} \boldsymbol{V}_m^\top \boldsymbol{\Phi}_m (\boldsymbol{P}_m^\top \boldsymbol{\Phi}_m)^{-1} \boldsymbol{P}_m^\top. \tag{5.15}$$

If $\mathcal{F}$ is evaluated elementwise at the components of $\widetilde{\boldsymbol{\mathcal{U}}}$, then it holds that

$$\widehat{\mathcal{F}(\widetilde{\boldsymbol{\mathcal{U}}}, t)} := \mathcal{F}(\widetilde{\boldsymbol{\mathcal{U}}}, t) \bigtimes_{m=1}^{d} \boldsymbol{P}_m^\top = \mathcal{F}(\widetilde{\boldsymbol{\mathcal{U}}} \bigtimes_{m=1}^{d} \boldsymbol{P}_m^\top, t). \tag{5.16}$$

Notice that $\mathcal{F}$ is then evaluated at $p_1 p_2 \cdots p_d \ll n^d$ entries. Next, we remark on a potential strategy for further reducing the online cost of the procedure. This strategy is, however, not considered in our implementations.

Remark 5.1. *For certain nonlinear functions, the evaluation of $p_1 p_2 \cdots p_d$ entries online may not be feasible. One possibility that can be considered is to further approximate the* HO-DEIM *reduced nonlinear function by a matrix-DEIM (MDEIM) type of interpolation (see e.g., [33, 119]). More precisely, if we consider the* HO-DEIM *approximations* (5.15) *and* (5.16) *then the snapshot matrix $\check{\boldsymbol{N}} = [\check{\boldsymbol{f}}(\mathrm{t}_1), \ldots, \check{\boldsymbol{f}}(\mathrm{t}_{n_s})] \in \mathbb{R}^{p_1 \cdots p_d \times n_s}$, can be considered, where*

$$\check{\boldsymbol{f}}(\mathrm{t}_j) = \mathrm{vec}\left(\widehat{\mathcal{F}(\widetilde{\boldsymbol{\mathcal{U}}}, \mathrm{t}_j)} \bigtimes_{m=1}^{d} (\boldsymbol{P}_m^\top \boldsymbol{\Phi}_m)^{-1} \right) \in \mathbb{R}^{p_1 \cdots p_d}.$$

If we define $\boldsymbol{\mathcal{M}}_q \in \mathbb{R}^{p_1 \times \cdots \times p_d}$ as the tensorization of the qth column of $\check{\boldsymbol{M}} \in \mathbb{R}^{p_1 \cdots p_d \times \mathfrak{p}}$ - the matrix of dominant left singular vectors of $\check{\boldsymbol{N}}$ - then

$$\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}, t) \approx \sum_{q=1}^{\mathfrak{p}} c_q \boldsymbol{\mathcal{M}}_q \bigtimes_{m=1}^{d} \boldsymbol{V}_m^\top \boldsymbol{\Phi}_m, \qquad c_q = \left[\left(\check{\boldsymbol{P}}^\top \check{\boldsymbol{M}} \right)^{-1} \boldsymbol{f}\left(\check{\boldsymbol{P}}^\top \mathrm{vec}\left(\widetilde{\boldsymbol{\mathcal{U}}} \bigtimes_{m=1}^{d} \boldsymbol{P}_m^\top \right), t \right) \right]_q,$$

where the columns of $\check{\boldsymbol{P}} \in \mathbb{R}^{p_1 \cdots p_d \times \mathfrak{p}}$ are related to the `q-deim` *interpolation indices of the matrix $\check{\boldsymbol{M}}$. Note that the nonlinear function is now evaluated at $\mathfrak{p} \ll p_1 \cdots p_d$ entries, even though no vectors of length N need to be stored. This type of approximation has a couple of drawbacks, however. Firstly, to determine the snapshots $\check{\boldsymbol{f}}(t_j)$, another (far cheaper) offline phase will be required. Moreover, potential structural properties such as symmetries, which are preserved by the approximation* (5.15)*, may be destroyed by the vectorization; see also the discussion in Section 4.2.2.*

We next provide an error bound for the HO-DEIM approximation (5.15), where we recall that the matrices $\boldsymbol{\Phi}_m$, for $m = 1, 2, \ldots, d$ all have orthonormal columns. This bound is a direct extension to the tensor setting of [41, Lemma 3.2]. A similar result is presented in Proposition 4.1 for $d = 2$.

Proposition 5.1. *Let $\boldsymbol{\mathfrak{D}}_m = \boldsymbol{\Phi}_m (\boldsymbol{P}_m^\top \boldsymbol{\Phi}_m)^{-1} \boldsymbol{P}_m^\top$, and consider an arbitrary tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n \times \cdots \times n}$, so that*

$$\widetilde{\boldsymbol{\mathcal{F}}} = \boldsymbol{\mathcal{F}} \bigtimes_{m=1}^{d} \boldsymbol{\Phi}_m (\boldsymbol{P}_m^\top \boldsymbol{\Phi}_m)^{-1} \boldsymbol{P}_m^\top = \boldsymbol{\mathcal{F}} \bigtimes_{m=1}^{d} \boldsymbol{\mathfrak{D}}_m.$$

Then,

$$\|\boldsymbol{\mathcal{F}} - \widetilde{\boldsymbol{\mathcal{F}}}\|_F \leq c_1 c_2 \cdots c_d \, \|\boldsymbol{\mathcal{F}} - \boldsymbol{\mathcal{F}} \bigtimes_{m=1}^{d} \boldsymbol{\Phi}_m \boldsymbol{\Phi}_m^\top\|_F \tag{5.17}$$

where $c_m = \left\| (\boldsymbol{P}_m^\top \boldsymbol{\Phi}_m)^{-1} \right\|_2$, for $m = 1, \ldots, d$.

Proof. Let $\boldsymbol{f} = \mathrm{vec}(\boldsymbol{\mathcal{F}}) \in \mathbb{R}^N$. Then, by the properties of the Kronecker product

$$\begin{aligned} \|\boldsymbol{\mathcal{F}} - \widetilde{\boldsymbol{\mathcal{F}}}\|_F &= \|\mathrm{vec}(\boldsymbol{\mathcal{F}}) - \mathrm{vec}(\widetilde{\boldsymbol{\mathcal{F}}})\|_2 = \|\boldsymbol{f} - (\boldsymbol{\mathfrak{D}}_d \otimes \cdots \otimes \boldsymbol{\mathfrak{D}}_1)\boldsymbol{f}\|_2 \\ &= \left\| \boldsymbol{f} - (\boldsymbol{\Phi}_d \otimes \cdots \otimes \boldsymbol{\Phi}_1) \left((\boldsymbol{P}_d \otimes \cdots \otimes \boldsymbol{P}_1)^\top (\boldsymbol{\Phi}_d \otimes \cdots \otimes \boldsymbol{\Phi}_1) \right)^{-1} (\boldsymbol{P}_d \otimes \cdots \otimes \boldsymbol{P}_1)^\top \boldsymbol{f} \right\|_2 \end{aligned}$$

Therefore, by [41, Lemma 3.2],

$$\begin{aligned}\|\boldsymbol{\mathcal{F}}-\widetilde{\boldsymbol{\mathcal{F}}}\|_F &\leq \left\|\left((\boldsymbol{P}_d\otimes\cdots\otimes\boldsymbol{P}_1)^\top(\boldsymbol{\Phi}_d\otimes\cdots\otimes\boldsymbol{\Phi}_1)\right)^{-1}\right\|_2 \left\|\boldsymbol{f}-(\boldsymbol{\Phi}_d\otimes\cdots\otimes\boldsymbol{\Phi}_1)(\boldsymbol{\Phi}_d\otimes\cdots\otimes\boldsymbol{\Phi}_1)^\top\boldsymbol{f}\right\|_2\\ &= \left\|(\boldsymbol{P}_d^\top\boldsymbol{\Phi}_d)^{-1}\right\|_2\cdots\left\|(\boldsymbol{P}_2^\top\boldsymbol{\Phi}_2)^{-1}\right\|_2\left\|(\boldsymbol{P}_1^\top\boldsymbol{\Phi}_1)^{-1}\right\|_2\left\|\boldsymbol{\mathcal{F}}-\boldsymbol{\mathcal{F}}\times_{m=1}^{d}\boldsymbol{\Phi}_m\boldsymbol{\Phi}_m^\top\right\|_F. \ \square\end{aligned}$$

The accuracy of the HO-DEIM approximation therefore depends on the contraction coefficients c_m, which are minimized by the use of `q-deim`; see, e.g., [57]. Furthermore it depends on the accuracy of the HO-POD bases, given by the term $\left\|\boldsymbol{\mathcal{F}}-\boldsymbol{\mathcal{F}}\times_{m=1}^{d}\boldsymbol{\Phi}_m\boldsymbol{\Phi}_m^\top\right\|_F$.

The full offline/online HO-POD-DEIM reduction procedure for reducing third-order tensor ODEs is presented below in Algorithm HO-POD-DEIM. In what follows, we

Algorithm HO-POD-DEIM for Tensor ODEs, $d=3$

Given: Coefficient matrices of (5.5) and function $\mathcal{F}:\mathbb{R}^{n\times\cdots\times n}\times[0,t_f]\to\mathbb{R}^{n\times\cdots\times n}$

Offline:

1. For each $j=1,2,\ldots,n_s$
   (i) Iteratively update $\{\boldsymbol{V}_m\}_{m=1}^3$ and $\{\boldsymbol{\Phi}_m\}_{m=1}^3$, for the snapshots $\boldsymbol{\mathcal{U}}(t_j)$ and $\mathcal{F}(\boldsymbol{\mathcal{U}}(t_j),t_j)$ respectively as (5.5) is integrated in time and discard the snapshots (HO-POD);
2. Compute $\widehat{\boldsymbol{A}}_m$, for $m=1,2,3$ and $\widehat{\boldsymbol{\mathcal{U}}}(0)$ from (5.7);
3. Determine $\{\boldsymbol{P}_m\}_{m=1}^3$ using `q-deim` (HO-DEIM);
4. Precompute $\{\boldsymbol{V}_m^\top\boldsymbol{\Phi}_m(\boldsymbol{P}_m^\top\boldsymbol{\Phi}_m)^{-1}\}_{m=1}^3$ and $\{\boldsymbol{P}_m^\top\boldsymbol{V}_m\}_{m=1}^3$;
5. Compute the real Schur decomposition $\widehat{\boldsymbol{A}}_3^\top=\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^\top$;

Online:

6. Determine $\widehat{\boldsymbol{\mathcal{U}}}^{(1)}$ from $\widehat{\boldsymbol{\mathcal{U}}}^{(0)}$;
7. For each $j=2,3,\ldots,n_{\mathrm{t}}$
   (i) Approximate $\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)},\mathrm{t}_{j-1})$ and $\widehat{\mathcal{F}}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-2)},\mathrm{t}_{j-2})$ as in (5.15) and (5.16) using the matrices computed above, and evaluate $\mathcal{G}(\widehat{\boldsymbol{\mathcal{U}}}^{(j-1)},\widehat{\boldsymbol{\mathcal{U}}}^{(j-2)})$;
   (ii) For each $\ell=1,2,\ldots,k_3$:
      (a) Evaluate $\boldsymbol{j}_{\ell-1}$ using (5.13) and compute $\boldsymbol{H}=\widehat{\boldsymbol{G}}\boldsymbol{Q}$;
      (b) Reshape column ℓ of $\boldsymbol{H}$ into a $k_1\times k_2$ matrix to form $\boldsymbol{H}_\ell$;
      (c) Reshape $\boldsymbol{j}_{\ell-1}$ into a $k_1\times k_2$ matrix to form $\boldsymbol{J}_{\ell-1}$;
      (d) Solve the Sylvester matrix equation by a direct solver:

$$\left((3-2h\boldsymbol{R}_{\ell,\ell})\boldsymbol{I}_k-2\,h\widehat{\boldsymbol{A}}_1\right)\boldsymbol{Z}_\ell+\boldsymbol{Z}_\ell\left(-2\,h\widehat{\boldsymbol{A}}_2^\top\right)=\boldsymbol{H}_\ell+2\,h\boldsymbol{J}_{\ell-1},$$

      (e) Update $\boldsymbol{Z}\leftarrow[\boldsymbol{Z},\mathrm{vec}(\boldsymbol{Z}_\ell)]$;
   (iii) Evaluate $\widehat{\boldsymbol{\mathcal{U}}}_{(3)}^{(j)}=\boldsymbol{Q}\boldsymbol{Z}^\top$ and reshape it into a $k_1\times k_2\times k_3$ tensor;
8. Return $\boldsymbol{V}_1,\boldsymbol{V}_2,\boldsymbol{V}_3$ and $\{\widehat{\boldsymbol{\mathcal{U}}}^{(j)}\}_{j=1}^{n_{\mathrm{t}}}$, so that $\widehat{\boldsymbol{\mathcal{U}}}^{(j)}\times_1\boldsymbol{V}_1\times_2\boldsymbol{V}_2\times_3\boldsymbol{V}_3\approx\boldsymbol{\mathcal{U}}(\mathrm{t}_j)$;

illustrate how the discussed higher-order POD-DEIM order reduction strategy can be applied to coupled systems of ODEs of the form (5.3).

