Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft Institute of Applied Mathematics

Nonlinear Model Order Reduction using POD/DEIM for Optimal Control of Burgers' Equation

A thesis submitted to the
Delft Institute of Applied Mathematics
in partial fulfillment of the requirements

for the degree

MASTER OF SCIENCE
in
APPLIED MATHEMATICS

by

Manuel Matthias Baumann

Delft, The Netherlands

July 1, 2013

TU Delft Delft University of Technology

MSc THESIS APPLIED MATHEMATICS

"Nonlinear Model Order Reduction using POD/DEIM for Optimal Control of Burgers' Equation"

Manuel M. Baumann

Delft University of Technology

Daily supervisor	**Responsible professor**
Dr. ir. Martin van Gijzen	Prof. dr. ir. Arnold Heemink
Dr. Marielba Rojas	

Committee members

Prof. dr. ir. Arnold Heemink	Prof. dr. ir. Cornelis Vuik
Dr. ir. Martin van Gijzen	Dr. Marielba Rojas

July 1, 2013

Delft, The Netherlands

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

This Master thesis has been written within the Erasmus Mundus Master's program *Computer Simulations for Science and Engineering (COSSE)*[1]. During the past two years of my studies, I had the opportunity to study for one year at the Royal Institute of Technology in Stockholm, Sweden, as well as the Technical University in Delft, the Netherlands. But COSSE offers much more than the possibility to study at two world-known universities in the field of Computer Science and Applied Mathematics and, therefore, I am mostly thankful for the friendship and scientific influence of all the great people I met from all over the world.

Since the beginning of my studies at TU Berlin more than five years ago, I was also supported by a scholarship of the *Friedrich-Ebert foundation (FES)*[2]. The FES is a political foundation that mostly supports students from a so-called working class background. During my time as a stipend, I went to many interesting workshops and seminars, especially in the field of development politics.

I would like to thank professor Volker Mehrmann and professor Günter Bärwolff from TU Berlin who supported my application for COSSE. Because of them I felt encouraged to apply for an international study program and the Erasmus Mundus scholarship. Professor Mehrmann also gave crucial scientific advices to my thesis work whenever I was in Berlin.

Most of all, I would like to thank the (daily) supervisors of my Master thesis, Martin van Gijzen and Marielba Rojas. Our weekly meetings, known as the *3M meetings*, always were inspiring, encouraging and, on a personal level, delightful. Both of you motivated me to continue my scientific career with a PhD. Moreover, I would like to thank Sławomir Szklarz who helped me in many respects since I arrived in Delft.

During the last two years of studying abroad, the support of my parents never ended and I am very thankful for many of their advices during our Skype talks. Last but not least, I would like to thank *Jana*. Only because of you, I never felt lonely away from home.

[1] For more information, please visit: `http://www.kth.se/en/studies/programmes/em/cosse`
[2] Again, more information is available online: `http://www.fes.de/studienfoerderung`

Chapter 1

Introduction

1.1 Motivation

The efficient numerical solution of time-dependent partial differential equations (PDE) plays an important role in various fields of application such as fluid dynamics [14, 54], electromagnetics [37, 47] or heat transfer [25]. Often, those differential equations consist of nonlinear terms which are challenging to solve numerically. Furthermore, in engineering applications, one is often interested in an optimal solution of the considered PDE with respect to a certain objective function. This leads to the mathematical field of PDE-constrained optimization where a cost function is minimized and the PDE is considered as a constraint. As an example, one might imagine the flow of a fluid in a certain domain which is described by the nonlinear, time-dependent Navier-Stokes equations, cf. [15]. The objective in that context might be to optimize the shape of the domain such that a desired flow behavior is obtained. This approach is, for example, been used in modern airplane design, cf. [6, 21, 34] and the introductory examples therein.

Since the numerical treatment of modern engineering problems requires a huge computational effort, one is interested in mathematical methods that reduce the dimension of the underlying dynamical system in order to speedup the optimization problem tremendously and, at the same time, lead to a good approximation to the optimal solution of the original full-size problem. This leads to the field of Model Order Reduction (MOR) which is a relatively new field of mathematics, especially when the considered dynamical systems are nonlinear.

1.2 Previous work

Algorithms for MOR are best known in the context of linear control theory. Prominent textbooks on the field of control theory are for instance [33, 38, 49], where so-called *input-*

state-output systems of the following form are considered,

$$\begin{aligned} \dot{x} &= Ax + Bu, \\ y &= Cx + Du, \end{aligned} \tag{1.1}$$

where u is the input, x is the state of the system, and y is the output of the system. The matrices A, B, C, D are assumed to have the appropriate dimensions. Note that (1.1) can be considered as a dynamical system for the special case when C equals the identity matrix and $D = 0$. Then (1.1) simplifies to $\dot{y} = Ay + Bu$, and we consider u as the control and $y(u)$ as the solution of the dynamical system that depends on u. Dynamical systems of this form arise for instance after spatial discretization of (linear) partial differential equation with a control.

The aim of model order reduction is to derive a system of the form (1.1) with a similar input-output relation but a state variable x of much smaller dimension. In the case of a linear system, there exist a wide range of algorithms that obtain a reduced system and that even guarantee a priori error estimates. The most important methods for linear MOR are:

- Balanced Truncation,
- Moment Matching,
- Hankel-norm approximation,
- Krylov methods.

All of the methods above are discussed in detail in textbooks on linear MOR (cf. [1, 44]) or in lectures notes of courses on that subject [40]. A brief presentation can also be found in the corresponding literature study for this thesis [5].

Nonlinear dynamical systems with control are of the form:

$$\dot{y} = f(y) + Bu, \tag{1.2}$$

with f nonlinear. There exist relatively new numerical methods for MOR of (1.2) which are of current scientific interest. Among them the method which is widely used in practice is the so-called Proper Orthogonal Decomposition (POD) which is explained in detail in [2, 53]. A detailed historical review of POD can be found in [28]. Therein, the authors state the correspondence of POD, which is also known as Karhunen-Loève decomposition (KLD), and the Principal Component Analysis (PCA) [26] which is used in statistics. According to [28], the first usage of POD for MOR of dynamical system date back to the 1990s, cf. [18]. Note that POD does not lead to a reduction of the involved nonlinearity and therefore, although the dimension of POD-reduced models is lower, the complexity of the nonlinear terms remains the same. The Discrete Empirical Interpolation Method (DEIM) proposed by [10, 12] in 2010 is an extension of POD that aims to construct reduced

systems that do not depend at all on the dimension of the full-order model. DEIM has already been applied to complex dynamical processes, cf. [11], and has shown to lead to a huge gain in computational complexity. Further methods for MOR of nonlinear system are a generalization of balanced truncation to nonlinear systems [30] as well as a generalization of moment matching [3, 13].

Since we want to apply MOR within the framework of optimal control of (nonlinear) partial differential equations, we also want to refer to standard literature from the field of optimal control and PDE-constrained optimization. A theoretical approach to that field is for instance given by [31, 50] where questions like existence and uniqueness of an optimal solution are considered. A more practical introduction to PDE-constrained optimization is given in [9, 22, 32] and the technical report [23]. An introduction to classical optimization methods is for instance given by [36, 39].

1.3 Research Goals

The aim of this Master thesis is the evaluation of MOR techniques when applied in the context of optimal control of nonlinear partial differential equations. In more detail, we are interested in the approximation error of the optimal state and the computational benefit when different optimization algorithms are applied to a reduced model obtained via a POD-DEIM scheme. In [52], the author presents a general approach of the application of POD to PDE-constrained optimization. As a standard test case, we consider the optimal control of Burgers' equation as in [23, 41]. In [29], the authors applied optimal control algorithms to a POD-reduced model of Burgers' equation with good results regarding accuracy and performance of the reduced model. The main contribution of the present work is, however, to extend the approach in [29] by applying DEIM to the POD-reduced Burgers' model. Especially for a small viscosity parameter ν, the full-size Burgers' model requires to be of large dimension in order to guarantee numerical stability. Since the POD-DEIM reduced model is completely independent of the full-model dimension, significant computational speedup can be obtained for any value of the parameter ν. We present a comparison of the optimal control of Burgers' equation when POD and POD-DEIM is applied using three different optimization algorithms. The computational benefit of DEIM within the framework of optimal control has been pointed out for each of the considered optimization methods.

1.4 Chapter outline

The thesis work is structured as follows: In Chapter 2 we present an overview of MOR methods for nonlinear dynamical system. This includes the method of POD and its improvement, the DEIM method. We demonstrate the two methods by deriving a POD-DEIM reduced model for the numerical solution of Burgers' equation in the end of

Chapter 2. In Chapter 3, we present some methods for PDE-constrained optimization which can be transformed in implicitly constrained optimization problems. We will focus on two classes of algorithms, one that takes only information of the first derivative of the objective function into account (first-order methods) and another approach that also considers information of the Hessian of the objective function (second-order methods). Also Chapter 3 ends with an application of the presented methods to the (full-order) one-dimensional unsteady Burgers' equation. Chapter 4 deals with a detailed comparison of POD and POD-DEIM when applied to the optimal control of Burgers' equation. We present a complete derivation and implementation of both MOR methods and give an algorithm that solves the optimal control problem on both reduced models. Also a detailed consideration of the approximation error and the computational gain is presented for both, a purely POD-reduced model and the model obtained by POD-DEIM. In chapter 5 we given an outlook on future research questions that have not been considered in this thesis.

Chapter 2

Model order reduction for nonlinear dynamical systems

In this chapter we present two methods for model order reduction of nonlinear dynamical systems. First, we present the method of Proper Orthogonal Decomposition (POD) which constructs a matrix $U_\ell \in \mathbb{R}^{N\times \ell}$ such that the subspace $\mathcal{U}_\ell := \text{range}(U_\ell)$ is low-dimensional and a Galerkin projection of the full-order dynamical system onto $\mathcal{U}_\ell$ still captures most of the dynamical behavior of the original system. An improvement of POD that also reduces the dimension of the involved nonlinearity is given by the Discrete Empirical Interpolation Method (DEIM). We end the chapter with an application of POD-DEIM to the nonlinear Burgers' equation and present both accuracy and performance of the reduced model in comparison to the numerical solution of the full-order model. We will put a special focus on the performance benefit of POD-DEIM compared to a purely POD-reduced model.

2.1 The Proper Orthogonal Decomposition (POD)

The focus of this chapter lies on nonlinear dynamical systems of the form

$$\frac{d}{dt}\mathbf{y}(t) = A\mathbf{y}(t) + \mathbf{F}(t, \mathbf{y}(t)), \quad t > 0, \tag{2.1}$$

$$\mathbf{y}(0) = \mathbf{y}_0, \tag{2.2}$$

which for example arise after spatial discretization of time-dependent nonlinear partial differential equations (PDEs). Therefore, we assume that the vector of unknowns $\mathbf{y}$ is of dimension N and the function $\mathbf{F} : [0, T] \times \mathbb{R}^N \to \mathbb{R}^N$ captures the nonlinearity of the dynamical behavior. We have chosen the form (2.1) in order to stress the difference between a linear and a nonlinear term in the dynamical behavior. We will see that a POD-reduction leads directly to a dimension reduction of the linear term while the evaluation of the nonlinear term still depends on the full dimension N. We will denote the dimension of the unknown vector $\mathbf{y}(t)$ with a capital N in order to remind that this is the *large* dimension which we seek to reduce. In the same way, we will call the (small) dimensions

that are obtained by MOR techniques with small letters ℓ, m in order to point out that $\ell, m \ll N$.

2.1.1 Optimality of the POD basis

The overall aim of POD is the projection of the governing equations (2.1)-(2.2) onto a suitable subspace $\mathcal{U}_\ell$ of dimension $\ell \ll N$ that captures most of the dynamical behavior of the original system. In order to obtain this subspace, the so-called matrix of snapshots defined as

$$Y := [\mathbf{y}(t_1), \mathbf{y}(t_2), ..., \mathbf{y}(t_{n_s})] \in \mathbb{R}^{N \times n_s}, \tag{2.3}$$

plays a key role. The solution vector $\mathbf{y}$ at n_s different time instances form the columns of the matrix Y. The integer n_s is called the *number of snapshots* and typically, $n_s \ll N$.

In this section, we will explain how a Singular Value Decomposition (SVD) of the snapshot matrix (2.3) can be used to obtain an optimal projection space. More specifically, we will refer to a resultfrom [53] that shows that the optimal ℓ-dimensional projection space is given by the span of those ℓ singular vectors of Y that correspond to the ℓ largest singular values. The singular value decomposition of a rectangular matrix Y is given by the following well-known theorem:

Theorem 2.1.1. (Singular value decomposition, [20]) *Let $Y \in \mathbb{R}^{N \times n_s}$ of rank d. Then there exists a decomposition of the form*

$$Y = U \begin{bmatrix} D & 0 \\ 0 & 0 \end{bmatrix} V^T =: U \Sigma V^T, \tag{2.4}$$

with $U \in \mathbb{R}^{N \times N}$, $V \in \mathbb{R}^{n_s \times n_s}$ orthogonal and $D = diag(\sigma_1, ..., \sigma_d) \in \mathbb{R}_+^{d \times d}$. The columns in $U = [\mathbf{u}_1, ..., \mathbf{u}_N]$ are called the (left) singular vectors of Y and for the singular values σ_i it holds: $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_d > 0$.

Proof. Can, for example, be found in [20, Theorem 2.5.2]. □

Note that due to the SVD, the following diagonalization holds

$$YY^T = (U \Sigma V^T)(U \Sigma V^T)^T = U \Sigma^2 U^T, \tag{2.5}$$

and, hence, the columns of U are eigenvectors of the matrix YY^T with corresponding eigenvalues $\lambda_i = \sigma_i^2 > 0,\ i = 1, ..., d$.

We will next consider the optimization problem (2.6) whose solution gives rise to the practical computation of the POD basis. We seek for an orthonormal basis $\varphi_1, ..., \varphi_\ell$ such that the components of the snapshots $\mathbf{y}(t_1), ..., \mathbf{y}(t_{n_s})$ are maximized when expressed in this basis. Therefore, it is desired to find the basis $\varphi_1, ..., \varphi_\ell$ and the following theorem states that the left singular vectors of Y solve the optimization problem (2.6).

Theorem 2.1.2. (POD basis, [53]) *Let $Y \in \mathbb{R}^{N\times n_s}$ be the snapshot matrix* (2.3) *with rank $d \leq \min\{N, n_s\}$. Further, let $Y = U\Sigma V^T$ be the singular value decomposition of Y with orthogonal matrices $U = [\mathbf{u}_1, ..., \mathbf{u}_N]$ and $V = [\mathbf{v}_1, ..., \mathbf{v}_{n_s}]$ as in* (2.4)*. Then, for any $\ell \in \{1, ..., d\}$ the solution to the optimization problem*

$$\max_{\varphi_1,...,\varphi_\ell} \sum_{i=1}^{\ell}\sum_{j=1}^{n_s} |\langle \mathbf{y}_j, \varphi_i\rangle|^2 \quad s.t. \quad \langle\varphi_i, \varphi_j\rangle = \delta_{i,j} \text{ for } 1 \leq i, j \leq \ell \tag{2.6}$$

is given by the left singular vectors $\{\mathbf{u}_i\}_{i=1}^{\ell}$. The vectors $\varphi_1, ..., \varphi_\ell$ are called the POD basis of rank ℓ. Here, $\delta_{i,j}$ denotes the Kronecker delta.

Proof. The proof is given in [53, p. 5-6] □

Note that this result is crucial for the practical usage of the POD method since it presents a simple way how to compute the POD basis from the snapshot matrix. The optimal projection space it then given by $\mathcal{U}_\ell = span\{\mathbf{u}_1, ..., \mathbf{u}_\ell\}$. The choice of the dimension ℓ of the POD basis is important for the quality of the approximation of the POD model. According to [53], there exists no theoretical bound for the approximation error depending on ℓ. Therefore, in practice the choice of ℓ has to be obtained heuristically. The ratio,

$$\varepsilon(\ell) := \frac{\sum_{i=1}^{\ell}\sigma_i^2}{\sum_{i=1}^{N}\sigma_i^2}, \quad 1 \leq \ell \leq N, \tag{2.7}$$

gives a good estimate of the relationship between the energy of the reduced system and the total energy. Note that $\varepsilon(\cdot)$ as a function of ℓ is growing fast when the considered dynamical system is suitable for model order reduction, i.e. the first singular values are large compared to the sum of all singular values. For instance, in the setting of Section 2.3 we present the distribution of the singular values for $\nu = 0.01$ in Figure 2.3. The numerical solution of the full-order model required a spatial discretization of $N = 80$ and a POD dimension of $\ell = 9$ already leads to $\varepsilon(9) = 0.98$ which seems to indicate that most of the dynamical behavior of the full-order system can be captured by a POD reduced model of dimension 9. This can also be seen in Figure 2.4.

2.1.2 The projected reduced-order model

In the previous section, we have shown that the optimal subspace $\mathcal{U}_\ell$ that captures most of the dynamical behavior of the original dynamical system is given by $\mathcal{U}_\ell = span\{\mathbf{u}_1, ..., \mathbf{u}_\ell\}$, where $\{\mathbf{u}_i\}_{i=1}^{\ell}$ form a POD basis of dimension ℓ. In order to construct the POD reduced-order model, we define the matrix $U_\ell := [\mathbf{u}_1, ..., \mathbf{u}_\ell]$. Then, we can construct an approximation of the solution vector $\mathbf{y}$ in $\mathcal{U}_\ell$ as:

$$\mathbf{y}(t) \approx U_\ell\tilde{\mathbf{y}}(t), \quad \tilde{\mathbf{y}}(t) \in \mathbb{R}^\ell.$$

A reduced-order model for (2.1)-(2.2) in terms of $\tilde{\mathbf{y}}(t)$ can be derived by a Galerkin projection of (2.1) onto $\mathcal{U}_\ell$,

$$\begin{aligned}\frac{d}{dt}\tilde{\mathbf{y}}(t) &= U_\ell^T A U_\ell \tilde{\mathbf{y}}(t) + U_\ell^T \mathbf{F}(t, U_\ell \tilde{\mathbf{y}}(t)) \\ &=: \tilde{A}\tilde{\mathbf{y}}(t) + \tilde{\mathbf{N}}(\tilde{\mathbf{y}}, t),\end{aligned} \tag{2.8}$$

with $\tilde{A} := U_\ell^T A U_\ell \in \mathbb{R}^{\ell\times\ell}$ and nonlinearity $\tilde{\mathbf{N}}(\tilde{\mathbf{y}}, t) := U_\ell^T \mathbf{F}(t, U_\ell\tilde{\mathbf{y}}(t))$.

The corresponding initial condition is given by

$$\tilde{\mathbf{y}}(0) = U_\ell^T \mathbf{y}_0.$$

We will refer to (2.8) as the POD-reduced model. Note that the linear part of the dynamical system in (2.8) is represented by the matrix $\tilde{A} \in \mathbb{R}^{\ell\times\ell}$. We see that the linear term is already independent of the full-order dimension N.

2.2 The Discrete Empirical Interpolation Method (DEIM)

In the previous sections, we used the POD-Galerkin approach (2.8) to construct a reduced-order model forthe full order dynamical system (2.1)-(2.2). The system (2.8) is of (small) dimension ℓ and the unknown $\tilde{\mathbf{y}}$ is an ℓ-dimensional approximation of $\mathbf{y}$. However, the evaluation of the nonlinear term $\tilde{\mathbf{N}}(\tilde{\mathbf{y}}, t)$ still has computational complexity that depends on the original problem size N as the following shows:

$$\tilde{\mathbf{N}}(\tilde{\mathbf{y}}, t) = \underbrace{U_\ell^T}_{\ell\times N}\underbrace{\mathbf{F}(U_\ell\tilde{\mathbf{y}}(t))}_{N\times 1}.$$

The DEIM method [12] is an efficient way to overcome this dependence on N and is, therefore, considered as an improvement of the POD algorithm. To simplifynotation, let us denote from now on $\mathbf{f}(t) := \mathbf{F}(U_\ell\tilde{\mathbf{y}}(t))$. We are looking for a low-dimensional approximation:

$$\mathbf{f}(t) \approx W\mathbf{c}(t),$$

where $W := [\mathbf{w}_1, ..., \mathbf{w}_m] \in \mathbb{R}^{N\times m}$ and the coefficient vector $\mathbf{c}(t) \in \mathbb{R}^m$, as in the POD approach. Indeed, in DEIM the projection space $span\{\mathbf{w}_1, ..., \mathbf{w}_m\}$ is obtained by an SVD of the snapshot matrix of the nonlinearity

$$F := [\mathbf{f}(t_1), \mathbf{f}(t_2), ..., \mathbf{f}(t_{n_s})] \in \mathbb{R}^{N\times n_s}. \tag{2.9}$$

Since $\mathbf{f}(t) = W\mathbf{c}(t)$ is an overdetermined linear system of equations in $\mathbf{c}(t)$, we select m distinguished rows from both sides of the system. Therefore, we define the matrix $\mathcal{P} = [\mathbf{e}_{\wp_1}, ..., \mathbf{e}_{\wp_m}] \in \mathbb{R}^{N\times m}$ with $\mathbf{e}_{\wp_i} = [0, ..., 0, 1, 0, ..., 0] \in \mathbb{R}^N$ having its non-zero entry at the $\wp_i$-th component. If $\mathcal{P}^T W$ is nonsingular, the coefficient vector $\mathbf{c}(t)$ can uniquely be determined from:

$$\mathcal{P}^T\mathbf{f}(t) = (\mathcal{P}^T W)\mathbf{c}(t),$$

and we derive an approximation of $\mathbf{f}(t)$ as:

$$\mathbf{f}(t) \approx W\mathbf{c}(t) = W(\mathcal{P}^T W)^{-1}\mathcal{P}^T\mathbf{f}(t) =: \hat{\mathbf{f}}(t). \tag{2.10}$$

Note that the matrix-vector multiplication $\mathcal{P}^T\mathbf{f}(t)$ is never computed in the standard way since this would imply a dependence of the computational complexity on N because $\mathbf{f}(t)$ is an N-dimensional vector. Instead, a left multiplication with $\mathcal{P}^T$ is by construction equivalent to the selection of m entries of the vector $\mathbf{f}(t)$ which is an $\mathcal{O}(m)$ operation.

The nonlinear term in (2.8) can, thus, be computed via

$$\begin{aligned}\tilde{\mathbf{N}}(\tilde{\mathbf{y}}, t) &\approx U_\ell^T W(\mathcal{P}^T W)^{-1}\mathcal{P}^T\mathbf{F}(U_\ell\tilde{\mathbf{y}}(t)) \\ &\overset{(*)}{=} \underbrace{U_\ell^T W(\mathcal{P}^T W)^{-1}}_{\ell\times m}\underbrace{\mathbf{F}(\mathcal{P}^T U_\ell\tilde{\mathbf{y}}(t))}_{m\times 1},\end{aligned}$$

where we have assumed that the function $\mathbf{F}(\cdot)$ only acts pointwise on its input vector. Hence, it is possible in step $(*)$ to first select the m components of the input vector and then evaluate the function $\mathbf{F}$. The resulting approximation of $\tilde{\mathbf{N}}(\tilde{\mathbf{y}}, t)$ does not depend on the dimension N of the full order system and, moreover, we note that the matrix $U_\ell^T W(\mathcal{P}^T W)^{-1}$ does not depend on time and can, thus, be pre-computed. Note that DEIM can also be applied to general nonlinear functions that are not pointwise. We refer the reader to [12] for more details.