5.2 Order reduction of coupled systems of array-valued ODEs

Here we illustrate how the HO-POD-DEIM order reduction scheme presented in the previous section can be applied to systems of array-valued ODEs of the form (5.3). Indeed, consider $d \cdot g$ tall basis matrices $\boldsymbol{V}_{m,i} \in \mathbb{R}^{n \times k_{mi}}$ with orthonormal columns, for $i = 1, 2, \ldots, g$ and $m = 1, 2, \ldots, d$, where $k_{mi} \ll n$. That is, we consider d basis matrices for each of the g equations in (5.3). Approximations to each $\boldsymbol{\mathcal{U}}_i(t)$, for $t \in T$, can then be written as

$$\boldsymbol{\mathcal{U}}_i(t) \approx \widetilde{\boldsymbol{\mathcal{U}}}_i(t) = \widehat{\boldsymbol{\mathcal{U}}}_i(t) \bigtimes_{m=1}^{d} \boldsymbol{V}_{m,i}, \qquad i = 1, 2, \ldots, g.$$

The functions $\widehat{\boldsymbol{\mathcal{U}}}_i(t) \in \mathbb{R}^{k_{1i} \times k_{2i} \times \cdots \times k_{di}}$ are determined as an approximation to the solution of the reduced, coupled problem

$$\begin{cases} \dot{\widehat{\boldsymbol{\mathcal{U}}}}_1 &= \widehat{\mathcal{A}}_1(\widehat{\boldsymbol{\mathcal{U}}}_1) + \widehat{\mathcal{F}}_1\left(\widehat{\mathcal{D}}_1(\widehat{\boldsymbol{\mathcal{U}}}_1), \widehat{\boldsymbol{\mathcal{U}}}_1, \widehat{\boldsymbol{\mathcal{U}}}_2, \cdots, \widehat{\boldsymbol{\mathcal{U}}}_g, t\right) \\ &\vdots \\ \dot{\widehat{\boldsymbol{\mathcal{U}}}}_g &= \widehat{\mathcal{A}}_g(\widehat{\boldsymbol{\mathcal{U}}}_g) + \widehat{\mathcal{F}}_g\left(\widehat{\mathcal{D}}_g(\widehat{\boldsymbol{\mathcal{U}}}_g), \widehat{\boldsymbol{\mathcal{U}}}_1, \widehat{\boldsymbol{\mathcal{U}}}_2, \cdots, \widehat{\boldsymbol{\mathcal{U}}}_g, t\right), \end{cases} \tag{5.18}$$

where

$$\widehat{\mathcal{A}}_i(\widehat{\boldsymbol{\mathcal{U}}}_i) := \sum_{m=1}^{d} \widehat{\boldsymbol{\mathcal{U}}}_i \times_m \widehat{\boldsymbol{A}}_{mi}, \quad \widehat{\boldsymbol{A}}_{mi} = \boldsymbol{V}_{m,i}^{\top} \boldsymbol{A}_{mi} \boldsymbol{V}_{m,i}, \quad \widehat{\boldsymbol{\mathcal{U}}}(0) = \boldsymbol{\mathcal{U}}_{i0} \bigtimes_{m=1}^{d} \boldsymbol{V}_{m,i}^{\top} \tag{5.19}$$

and

$$\widehat{\mathcal{F}}_i\left(\widehat{\mathcal{D}}_i(\widehat{\boldsymbol{\mathcal{U}}}_i), \widehat{\boldsymbol{\mathcal{U}}}_1, \cdots, \widehat{\boldsymbol{\mathcal{U}}}_g, t\right) = \mathcal{F}_i\left(\mathcal{D}_i(\widetilde{\boldsymbol{\mathcal{U}}}_i), \widetilde{\boldsymbol{\mathcal{U}}}_1, \cdots, \widetilde{\boldsymbol{\mathcal{U}}}_g, t\right) \bigtimes_{m=1}^{d} \boldsymbol{V}_{m,i}^{\top}. \tag{5.20}$$

We use the HO-POD procedure from the previous section to determine the basis matrices. In particular, given the set of snapshot solutions $\{\boldsymbol{\mathcal{U}}_i(t_j)\}_{j=1}^{n_s}$, the basis matrices $\boldsymbol{V}_{m,i} \in \mathbb{R}^{n \times k_{mi}}$, $m = 1, 2, \ldots, d$, are determined following the HO-POD procedure from Section 5.1, for each $i = 1, 2, \ldots, g$.

The reduced order model (5.18), can also be integrated by means of the IMEX 2 - SBDF scheme for systems. Indeed, the $\widehat{\boldsymbol{\mathcal{U}}}_1^{(j)}, \widehat{\boldsymbol{\mathcal{U}}}_2^{(j)}, \ldots, \widehat{\boldsymbol{\mathcal{U}}}_g^{(j)}$ approximations to $\widehat{\boldsymbol{\mathcal{U}}}_1(\mathrm{t}_j), \widehat{\boldsymbol{\mathcal{U}}}_2(\mathrm{t}_j), \ldots, \widehat{\boldsymbol{\mathcal{U}}}_g(\mathrm{t}_j)$ are determined by solving the linear systems

$$\begin{cases} (3\,\widehat{\mathcal{I}} - 2\,h\widehat{\mathcal{A}}_1)(\widehat{\boldsymbol{\mathcal{U}}}_1^{(j)}) &= \widehat{\mathcal{G}}_1\left(\{\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-1)}\}_{i=1}^{g}, \{\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-2)}\}_{i=1}^{g}\right) \\ &\vdots \\ (3\,\widehat{\mathcal{I}} - 2\,h\widehat{\mathcal{A}}_g)(\widehat{\boldsymbol{\mathcal{U}}}_g^{(j)}) &= \widehat{\mathcal{G}}_g\left(\{\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-1)}\}_{i=1}^{g}, \{\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-2)}\}_{i=1}^{g}\right), \end{cases} \tag{5.21}$$

at each $\mathfrak{t}_j$, where

$$\widehat{\mathcal{G}}_i\left(\{\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-1)}\}_{i=1}^g,\{\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-2)}\}_{i=1}^g\right)=4\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-1)}-\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-2)}+4h\widehat{\mathcal{F}}_i\left(\widehat{\mathcal{D}}_i(\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-1)}),\widehat{\boldsymbol{\mathcal{U}}}_1^{(j-1)},\cdots,\widehat{\boldsymbol{\mathcal{U}}}_g^{(j-1)},\mathfrak{t}_{j-1}\right)$$
$$-2h\widehat{\mathcal{F}}_i\left(\widehat{\mathcal{D}}_i(\widehat{\boldsymbol{\mathcal{U}}}_i^{(j-2)}),\widehat{\boldsymbol{\mathcal{U}}}_1^{(j-2)},\cdots,\widehat{\boldsymbol{\mathcal{U}}}_g^{(j-2)},\mathfrak{t}_{j-2}\right).$$

Each of the g linear systems in (5.21) can be solved via the procedures set forth in Section 2.1 for $d=2$ and Section 5.1 for $d=3$.

Adjacent to the setting discussed in Section 5.1, the nonlinear functions need to be interpolated to avoid evaluating the functions in full dimension at each timestep. To this end, we approximate $\widehat{\mathcal{F}}_i\left(\widehat{\mathcal{D}}_i(\widehat{\boldsymbol{\mathcal{U}}}_i),\widehat{\boldsymbol{\mathcal{U}}}_1,\cdots,\widehat{\boldsymbol{\mathcal{U}}}_g,t\right)$ in the space spanned by the columns of the matrices $\boldsymbol{\Phi}_{m,i}\in\mathbb{R}^{n\times p_{mi}}$, $m=1,2,\ldots,d$, for each $i=1,2,\ldots g$, where $p_{mi}\ll n$. Given the selection matrices $\boldsymbol{P}_{m,i}\in\mathbb{R}^{n\times p_{mi}}$, $m=1,2,\ldots,d$, we obtain

$$\widehat{\mathcal{F}}_i(\widehat{\mathcal{D}}_i(\widehat{\boldsymbol{\mathcal{U}}}_i),\widehat{\boldsymbol{\mathcal{U}}}_1,\ldots,\widehat{\boldsymbol{\mathcal{U}}}_g,t)\approx\mathcal{F}_i(\mathcal{D}_i(\widetilde{\boldsymbol{\mathcal{U}}}_i),\widetilde{\boldsymbol{\mathcal{U}}}_1,\ldots,\widetilde{\boldsymbol{\mathcal{U}}}_g,t)\bigtimes_{m=1}^{d}\boldsymbol{V}_{m,i}^\top\boldsymbol{\mathfrak{D}}_{m,i},\tag{5.22}$$

with the oblique projectors

$$\boldsymbol{\mathfrak{D}}_{m,i}=\boldsymbol{\Phi}_{m,i}(\boldsymbol{P}_{m,i}^\top\boldsymbol{\Phi}_{m,i})^{-1}\boldsymbol{P}_{m,i}^\top,\quad\text{for}\quad m=1,2,\ldots,d\quad\text{and}\quad i=1,2,\ldots,g.$$

The basis matrices $\boldsymbol{\Phi}_{m,i}$, $m=1,2,\ldots,d$ are determined via the HO-POD procedure described in Section 5.1, given the set of nonlinear snapshots

$$\{\mathcal{F}_i\left(\mathcal{D}_i(\boldsymbol{\mathcal{U}}_i(t_j)),\boldsymbol{\mathcal{U}}_1(t_j),\boldsymbol{\mathcal{U}}_2(t_j),\cdots,\boldsymbol{\mathcal{U}}_g(t_j),t_j\right)\}_{j=1}^{n_s},\tag{5.23}$$

whereas the selection matrices $\boldsymbol{P}_{m,i}$ are determined via `q-deim` with inputs $\boldsymbol{\Phi}_{m,i}^\top$ respectively, for each $i=1,2,\ldots,g$. In this chapter we assume that there is a componentwise relationship between the arrays $\boldsymbol{\mathcal{U}}_1(t_j),\boldsymbol{\mathcal{U}}_2(t_j),\cdots,\boldsymbol{\mathcal{U}}_g(t_j)$ and the approximation to the gradient $\mathcal{D}_i(\boldsymbol{\mathcal{U}}_i)$ in the nonlinear function $\mathcal{F}_i$. Therefore, since the matrix $\boldsymbol{P}_{m,i}$ is merely responsible for selecting rows in the respective modes, it holds that

$$\widetilde{\mathcal{F}_i\left(\mathcal{D}_i(\widetilde{\boldsymbol{\mathcal{U}}}_i),\widetilde{\boldsymbol{\mathcal{U}}}_1,\ldots,\widetilde{\boldsymbol{\mathcal{U}}}_g,t\right)}:=\mathcal{F}_i\left(\mathcal{D}_i(\widetilde{\boldsymbol{\mathcal{U}}}_i),\widetilde{\boldsymbol{\mathcal{U}}}_1,\ldots,\widetilde{\boldsymbol{\mathcal{U}}}_g,t\right)\bigtimes_{m=1}^{d}\boldsymbol{P}_{m,i}^\top$$
$$=\mathcal{F}_i\left(\mathcal{D}_i(\widetilde{\boldsymbol{\mathcal{U}}}_i)\bigtimes_{m=1}^{d}\boldsymbol{P}_{m,i}^\top,\,\widetilde{\boldsymbol{\mathcal{U}}}_1\bigtimes_{m=1}^{d}\boldsymbol{P}_{m,i}^\top,\ldots,\widetilde{\boldsymbol{\mathcal{U}}}_g\bigtimes_{m=1}^{d}\boldsymbol{P}_{m,i}^\top,t\right).$$

By definition of the operator $\mathcal{D}_i$, the first term in the function $\mathcal{F}_i$ can be expanded as

$$\mathcal{D}_i(\widetilde{\boldsymbol{\mathcal{U}}}_i)\bigtimes_{m=1}^{d}\boldsymbol{P}_{m,i}^\top=\sum_{m=1}^{d}\widetilde{\boldsymbol{\mathcal{U}}}_i\times_m\boldsymbol{P}_{m,i}^\top\boldsymbol{D}_{mi},\qquad i=1,2,\ldots,g,$$

which corresponds to merely selecting the appropriate rows of the matrix approximating the gradient in each of the d spatial directions.

5.3 Numerical experiments

In this section we illustrate the efficiency of the discussed methods via benchmark problems from Biology and Engineering. For all problems, the accuracy of the reduced order model is tested through the average error measure

$$\bar{\mathcal{E}}(\boldsymbol{\mathcal{U}}) = \frac{1}{n_{\mathfrak{t}}} \sum_{j=1}^{n_{\mathfrak{t}}} \frac{\|\boldsymbol{\mathcal{U}}^{(j)} - \widetilde{\boldsymbol{\mathcal{U}}}^{(j)}\|_F}{\|\boldsymbol{\mathcal{U}}^{(j)}\|_F}, \tag{5.24}$$

and the truncation of the singular values is done by monitoring the quality of the approximation in the Frobenius norm. That is, if $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\gamma$ are the singular values of the matrix that needs to be truncated, then the new dimension $\nu \leq \gamma$ is determined as

$$\sqrt{\sum_{i=\nu+1}^{\gamma} \sigma_i^2} < \tau \sqrt{\sum_{i=1}^{\gamma} \sigma_i^2}. \tag{5.25}$$

We first illustrate the efficiency with two examples where $d = 2$, after which we investigate two further coupled problems in three dimensions.

Example 5.1. *The 2D FitzHugh-Nagumo model (FN).* Consider the following classical problem, given in adimensional form,

$$\dot{u}_1 = \delta_1 \Delta u_1 + \alpha(-u_1^3 + u_1 - u_2), \qquad \dot{u}_2 = \delta_2 \Delta u_2 + \alpha(\beta u_1 - \beta\eta u_2), \tag{5.26}$$

where the functions $u_1(x, y, t)$ and $u_2(x, y, t)$ model the densities of two species for $t \in [0, 1]$, and $\boldsymbol{x} = (x, y) \in [-1, 1]^2$. We refer the reader to, e.g., [66] for a description of the role of the nonnegative coefficients η, β, and α. For this example we set $\eta = 0.5, \beta = 2.1, \alpha = 9.65$, $\delta_1 = 0.01$ and $\delta_2 = 0.1$. Furthermore, homogeneous Neumann boundary conditions are imposed and the initial state is given by

$$\begin{aligned} u_1(x, y, 0) &= (1 - x^2)(1 - y^2)\sin(2\pi x)\cos(2\pi(y + 0.3)) \\ u_2(x, y, 0) &= (1 - x^2)(1 - y^2)e^{-\sin(2\pi(x-0.3)y)}. \end{aligned}$$

This example investigates the efficiency of the reduced-order model in terms of accuracy and online CPU time. To this end, the system (5.26) is discretized with $n = 1200$ spatial nodes in each direction yielding the form (5.3) . Note that this is equivalent to the system (5.2) with dimension $N = 1\,440\,000$.