It remains to describe how the m entries of $\mathbf{f}(t)$ are selected in DEIM such that the approximation $\hat{\mathbf{f}}(t)$ in formula (2.10) is optimal. An algorithm that successively constructs the matrix $\mathcal{P}$ for a given input basis $\mathbf{w}_1, ..., \mathbf{w}_m$ is proposed in [12, Section 3.1] and presented here as Algorithm 2.1. In the initialization step, the first index $\wp_1 \in \{1, ..., N\}$ is selected corresponding to the largest components in magnitude of the first input basis vector $\mathbf{w}_1$. The loop over $i = 2$ to $i = m$ selects the indices corresponding to the largest component of the residual $\mathbf{r}$ between the input basis $\mathbf{w}_i$ and its approximation within the subspace $span\{\mathbf{w}_1, ..., \mathbf{w}_{i-1}\}$.

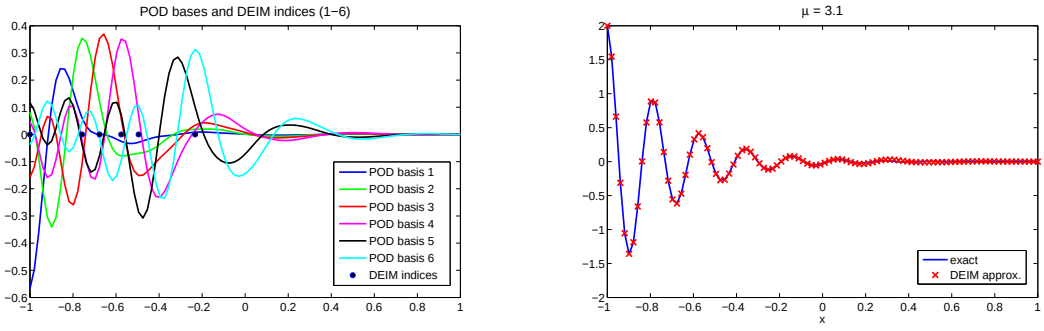
Algorithm 2.1 The DEIM algorithm, [12]

1: **INPUT:** $\{\mathbf{w}_i\}_{i=1}^m \subset \mathbb{R}^N$ linear independent
2: **OUTPUT:** $\vec{\wp} = [\wp_1, ..., \wp_m]^T \in \mathbb{R}^m$, $\mathcal{P} \in \mathbb{R}^{N\times m}$
3: $[|\rho|, \wp_1] = \max\{|\mathbf{w}_1|\}$
4: $W = [\mathbf{w}_1], \mathcal{P} = [\mathbf{e}_{\wp_1}], \vec{\wp} = [\wp_1]$
5: **for** $i = 2$ to m **do**
6: Solve $(\mathcal{P}^T W)\mathbf{c} = \mathcal{P}^T\mathbf{w}_i$ for $\mathbf{c}$
7: $\mathbf{r} = \mathbf{w}_i - W\mathbf{c}$
8: $[|\rho|, \wp_i] = \max\{|\mathbf{r}|\}$
9: $W \leftarrow [W\ \mathbf{w}_i], \mathcal{P} \leftarrow [\mathcal{P}\ \mathbf{e}_{\wp_i}], \vec{\wp} \leftarrow \begin{bmatrix}\vec{\wp}\\ \wp_i\end{bmatrix}$
10: **end for**

An illustrative example taken from [12] is given by the nonlinear parameterized function:

$$s(x,\mu) = (1-x)\cos(3\pi\mu(x+1))e^{-(1+x)\mu}, \quad x \in [-1,1], \mu \in [1,\pi], \tag{2.11}$$

to which the DEIM algorithm was applied based on 100 equidistantly spaced points $x_i \in [-1,1]$ and 51 snapshots for $\mu_j \in [1,\pi]$. Figure 2.1a shows that the first DEIM indices obtained by Algorithm 2.1 are chosen in a region where most of the dynamics of s occurs. In Figure 2.1b, it can be seen that for $\mu = 3.1$, the approximation $\hat{s}$ based on DEIM with dimension $m = 10$ gives a good result compared to the 100-dimensional original model.



(a) First six POD bases with DEIM points.

(b) DEIM approximation of dimension 10.

Figure 2.1: The location of the DEIM indices for example (2.11).

Theorem 2.2.1. (Error bound of DEIM approximation, [12]) *Let* $\mathbf{f} \in \mathbb{R}^N$ *be an arbitrary vector. Let* $\{\mathbf{w}_i\}_{i=1}^m$ *be the first* m *left singular vectors of the snapshot matrix* (2.9) *(POD basis). From* (2.10)*, the DEIM approximation of order* m *for* $\mathbf{f}$ *in the space span*$\{\mathbf{u}_1, ..., \mathbf{u}_m\}$ *is*

$$\hat{\mathbf{f}} = W(\mathcal{P}^T W)^{-1}\mathcal{P}^T\mathbf{f},$$

where $W := [\mathbf{w}_1, ..., \mathbf{w}_m] \in \mathbb{R}^{N\times m}$ *and* $\mathcal{P} := [\mathbf{e}_{\wp_1}, ..., \mathbf{e}_{\wp_m}] \in \mathbb{R}^{N\times m}$*, with* $\{\wp_1, ..., \wp_m\}$ *being obtained from Algorithm 2.1 with input basis* $\{\mathbf{w}_1, ..., \mathbf{w}_m\}$*. An error bound for* $\hat{\mathbf{f}}$ *is then given by*

$$\|\mathbf{f} - \hat{\mathbf{f}}\|_2 \leq \mathcal{C} \cdot \mathcal{E}_*(\mathbf{f}),$$

where

$$\mathcal{C} = \|(\mathcal{P}^T W)^{-1}\|_2, \quad \mathcal{E}_*(\mathbf{f}) = \|(I - WW^T)\mathbf{f}\|_2.$$

The constant $\mathcal{C}$ *is bounded by*

$$\mathcal{C} \leq \frac{(1+\sqrt{2N})^{m-1}}{|\mathbf{e}_{\wp_1}^T\mathbf{w}_1|} = (1+\sqrt{2N})^{m-1}\|\mathbf{w}_1\|_\infty^{-1}.$$

Proof. The proof is given in [12, p. 2747-2749]. □

Not only does Theorem 2.2.1 give an error bound for the approximation obtained by the DEIM algorithm, but the proof also established that the choice of the DEIM indices in Algorithm 2.1 in fact minimizes the growth of $\|(\mathcal{P}^T W)^{-1}\|_2$. Therefore, the algorithm is optimal in the sense that the approximation error is minimized.

2.3 Application: POD-DEIM for the unsteady Burgers' equation

The following example is taken from [29] where the authors applied POD to the one-dimensional Burgers' equation together with homogeneous Dirichlet boundary conditions and a step function $y_0(x)$ as initial condition has been considered,

$$\begin{aligned} y_t + \left(\frac{1}{2}y^2 - \nu y_x\right)_x &= f, \quad (x,t) \in (0,L)\times(0,T), \\ y(t,0) = y(t,L) &= 0, \quad t \in (0,T), \\ y(0,x) &= y_0(x), \quad x \in (0,L). \end{aligned} \tag{2.12}$$

Burgers' equation is a fundamental partial differential equation (PDE) from fluid dynamics and is, for instance, used in gas dynamics. The formulation (2.12) is known as the conservative form of Burgers' equation. Note that the numerical properties of (2.12) highly depend on the viscosity parameter ν. For instance, when ν is small, the nonlinear term influences the numerical solution more and the PDE is called *stiff*. This requires a small time step for the time integration.

For the numerical solution of the full-order system (2.12), we used a finite element discretization in space using linear basis functions as described in Appendix A.1. This approach leads to the following system of ODEs:

$$M\dot{\mathbf{y}}(t) = -\frac{1}{2}B\mathbf{y}^2(t) - \nu C\mathbf{y}(t) + \mathbf{f} \tag{2.13}$$

where M is the mass matrix, C is the stiffness matrix, and B represents the convective term. For simplicity, we assume that the source term is zero, i.e. $f \equiv 0$. At this point, we also need to specify the initial condition $y_0(x)$:

$$y_0(x) = \begin{cases} 1, & \text{if } 0 \le x \le \frac{L}{2} \\ 0, & \text{if } \frac{L}{2} < x \le L \end{cases} \quad \Rightarrow \quad \mathbf{y}(0) = [1,...,1,0,...,0]^T \in \mathbb{R}^N. \tag{2.14}$$

The system (2.13) with initial condition (2.14) can be integrated in time using the implicit Euler method (see Appendix A.2). In Figure 2.2, we present the numerical solution of (2.13)-(2.14) for different values of ν and $L = T = 1$.

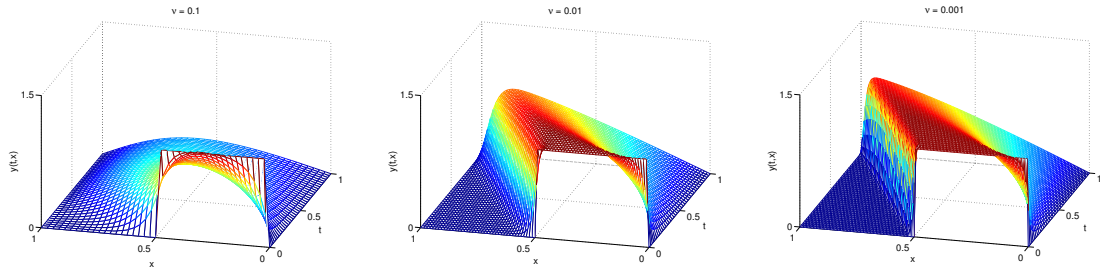


Figure 2.2: Numerical solution of the full-order Burgers' equation for different viscosity parameters $\nu = \{0.1, 0.01, 0.001\}$ and initial condition (2.14).

According to [29] the POD basis $\{\varphi_i\}_{i=1}^{\ell}$ can be obtained by solving the eigenvalue problem:

$$YY^T M\varphi = \sigma^2\varphi, \tag{2.15}$$

where Y is the matrix of snapshots as previously defined in (2.3), i.e. the columns of Y are the solution of Burgers' equation at different time instances.

Since the mass matrix M is symmetric and positive definite, there exists a Cholesky decomposition of the form $M = R^T R$. Multiplication of (2.15) from the left with R then yields,

$$RYY^T R^T R\varphi = \sigma^2 R\varphi,$$

which is equal to

$$\bar{Y}\bar{Y}^T\bar{\varphi} = \sigma^2\bar{\varphi}, \tag{2.16}$$

for $\bar{Y} := RY$ and $\bar{\varphi} := R\varphi$. Because of the relation (2.5) between the SVD and eigenvalues, we can solve (2.16) and, thus, also (2.15) by the computation of an SVD of the matrix $\bar{Y} = RY$,

$$RY = U\Sigma V^T. \tag{2.17}$$

After choosing a suitable POD dimension $\ell \ll N$, let us define the matrix of left singular vectors, $U_\ell := [u_1, ..., u_\ell]$, where u_i are the columns of the matrix U in (2.17). Then, the POD basis is given by:

$$\Phi_\ell := R^{-1}U_\ell, \tag{2.18}$$

and we denote the columns of Φ_ℓ by φ_i such that $\Phi_\ell = [\varphi_1, ..., \varphi_\ell]$. The following ansatz leads to an approximation of the original full-order solution,

$$\mathbf{y}(t) \approx \Phi_\ell\tilde{\mathbf{y}}(t) \quad \text{with } \tilde{\mathbf{y}}(t) \in \mathbb{R}^\ell. \tag{2.19}$$

The Galerkin projection of (2.13) onto $span\{\varphi_1, ..., \varphi_\ell\}$ is given by

$$\dot{\tilde{\mathbf{y}}}(t) = -\frac{1}{2}B_\ell(\Phi_\ell\tilde{\mathbf{y}}(t))^2 - \nu C_\ell\tilde{\mathbf{y}}(t), \tag{2.20}$$

with the matrices

$$B_\ell := \Phi_\ell^T B \in \mathbb{R}^{\ell \times N}, \tag{2.21}$$

$$C_\ell := \Phi_\ell^T C \Phi_\ell \in \mathbb{R}^{\ell \times \ell}, \tag{2.22}$$

and with the new mass matrix equal to the $\ell \times \ell$ identity since:

$$M_\ell := \Phi_\ell^T M \Phi_\ell = U_\ell^T U_\ell = I_\ell \in \mathbb{R}^{\ell \times \ell}. \tag{2.23}$$

We will refer to (2.20) as the POD-reduced system. As shown in (2.23), due to the M-orthogonality of the projection matrix Φ_ℓ, the mass matrix in the POD-reduced system vanishes, and this leads to a simpler numerical treatment of (2.20) compared to the full-order system (2.13). Note, that $\Phi_\ell \tilde{\mathbf{y}}(t) \in \mathbb{R}^N$ is still of large dimension and, therefore, no dimension reduction for the nonlinearity has been obtained so far. Also, the number of columns of the reduced matrix B_ℓ is still of large dimension N.

The projected initial condition for the original system is:

$$\Phi_\ell \tilde{\mathbf{y}}(0) = \mathbf{y}(0),$$
$$R^{-1} U_\ell \tilde{\mathbf{y}}(0) = \mathbf{y}(0),$$

and, therefore, the initial condition for (2.20) can be obtained by pre-multiplication of the full-size initial condition with the matrix product $\Phi_\ell^T M$ as shown below:

$$\tilde{\mathbf{y}}(0) = U_\ell^T R \mathbf{y}(0) = \Phi_\ell^T R^T R \mathbf{y}(0) = \Phi_\ell^T M \mathbf{y}(0). \tag{2.24}$$

As it has already been pointed out, the nonlinear term in (2.20) is still of large dimension N. We will now apply the DEIM method in order to reduce the computational complexity of evaluating the nonlinearity. Consider the snapshot matrix of the nonlinearity, $F := [\mathbf{y}(t_1)^2, ..., \mathbf{y}(t_{n_s})^2]$ and the corresponding Singular Value Decomposition:

$$F = U_f \Sigma_f V_f^T.$$

Again, by choosing the DEIM-dimension m, we are able to define the matrix of left singular vectors, $U_m := [u_1^f, ..., u_m^f]^T$, which is used as an input basis for Algorithm 2.1. From Algorithm 2.1, we obtain the projection matrix $\mathcal{P}$ and the nonlinear term becomes:

$$\Phi_\ell^T B(\Phi_\ell \tilde{\mathbf{y}})^2 = \Phi_\ell^T B U_m (\mathcal{P}^T U_m)^{-1} \mathcal{P}^T (\Phi_\ell \tilde{\mathbf{y}})^2 \overset{(*)}{=} \underbrace{\Phi_\ell^T B U_m (\mathcal{P}^T U_m)^{-1}}_{\ell \times m} (\underbrace{\mathcal{P}^T \Phi_\ell}_{m \times \ell} \tilde{\mathbf{y}})^2,$$

where the step $(*)$ is allowed since squaring is a componentwise operation.

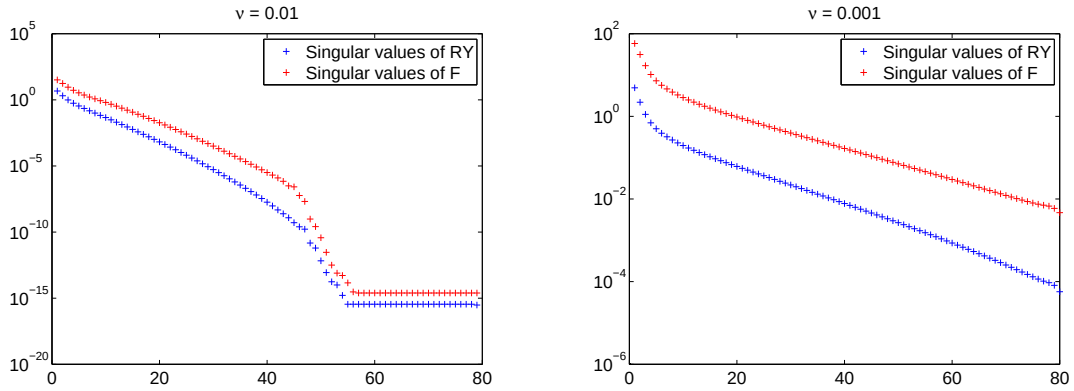


Figure 2.3: Distribution of the singular values for $\nu = 0.01$ (left) and $\nu = 0.001$ (right).

The fully reduced POD-DEIM system is then given by

$$\dot{\tilde{\mathbf{y}}}(t) = -\frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}}(t))^2 - \nu\tilde{C}\tilde{\mathbf{y}}(t) + \tilde{\mathbf{f}}, \tag{2.25}$$

where

$$\tilde{B} := \Phi_\ell^T B U_f (\mathcal{P}^T U_f)^{-1} \in \mathbb{R}^{\ell\times m}, \tag{2.26}$$

$$\tilde{C} := \Phi_\ell^T C \Phi_\ell \in \mathbb{R}^{\ell\times\ell}, \tag{2.27}$$

$$\tilde{F} := \mathcal{P}^T \Phi_\ell \in \mathbb{R}^{m\times\ell}. \tag{2.28}$$

Note that both matrices $\tilde{B}$ and $\tilde{F}$ can be pre-computed such that the system (2.25) is indeed of dimension $\ell \times \ell$ and no dependence on the original size N exists anymore. As for the full-order system (2.13), the POD-DEIM reduced system (2.25) was solved by means of the implicit Euler method, see Appendix A.2. The choice of the dimensions of the reduced model ℓ and m is crucial in order to obtain an accurate result of the reduced system that reproduces the dynamics of the original system. A good estimate for the choice of ℓ, m is given by the ration of the truncated sum of the singular values and the sum of all singular values given in (2.7). Therefore, we consider the distribution of the singular values of the respective snapshot matrices in Figure 2.3. We see that especially for $\nu = 0.01$, the first singular values decay tremendously which gives rise to a good approximation of the reduced system when only a small number for ℓ, m is chosen. We will present in Figure 2.4 the behavior of the numerical solution of the POD-DEIM reduced system when the DEIM-dimension m is fixed and ℓ is increased.

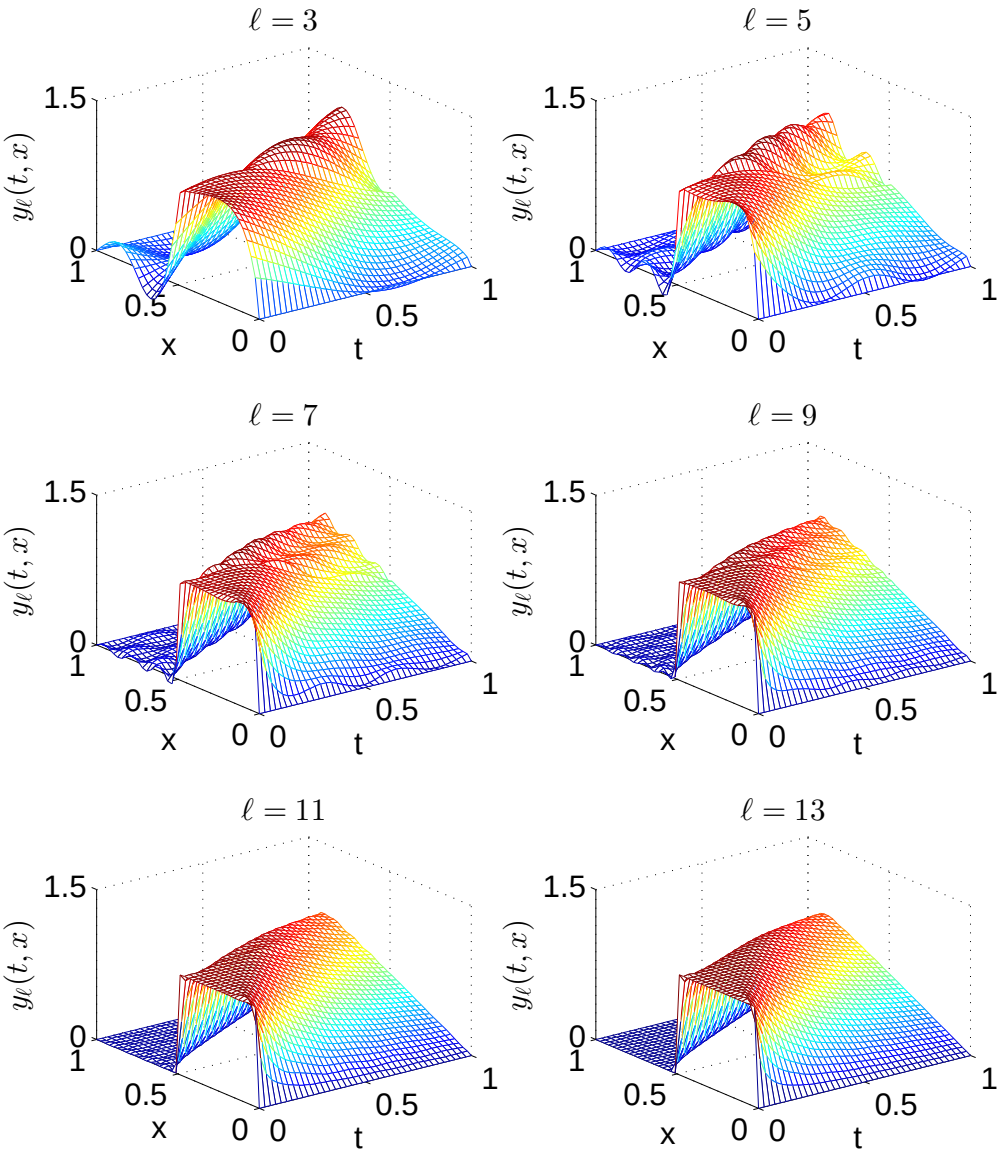


Figure 2.4: POD-DEIM approximations for different dimensions ℓ and $m = 15$, $\nu = 0.01$.

Figure 2.4 shows the convergence of the POD-DEIM reduced model to the full-order solution. In each computation, a fixed DEIM dimension of $m = 15$ was used to approximate the nonlinear term in Burgers' equation. If the dimension of the POD basis ℓ, i.e. the size of the projected model is increased, we see from Figure 2.4 that the numerical solution of the reduced system converges towards a smooth solution that corresponds to the solution of the original model.

2.3.1 Approximation error

We now focus on the accuracy of the approximated system's response computed with reduced models. Let $\mathbf{y}$ be the response of the full-order system, $\mathbf{y}^{(\ell)} = \Phi_\ell \tilde{\mathbf{y}}$ be the response of the POD-reduced model (2.20) obtained with a POD basis of dimension ℓ, and let $\mathbf{y}^{(\ell,m)} = \Phi_\ell \tilde{\mathbf{y}}$ be the response of a reduced model obtained with POD-DEIM (2.25), with a POD basis is of dimension ℓ and a DEIM basis of dimension m. Then, we define the $L_2(\Omega)$ relative error in an approximate response $\bar{\mathbf{y}}$ with respect to the full-order-system's response $\mathbf{y}$ as:

$$\bar{e} := \frac{\|\mathbf{y} - \bar{\mathbf{y}}\|_{L_2(\Omega)}}{\|\mathbf{y}\|_{L_2(\Omega)}} \approx \frac{\sqrt{h \cdot \delta t \cdot \sum_{i,j} [y_i(t_j) - \bar{y}_i(t_j)]^2}}{\sqrt{h \cdot \delta t \cdot \sum_{i,j} y_i^2(t_j)}}, \tag{2.29}$$

where $\Omega = [0, L] \times [0, T]$, and h and δt are the step size in space and time, respectively. In our test cases, we have chosen $T = L = 1$ and, hence, $\Omega = [0, 1] \times [0, 1]$. The approximate response $\bar{\mathbf{y}}$ in (2.29) is $\bar{\mathbf{y}} = \mathbf{y}^{(\ell)}$ for POD and $\bar{\mathbf{y}} = \mathbf{y}^{(\ell,m)}$ for POD-DEIM.