In particular, if we let $\boldsymbol{T} = \text{tridiag}(1, \underline{-2}, 1) + \boldsymbol{N}_B$, $\boldsymbol{T} \in \mathbb{R}^{n\times n}$, where

$$\boldsymbol{N}_B = \frac{2}{3} \begin{pmatrix} 2 & -1/2 & \cdots & 0 & 0 \\ 0 & 0 & \cdots & \cdots & 0 \\ \vdots & & & \vdots & \\ 0 & 0 & \cdots & -1/2 & 2 \end{pmatrix} \in \mathbb{R}^{n\times n}$$

contains the Neumann boundary conditions (see, e.g., [51]), then the coefficient matrices of (5.3) are defined as

$$\boldsymbol{A}_{11} = \alpha\, \boldsymbol{I}_n + \frac{\delta_1}{\ell_x^2}\boldsymbol{T}, \quad \boldsymbol{A}_{21} = -\alpha\beta\eta\boldsymbol{I}_n + \frac{\delta_2}{\ell_x^2}\boldsymbol{T}, \quad \boldsymbol{A}_{12} = \frac{\delta_1}{\ell_y^2}\boldsymbol{T}, \quad \text{and} \quad \boldsymbol{A}_{22} = \frac{\delta_2}{\ell_y^2}\boldsymbol{T}$$

where $\ell_x = \ell_y = 2/(n-1)$. Notice that the discretized linear terms $\alpha\boldsymbol{U}_1$ and $-\alpha\beta\eta\boldsymbol{U}_2$ have been incorporated into the coefficient matrices $\boldsymbol{A}_{11}$ and $\boldsymbol{A}_{21}$ respectively. Furthermore the matrix $\mathcal{F}_1(\boldsymbol{U}_1, \boldsymbol{U}_2, t)$ stems from evaluating the function $f_1(u_1, u_2) = -\alpha u_1^3$ elementwise, whereas $\mathcal{F}_2(\boldsymbol{U}_1, \boldsymbol{U}_2, t) = \alpha\beta\boldsymbol{U}_1$ is linear and requires no HO-DEIM interpolation. Finally, the remaining linear term in the first equation $-\alpha\,\boldsymbol{U}_2$ can be projected explicitly onto range$(\boldsymbol{V}_{m,1})$, the HO-POD subspaces of the first equation for $m = 1, 2, \ldots, d$.

In our experiments we found that $n_s = 20$ equispaced snapshots $\boldsymbol{U}_1(t)$ and $\boldsymbol{U}_2(t)$ in the timespan $[0, 1]$ are sufficient for constructing the basis vectors. Furthermore, we consider four different truncation tolerances, namely $\tau = \{10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}\}$, for this experiment. Table 5.1 reports all the basis dimensions obtained for each τ, by means of (5.25).

TABLE 5.1: Example 5.1. Dim. of HO-POD and HO-DEIM bases obtained for different τ. The full order model has dimension $n = 1200$.

τ	$\boldsymbol{U}_i$	left dim. HO-POD	right dim. HO-POD	left dim. HO-DEIM	right dim. HO-DEIM
10^{-2}	$\boldsymbol{U}_1$	7	7	11	11
	$\boldsymbol{U}_2$	9	10	–	–
10^{-4}	$\boldsymbol{U}_1$	18	20	23	23
	$\boldsymbol{U}_2$	19	20	–	–
10^{-6}	$\boldsymbol{U}_1$	31	33	32	34
	$\boldsymbol{U}_2$	29	31	–	–
10^{-8}	$\boldsymbol{U}_1$	43	46	44	47
	$\boldsymbol{U}_2$	37	40	–	–

In Figure 5.1 (left) we plot the average error (5.24) for both $\widetilde{\boldsymbol{U}}_1$ and $\widetilde{\boldsymbol{U}}_2$ integrated from 0 to t_f at $n_t = 300$ timesteps, for the different values of τ presented in Table 5.1. For the error computation, both the full order model and the reduced order model are integrated with the IMEX 2-SBDF scheme. On the right of Figure 5.1 we plot the CPU time for integrating the full order model and the reduced order model at $n_t = 300$ timesteps for decreasing τ. The figures indicate that even when the HO-POD-DEIM reduced-order model approximates the full order model with eight digits of accuracy, the time needed to integrate the model is almost three orders of magnitude faster.□

In what follows we analyze the efficiency of the offline phase in comparison to the the standard POD-DEIM procedure applied to the coupled 2D Burgers equation in [161].

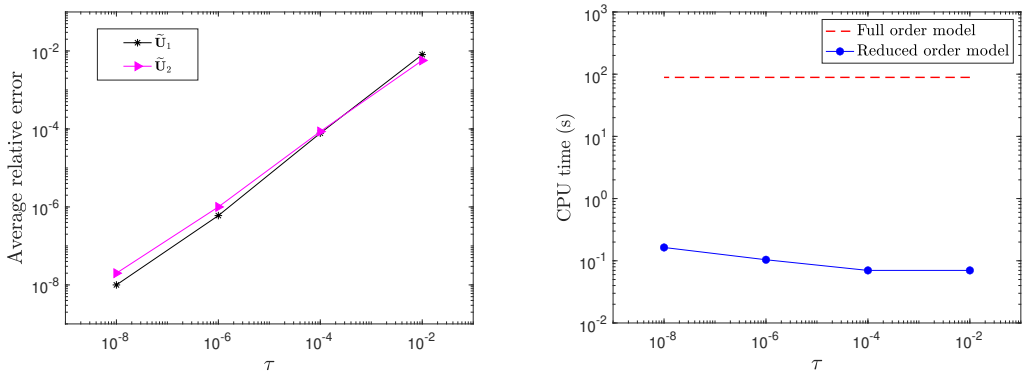

FIGURE 5.1: Example 5.1: Average relative error (5.24) (left) and online computational time (right) of the reduced order model and the full order model for different values of τ.

Example 5.2. *The 2D coupled Burgers equation (BE) [161].* Here we consider the semilinear 2D coupled Burgers equation given by

$$\begin{cases} \dot{u}_1 = & \frac{1}{r}\Delta u_1 - u_1(u_1)_x - u_2(u_1)_y \\ \dot{u}_2 = & \frac{1}{r}\Delta u_2 - -u_1(u_2)_x - u_2(u_2)_y \end{cases} \tag{5.27}$$

where $u_1(x, y, t)$ and $u_2(x, y, t)$ $(t \in [0, 1])$ are the velocities to be determined, with $\boldsymbol{x} = (x, y) \in [0, 1]^2$, and r is the Reynold's number. As is done in [161], we derive the initial and boundary conditions from the exact traveling wave solution of the 2D Burgers equation, given by (see e.g., [62])

$$u_1(x, y, t) = \frac{3}{4} - \frac{1}{4}\left(1 + e^{\frac{r(-4x+4y-t)}{32}}\right)^{-1} \qquad u_2(x, y, t) = \frac{3}{4} + \frac{1}{4}\left(1 + e^{\frac{r(-4x+4y-t)}{32}}\right)^{-1}.$$

We consider the case $r = 100$ and discretize the model on a grid with n spatial nodes in each direction, yielding a system of the form (5.3) , with nonlinear functions

$$\mathcal{F}_i(\mathcal{D}_i(\boldsymbol{\mathcal{U}}_i), \boldsymbol{\mathcal{U}}_1, \boldsymbol{\mathcal{U}}_2, t) = \mathcal{F}_i(\mathcal{D}_i(\boldsymbol{U}_i), \boldsymbol{U}_1, \boldsymbol{U}_2, t) := (\boldsymbol{D}_{1i}\boldsymbol{U}_i) \circ \boldsymbol{U}_1 + (\boldsymbol{U}_i\boldsymbol{D}_{2i}^\top) \circ \boldsymbol{U}_2, \tag{5.28}$$

for $i = 1, 2$, where the matrices $\boldsymbol{D}_{1i} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{D}_{2i} \in \mathbb{R}^{n\times n}$ contain the coefficients for a first order centered difference space discretization in the $x-$ and $y-$ directions respectively (i.e., $\boldsymbol{D}_{1i} = \boldsymbol{D}_{2i} = \frac{n-1}{2}\,\text{tridiag}(-1, \underline{0}, 1)$), and $\circ$ is the matrix Hadamard product. An upwind scheme can also be considered for $\boldsymbol{D}_{1i}$ and $\boldsymbol{D}_{2i}$, as is typically done for the coupled Burgers equation, however, in order to reproduce the results of [161] we consider centered finite differences.

It is also worth motivating the use of DEIM for the nonlinear function (5.28). Indeed, this type of nonlinearity can also be efficiently treated in the vectorized POD reduced model by writing it as a tensor (see, e.g., [104]) in order to avoid the use of DEIM. Nevertheless, in [148, Table I] this idea is compared to that of POD-DEIM and it is concluded that for quadratic nonlinearities, the POD-DEIM model requires considerably

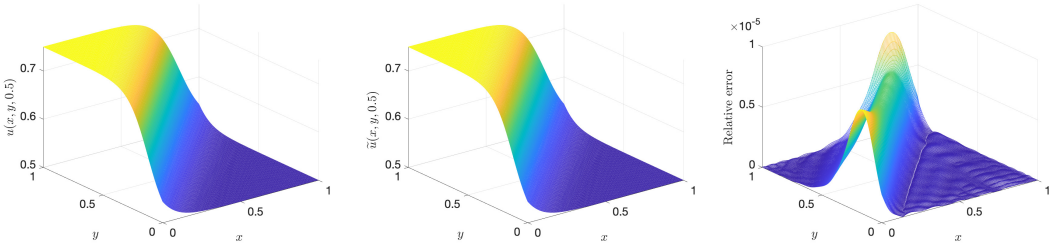

FIGURE 5.2: Example 5.2: $u_1(x, y, 0.5)$ discretized with $n = 200$. The exact solution (left), the HO-POD-DEIM approximation (middle), and the relative error mesh between the two (right).

fewer floating-point operations online for moderate DEIM dimension p. Furthermore, we use DEIM to reproduce and compare to the results of [161], where it is indeed used.

As mentioned above, the presented HO-POD-DEIM order reduction strategy is compared to the standard POD-DEIM applied to (5.27) in [161]. We did not have access to the codes of [161], but the POD-DEIM algorithm was implemented as discussed in their paper, and the results in terms of basis dimension to accuracy are comparable to the ones reported in [161]. Moreover, in [161], the reduced-order model is integrated by a fully implicit scheme, whereas for this experiment, we use the IMEX 2-SBDF method to integrate both the HO-POD-DEIM and POD-DEIM reduced-order models, which accounts for the faster online phase for POD-DEIM in comparison to the times reported in [161].

To this end, we consider four different space discretizations, namely $n = \{60, 200, 600, 1200\}$, and compare the computational details of HO-POD-DEIM to that of POD-DEIM [161]. Therefore, to ensure stability in the numerical integration we consider $n_t = 2n$ discrete timesteps for integrating the reduced order model. Moreover, to correspond with the space discretization error, we set $\tau = 1/n^2$.

In our experiments, we have observed that the HO-POD-DEIM strategy requires far fewer snapshots than POD-DEIM to construct an equally accurate reduced-order model. We hypothesize that this is because one matrix snapshot contains information about several spatial directions in $\mathbb{R}^n$, whereas one vector snapshot only offers information about one spatial direction in $\mathbb{R}^N$. Therefore, to obtain a POD basis of dimension k in the vectorized setting, at least k snapshots are required, even though one vector in $\mathbb{R}^N$ contains many spatial directions from $\mathbb{R}^n$. This corresponds to the behavior seen in Figure 4.7. To this end, we consider $n_s = 20$ equispaced snapshots for HO-POD-DEIM and $n_s = 100$ for POD-DEIM.

A visual comparison of the accuracy of the HO-POD-DEIM reduced model at $t = 0.5$, when $n = 200$ is plotted in Figures 5.2 and 5.3 for u_1 and u_2 respectively. Moreover, in Figure 5.4 we plot the average relative error through $n_t = 2n$ timesteps between the HO-POD-DEIM approximation and the exact solution at the relevant nodes, for the

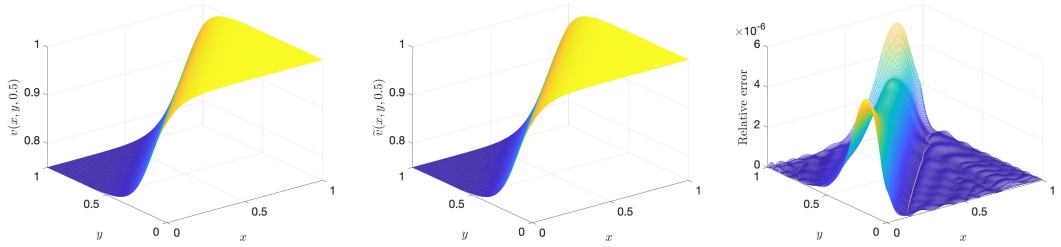

FIGURE 5.3: Example 5.2: $u_2(x, y, 0.5)$ discretized with $n = 200$. The exact solution (left), the HO-POD-DEIM approximation (middle), and the relative error mesh between the two (right).

four different space dimensions n. We investigate the computational load required by both strategies to achieve this accuracy.