In Figure 2.5, show the behavior of the relative error when either the DEIM-dimension is fixed (left) or the POD dimension is fixed (right). From the numerical results that have been obtained for a viscosity parameter $\nu = 0.01$ and a full-order dimension of $N = 80$, we observe that the error of the POD and the POD-DEIM reduced model are of the same magnitude until the POD dimension exceeds the DEIM-dimension which has been fixed in the left plot to $m = 15$. When ℓ is larger than m in the left plot, we observe that only the POD-reduced model is still improved while the error of the POD-DEIM reduced model seems to be dominated by the error caused by the DEIM approximation of size $m = 15$.

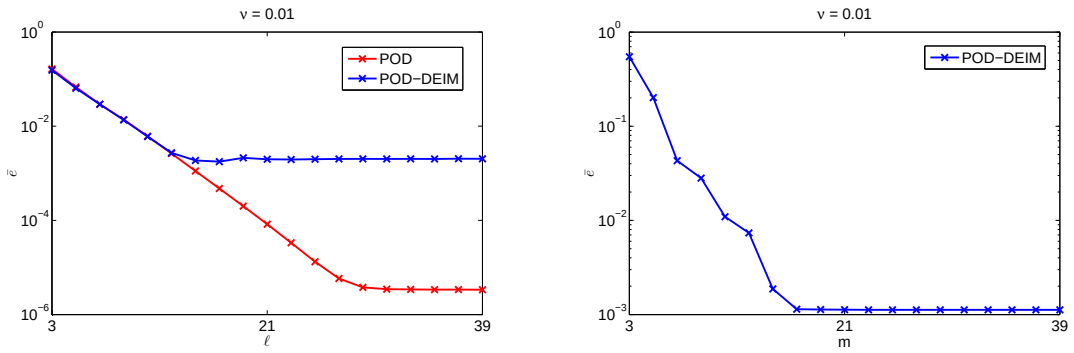


Figure 2.5: Relative error of the POD and POD-DEIM approximation for a fixed number of DEIM basis $m = 15$ (left) and a fixed number of POD basis $\ell = 15$ (right).

The right plot in Figure 2.5 shows the same behavior, i.e. an increase of the DEIM-dimension does not lead to a better approximation once it is larger than a given POD basis dimension. In Figure 2.6, we observe this behavior for all combinations of (ℓ, m) in a range of `3:2:39`. These observations seem to indicate that for optimal dimension reduction using the POD-DEIM approach, we need to choose ℓ and m in such a way that they are almost of the same size.

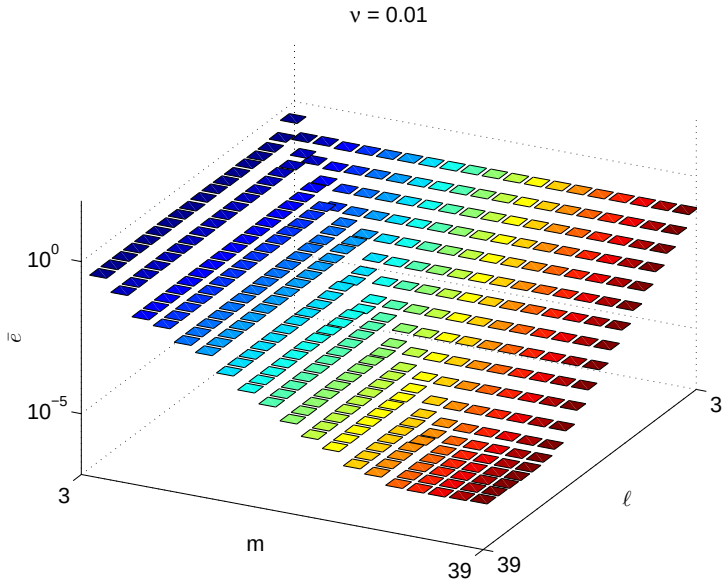


Figure 2.6: Relative error of the POD-DEIM reduced model as a function of the reduced-model dimensions ℓ and m.

2.3.2 Use of POD-DEIM for parameter studies

In [11], the authors propose to use POD-DEIM also for parameter studies of a given dynamical system. In the case of Burgers' equation, this means that we simulate the full-order system twice for two different values of the viscosity parameters $\nu_{left} = 0.001$ and $\nu_{right} = 0.1$, using $N = 400$ grid points in space which is necessary when dealing with very small values for ν. The numerical solutions of the full-order system for both viscosity parameters were presented in Figure 2.2 where we can observe that the value of the viscosity parameter plays a crucial role in the propagation of the initial impulse in time. For large ν, the solution becomes much more diffusive and, therefore, the initial condition vanishes very fast. For small ν, the initial condition is propagated in time almost without losing height. The benefit of POD-DEIM for parameter studies is that a reduced model can be obtained for any parameter ν that lies within the interval $[\nu_{left}, \nu_{right}]$ only from data of the full-order model corresponding to ν_{left} and ν_{right}. Suppose we are interested in an approximation of the numerical solution for 100 different viscosity parameter in $[\nu_{left}, \nu_{right}]$, we only need to solve the full-order system twice and are able to obtain a

reduced model from these data.

The reduction is based on a singular value decomposition of the combined matrices

$$Y = [Y_\ell | Y_r], \quad F = [F_\ell | F_r],$$

where Y_ℓ and F_ℓ are snapshot matrices of the solution and the nonlinearity corresponding to ν_{left} and Y_r and F_r correspond to ν_{right}, respectively. The POD basis is then derived using the snapshot matrix Y that contains information of the dynamical behavior of Burgers' equation with viscosity parameter ν_{left} and ν_{right}. Similarly, the DEIM projection matrix $\mathcal{P}$ is obtained from the matrix F.

In Figure 2.7, the POD-DEIM approximation for the viscosity parameter $\nu = 0.01$ using the dimensions $\ell = m = 45$ is presented as well as the full-order model of dimension $N = 400$ for the same viscosity parameter. The plot shows the good quality of the reduced system. At this point, we want to stress that in order to obtain the POD-DEIM model in Figure 2.7, the simulation of the full-order system with parameter $\nu = 0.01$ was not taken into account. Therefore, the derivation of a reduced model for any parameter $\nu \in [\nu_{left}, \nu_{right}]$ only requires the computational work of the numerical solution of the reduced system once the matrices Y and F are obtained.

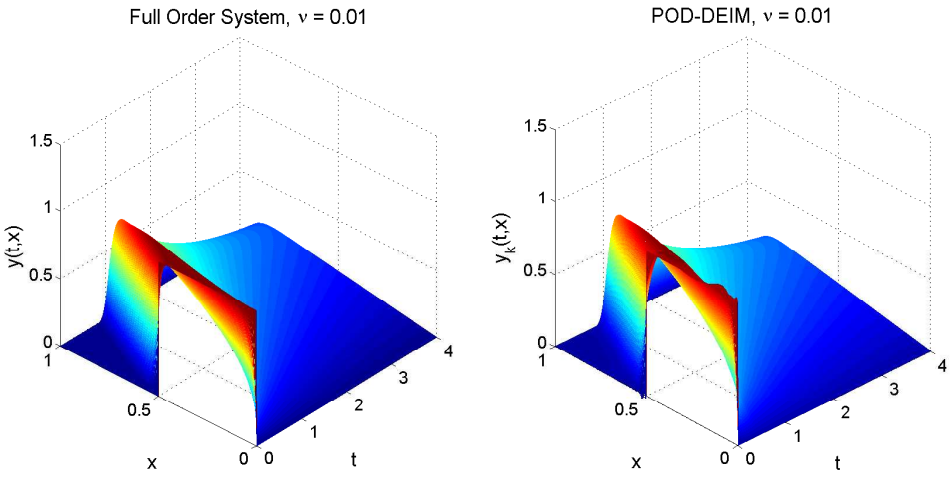


Figure 2.7: Response of the full-order system and of the POD-DEIM reduced model for $\nu = 0.01$.

2.3.3 Computational speedup for POD and POD-DEIM

In the previous chapters, the main focus lies on the size of the reduced system and the quality of the approximation compared to the full-order system. In this section, we are interested in the speedup of the computation when POD or POD-DEIM is applied. Therefore, we consider the numerical solution of Burgers' equation as described in Appendix A for different viscosity parameters ν. From the numerical analysis of the full model, it is well-known that for different sizes of ν a different size of the spatial discretization is required in order to obtain a stable numerical solution. In 2.1, we have chosen the smallest value of N for which the full-order solution is stable.

The following quantities have been measured during the numerical simulation:

- t_{setup} - The time which is required to pre-compute the matrices (2.21), (2.22) in the case of pure POD and (2.26)-(2.28) in the case of POD-DEIM. Note that this only has to be performed once.
- t_{PDEsol} - The time for the numerical solution of Burgers' equation. This has been done via the implicit Euler method and a Newton iteration for the time integration and a finite elements approach for the spatial discretization.
- $\bar{e}[10^{-4}]$ - The relative error in $L_2([0,1]\times[0,1])$ as defined in (2.29).
- $S_P^{(1)}$ - The overall speedup which is defined as the ratio of the computational time for the full-order model and the respective reduced model.
- $S_P^{(2)}$ - The speedup when only the numerical solution of Burgers' equation is taken into account. Here, we neglect the time that is required for the pre-computation in t_{setup}.

In order to obtain comparable results, we decided to choose the reduced dimensions ℓ and m such that the relative error $\bar{e}$ in the response of each of the reduced models has the same order of magnitude, $\bar{e} \in \mathcal{O}(10^{-4})$. Therefore, we indicate in the first row of Table 2.1 the important dimension from the respective model. In case of the full-order model, we present the number N of ansatz function of the FEM discretization. For the pure POD reduced model we give the size of the POD basis, ℓ, and for the the POD-DEIM reduced model we present the dimensions as a tuple (ℓ, m).

	$\nu = 0.01$			$\nu = 0.001$			$\nu = 0.0001$		
	Full	POD	DEIM	Full	POD	DEIM	Full	POD	DEIM
$N/\ell/m$	80	11	$(11,13)$	200	35	$(35,40)$	800	40	$(40,55)$
$t_{setup}[s]$	-	0.003	0.011	-	0.009	0.021	-	0.0729	0.182
$t_{PDEsol}[s]$	0.068	0.047	0.040	0.232	0.12	0.055	4.416	0.276	0.085
$\bar{e}[10^{-4}]$	-	6.13	6.46	-	6.12	6.85	-	7.43	7.66
$S_P^{(1)}$	-	1.35	1.34	-	1.74	3.03	-	12.54	16.52
$S_P^{(2)}$	-	1.44	1.71	-	1.88	4.21	-	15.85	51.85

Table 2.1: Comparison between POD and POD-DEIM for different values of ν.

We first note that for all different sizes of the viscosity parameter $\nu \in \{0.01, 0.001, 0.0001\}$ it is in general possible to derive a POD and POD-DEIM reduced model of a tremendously smaller dimension and a high accuracy of $\bar{e} \in \mathcal{O}(10^{-4})$. The numerical tests of Table 2.1 are presented in order to illustrate two features of the POD-DEIM approach. Firstly, we see that the time which is required for the numerical solution of the POD-DEIM reduced model does not increase when the size of the original problem, N, increases. On the other

hand, we see that the solution of the pure POD model still depends on the original problem size and increases with N. This behavior is illustrated in Figure 2.8 where we see that the increase in N only affects the computation time of the POD model. Its independence of the original (large) dimension is the reason for the large speedup obtained when using the POD-DEIM reduced model. We see that for $N = 3,000$, the POD-DEIM model is almost five times faster than the pure POD model. Secondly, we observe that in order to apply DEIM, it is more costly to apply the required pre-computations of the matrices (2.26)-(2.28). This is mostly because two singular value decompositions are required in order to obtain the projection basis and the input basis for the DEIM Algorithm 2.1. This is also the reason why the overall speedup $S_P^{(1)}$ of POD and POD-DEIM for the presented test configurations is comparable even though DEIM leads to a higher speedup. The most important observation from the results in Table 2.1 is, however, that for the case $\nu = 0.0001$ and $N = 800$, we are able to show a speedup of the POD-DEIM method of more than 50 compared to the full model while POD only leads to a speedup of $\sim$16. This speedup computation only takes into account the time that is required to actually solve the reduced system numerically. Therefore, we conclude that DEIM leads to a tremendous reduction in computational cost as long as we are able to pre-compute the matrices (2.26)-(2.28) only once and then solve the reduced system many times. This conclusion gives rise to the application of POD-DEIM within an optimization iteration as described in the Chapter 3.

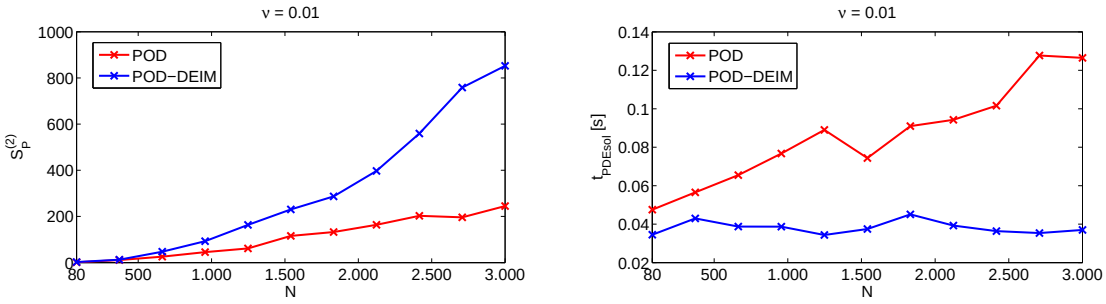


Figure 2.8: Dependence of the reduced models on the full-order dimension N.

Chapter 3

Optimal control of partial differential equations

We consider the following optimization problem,

$$\min_u \mathcal{J}(y(u), u), \tag{3.1}$$

where y is the solution to a nonlinear, possibly time-dependent partial differential equation,

$$c(y, u) = 0, \tag{3.2}$$

and $\mathcal{J}$ is called the *cost function* or the *objective function*. Since in order to evaluate (3.1) as a function of the control u we first need to solve the constraining PDE (3.2), the optimization problem (3.1)-(3.2) is referred to as an *implicitly-constrained optimization problem* in [22]. Standard references for the numerical solution of optimization problems are in [16, 27, 36]. In general, one distinguishes between gradient-based and Newton-type methods which use information of the first and second derivative of the cost function, respectively.

An alternative way to look at (3.1)-(3.2) is to consider the following constrained optimization problem,

$$\begin{aligned} &\min_u \ \mathcal{J}(y, u), \\ &\text{subject to} \quad c(y, u) = 0, \end{aligned} \tag{3.3}$$

where y is called the *state* and u is considered to be the *control* or the *input* of the problem (3.3). Again, the scalar function $\mathcal{J}$ is usually called the *cost function* and the *constraint* c is given by a nonlinear partial differential equation. Note, that $y \in \mathbb{R}^{n_y}$ and $u \in \mathbb{R}^{n_u}$ are typically high-dimensional and, therefore, the (given) functions $\mathcal{J}$ and c map as follows,

$$\mathcal{J} : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \to \mathbb{R}, \quad c : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_y}.$$

As it is stated in (3.3), we seek for an optimal control $u^* = \operatorname{argmin}_u \mathcal{J}(y, u)$ that minimizes the cost function $\mathcal{J}$ and at the same time fulfills the PDE c. In more detail, the control u will appear on the right-hand side of the constraining partial differential equation c such that the solution y will depend on u. We will therefore sometimes write $y(u)$ in order to indicate that the solution to the PDE is only unique after u is specified. Reference for optimal control of partial differential equations are [9, 32, 50].

3.1 Newton-type methods using adjoint techniques for derivative computation

For an easier notation, let us introduce the objective function as a function of the control only,

$$\hat{\mathcal{J}}(u) := \mathcal{J}(y(u), u). \tag{3.4}$$

The unconstrained optimization problem (3.1) then becomes,

$$\min_u \hat{\mathcal{J}}(u), \tag{3.5}$$

where $\hat{\mathcal{J}} : \mathbb{R}^{n_u} \to \mathbb{R}$. Many textbooks on optimization deal with the numerical solution of (3.5), cf. [16, 27, 36, 39].

A standard approach to solve the optimization problem (3.5) is by solving iteratively the so-called Newton equation

$$\nabla^2 \hat{\mathcal{J}}(u_k) s_k = -\nabla \hat{\mathcal{J}}(u_k), \tag{3.6}$$

where an initial control u_0 has to be chosen and in an outermost loop on k one tries to reach convergence. In (3.6), s_k is the *search direction* and $\nabla \hat{\mathcal{J}}$ and $\nabla^2 \hat{\mathcal{J}}$ are the gradient and the Hessian of the objective function, respectively. Note that (3.6) is a system of linear equations in the unknown search direction s_k. The new control is computed as:

$$u_{k+1} = u_k + \alpha_k^* \cdot s_k, \tag{3.7}$$

where the optimal step size $\alpha_k^* \in \mathbb{R}_+$ has to be determined such that:

$$\alpha_k^* = \operatorname*{argmin}_{\alpha_k \in \mathbb{R}_+} \mathcal{J}(y(u_k + \alpha_k \cdot s_k), u_k + \alpha_k \cdot s_k). \tag{3.8}$$

In practice, problem (3.8) is solved approximately using a selected criterion. The optimization method in Algorithm 3.1 is a truncated-CG Newton method from [23]. In Algorithm 3.1, an Armijo line search strategy (see Appendix B.2) is used for approximately solving (3.8) and a truncated Conjugate Gradient (CG) method is used for solving the Newton equation (3.6), see Appendix B.1. The gradient and Hessian of the objective function are computed efficiently by means of the adjoint technique as described in Section 3.1.1 and 3.1.2, respectively.

Algorithm 3.1 Truncated Newton-CG method with Armijo line search, [23]

1: Set initial control $u_0 = 0$, stopping tolerances $\varepsilon_{\mathcal{J}}, \varepsilon_{\nabla} > 0$, `max_newtoncg` $\in \mathbb{N}$
2: **for** $k = 0$ to `max_newtoncg` **do**
3: Solve $c(y_k, u_k) = 0$ for y_k
4: Save $\mathcal{J}_{old} := \mathcal{J}(y_k, u_k)$
5: Compute $\nabla\hat{\mathcal{J}}(u_k)$ via algorithm 3.2
6: **if** $\|\nabla\hat{\mathcal{J}}(u_k)\| < \varepsilon_{\nabla}$ **then**
7: **return**
8: **end if**
9: Solve the Newton equation $\nabla^2\hat{\mathcal{J}}(u_k)s_k = -\nabla\hat{\mathcal{J}}(u_k)$ via the truncated conjugate gradient method, see Appendix B.1, and using Algorithm 3.3 to compute matrix-vector products with $\nabla^2\hat{\mathcal{J}}(u_k)$
10: Obtain $\alpha_k^* \approx \operatorname{argmin}_{\alpha_k \in \mathbb{R}_+} \mathcal{J}(y(u_k + \alpha_k s_k), u_k + \alpha s_k)$ via the Armijo line search algorithm, see Appendix B.2
11: Update $u_{k+1} = u_k + \alpha_k^* s_k$
12: **if** $|\mathcal{J}_{old} - \mathcal{J}(y(u_{k+1}), u_{k+1})| < \varepsilon_{\mathcal{J}}$ **then**
13: **return**
14: **end if**
15: **end for**

Note that in Algorithm 3.1, we have used two so-called numerical stopping criteria. The tolerance ε_{∇} in line 6 determines when the gradient of the objective function is almost zero which is a necessary condition to find a minimum. In line 12, the tolerance $\varepsilon_{\mathcal{J}}$ checks if the change of the value of the objective function during the optimization iteration is sufficiently small.

3.1.1 Using adjoint equations for gradient computation

In order to derive an efficient numerical method to compute the gradient of $\hat{\mathcal{J}}$ as proposed in [23], let us introduce the Lagrangian function $\mathcal{L}$ which converts the constrained optimization problem (3.3) into an unconstrained optimization problem. The Lagrangian function is defined via

$$\begin{aligned} &\mathcal{L} : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R} \\ &\mathcal{L}(y, u, \lambda) = \mathcal{J}(y, u) + \lambda^T c(y, u), \end{aligned} \tag{3.9}$$

where λ is a new variable called the *Lagrange multiplier*. It is well-known that for all optimal points (y^*, u^*) to the original constrained problem (3.3), there exists a λ^* such that (y^*, u^*, λ^*) is a stationary point of the Lagrangian function (3.9), cf. [46]. Stationary points of (3.9) fulfill the first order optimality conditions (zero-gradient condition),

$$\nabla_y \mathcal{L}(y, u, \lambda) = 0, \tag{3.10}$$

$$\nabla_u \mathcal{L}(y, u, \lambda) = 0, \tag{3.11}$$

$$\nabla_\lambda \mathcal{L}(y, u, \lambda) = 0, \tag{3.12}$$

where the subscripts are used to denote partial derivatives with respect to the indicated variable. Note, that from (3.12) the original constraint $c(y,u)=0$ follows directly. Furthermore, we will use (3.10) in order to derive the so-called *adjoint equation* which determines the auxiliary variable λ. In the following, we will describe an efficient way to compute $\nabla\hat{\mathcal{J}}(u)$ which appears on the right-hand side of (3.9) in Algorithm 3.1 based on the conditions (3.10)-(3.12).

If we apply the gradient with respect to y to the Lagrangian function (3.9), i.e. use relation (3.10), we derive:

$$c_y(y(u),u)^T\lambda = -\nabla_y\mathcal{J}(y(u),u), \tag{3.13}$$

which is called the *adjoint equations* because from (3.13) we are able to determine the adjoint variable λ.

Furthermore, from condition (3.12) we derive the constraint $c(y,u)=0$ and, therefore, we conclude by differentiation:

$$c_y(y(u),u)y_u(u) + c_u(y(u),u) = 0,$$

which can be written as:

$$y_u(u) = -c_y(y(u),u)^{-1}c_u(y(u),u). \tag{3.14}$$

The gradient in the right-hand side of (3.6) can, thus, be computed as:

$$\begin{aligned}\nabla\hat{\mathcal{J}}(u) &= y_u(u)^T\nabla_y\mathcal{J}(y(u),u) + \nabla_u\mathcal{J}(y(u),u)\\ &\overset{(3.14)}{=} -c_u(y(u),u)^Tc_y(y(u),u)^{-T}\nabla_y\mathcal{J}(y(u),u) + \nabla_u\mathcal{J}(y(u),u)\\ &\overset{(3.13)}{=} c_u(y(u),u)^T\lambda(u) + \nabla_u\mathcal{J}(y(u),u),\end{aligned} \tag{3.15}$$

where $\lambda(u)$ is the solution to (3.13). Note, that in order to use (3.15) to compute the gradient, the state y and also the adjoint λ are assumed to be computed after a specific control u has been chosen. We emphasize this by the notation $y(u)$ and $\lambda(u)$, respectively. A summary of the computation of $\nabla\hat{\mathcal{J}}(u)$ is stated in Algorithm 3.2.