Firstly, we report in Table 5.2 the reduced basis dimensions for HO-POD-DEIM and POD-DEIM for all four space discretizations, as well as the memory requirements. In particular, for each $\boldsymbol{U}_i$ we report the dimensions k_1/k_2 (p_1/p_2) of the HO-POD (HO-DEIM) bases and the dimension k (p) of the POD (DEIM) bases. Moreover, the reported global memory requirements include the number of stored vectors multiplied by their length $(\# \cdot \mathit{length})$ in each phase. The table indicates a great reduction in

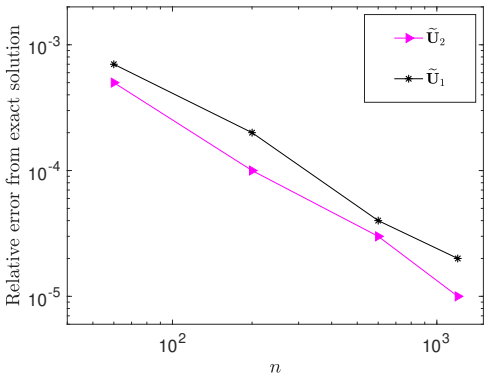

FIGURE 5.4: Example 5.2: The average relative error through $n_t = 2n$ timesteps between the HO-POD-DEIM approximation $\widetilde{\boldsymbol{U}}_1(t)$ $(\widetilde{\boldsymbol{U}}_2(t))$ and the exact solution $u_1(x, y, t)$ $(u_2(x, y, t))$.

memory requirements for HO-POD-DEIM in comparison to POD-DEIM, but the POD-DEIM strategy produces a smaller reduced order model, as is expected from a one-sided reduction strategy. That is, the POD-DEIM reduced model requires evaluating the nonlinear function at merely p entries, whereas the HO-POD-DEIM reduced model requires evaluating the nonlinear function at $p_1 \cdot p_2$ entries. From Table 5.2 it is clear that $p_1 \cdot p_2 \geq p$ for all n.

We investigate the pros and cons, in terms of computational time, of both strategies in Figure 5.5. On the left of Figure 5.5 we plot the time needed offline to construct the basis vectors for both strategies, for increasing n. For HO-POD-DEIM this includes the time needed to perform the SVD of each snapshot and the time needed to orthogonalize

TABLE 5.2: A breakdown of the (HO)-POD and (HO)-DEIM basis dimensions and the memory requirements for four different state space dimensions. Note that $\tau = 1/n^2$.

n	ALGORITHM	$\boldsymbol{U}_i$	POD DIM.	DEIM DIM.	OFFLINE MEMORY	ONLINE MEMORY
60	HO-POD-DEIM	$\boldsymbol{U}_1$	9/9	18/18	$98n$	$54n$
		$\boldsymbol{U}_2$	9/9	18/18	$98n$	$54n$
	POD-DEIM [161]	$\boldsymbol{U}_1$	5	14	$400n^2$	$19n^2$
		$\boldsymbol{U}_2$	4	14	$400n^2$	$18n^2$
200	HO-POD-DEIM	$\boldsymbol{U}_1$	13/13	24/25	$153n$	$75n$
		$\boldsymbol{U}_2$	12/12	24/25	$153n$	$73n$
	POD-DEIM [161]	$\boldsymbol{U}_1$	9	23	$400n^2$	$32n^2$
		$\boldsymbol{U}_2$	8	23	$400n^2$	$31n^2$
600	HO-POD-DEIM	$\boldsymbol{U}_1$	16/17	32/32	$196n$	$97n$
		$\boldsymbol{U}_2$	16/16	32/32	$194n$	$96n$
	POD-DEIM [161]	$\boldsymbol{U}_1$	15	28	$400n^2$	$43n^2$
		$\boldsymbol{U}_2$	14	28	$400n^2$	$42n^2$
1200	HO-POD-DEIM	$\boldsymbol{U}_1$	19/19	36/39	$219n$	$113n$
		$\boldsymbol{U}_2$	19/19	36/39	$215n$	$113n$
	POD-DEIM [161]	$\boldsymbol{U}_1$	19	31	$400n^2$	$50n^2$
		$\boldsymbol{U}_2$	18	31	$400n^2$	$50n^2$

and truncate the new basis vectors for all 8 bases[5], whereas for POD-DEIM this includes the time to vectorize each snapshot and the time to perform the economy SVD of all four $n^2 \times n_s$ matrices of snapshots. On the right of Figure 5.5 we report the time needed to evaluate (5.18) and a POD-DEIM reduced system of the form (5.2) at n_t timesteps online and compare it to the time needed to evaluate the full order model (5.3) . The timings in Figure 5.5 (right) correspond to the reduced dimensions reported in Table 5.2.

Figure 5.5 (left) indicates the large gain in offline computational time by the new strategy, with almost two orders of magnitude difference as n increases. However, due to the larger reduced dimensions of HO-DEIM reported in Table 5.2, fractionally more time is required online, as presented in Figure 5.5 (right). Nevertheless the online times are very comparable and orders of magnitude lower than the full order model (5.3) . In problems where the nonlinear term is more expensive to evaluate this drawback of HO-POD-DEIM will become more evident, however, and will need to be further investigated in future work; see also Remark 5.1.

This experiment indicated that a greater accuracy with respect to the exact solution can be achieved by the discrete HO-POD-DEIM reduced order model in a fraction of the offline computational time compared to POD-DEIM. Moreover the online time remains comparable, and a large gain in memory requirements is witnessed. □

[5]Each equation u_1 and u_2 require four basis matrices when $d = 2$. Two stemming from the snapshot solutions $\{\boldsymbol{\mathcal{U}}_i(t_j)\}_{j=1}^{n_s}$ for the HO-POD dimension reduction and two stemming from the nonlinear snapshots (5.23) for the HO-DEIM interpolation.

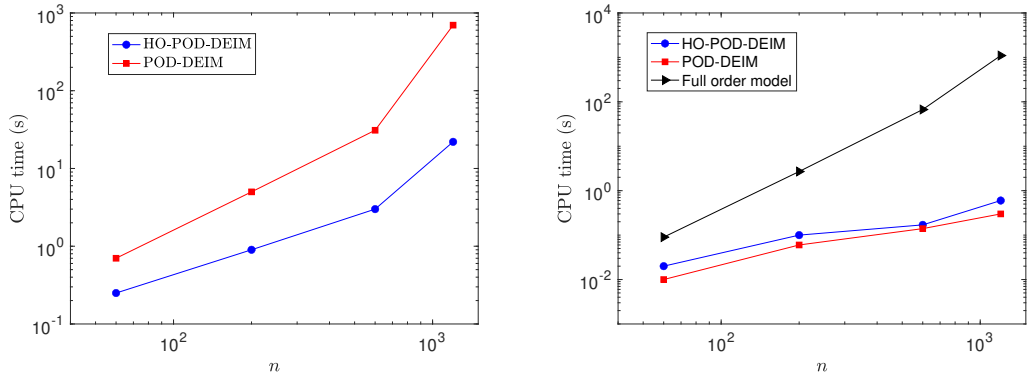

FIGURE 5.5: Example 5.2: A comparison of the time required offline for basis construction (left) and online for integration (right) between HO-POD-DEIM and POD-DEIM [161] for increasing n.

In what follows we illustrate the efficiency of the procedure in the multilinear setting.

Example 5.3. *The 3D coupled Burgers equation (BE).* Here we consider the semilinear 3D coupled Burgers equation (see, e.g., [67]) given by

$$\begin{cases} \dot{u}_1 &= \frac{1}{r}\Delta u_1 - \underline{u}\cdot\nabla u_1 \\ \dot{u}_2 &= \frac{1}{r}\Delta u_2 - \underline{u}\cdot\nabla u_2, \\ \dot{u}_3 &= \frac{1}{r}\Delta u_3 - \underline{u}\cdot\nabla u_3, \end{cases} \tag{5.29}$$

where $u_1(x,y,z,t)$, $u_2(x,y,z,t)$ and $u_3(x,y,z,t)$ are the three velocities to be determined, with $\boldsymbol{x} = (x,y,z) \in (0,1)^3$ and $t \in [0,1]$. Furthermore, the system is subject to homogeneous Dirichlet boundary conditions and initial states

$$\begin{aligned} u_1(x,y,z,0) &= \frac{1}{10}\sin(2\pi x)\sin(2\pi y)\cos(2\pi z) \\ u_2(x,y,z,0) &= \frac{1}{10}\sin(2\pi x)\cos(2\pi y)\sin(2\pi z) \\ u_3(x,y,z,0) &= \frac{1}{10}\cos(2\pi x)\sin(2\pi y)\sin(2\pi z). \end{aligned}$$

A finite difference space discretization inside the cube yields a system of ODEs of the form (5.3) , with nonlinear functions given by

$$\mathcal{F}_i(\mathcal{D}_i(\boldsymbol{\mathcal{U}}_i),\boldsymbol{\mathcal{U}}_1,\boldsymbol{\mathcal{U}}_2,\boldsymbol{\mathcal{U}}_3,t) = (\boldsymbol{\mathcal{U}}_i \times_1 \boldsymbol{D}_{1i}) \circ \boldsymbol{\mathcal{U}}_1 + (\boldsymbol{\mathcal{U}}_i \times_2 \boldsymbol{D}_{2i}) \circ \boldsymbol{\mathcal{U}}_2 + (\boldsymbol{\mathcal{U}}_i \times_3 \boldsymbol{D}_{3i}) \circ \boldsymbol{\mathcal{U}}_3,$$

for $i = 1,2,3$, where $\boldsymbol{D}_{1i} \in \mathbb{R}^{n\times n}$, $\boldsymbol{D}_{2i} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{D}_{3i} \in \mathbb{R}^{n\times n}$ contain the coefficients for a first order centered difference space discretization in the $x-$, $y-$ and $z-$ directions respectively. Furthermore we vary r through the experiment, and calculate the reduced order model through $n_s = 50$ equispaced snapshots for both HO-POD-DEIM and POD-DEIM.

Firstly, we set $r = 10$ and consider five different space discretizations, namely $n =$

$\{50, 80, 100, 150, 200\}$, and investigate the efficiency of the offline phase of the HO-POD-DEIM procedure for systems when $d = 3$, in comparison to standard POD-DEIM. Note that the system (5.3) of dimension $n = 200$ is equivalent to the system (5.2) with dimension $N = 8\,000\,000$ when $d = 3$.

The improvement in memory requirements is immediately evident, since the new procedure requires storing basis vectors of length n, whereas the vectorization procedure needs to store many basis vectors of length n^3, as has been witnessed in Table 5.2 for $d = 2$. In Figure 5.6 (left) we compare the computational time needed to determine

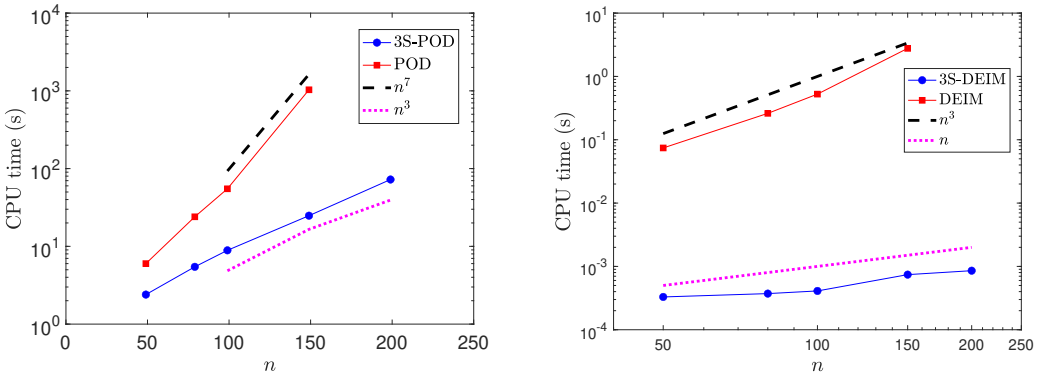

FIGURE 5.6: Example 5.3: A comparison of the offline time for increasing dimension n, between HO-POD and POD (left) and HO-DEIM and DEIM (right).

the basis vectors, given $\tau = 10^{-4}$, for increasing dimension n. For the new procedure that includes the time needed to perform the STHOSVD of each snapshot and the time needed to orthogonalize and truncate the new basis vectors, whereas for POD-DEIM this includes the time to vectorize each snapshot and the time to perform the economy SVD of the $n^3 \times n_s$ matrix of snapshots. The times are added together for all 18 bases[6] required by HO-POD-DEIM and all 6 bases required by POD-DEIM. We explicitly remark that both the STHOSVD and the economy SVD can potentially be further accelerated by using randomized algorithms; see e.g., [76, 116]. This is, however, not considered in our experiments.

In Figure 5.6 (right) we compare the time needed to determine the (HO)-DEIM interpolation indices. That is, the cumulative time taken by `q-deim` for all 9 nonlinear bases for HO-POD-DEIM and all 3 nonlinear bases required by POD-DEIM.

For both POD and DEIM the improvement in computational time is very evident in the plots, with a few orders of magnitude difference. For example, on the standard laptop computer on which these experiments were performed, the HO-POD-DEIM bases were created in just more than a minute for $n = 200$, whereas the computer ran out

[6]Each equation u_1,u_2 and u_3 requires six basis matrices when $d = 3$. Three stemming from the snapshot solutions $\{\boldsymbol{\mathcal{U}}_i(t_j)\}_{j=1}^{n_s}$ for the HO-POD dimension reduction and three stemming from the nonlinear snapshots (5.23) for the HO-DEIM interpolation. For standard POD-DEIM each equation requires only two basis matrices, one for dimension reduction and one for DEIM interpolation, hence six bases in total.

of memory for the vectorization procedure processing 30 snapshots in more than an hour.

In what follows, we investigate the online phase. We set $n = 150$ and illustrate the efficiency of the three-sided reduction procedure, together with the new T3-SYLV method for solving the low-dimensional, dense tensor-valued system of equations. To this end, we investigate the total time needed for solving all inner linear systems at $n_t = 100$ timesteps, that is 300 linear systems in total, using T3-SYLV and compare it to the time needed if the system (5.21) is vectorized and solved as a standard (nearly dense) linear system (Vec-lin). This is done for different values of τ, which result in different reduced dimensions. In particular, we plot the computational time with respect to the maximum dimension of the vectorized systems for the different values of τ. That is, the value $\max(k_1 k_2 k_3)$ on the $x-$axis is the maximum value value of $k_{1i} k_{2i} k_{3i}$ for all $i = 1, 2, 3$, given τ.

For the solution of the vectorized system we perform a reverse Cuthill–McKee reordering of the coefficient matrices, to exploit any remaining sparsity pattern, and perform an LU decomposition once for all, so that only front- and back-substitution is required for all system solves. The results are reported in Figure 5.7.

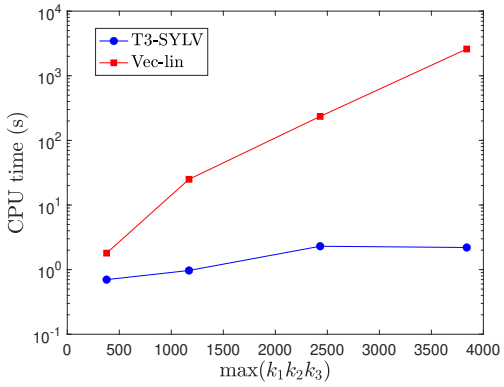

FIGURE 5.7: Example 5.3: A comparison of the time to solve all linear systems of the form (5.21), for different values of τ, between T3-SYLV and Vec-lin. The $x-$axis displays the maximum dimension of the three vectorized equations for different values of τ.

The advantage that the three-sided reduction procedure poses in combination with the T3-SYLV inner solver is evident from Figure 5.7. This figure, together with Figure 5.6 illustrates that much time can be saved offline and online. Without the T3-SYLV inner solver, it is evident that the tensor structure of the coefficient matrices retained by the three-sided projection would result in expensive, dense linear system solves, which would cancel the time that has been saved in the offline phase. Nevertheless, with the availability of the T3-SYLV solver, a considerable speedup is seen in both the offline and online phases, together with a massive gain in memory requirements.

Finally, Table 5.3 and Table 5.4 contain the details of the reduced order model, given $n = 150$, $\tau = 10^{-4}$ and the error measure (5.24) for increasing values of the Reynold's number r. Table 5.3 illustrates that a large reduction in all dimensions is achieved,

TABLE 5.3: Example 5.3. Dim. of HO-POD and HO-DEIM bases and the average error at 300 timesteps for increasing r. The full order model has dimension $n = 150$ and $\tau = 10^{-4}$.

r	u	k_1	k_2	k_3	p_1	p_2	p_3	ERROR $\bar{\mathcal{E}}(\boldsymbol{\mathcal{U}})$
10	u_1	4	7	10	7	12	16	$1 \cdot 10^{-4}$
	u_2	7	7	7	9	12	13	$6 \cdot 10^{-5}$
	u_3	8	12	8	9	16	13	$1 \cdot 10^{-4}$
100	u_1	6	11	15	10	17	20	$3 \cdot 10^{-5}$
	u_2	10	11	11	12	17	17	$4 \cdot 10^{-5}$
	u_3	10	16	12	12	21	17	$4 \cdot 10^{-5}$
500	u_1	9	15	19	13	23	26	$2 \cdot 10^{-5}$
	u_2	11	16	17	14	23	23	$3 \cdot 10^{-5}$
	u_3	12	19	16	14	25	23	$4 \cdot 10^{-5}$

with a very acceptable accuracy over 300 timesteps, even for large r. Nevertheless it is clear that for larger Reynold's number the singular value decay in each mode becomes slower, resulting in larger reduced dimensions. Moreover we mention that for this same problem, when $r = 100$, standard POD-DEIM would require storing 57 vectors of length n^3 in the online phase, as opposed to the 245 vectors of length n that need to be stored for HO-POD-DEIM. Furthermore, Table 5.4 confirms that even for large r we observe a large gain in online computational time achieved by the reduced model.□

TABLE 5.4: Example 5.3. Memory and CPU time required for basis construction and integration. The full order model has dimension $n = 150$ and $\tau = 10^{-4}$.

r	Online memory	Basis time(s)	FOM time(s)	ROM time(s)
10	$177n$	20	1641	1.9
100	$245n$	20	1641	2.2
500	$318n$	20	1641	3.3

Example 5.4. *A 3D reaction-diffusion model for cell apoptosis.* As a final example we consider a reaction-diffusion system, orginally introduced in [49] to investigate the behavior of protein concentrations (in space and time) of a cell apoptosis model in 1D. The model was later extended to higher dimension in [50, Chapter 2.3]. The proteins build a network called "caspase–cascade" and the dynamics, with homogeneous Neumann boundary conditions, are given by

$$\begin{aligned} \dot{u}_1 &= \delta_1 \Delta u_1 - c_4 u_1 + c_1 \sin(u_3 u_2), & \dot{u}_2 &= \delta_2 \Delta u_2 - c_4 u_2 + c_2 u_4 u_1^3, \\ \dot{u}_3 &= \delta_3 \Delta u_3 - c_4 u_3 - c_1 \sin(u_3 u_2) + c_3, & \dot{u}_4 &= \delta_4 \Delta u_4 - c_4 u_4 - c_2 u_4 u_1^3 + c_3, \end{aligned} \tag{5.30}$$

where $u_1(\boldsymbol{x},t)$, $u_2(\boldsymbol{x},t)$, $u_3(\boldsymbol{x},t)$ and $u_4(\boldsymbol{x},t)$ are four different reactants called Procaspase-8, Procaspase-3, Caspase-8 and Caspase-3 respectively, with $\boldsymbol{x}=(x,y,z)\in(0,1)^3=:\Omega$ and $t\in[0,1]$. For the values and derivation of the constants δ_i and c_i we refer the reader to [50, Chapter 2.3] and we consider the initial condition

$$(u_1,u_2,u_3,u_4)(\boldsymbol{x},0)=\begin{cases}\left(u_1^{(d)},u_2^{(d)},u_3^{(d)},u_4^{(d)}\right) & \text{for} \quad \boldsymbol{x}\in\Omega_{\text{ext}}\\ \left(u_1^{(\ell)},u_2^{(\ell)},u_3^{(\ell)},u_4^{(\ell)}\right) & \text{for} \quad \boldsymbol{x}\in\Omega_{\text{in}}\end{cases},$$

where $\Omega_{\text{ext}}:=\{\boldsymbol{x}\in\Omega, r_0\le\|\boldsymbol{x}\|_2\le 1\}$ and $\Omega_{\text{in}}:=\{\boldsymbol{x}\in\Omega,\|\boldsymbol{x}\|_2<r_0\}$ and we consider $r_0=\{0.1,0.3\}$. Furthermore, $u_i^{(\ell)}$ and $u_i^{(d)}$ represent respectively the life and death states of the reactants and they are defined in [50, Chapter 2.4]. Note that we have introduced the sin function in equations one and three to also test the strength of the procedure on non-polynomial nonlinearities.

For the experimental setup we discretize Equation (5.30) with $n=150$ nodes in each of the spatial directions. Therefore, if $\boldsymbol{T}\in\mathbb{R}^{n\times n}$ is defined as in Example 5.1, this yields a system of the form (5.3) , with

$$\boldsymbol{A}_{1i}=-c_4\boldsymbol{I}_n-\frac{\delta_i}{\ell_x^2}\boldsymbol{T},\quad \boldsymbol{A}_{2i}=\frac{\delta_i}{\ell_y^2}\boldsymbol{T},\quad \boldsymbol{A}_{3i}=\frac{\delta_i}{\ell_z^2}\boldsymbol{T},\qquad i=1,2,3,4,$$

and $\ell_x=\ell_y=\ell_z=1/(n-1)$, where the additional linear terms have been incorporated into the matrices $\boldsymbol{A}_{1i}$. Furthermore, the nonlinear functions $\mathcal{F}_1(\boldsymbol{\mathcal{U}}_1,\boldsymbol{\mathcal{U}}_2,\boldsymbol{\mathcal{U}}_3,\boldsymbol{\mathcal{U}}_4,t)$ and $\mathcal{F}_2(\boldsymbol{\mathcal{U}}_1,\boldsymbol{\mathcal{U}}_2,\boldsymbol{\mathcal{U}}_3,\boldsymbol{\mathcal{U}}_4,t)$ stem from respectively evaluating the nonlinear terms $c_1\sin(u_3u_2)$ and $c_2u_4u_1^3$ elementwise. Notice, furthermore, that $\mathcal{F}_3=-\mathcal{F}_1$ and $\mathcal{F}_4=-\mathcal{F}_2$, so that only two HO-DEIM bases are required, instead of four. The constant matrices stemming from the discretization of the constant c_3 are treated separately. Finally we consider $\tau=10^{-2}$ and $n_s=30$ equispaced snapshots of each $\boldsymbol{\mathcal{U}}_i$, $\mathcal{F}_1$ and $\mathcal{F}_2$ in the timespan.

In Table 5.5 we report, for all four equations and both values of r_0, the dimension of the HO-POD and HO-DEIM bases, the online memory requirements (# of vectors times the length, as before), the online time for $n_t=300$ (for each equation separately) and the average relative error (5.24).

We observe a significant decrease in the state dimension for both values of r_0, with a very acceptable average relative error in all equations. Equations three and four require a larger basis than one and two, but in turn, they do not require the additional cost online of HO-DEIM interpolation and the evaluation of the nonlinear function. Furthermore, we observe that all equations can be solved in a rapid online phase, whereas the full order model needs approximately 4064 seconds to be integrated at $n_t=300$ timesteps, independent of r_0. □

TABLE 5.5: Example 5.4. Dim. of HO-POD and HO-DEIM bases and further computational detalis for $\tau = 10^{-2}$ and $n = 150$.

r_0	$\boldsymbol{U}_i$	POD DIM. $(k_1/k_2/k_3)$	DEIM DIM. $(p_1/p_2/p_3)$	ONLINE MEMORY	ONLINE TIME (S)	ERROR
0.1	$\boldsymbol{U}_1$	2/2/2	5/5/5	$21n$	1.29	$3 \cdot 10^{-4}$
	$\boldsymbol{U}_2$	2/2/2	3/3/3	$15n$	1.20	$4 \cdot 10^{-4}$
	$\boldsymbol{U}_3$	18/18/18	–	$54n$	1.50	$3 \cdot 10^{-4}$
	$\boldsymbol{U}_4$	9/9/9	–	$27n$	0.63	$1 \cdot 10^{-2}$
0.3	$\boldsymbol{U}_1$	8/8/8	9/9/9	$51n$	1.56	$3 \cdot 10^{-4}$
	$\boldsymbol{U}_2$	8/8/8	9/9/9	$51n$	1.13	$3 \cdot 10^{-4}$
	$\boldsymbol{U}_3$	43/43/42	–	$128n$	11.25	$3 \cdot 10^{-3}$
	$\boldsymbol{U}_4$	31/31/30	–	$92n$	4.27	$5 \cdot 10^{-3}$

5.4 Concluding Remarks

In this chapter, we have illustrated that systems of the form (5.1), with linear operators with separable coefficients, discretized by a tensor basis on certain domains, can be treated directly in matrix or tensor form. In this setting, we have extended the POD-DEIM model order reduction method to the multilinear setting and illustrated how it could be used to massively reduce the dimension and complexity of systems of ODEs in two and three spatial dimensions. Furthermore, some very encouraging numerical experiments on complex problems such as the 2D and 3D viscous Burgers equation indicate a dramatic decrease in CPU time and memory requirements in the offline phase to construct the bases, especially when $d = 3$.

Nevertheless, the dense Kronecker structure of the reduced-order model obtained by the HO-POD-DEIM projection would incur unnecessary computational costs in the online phase when $d = 3$. To this end, we have shown how the novel T3-SYLV linear system solver from [143] can exploit the structure of the reduced-order model, resulting in a significant decrease in computational time in the online phase as well.

Future work would entail an analysis of the number of snapshots required by HO-POD-DEIM in comparison to POD-DEIM. It could also be of interest to extend the T3-SYLV solver to higher dimensions so that the HO-POD-DEIM strategy can be applied to PDEs with $d > 3$. Furthermore, the extension to the parameter-dependent setting can also be considered. This will result in an additional dimension that needs to be treated. Finally, the presented algorithm can certainly also benefit from a dynamic implementation as presented for the matrix setting in Section 4.2.3.

Chapter 6

Conclusion and Future Work

In this thesis, we have investigated order reduction strategies for semilinear matrix- and tensor-valued semilinear ODEs, and also systems of such equations. Our framework relies on forming reduced semilinear ODEs with the same structure as the original problem, depending on far smaller arrays. The framework also encompasses integrating the reduced model in its array form, which leads to further computational advantages. We first investigated the matrix-valued problem. Under the same framework, we studied separately the cases of the quadratic differential Riccati equation and of more general nonlinear functions, with the crucial difference being how the approximation spaces are chosen and constructed and how the nonlinear term is treated.