Algorithm 3.2 Computing $\nabla\hat{\mathcal{J}}(u)$ via adjoints, [23]

1: For a given control u, solve $c(y,u)=0$ for the state $y(u)$
2: Solve the adjoint equation $c_y(y(u),u)^T\lambda = -\nabla_y\mathcal{J}(y(u),u)$ for $\lambda(u)$
3: Compute $\nabla\hat{\mathcal{J}}(u) = \nabla_u\mathcal{J}(y(u),u) + c_u(y(u),u)^T\lambda(u)$

3.1.2 Using adjoint equations for Hessian computation

In order to compute the Hessian of $\hat{\mathcal{J}}$, we first note that equation (3.15) can be written as,

$$\nabla\hat{\mathcal{J}}(u) = \nabla_u\mathcal{L}(y(u),u,\lambda(u)). \tag{3.16}$$

The differentiation of (3.16) then gives,

$$\nabla^2 \hat{\mathcal{J}}(u) = \nabla_{uy}\mathcal{L}(y(u), u, \lambda(u))y_u(u) + \nabla_{uu}\mathcal{L}(y(u), u, \lambda(u)) + \nabla_{u\lambda}\mathcal{L}(y(u), u, \lambda(u))\lambda_u(u), \tag{3.17}$$

where inner derivatives exist due to the dependence of y and λ on u. In order to compute (3.17), we only need to determine the derivative $\lambda_u(\cdot)$ since an expression for $y_u(\cdot)$ is given by (3.14). In order to derive an expression for $\lambda_u(\cdot)$, we use relation (3.11) and differentiate to obtain:

$$\nabla_{yy}\mathcal{L}(y(u), u, \lambda(u))y_u(u) + \nabla_{yu}\mathcal{L}(y(u), u, \lambda(u)) + \nabla_{y\lambda}\mathcal{L}(y(u), u, \lambda(u))\lambda_u(u) = 0$$

and, therefore,

$$\begin{aligned}\lambda_u(u) &= (\nabla_{y\lambda}\mathcal{L}(y(u), u, \lambda))^{-1}\left[-\nabla_{yy}\mathcal{L}(y(u), u, \lambda)y_u(u) - \nabla_{yu}\mathcal{L}(y, u, \lambda)\right] \\ &= (\nabla_{y\lambda}\mathcal{L}(y(u), u, \lambda))^{-1}\left[\nabla_{yy}\mathcal{L}(y(u), u, \lambda)c_y(y(u), u)^{-1}c_u(y(u), u) - \nabla_{yu}\mathcal{L}(y, u, \lambda)\right],\end{aligned} \tag{3.18}$$

where (3.14) has been used to substitute $y_u(u)$.

From the definition (3.9), we see that $\nabla_\lambda \mathcal{L}(y, u, \lambda) = c(y, u)^T$ and, therefore the second mixed derivatives are simply given by,

$$\nabla_{y\lambda}\mathcal{L}(y, u, \lambda) = c_y(y, u)^T, \quad \nabla_{u\lambda}\mathcal{L}(y, u, \lambda) = c_u(y, u)^T. \tag{3.19}$$

If we plug-in (3.14) and (3.18) into (3.17) and use relation (3.19), we end up with,

$$\begin{aligned}\nabla^2\hat{\mathcal{L}}(u) = {} & c_u(y(u), u)^T c_y(y(u), u)^{-T}\nabla_{yy}\mathcal{L}(y(u), u, \lambda(u))c_y(y(u), u)^{-1}c_u(y(u), u) \\ & - c_u(y(u), u)^T c_y(y(u), u)^{-T}\nabla_{yu}\mathcal{L}(y(u), u, \lambda(u)) \\ & - \nabla_{uy}\mathcal{L}(y(u), u, \lambda(u))c_y(y(u), u)^{-1}c_u(y(u), u) + \nabla_{uu}\mathcal{L}(y(u), u, \lambda(u)),\end{aligned} \tag{3.20}$$

which is obviously an identity that can be used to compute the $n_u \times n_u$ Hessian matrix. In an efficient implementation, however, we want to avoid the computation of inverses. Therefore, we introduce the following auxiliary variables,

$$w := c_y(y(u), u)^{-1}c_u(y(u), u), \tag{3.21}$$

$$\begin{aligned}p := {} & c_y(y(u), u)^{-T}\nabla_{yy}\mathcal{L}(y(u), u, \lambda(u))c_y(y(u), u)^{-1}c_u(y(u), u) \\ & - c_y(y(u), u)^{-T}\nabla_{yu}\mathcal{L}(y(u), u, \lambda(u)) \\ = {} & c_y(y(u), u)^{-T}\left(\nabla_{yy}\mathcal{L}(y(u), u, \lambda(u))w - \nabla_{yu}\mathcal{L}(y(u), u, \lambda(u))\right)\end{aligned} \tag{3.22}$$

which is equivalent to solving the systems for w and p respectively,

$$c_y(y(u), u)w = c_u(y(u), u), \tag{3.23}$$

$$c_y(y(u), u)^T p = \nabla_{yy}\mathcal{L}(y(u), u, \lambda(u))w - \nabla_{yu}\mathcal{L}(y(u), u, \lambda(u)). \tag{3.24}$$

The computation of the Hessian can, thus, be obtained in three steps as shown in Algorithm 3.3. Note, that in Algorithm 3.3 we do not compute the whole Hessian matrix but

instead a matrix-vector product with the Hessian. This is the main computation required by iterative linear solvers, such as the truncated CG algorithm (see Section B.1). Iterative solvers are needed for large problems when the computation and storage of the full Hessian is expensive.

Algorithm 3.3 Computing the product $\nabla^2\hat{\mathcal{J}}(u)\cdot v$ via adjoints, [23]

1: Assuming that for a given u we have already computed $y(u), \lambda(u)$ in Algorithm 3.2
2: Solve the equation $c_y(y(u),u)w = c_u(y(u),u)v$ for w
3: Solve the equation $c_y(y(u),u)^T p = \nabla_{yy}\mathcal{L}(y(u),u,\lambda(u))w - \nabla_{yu}\mathcal{L}(y(u),u,\lambda(u))v$ for p
4: Compute $\nabla^2\hat{\mathcal{J}}(u)v = c_u(y(u),u)^T p - \nabla_{uy}\mathcal{L}(y(u),u,\lambda(u))w + \nabla_{uu}\mathcal{L}(y(u),u,\lambda(u))v$

Note, that in most applications, mixed derivatives of the Lagrangian function vanish which makes the computation of the Hessian easier. However, we will also point out in Section 3.3 and 4.2 that the computation of second derivatives that are required in Algorithm 3.3 are not trivial to compute in practice.

3.2 Gradient-based optimization techniques

Especially when dealing with large-scale optimization problems of the form (3.5), one is interested in numerical algorithms that do not require the computation of the Hessian matrix of the objective function. A straightforward approach to minimize a function is to calculate the gradient of that function at each step of the iteration and set the search direction equal to the negative gradient, $s_k = -\nabla\hat{\mathcal{J}}(u_k)$. This method is referred to as the *steepest descent method* and it is well-know that its convergence can be very slow, cf. [51]. Therefore, the so-called quasi-Newton methods have been developed. The idea of the quasi-Newton methods is to solve the Newton equation (3.6) without computing the Hessian matrix explicitly but using an approximation to the Hessian. We then solve a linear system of the form

$$H_k s_k = -\nabla\hat{\mathcal{J}}(u_k), \tag{3.25}$$

where H_k is an approximation of the Hessian at u_k. In this section, we present the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method and the spectral projected gradient (SPG) method which are two algorithms of the family of quasi-Newton methods. For the computation of the gradient in (3.25), we refer to Algorithm 3.2 of the previous section.

3.2.1 The Broyden–Fletcher–Goldfarb–Shanno (BFGS) method

We will present a short motivation of the famous BFGS method based on the textbooks [36, 39]. The MATLAB implementation that has been used in Section 3.3.2 can be found in the `immoptibox` developed at the Technical University of Denmark, cf. [35]. We start with the basic idea, that the approximation of the Hessian is improved in every iteration

of the BFGS method by an update,

$$H_{k+1} = H_k + (\Delta H)_k, \quad H_0 = I.$$

We will see that the initial choice $H_0 = I$ is equivalent to a steepest descent step in the first iteration. In order to motivate the formula for $(\Delta H)_k$, we introduce the second order Taylor expansion of $\hat{\mathcal{J}}(\cdot)$ at $u = u_{k+1}$,

$$T_{k+1}(u) = \hat{\mathcal{J}}(u_{k+1}) + \nabla\hat{\mathcal{J}}(u_{k+1})(u - u_{k+1}) + \frac{1}{2}(u - u_{k+1})^T H_{k+1}(u - u_{k+1}). \tag{3.26}$$

In order to improve the accuracy of the approximation (3.26) to the objective function $\hat{\mathcal{J}}$, we want to impose the condition that the gradients at $u = u_k$ are the same, i.e.

$$\nabla T_{k+1}(u_k) = \nabla\hat{\mathcal{J}}(u_k). \tag{3.27}$$

Note that this is an extra condition since from the Taylor approximation, it only follows that $T_{k+1}(u_{k+1}) = \hat{\mathcal{J}}(u_{k+1})$ and $\nabla T_{k+1}(u_{k+1}) = \nabla\hat{\mathcal{J}}(u_{k+1})$.

In order to fulfill (3.27), we derive

$$\nabla T_{k+1}(u_k) = \nabla\hat{\mathcal{J}}(u_{k+1}) - \alpha_k^* H_{k+1} s_k \stackrel{!}{=} \nabla\hat{\mathcal{J}}(u_k),$$

where the left-hand side of the above equation holds since $u_k - u_{k+1} = -\alpha_k^* s_k$ and the approximate Hessian H_{k+1} is assumed to be symmetric. This leads to the so-called *Secant equation*

$$H_{k+1} p_k = g_k, \tag{3.28}$$

where $p_k := \alpha_k^* s_k = u_{k+1} - u_k$ and $g_k := \nabla\hat{\mathcal{J}}(u_{k+1}) - \nabla\hat{\mathcal{J}}(u_k)$ are introduced.

Since the condition (3.27) does not lead to a unique solution of the update, we formulate instead the following minimization problem,

$$\begin{aligned} H_{k+1} &= \underset{H}{\operatorname{argmin}} \|H - H_k\|, \\ s.t. \quad & H = H^T, Hp_k = g_k, \end{aligned} \tag{3.29}$$

where we are looking for a next iterate H_{k+1} that is *close* to the previous approximate Hessian and which is symmetric and fulfills the Secant equation. In [39], the solution to (3.29) is presented when an appropriate matrix norm is chosen. For a certain weighted Frobenius norm, the result for the update is given by,

$$(\Delta H)_k = \frac{g_k g_k^T}{g_k^T p_k} - \frac{H_k p_k p_k^T H_k}{p_k^T H_k p_k}.$$

We can summarize the BFGS method in the following algorithm.