In the quadratic setting, thanks to important theoretical results from [19, 21] illustrating that the solution of the DRE lives in an invariant Krylov subspace, we have devised an efficient reduction procedure onto rational Krylov subspaces. In particular, we have exploited the fact that these spaces are nested to derive a novel two-phase algorithm consisting of independently constructing the approximation space and refining the accuracy of the final set of solutions. The algorithm relies on a new stopping criterion that considers the different approximation behavior of the algebraic and differential portions of the equation. Numerical experiments on benchmark problems and comparisons to state-of-the-art methods for solving the DRE have illustrated the efficiency and reliability of the new procedure.

In the more general setting, the challenge was to determine a pair of left-right approximation spaces that accurately captures the behavior of the nonlinear function, which also needs to be interpolated for efficiency. To this end, we have proposed a matrix-oriented POD-DEIM type order reduction strategy to drastically reduce the discretized dimension of the considered problem in two space variables, following the framework described above. The approximation spaces are constructed through a novel interpretation of the POD when applied to functions in two variables, preserving the (possible) symmetric structure in the original nonlinear function and the approximate solution. A dynamic procedure has also been proposed to limit the number of high-dimensional snapshots that contribute to the approximation space. Moreover, a two-sided DEIM methodology has been derived to handle the nonlinear term in low dimension. Hence,

the numerical treatment of the matrix reduced-order differential problem can take full advantage of both the matrix setting and the small dimension by exploiting effective, reliable exponential integrators.

In the final part of the thesis, we studied tensor-valued semilinear ODEs and extended the matrix-oriented POD-DEIM algorithm to the tensor setting. We have illustrated how to construct the approximation spaces efficiently, how to interpolate the nonlinear function, and, more importantly, how to integrate the tensor-valued reduced-order model. Furthermore, we also investigated systems of semilinear ODEs, which allows us to illustrate the efficiency of the proposed procedure on typical benchmark problems coming from industrial applications in fields such as Biology and Engineering.

The range of applicability of the presented procedures clearly relies on the ability to express the underlying problem in matrix or tensor form. For the quadratic DRE, as it appears in the numerical treatment of the LQR problem, the matrix structure is inherent. However, in the more general setting, where the ODEs stem from the space discretization of semilinear PDEs, the resulting structure of the problem is dependent on i) the computational domain $\mathbf{\Omega} \subset \mathbb{R}^d$; ii) the differential operator $\mathcal{L} : S_{\text{PDE}} \to \mathbb{R}$; iii) the discretization basis.

In this thesis, we have considered problems that stem from *simple* differential operators such as the Laplace operator (with a possible constant convection term), solved on parallelepipedal computational domains discretized by centered finite differences or spectral methods. The applicability of the proposed algorithms is, however, not restricted to this setting. As discussed in Section 1.2, more general operators, computational domains, and discretization bases (including certain finite elements) have already been successfully treated in array form. A future step would be to extend our framework to also deal with these more demanding problems. For instance, semilinear PDEs with non-constant convection terms on polygonal domains can be addressed following the framework outlined in [78]. At first glance it seems that the matrix-oriented reduction framework will be well suited to this setting, however, the time integration of the reduced model will need to be carefully adapted to deal with special multiterm semilinear matrix equations, involving Hadamard products.

Another restriction of the algorithms proposed in Chapters 4 and 5 is the number of nonlinear functional evaluations in the online phase. The numerical experiments have indicated that the presented algorithms typically have more DEIM interpolation indices than in the vector setting, due to the tensorized nature of the interpolation. Despite comparable online times to the vector setting, this could become more restrictive with more demanding nonlinear functions. In Remark 5.1 we suggest a possible alleviation to this restriction, which comes at the cost of losing the tensor structure of the interpolation. Alleviating this issue without affecting the structure of the approximation is certainly also a research line that can be explored in future work.

In this thesis, we have not considered any parameter-dependencies apart from time. It is necessary to extend the proposed framework to the parameter-dependent setting to broaden the range of applicability of the procedures. This will result in extra dimensions that need to be treated in the HO-POD-DEIM framework, resulting in a more demanding offline phase and a challenging online phase to deal with parameters beyond the training setup. This extension could go hand in hand with an adaptation of the framework to handle finite-element discretizations on tensorized bases with a careful selection of the norms to deal with more challenging industrial problems.

This work can, furthermore, be expanded in several other directions. In particular, for the DRE, several open questions remain regarding the stability properties of the rational Krylov approximate solution in the control theory context. These results could build on preliminary results presented for the Polynomial Krylov subspace [99]. Furthermore, the HO-POD-DEIM algorithm naturally calls for an extension beyond three dimensions, which will mainly rely on extending the T3-SYLV solver from Section 5.1 to higher dimensions. Finally, the new matrix- and tensor-oriented POD-DEIM algorithms will also benefit from a thorough analysis of the approximations' quality.

Bibliography

[1] H. Abou-Kandil, G. Freiling, V. Ionescu, and G. Jank, *Matrix Riccati Equations in Control and Systems Theory*. Birkhauser, Basel, 2003 (cit. on pp. 5, 41).

[2] J. Achterberg, "Model reduction of multidimensional dynamical systems by tensor decompositions", Master's thesis, Eindhoven University of Technology, Eindhoven, The Netherlands, 2008 (cit. on p. 88).

[3] J. V. Aguado, D. Borzacchiello, K. S. Kollepara, F. Chinesta, and A. Huerta, "Tensor representation of non-linear models using cross approximations", *J. Sci. Comput.*, vol. 81, no. 1, pp. 22–47, 2019 (cit. on p. 88).

[4] A. H. Al-Mohy and N. J. Higham, "A new scaling and squaring algorithm for the matrix exponential", *SIAM J. Matrix Anal. Appl.*, vol. 31, no. 3, pp. 970–989, 2010 (cit. on p. 31).

[5] S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening", *Acta Metall*, vol. 27, no. 6, pp. 1085–1095, 1979 (cit. on p. 82).

[6] D. Amsallem and C. Farhat, "An online method for interpolating linear parametric reduced-order models", *SIAM J. Sci. Comput.*, vol. 33, no. 5, pp. 2169–2198, 2011 (cit. on p. 70).

[7] V. Angelova, M. Hached, and K. Jbilou, "Approximate solutions to large nonsymmetric differential Riccati problems with applications to transport theory", *Numer. Linear Algebra Appl.*, vol. 27, no. 1, e2272, 2020 (cit. on p. 50).

[8] H. Antil, M. Heinkenschloss, and D. C. Sorensen, "Application of the discrete empirical interpolation method to reduced order modeling of nonlinear and parametric systems", in *Reduced order methods for modeling and computational reduction*, Springer, 2014, pp. 101–136 (cit. on p. 69).

[9] P. Antolin, A. Buffa, F. Calabro, M. Martinelli, and G. Sangalli, "Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization", *Comput. Methods Appl. Mech. Eng.*, vol. 285, pp. 817–828, 2015 (cit. on p. 15).

[10] A. Antoulas, C. Beattie, and S Gugercin, *Interpolatory methods for model reduction*. SIAM, Philidelphia, 2020 (cit. on p. 6).

[11] A. C. Antoulas, *Approximation of large-scale dynamical systems*. SIAM, Philidelphia, 2005, vol. 6 (cit. on pp. 4, 5, 42–44).

[12] W. F. Arnold and A. J. Laub, "Generalized eigenproblem algorithms and software for algebraic Riccati equations", *Proceedings of the IEEE*, vol. 72, no. 12, pp. 1746–1754, 1984 (cit. on p. 36).

[13] U. Ascher and L. Petzold, *Computer methods for ordinary differential equations and differential-algebraic equations*. SIAM, Philidelphia, 1998 (cit. on pp. 35, 37).

[14] U. M. Ascher, S. J. Ruuth, and B. T. Wetton, "Implicit-explicit methods for time-dependent partial differential equations", *SIAM J. Numer. Anal.*, vol. 32, no. 3, pp. 797–823, 1995 (cit. on pp. 24, 27–29, 74, 90).

[15] P. Astrid, S. Weiland, K. Willcox, and T. Backx, "Missing point estimation in models described by proper orthogonal decomposition", *IEEE Trans. Autom. Control*, vol. 53, no. 10, pp. 2237–2251, 2008 (cit. on p. 6).

[16] J. Baglama and L. Reichel, "Augmented implicitly restarted Lanczos bidiagonalization methods", *SIAM J. Sci. Comput.*, vol. 27, no. 1, pp. 19–42, 2005 (cit. on p. 76).

[17] M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera, "An 'empirical interpolation' method: Application to efficient reduced-basis discretization of partial differential equations", *C. R. Math. Acad. Sci. Paris*, vol. 339, no. 9, pp. 667–672, 2004 (cit. on pp. 6, 19).

[18] R. Bartels and G. Stewart, "Algorithm 432: Solution of the matrix equation $AX+XB = C$", *Comm. ACM*, vol. 15, no. 2, pp. 820 –826, 1972 (cit. on pp. 31, 32).

[19] M. Behr, P. Benner, and J. Heiland, "On an invariance principle for the solution space of the differential Riccati equation", *Proc. Appl. Math. Mech.*, vol. 18, no. 1, e201800031, 2018 (cit. on pp. 6, 46, 109).

[20] ——, "Solution formulas for differential Sylvester and Lyapunov equations", *Calcolo*, vol. 56:51, 2019 (cit. on p. 5).

[21] ——, "Galerkin trial spaces and Davison-Maki methods for the numerical solution of differential Riccati equations", *Appl. Math. Comput.*, p. 126 401, 2021 (cit. on pp. 5, 6, 46, 109).

[22] P. Benner and R. Byers, "An exact line search method for solving generalized continuous algebraic Riccati equations", *IEEE Trans. Automat Control*, vol. 43, no. 1, pp. 101–107, 1998 (cit. on p. 36).

[23] P. Benner, A. Cohen, M. Ohlberger, and K. Willcox, *Model reduction and approximation theory and algorithms*. SIAM, Philidelphia, 2017 (cit. on pp. 6, 41).

[24] P. Benner, V. Mehrmann, and D. Sorensen, *Dimension Reduction of Large-Scale Systems*. Springer-Verlag, Berlin/Heidelberg, Germany, 2005 (cit. on pp. 6, 52, 53).

[25] P. Benner and T. Breiten, "Two-sided projection methods for nonlinear model order reduction", *SIAM J. Sci. Comput.*, vol. 37, no. 2, B239–B260, 2015 (cit. on pp. 4, 6).

[26] P. Benner, Z. Bujanovic, P. Kurschner, and J. Saak, "A numerical comparison of different solvers for large-scale, continuous-time algebraic Riccati equations and LQR problems", *SIAM J. Sci. Comput.*, vol. 42, no. 2, A957–A996, 2020 (cit. on pp. 36, 39).

[27] P. Benner, S. Gugercin, and K. Willcox, "A survey of projection-based model reduction methods for parametric dynamical systems", *SIAM Rev*, vol. 57, no. 4, pp. 483–531, 2015 (cit. on pp. 6, 69, 70).

[28] P. Benner and H. Mena, "Rosenbrock methods for solving Riccati differential equations", *IEEE Trans Autom Control*, vol. 58, no. 11, pp. 2950–2956, 2013 (cit. on p. 35).

[29] G. Beylkin, J. M. Keiser, and L. Vozovoi, "A new class of time discretization schemes for the solution of nonlinear PDEs", *J. Comput. Phys.*, vol. 147, no. 2, pp. 362–387, 1998 (cit. on p. 25).

[30] D. A. Bini, B. Lannazzo, and B. Meini, *Numerical solution of algebraic Riccati equations*. SIAM, Philidelphia, 2012, vol. 9 (cit. on pp. 36, 39).

[31] R. Bitmead, M. Gevers, I. Petersen, and J. Kaye, "Monotonicity and stabilizability-properties of solutions of the Riccati difference equation: Propositions, lemmas, theorems, fallacious conjectures and counterexamples", *Systems & Control Letters*, vol. 5, no. 5, pp. 309–315, 1985 (cit. on p. 62).

[32] S. Blanes, "High order structure preserving explicit methods for solving linear-quadratic optimal control problems and differential games", *Numer. Algor.*, vol. 69, 271—290, 2015 (cit. on p. 41).

[33] D. Bonomi, A. Manzoni, and A. Quarteroni, "A matrix DEIM technique for model reduction of nonlinear parametrized problems in cardiac mechanics", *Comput. Methods Appl. Mech. Eng.*, vol. 324, pp. 300–326, 2017 (cit. on pp. 69, 93).

[34] T. Breiten, S. Dolgov, and M. Stoll, "Solving differential Riccati equations: A nonlinear space-time method using tensor trains", *Numer. Algebra Control Optim.*, vol. 11, no. 3, pp. 407–429, 2021 (cit. on p. 45).

[35] J. C. Butcher and N. Goodwin, *Numerical methods for ordinary differential equations*. Wiley Online Library, 2008, vol. 2 (cit. on p. 23).

[36] T. Byers, "Solving the algebraic Riccati equation with the matrix sign function", *Linear Alg. Appl.*, vol. 85, pp. 267–279, 1987 (cit. on p. 36).

[37] M. Caliari, P. Kandolf, A. Ostermann, and S. Rainer, "Comparison of software for computing the action of the matrix exponential", *BIT Numer. Math.*, vol. 54, no. 1, pp. 113–128, 2014 (cit. on p. 34).

[38] M. Caliari and A. Ostermann, "Implementation of exponential Rosenbrock-type integrators", *Applied Numerical Mathematics*, vol. 59, no. 3-4, pp. 568–581, 2009 (cit. on p. 83).

[39] K. Carlberg, R. Tuminaro, and P. Boggs, "Preserving Lagrangian structure in nonlinear model reduction with application to structural dynamics", *SIAM J. Sci. Comput.*, vol. 37, no. 2, B153–B184, 2015 (cit. on p. 69).

[40] J. E. Castillo, *Mathematical aspects of numerical grid generation*. SIAM, 1991 (cit. on p. 15).

[41] S. Chaturantabut and D. C. Sorensen, "Nonlinear model reduction via discrete empirical interpolation", *SIAM J. Sci. Comput.*, vol. 32, no. 5, pp. 2737–2764, 2010 (cit. on pp. 6, 19, 20, 64, 65, 72, 73, 76, 93, 94).

[42] ——, "Application of POD and DEIM on dimension reduction of non-linear miscible viscous fingering in porous media", *Math. Comput. Modell. Dyn. Syst.*, vol. 17, no. 4, pp. 337–353, 2011 (cit. on p. 88).

[43] ——, "A state space error estimate for POD-DEIM nonlinear model reduction", *SIAM J. Numer. Anal.*, vol. 50, no. 1, pp. 46–63, 2012 (cit. on p. 74).

[44] J. Chehab and M. Raydan, "Inexact Newton's method with inner implicit preconditioning of algebraic Riccati equations", vol. 36, no. 2, pp. 955–969, (cit. on p. 36).

[45] U. S.P.I.T. A. Committee, *Computational Science: Ensuring America's Competitiveness*. National Coordination Office for Information Technology Research & Development, 2005 (cit. on p. 3).

[46] M. J. Corless and A. E. Frazho, *Linear systems and control - an operator perspective*. Marcel Dekker, New York, 2003 (cit. on pp. 5, 41, 49).

[47] S. Cox and P. Matthews, "Exponential time differencing for stiff systems", *J. Comput. Phys.*, vol. 176, no. 2, pp. 430–455, 2002 (cit. on p. 25).

[48] M. Crouzeix, "Une méthode multipas implicite-explicite pour l'approximation des équations d'évolution paraboliques", *Numer. Math.*, vol. 35, no. 3, pp. 257–276, 1980 (cit. on pp. 26, 28).

[49] M. Daub, S. Waldherr, F. Allgöwer, P. Scheurich, and G. Schneider, "Death wins against life in a spatially extended apoptosis model", *Biosystems*, vol. 108, pp. 45–51, 2012 (cit. on p. 106).

[50] M. Daub, "Mathematical modeling and numerical simulations of the extrinsic pro-apoptotic signaling pathway", PhD thesis, University of Stuttgart, 2013 (cit. on pp. 106, 107).

[51] M. C. D'Autilia, I. Sgura, and V. Simoncini, "Matrix-oriented discretization methods for reaction–diffusion PDEs: Comparisons and applications", *Comput. Math. Appl.*, pp. 2067–2085, 2020 (cit. on pp. 4, 13, 14, 23, 25, 26, 29, 32, 74, 75, 90, 98).

[52] E. Davison and M. Maki, "The numerical solution of the matrix Riccati differential equation", *IEEE Trans. Autom. Control*, pp. 71–73, 1973 (cit. on pp. 23, 24, 35).

[53] A. De Wit, "Spatial patterns and spatiotemporal dynamics in chemical systems", in. John Wiley & Sons, Ltd, 1999, pp. 435–513 (cit. on p. 87).

[54] R. Dedden, "Model order reduction using the discrete empirical interpolation method", Master's thesis, TU Delft, 2012 (cit. on p. 69).

[55] L. Dieci, "Numerical integration of the differential Riccati equation and some related issues", *SIAM J. Numer. Anal.*, vol. 29, no. 3, pp. 781–815, 1992 (cit. on pp. 23, 24, 35).

[56] L. Dieci and T. Eirola, "Preserving monotonicity in the numerical solution of Riccati differential equations", *Numer. Math.*, vol. 74, no. 1, pp. 35–47, 1996 (cit. on p. 62).

[57] Z. Drmač and S. Gugercin, "A new selection operator for the discrete empirical interpolation method—improved a priori error bound and extensions", *SIAM J. Sci. Comput.*, vol. 38, no. 2, A631–A648, 2016 (cit. on pp. 21, 72, 73, 76, 94).

[58] V. Druskin and L. Knizhnerman, "Extended Krylov subspaces: Approximation of the matrix square root and related functions", *SIAM J. Matrix Anal. Appl.*, vol. 19, no. 3, pp. 755–771, 1998 (cit. on p. 17).

[59] V. Druskin and V. Simoncini, "Adaptive rational Krylov subspaces for large-scale dynamical systems", *Systems & Control Letters*, vol. 60, pp. 546–560, 2011 (cit. on pp. 16, 18, 44, 53).

[60] L. C. Evans and J. Spruck, "Motion of level sets by mean curvature. I", *J. Differ. Geom.*, vol. 33, no. 3, pp. 635–681, 1991 (cit. on p. 82).

[61] M. Farrashkhalvat and J. P. Miles, *Basic structured grid generation*. Butterworth & Heinemann, 2003 (cit. on p. 15).

[62] C. A. Fletcher, "Generating exact solutions of the two-dimensional Burgers' equations", *Int. J. Numer. Methods Fluids*, vol. 3, pp. 213–216, 1983 (cit. on p. 99).

[63] D. Fortunato and A. Townsend, "Fast Poisson solvers for spectral methods", *IMA Journal of Numerical Analysis*, vol. 40, no. 3, pp. 1994–2018, 2020 (cit. on p. 15).

[64] R. A Friesner, L. Tuckerman, B. Dornblaser, and T. Russo, "A method for exponential propagation of large systems of stiff nonlinear differential equations", *J. Sci. Comput.*, vol. 4, no. 4, pp. 327–354, 1989 (cit. on p. 25).

[65] M. Frittelli and I. Sgura, "Matrix-oriented FEM formulation for stationary and time-dependent PDEs on x-normal domains", *arXiv preprint arXiv:2109.01173*, 2021 (cit. on p. 15).

[66] G. Gambino, M. Lombardo, and M. Sammartino, "Pattern selection in the 2D FitzHugh–Nagumo model", *Ricerche di Matematica*, vol. 68, no. 2, pp. 535–549, 2019 (cit. on p. 97).

[67] Q. Gao and M. Zou, "An analytical solution for two and three dimensional nonlinear Burgers' equation", *Appl. Math. Modell.*, vol. 45, pp. 255 –270, 2017 (cit. on p. 103).

[68] U. Z. George, A. Stéphanou, and A. Madzvamuse, "Mathematical modelling and numerical simulations of actin dynamics in the eukaryotic cell", *J. Math. Biol.*, vol. 66, no. 3, pp. 547–593, 2013 (cit. on p. 87).

[69] M. Gevers, R. Bitmead, I. Petersen, and J. Kaye, "When is the solution of the Riccati equation stabilizing at every instant?", in *Frequency Domain and State Space Methods for Linear Systems*, North-Holland, 1986, pp. 531–540 (cit. on p. 62).

[70] G. H. Golub and C. F. van Loan, *Matrix Computations*, Fourth. Johns Hopkins University Press, Baltimore, 2013 (cit. on pp. 12, 19, 31, 76).

[71] L. Grasedyck, "Existence and computation of low Kronecker-rank approximations for large linear systems of tensor product structure", *Computing*, vol. 72, no. 3–4, pp. 247–265, Oct. 2004 (cit. on p. 41).

[72] ——, "Existence of a low rank or $\mathcal{H}$-matrix approximant to the solution of a Sylvester equation", *Numer. Linear Algebra Appl.*, vol. 11, no. 4, pp. 371–389, Oct. 2004 (cit. on p. 41).

[73] C. Gu, "QLMOR: A projection-based nonlinear model order reduction approach using quadratic-linear representation of nonlinear systems", *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 9, pp. 1307–1320, 2011 (cit. on pp. 6, 88).

[74] N. Guglielmi and V. Simoncini, "On the existence and approximation of a dissipating feedback", *arXiv preprint arXiv:1811.00069*, 2019 (cit. on p. 50).

[75] Y. Güldogan, M. Hached, K. Jbilou, and M. Kurulaya, "Low rank approximate solutions to large-scale differential matrix Riccati equations", *Applicationes Mathematicae*, vol. 45, no. 2, pp. 233–254, 2018 (cit. on pp. 5, 43, 44, 46).

[76] N. Halko, P.-G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions", *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011 (cit. on p. 104).

[77] E. Hansen and T. Stillfjord, "Convergence analysis for splitting of the abstract differential Riccati equation", *SIAM J. Numer. Anal.*, vol. 52, no. 6, pp. 3128–3139, 2014 (cit. on p. 39).

[78] Y. Hao and V. Simoncini, "Matrix equation solving of PDEs in polygonal domains using conformal mappings", *J. Numer. Math.*, vol. 29, no. 3, pp. 221–244, 2021 (cit. on pp. 15, 110).

[79] M. Heyouni and K. Jbilou, "An extended block Arnoldi algorithm for large-scale solutions of the continuous-time algebraic Riccati equation", *Electron. Trans. Numer Anal.*, vol. 33, pp. 53–62, 2009 (cit. on p. 44).

[80] N. J. Higham, "The scaling and squaring method for the matrix exponential revisited", *SIAM J. Matrix Anal. Appl.*, vol. 26, no. 4, pp. 1179–1193, 2005 (cit. on p. 31).

[81] ——, *Functions of matrices: theory and computation*. SIAM, Philidelphia, 2008 (cit. on pp. 25, 73).

[82] M. Hinze and S. Volkwein, "Proper orthogonal decomposition surrogate models for nonlinear dynamical systems: Error estimates and suboptimal control", in

Dimension reduction of large-scale systems, Springer-Verlag, Berlin/Heidelberg, 2005, pp. 261–306 (cit. on p. 6).

[83] M. Hochbruck and A. Ostermann, "Explicit exponential Runge–Kutta methods for semilinear parabolic problems", *SIAM J. Numer. Anal.*, vol. 43, no. 3, pp. 1069–1090, 2005 (cit. on pp. 24–26).

[84] ——, "Exponential integrators.", *Acta Numer.*, vol. 19, pp. 209–286, 2010 (cit. on pp. 24–26).

[85] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve", *The Journal of physiology*, vol. 117, no. 4, p. 500, 1952 (cit. on p. 87).

[86] W. Hundsdorfer and J. G. Verwer, *Numerical solution of time-dependent advection-diffusion-reaction equations*. Springer Science & Business Media, Berlin, Germany, 2013, vol. 33 (cit. on pp. 23, 27, 28, 39).

[87] I. M. Jaimoukha and E. M. Kasenally, "Krylov subspace methods for solving large Lyapunov equations", *SIAM J. Numer. Anal.*, vol. 31, no. 1, pp. 227–251, 1994 (cit. on p. 44).

[88] L. Ju, J. Zhang, L. Zhu, and Q. Du, "Fast explicit integration factor methods for semilinear parabolic equations", *J. Sci. Comput.*, vol. 62, no. 2, pp. 431–455, 2015 (cit. on p. 82).

[89] B. Karasözen, M. Uzunca, and T. Küçükseyhan, "Model order reduction for pattern formation in Fitzhugh-Nagumo equations", in *Numerical Mathematics and Advanced Applications ENUMATH 2015*, Springer, 2016, pp. 369–377 (cit. on p. 88).

[90] ——, "Reduced order optimal control of the convective Fitzhugh–Nagumo equations", *Comput. Math. Appl.*, vol. 79, no. 4, pp. 982–995, 2020 (cit. on p. 88).

[91] B. Karasözen, S. Yıldız, and M. Uzunca, "Structure preserving model order reduction of shallow water equations", *Math. Methods Appl. Sci.*, vol. 44, no. 1, pp. 476–492, 2021 (cit. on p. 88).

[92] C. Kenny, A. Laub, and P. Papadopoulos, "Matrix sign function algorithms for Riccati equations", in *IMA Conference on Control: Modelling, Computation, Information*, I. C. S. Press, Ed., 1992 (cit. on p. 36).

[93] G. Kirsten, "Multilinear POD-DEIM model reduction for 2D and 3D semilinear systems of differential equations", *To appear in J. Comput. Dyn.*, 2021 (cit. on p. 87).

[94] G. Kirsten and V. Simoncini, "Order reduction methods for solving large-scale differential matrix Riccati equations", *SIAM J. Sci. Comput.*, vol. 42, no. 4, A2182–A2205, 2020 (cit. on p. 41).

[95] D. Kleinman, "On an iterative technique for Riccati equation computations", *IEEE Trans. Autom. Contr.*, vol. 13, no. 1, 1968 (cit. on p. 36).

[96] P. Knupp and S. Steinberg, *Fundamentals of grid generation*. CRC press, 2020 (cit. on p. 15).

[97] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications", *SIAM Rev*, vol. 51, no. 3, pp. 455–500, 2009 (cit. on pp. 12, 90).

[98] M. Konstantinov and G. Pelova, "Sensitivity of the solutions to differential matrix Riccati equations", *IEEE Trans. Autom. Control*, vol. 36, no. 2, pp. 213–215, 1991 (cit. on p. 50).

[99] A. Koskela and H. Mena, "Analysis of Krylov subspace approximation to large scale differential Riccati equations", *Electron. Trans. Numer. Anal.*, vol. 52, pp. 431–454, 2020 (cit. on pp. 5, 43, 44, 46, 62, 111).

[100] B. Kramer, "Model reduction of the coupled Burgers equation in conservation form", PhD thesis, Virginia Tech, 2011 (cit. on p. 88).

[101] B. Kramer and K. E. Willcox, "Nonlinear model order reduction via lifting transformations and proper orthogonal decomposition", *AIAA Journal*, vol. 57, no. 6, pp. 2297–2307, 2019 (cit. on pp. 6, 88).

[102] M. Krusemeyer, *Differential Equations*. Macmillan College Publishing, New York, 1994 (cit. on p. 25).

[103] V. Kucera, "A review of the matrix Riccati equation", *Kibernetika*, vol. 9, no. 1, pp. 42–61, 1973 (cit. on pp. 42, 45, 50–52).

[104] K. Kunisch and S. Volkwein, "Control of the Burgers equation by a reduced-order approach using proper orthogonal decomposition", *J. Optim. Theory Appl.*, vol. 102, no. 2, pp. 345–371, 1999 (cit. on pp. 6, 99).

[105] H. Kwakernaak and R. Sivan, *Linear optimal control systems*. Wiley-interscience, New York, 1972, vol. 1 (cit. on pp. 5, 41, 49).

[106] P. Lancaster and L. Rodman, *The Algebraic Riccati Equations*, Oxford, Ed. Clarendon Press, 1995 (cit. on p. 36).

[107] N. Lang, "Numerical methods for large-scale linear time-varying control systems and related differential matrix equations", PhD thesis, Technische Universitaet Chemnitz, 2017 (cit. on pp. 35, 45).

[108] Y. Lin and V. Simoncini, "Minimal residual methods for large scale Lyapunov equations", *Applied Num. Math.*, vol. 72, pp. 52–71, 2013 (cit. on p. 18).

[109] ——, "A new subspace iteration method for the algebraic Riccati equation", *Numer. Linear Algebra Appl.*, vol. 22, no. 1, pp. 26–47, 2015 (cit. on p. 44).

[110] P. K. Maini and H. G. Othmer, *Mathematical Models for Biological Pattern Formation*, ser. The IMA Volumes in Mathematics and its Applications - Frontiers in application of Mathematics. Springer-Verlag, New York, 2001 (cit. on p. 87).

[111] H. Malchow, S. Petrovskii, and E. Venturino, *Spatiotemporal Patterns in Ecology and Epidemiology: Theory, Models, and Simulations*. Chapman & Hall, CRC, London, 2008 (cit. on p. 87).

[112] A. Mantzaflaris, B. Jüttler, B. N. Khoromskij, and U. Langer, "Low rank tensor methods in Galerkin-based isogeometric analysis", *Comput. Methods Appl. Mech. Eng.*, vol. 316, pp. 1062–1085, 2017 (cit. on p. 15).

[113] *Matlab 7*, r2013b, The MathWorks, 2013 (cit. on p. 9).

[114] H. Mena, "Numerical solution of differential Riccati equations arising in optimal control problems for parabolic partial differential equations", PhD thesis, Escuela Politécnica Nacional, 2007 (cit. on pp. 23, 24, 35).

[115] H. Mena, A. Ostermann, L.-M. Pfurtscheller, and C. Piazzola, "Numerical low-rank approximation of matrix differential equations", *J. Comput. Appl. Math.*, vol. 340, pp. 602–614, 2018 (cit. on pp. 23, 24, 35, 37, 60).

[116] R. Minster, A. K. Saibaba, and M. E. Kilmer, "Randomized algorithms for low-rank tensor decompositions in the Tucker format", *SIAM J. Math. Data Sci.*, vol. 2, no. 1, pp. 189–215, 2020 (cit. on p. 104).

[117] D. Mott, E. Oran, and B. van Leer, "A quasi-steady-state solver for the stiff ordinary differential equations of reaction kinetics", *J. Comput. Phys.*, vol. 164, no. 2, pp. 407–428, 2000 (cit. on p. 25).

[118] J. Murray, *Mathematical biology II: spatial models and biomedical applications*. Springer-Verlag, Berlin, 2001, vol. 3 (cit. on p. 87).

[119] F. Negri, A. Manzoni, and D. Amsallem, "Efficient model reduction of parametrized systems by matrix discrete empirical interpolation", *J. Comput. Phys.*, vol. 303, pp. 431–454, 2015 (cit. on pp. 69, 93).

[120] N. C. Nguyen, A. T. Patera, and J. Peraire, "A 'best points' interpolation method for efficient approximation of parametrized functions", *Int J. Numer. Methods Eng.*, vol. 73, no. 4, pp. 521–543, 2008 (cit. on p. 6).

[121] *Oberwolfach model reduction benchmark collection*. 2003. [Online]. Available: https://morwiki.mpi-magdeburg.mpg.de/morwiki/index.php/Category:Oberwolfach (cit. on p. 52).

[122] G. M. Oxberry, T. Kostova-Vassilevska, W. Arrighi, and K. Chand, "Limited-memory adaptive snapshot selection for proper orthogonal decomposition", *Int. J. Numer. Meth. Eng.*, vol. 109, pp. 198–217, 2017 (cit. on p. 78).

[123] D. Palitta and V. Simoncini, "Matrix-equation-based strategies for convection–diffusion equations", *BIT Numer. Math.*, vol. 56, no. 2, pp. 751–776, 2016 (cit. on pp. 4, 13–15).

[124] A. T. Patera and G. Rozza, *Reduced basis approximation and a posteriori error estimation for parametrized partial differential equations*. MIT Cambridge, MA, USA, 2007 (cit. on p. 6).

[125] T. Penzl, "A cyclic low-rank Smith method for large sparse Lyapunov equations", *SIAM J. Sci. Comput.*, vol. 21, no. 4, pp. 1401–1418, 2000 (cit. on p. 44).

[126] M. A. Poubelle, R. Bitmead, and M. Gevers, "Fake algebraic Riccati techniques and stability", *IEEE Trans. Autom. Control*, vol. 33, no. 4, pp. 379–381, 1988 (cit. on p. 62).

[127] C. E. Powell, D. Silvester, and V. Simoncini, "An efficient reduced basis solver for stochastic Galerkin matrix equations", *SIAM J. Sci. Comput.*, vol. 39, no. 1, A141–A163, 2017 (cit. on p. 15).

[128] A. Quarteroni, *Numerical Models for Differential Problems*, ser. MS&A - Modeling, Simulation and Applications. Springer-Verlag, Milan, 2017, vol. 8 (cit. on p. 87).

[129] W. Reid, *Riccati differential equations*. Academic Press, New York, 1972 (cit. on p. 41).

[130] A. Ruhe, "Rational Krylov sequence methods for eigenvalue computation", *Lin. Alg. Appl.*, vol. 58, pp. 391–405, 1984 (cit. on pp. 17, 44).

[131] ——, "The rational Krylov algorithm for nonsymmetric eigenvalue problems. III:Complex shifts for real matrices", *BIT Numer. Math.*, vol. 34, pp. 165–176, 1994 (cit. on pp. 18, 44).

[132] S. J. Ruuth, "Implicit-explicit methods for reaction-diffusion problems in pattern formation", *J. Math. Biol.*, vol. 34, no. 2, pp. 148–176, 1995 (cit. on pp. 24, 27).

[133] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, Philidelphia, 2003 (cit. on p. 16).

[134] J. Saak, M. Köhler, and P. Benner, *M-m.e.s.s.-1.0.1 – the matrix equations sparse solvers library*, DOI:10.5281/zenodo.50575, see also:www.mpi-magdeburg.mpg.de/projects/mess, Apr. 2016 (cit. on pp. 36, 42, 45, 56, 59).

[135] S. Sahyoun and S. M. Djouadi, "Nonlinear model reduction using space vectors clustering POD with application to the Burgers' equation", in *2014 American Control Conference*, IEEE, 2014, pp. 1661–1666 (cit. on p. 88).

[136] A. K. Saibaba, "HOID: Higher order interpolatory decomposition for tensors based on Tucker representation", *SIAM J. Matrix Anal. Appl.*, vol. 37, no. 3, pp. 1223–1249, 2016 (cit. on pp. 88, 90).

[137] G. Sangalli and M. Tani, "Isogeometric preconditioners based on fast solvers for the Sylvester equation", *SIAM J. Sci. Comput.*, vol. 38, no. 6, A3644–A3671, 2016 (cit. on p. 15).

[138] I. Sgura, B. Bozzini, and D. Lacitignola, "Numerical approximation of Turing patterns in electrodeposition by ADI methods", *J. Comput. Appl. Math.*, vol. 236, no. 16, pp. 4132–4147, 2012 (cit. on p. 29).

[139] J. A. Sherratt and M. A. Chaplain, "A new mathematical model for avascular tumour growth", *J. Math. Biol.*, vol. 43, no. 4, pp. 291–312, 2001 (cit. on p. 87).

[140] V. Simoncini, "A new iterative method for solving large-scale lyapunov matrix equations", *SIAM J. Sci. Comput.*, vol. 29, no. 3, pp. 1268–1288, 2007 (cit. on pp. 49, 53).

[141] ——, "Analysis of the rational Krylov subspace projection method for large-scale algebraic Riccati equations", *SIAM J. Matrix Anal. Appl.*, vol. 37, no. 4, pp. 1655–1674, 2016 (cit. on pp. 6, 18, 32, 42, 44, 49, 50).

[142] ——, "Computational methods for linear matrix equations", *SIAM Rev.*, vol. 58, no. 3, pp. 377–441, 2016 (cit. on pp. 4, 13, 14, 16).

[143] ——, "Numerical solution of a class of third order tensor linear equations", *BUMI*, vol. 13, no. 3, pp. 429–439, 2020 (cit. on pp. 91, 92, 108).

[144] V. Simoncini, D. B. Szyld, and M. Monsalve, "On two numerical methods for the solution of large-scale algebraic Riccati equations", *IMA J. Numer. Anal.*, vol. 34, no. 3, pp. 904–920, 2014 (cit. on pp. 6, 44, 58).

[145] H. Song, L. Jiang, and Q. Li, "A reduced order method for Allen–Cahn equations", *J. Comput. Appl. Math.*, vol. 292, pp. 213–229, 2016 (cit. on p. 82).

[146] D. C. Sorensen and M. Embree, "A DEIM induced CUR factorization", *SIAM J. Sci. Comput.*, vol. 38, no. 3, A1454–A1482, 2016 (cit. on p. 72).

[147] R. Ştefănescu and A. Sandu, "Efficient approximation of sparse Jacobians for time-implicit reduced order models", *Int. J. Numer. Methods Fluids*, vol. 83, no. 2, pp. 175–204, 2017 (cit. on p. 69).

[148] R. Ştefănescu, A. Sandu, and I. M. Navon, "Comparison of POD reduced order strategies for the nonlinear 2D shallow water equations", *Int. J. Numer. Methods Fluids*, vol. 76, no. 8, pp. 497–521, 2014 (cit. on p. 99).

[149] T. Stillfjord, "Low-rank second-order splitting of large-scale differential Riccati equations", *IEEE Trans. Automat. Control*, vol. 60, no. 10, pp. 2791–2796, 2015 (cit. on pp. 23, 24, 35, 38).

[150] ——, "Adaptive high-order splitting schemes for large-scale differential Riccati equations", *Numer. Algor.*, vol. 78, no. 4, pp. 1129–1151, 2018 (cit. on pp. 35, 38, 45, 59, 61).

[151] ——, "Singular value decay of operator-valued differential Lyapunov and Riccati equations", *SIAM J. Control Optim.*, vol. 56, no. 5, pp. 3598–3618, 2018 (cit. on p. 41).

[152] J. C. Strikwerda, *Finite difference schemes and partial differential equations*. SIAM, Philidelphia, 2004 (cit. on p. 23).

[153] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Artech House, Boston, 1995 (cit. on p. 25).

[154] J. Thompson, Z. Warsi, and C. Mastin, *Numerical grid generation: foundations and applications*. Elsevier North-Holland, Inc., 1985 (cit. on p. 15).

[155] P. Tiso and D. J. Rixen, "Discrete empirical interpolation method for finite element structural dynamics", in *Topics in Nonlinear Dynamics, Volume 1*, Springer, 2013, pp. 203–212 (cit. on p. 69).

[156] L. N. Trefethen, *Spectral Methods in MatLab*. SIAM, Philidelphia, 2000, ISBN: 0898714656 (cit. on p. 33).

[157] A. Tveito, H. P. Langtangen, B. F. Nielsen, and X. Cai, *Elements of Scientific Computing*, ser. Texts in Computational Science and Engineering. Springer-Verlag, Berlin, 2010 (cit. on p. 87).

[158] V. K. Vanag, "Waves and patterns in reaction–diffusion systems. Belousov–Zhabotinsky reaction in water-in-oil microemulsions", *Phys. Usp.*, vol. 47, no. 9, p. 923, 2004 (cit. on p. 87).

[159] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A new truncation strategy for the higher-order singular value decomposition", *SIAM J. Sci. Comput.*, vol. 34, no. 2, A1027–A1052, 2012 (cit. on p. 89).

[160] J. Varah, "Stability restrictions on second order, three level finite difference schemes for parabolic equations", *SIAM J. Numer. Anal.*, vol. 17, no. 2, pp. 300–309, 1980 (cit. on p. 26).

[161] Y. Wang, I. M. Navon, X. Wang, and Y. Cheng, "2D Burgers equation with large Reynolds number using POD/DEIM and calibration", *Int. J. Numer. Methods Fluids*, vol. 82, no. 12, pp. 909–931, 2016 (cit. on pp. 88, 98–100, 102, 103).

[162] J. K. White, "A trajectory piecewise-linear approach to model order reduction of nonlinear dynamical systems", PhD thesis, Massachusetts Institute of Technology, 2003 (cit. on p. 6).

[163] D. Wirtz, D. C. Sorensen, and B. Haasdonk, "A posteriori error estimation for DEIM reduced nonlinear dynamical systems", *SIAM J. Sci. Comput.*, vol. 36, no. 2, A311–A338, 2014 (cit. on pp. 69, 74).