Algorithm 3.4 Broyden–Fletcher–Goldfarb–Shanno (BFGS) method, [39, Section 6.15]

1: Set initial control $u_0 = 0$, $H_0 = I$, stopping tolerances $\varepsilon_{\mathcal{J}}, \varepsilon_{\nabla} > 0$, `max_bfgs` $\in \mathbb{N}$
2: **for** $k = 0$ to `max_bfgs` **do**
3: Solve $c(y_k, u_k) = 0$ for y_k
4: Save $\mathcal{J}_{old} := \mathcal{J}(y_k, u_k)$
5: Compute $\nabla\hat{\mathcal{J}}(u_k)$ via Algorithm 3.2
6: **if** $\|\nabla\hat{\mathcal{J}}(u_k)\| < \varepsilon_{\nabla}$ **then**
7: **return**
8: **end if**
9: Solve the Newton equation using the approximate Hessian, $H_k s_k = -\nabla\hat{\mathcal{J}}(u_k)$
10: Obtain $\alpha_k^* \approx \operatorname{argmin}_{\alpha_k \in \mathbb{R}_+} \mathcal{J}(y(u_k + \alpha_k s_k), u_k + \alpha s_k)$ using an appropriate line search algorithm
11: Set new control, $u_{k+1} = u_k + \alpha_k^* s_k$
12: **if** $|\mathcal{J}_{old} - \mathcal{J}(y(u_{k+1}), u_{k+1})| < \varepsilon_{\mathcal{J}}$ **then**
13: **return**
14: **end if**
15: Set $p_k := \alpha_k^* s_k$
16: Set $g_k := \nabla\hat{\mathcal{J}}(u_{k+1}) - \nabla\hat{\mathcal{J}}(u_k)$
17: Compute $H_{k+1} = H_k + \frac{g_k g_k^T}{g_k^T p_k} - \frac{H_k p_k p_k^T H_k}{p_k^T H_k p_k}$
18: **end for**

In [39], we also find that the linear system in line 9 of Algorithm 3.4 can be solved easily since an analytic formula for the inverse is known:

$$H_{k+1}^{-1} = H_k^{-1} + \frac{(p_k g_k + g_k^T H_k^{-1} g_k)(p_k p_k^T)}{(p_k^T g_k)^2} - \frac{H_k^{-1} g_k p_k^T p_k g_k^T H_k^{-1}}{p_k^T g_k}. \tag{3.30}$$

3.2.2 The spectral projected gradient (SPG) method

The advantage of the SPG method is that we are able to solve optimization problems with additional convex constraints, i.e.

$$\min_u \hat{\mathcal{J}}(u), \quad \text{subject to } u \in \Omega_c, \tag{3.31}$$

where Ω_c is a convex subset of $\mathbb{R}^{n_u}$. Let $u_l < u_r$ be an upper and a lower bound for the control. In many applications, one is interested in limiting the control u in the following way, $u_l \leq u \leq u_r$, which leads to a convex feasible set. Note that the control u can be a multi-dimensional vector. In this case, the inequalities of the bound constraint are understood componentwise and the domain Ω_c is understood to be a hypercube. We will denote the Euclidean projector onto Ω_c by $\mathcal{P}_{\Omega_c}$. Note that this projection can be implemented efficiently in MATLAB via,

```
proj(u) = min(max(u,ul),ur);
```

The following derivation of the SPG method is based on [7, 8]. We follow the same approach as in Section 3.2.1 in the sense that we seek an approximate Hessian that fulfills

the Secant equation (3.28). Furthermore, we make a rank-1 ansatz for the approximate Hessian,

$$H_{k+1} = \gamma_{k+1} I, \quad \text{with } \gamma_{k+1} \in \mathbb{R}.$$

The Secant equation (3.28) then reduces to $\gamma_{k+1} p_k = g_k$ which is an over-determined system of equations. The least-squares solution is given by,

$$\gamma_{k+1} = \underset{\gamma}{\operatorname{argmin}} \|\gamma p_k - g_k\|_2^2 = \frac{p_k^T g_k}{p_k^T p_k},$$

which is known to be the *Rayleigh* quotient corresponding to an average Hessian matrix as shown in detail in [7]. Because of the usage of the Rayleigh quotient in eigenvalue computation, this method is also called a *spectral method*. Since by design, the approximate Hessian is easily invertible, we can derive a search direction from the approximate Newton equation (3.25),

$$\hat{s}_k = -H_k^{-1} \nabla \hat{\mathcal{J}}(u_k) = -\underbrace{\frac{1}{\gamma_k}}_{=:\lambda_k} \nabla \hat{\mathcal{J}}(u_k) = -\lambda_k \nabla \hat{\mathcal{J}}(u_k),$$

where we denote the inverse Rayleigh quotient by λ_k. Using this direction, the new iterate $\hat{u}_{k+1} = u_k + \hat{s}_k = u_k - \lambda_k \nabla \hat{\mathcal{J}}(u_k)$ might not be within the bounds of Ω_c. Therefore, we need to project the new iterate onto Ω_c and derive the new search direction according to [7],

$$s_k = \mathcal{P}_{\Omega_c}(\hat{u}_{k+1}) - u_k = \mathcal{P}_{\Omega_c}(u_k - \lambda_k \nabla \hat{\mathcal{J}}(u_k)) - u_k.$$

The complete SPG algorithm is presented next. For the line search algorithm used within the SPG method, we refer to the original paper [7]. Also some details like the usage of so-called safeguards $0 < \lambda_{min} \leq \lambda_{max} < \infty$ are not discussed here but in references [7, 8].

In Algorithm 3.5 we present to SPG method according to [7]. Note that in a similar way as in Algorithm 3.1, we have included the numerical stopping criteria $\varepsilon_{\mathcal{J}}$ and ε_{∇} in order to stop the optimization procedure when either the zero-gradient condition is fulfilled or the change in the objective function is small.

Algorithm 3.5 Spectral projected gradient (SPG) method, [7]

Set initial control u_0, $\lambda_0 \in [\lambda_{min}, \lambda_{max}]$, stopping tolerances $\varepsilon_{\mathcal{J}}, \varepsilon_{\nabla} > 0$, `max_spg` $\in \mathbb{N}$
Project $u_0 = \mathcal{P}_{\Omega_c}(u_0)$
for $k = 0$ to `max_spg` **do**
  Save $\mathcal{J}_{old} := \mathcal{J}(y(u_k), u_k)$
  if $\|\mathcal{P}_{\Omega_c}(u_k - \nabla\hat{\mathcal{J}}(u_k)) - u_k\|_\infty \leq \varepsilon_{\nabla}$ **then**
    return
  end if
  Compute search direction, $s_k = \mathcal{P}_{\Omega_c}(u_k - \lambda_k \nabla\hat{\mathcal{J}}(u_k)) - u_k$
  Perform non-monotone line search as described in [8, Algorithm 2.2], obtain step length α_k^*
  Update $u_{k+1} = u_k + \alpha_k^* s_k$
  if $|\mathcal{J}_{old} - \mathcal{J}(y(u_{k+1}), u_{k+1})| < \varepsilon_{\mathcal{J}}$ **then**
    return
  end if
  Set $p_k = u_{k+1} - u_k$
  Set $g_k = \nabla\hat{\mathcal{J}}(u_{k+1}) - \nabla\hat{\mathcal{J}}(u_k)$
  if $p_k^T g_k \leq 0$ **then**
    Set $\lambda_{k+1} := \lambda_{max}$
  else
    Set $\lambda_{k+1} := \max\{\lambda_{min}, \min\{p_k^T p_k / p_k^T g_k, \lambda_{max}\}\}$
  end if
end for

3.3 Application: Optimal control of Burgers' equation

In order to apply the optimization algorithms described in the previous sections, we need to specify a cost function $\mathcal{J}$ as well as the nonlinear PDE c that is constraining the optimization problem (3.3). In this section, we will consider a test problem that has been widely used in the literature and can, thus, be considered as a standard test problem, cf. [23, 29]. We want to minimize the following cost functional,

$$\min_u \frac{1}{2} \int_0^T \int_0^L [y(t,x) - z(t,x)]^2 + \omega u^2(t,x) \; dx \; dt, \tag{3.32}$$

where y is a solution the one-dimensional, unsteady Burgers' equation with homogeneous Dirichlet boundary conditions and initial condition $y_0(x)$,

$$\begin{aligned} y_t + \left(\frac{1}{2}y^2 - \nu y_x\right)_x &= f + u, \quad (x,t) \in (0,L) \times (0,T), \\ y(t,0) = y(t,L) &= 0, \quad t \in (0,T), \\ y(0,x) &= y_0(x), \quad x \in (0,L). \end{aligned} \tag{3.33}$$

In (3.32), the function z is a given function defined on $\Omega \times [0, T]$. We consider z to be the *desired state* of the optimization problem (3.32)-(3.33) because if we are able to control the solution of Burgers' equation in such a way that the difference between y and z is small on the whole domain $\Omega \times [0, T]$, then the value of the objective function (3.32) is small.

The parameter $\omega \in \mathbb{R}_+$ is called the *control penalty*. Usually, ω is chosen to be small such that a relatively large control u is allowed that drives the state y into the desired state z. The control itself appears on the right-hand side of Burgers' equation and can be chosen arbitrarily as long as the initial and boundary conditions on y are not violated.

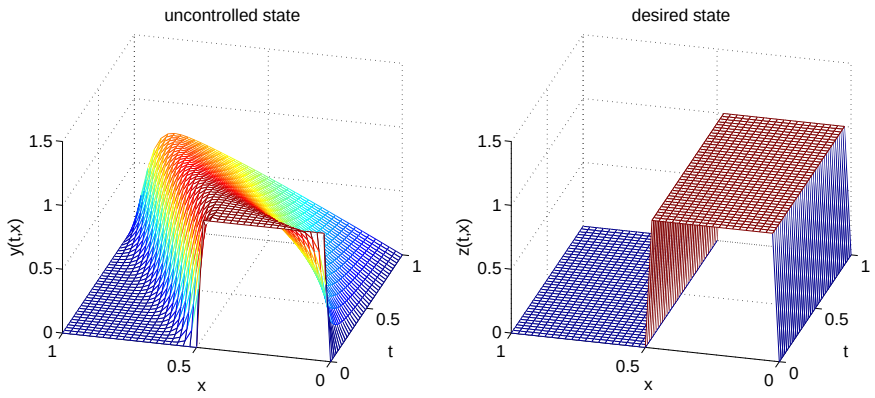


Figure 3.1: Uncontrolled and desired state for $\nu = 0.01$.

We now present a discretization of the cost functional (3.32) as well as Burgers' equation (3.33) which is again in conservative form as already seen in Section 2.3. Therefore, we make the ansatz that the control u can be approximated in a finite element way as the superposition of piecewise linear test functions ϕ_j as introduced in Appendix A.1. This approach can be written as,

$$u(t,x) \approx \sum_{j=1}^{N} u_j(t)\phi_j(x), \tag{3.34}$$

where u_j are the respective coefficients of ϕ_j that only depend on time, see for example [17]. Note, that this ansatz also implies that the control is zero at the boundary of Ω and, therefore, does not change the behavior of the solution y at those points. Furthermore, we will choose $u(0,x) = 0$ as initial control which, again, does not affect the initial condition on y.

In order to discretize (3.32), the outermost time integral has been approximated by a simple sum using a constant step size δt for the discretization of the time interval $[0, T]$. We therefore obtain all time-dependent quantities at discrete time instances t_i, where $t_0 = 0$ and $t_{N_t} = T$. Recall, that in Appendix A.1 the state has been approximated in the following way, $y(t,x) \approx \sum_{j=1}^{N} y_j(t)\phi_j(x)$, a fully discrete version of the cost functional is given by,

$$\min_{\mathbf{u}_0,\dots,\mathbf{u}_{N_t}} \mathcal{J}(\mathbf{y}_0,\dots,\mathbf{y}_{N_t},\mathbf{u}_0,\dots,\mathbf{u}_{N_t}) = \min_{\mathbf{u}_0,\dots,\mathbf{u}_{N_t}} \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2}\mathbf{y}_i^T M \mathbf{y}_i - \mathbf{z}^T \mathbf{y}_i + \frac{\omega}{2}\mathbf{u}_i^T M \mathbf{u}_i \right), \tag{3.35}$$

where the vector-valued quantities

$$\mathbf{y}_i = \begin{pmatrix} y_1(t_i) \\ \vdots \\ y_N(t_i) \end{pmatrix} \in \mathbb{R}^N, \quad \mathbf{u}_i = \begin{pmatrix} u_1(t_i) \\ \vdots \\ u_N(t_i) \end{pmatrix} \in \mathbb{R}^N, \quad \text{for } t_i \in [0, T]$$

are introduced and M is the mass matrix as defined in Appendix A.1. Note, that in (3.35) the constant and positive term

$$\frac{1}{2} \int_0^T \int_0^L z^2(t, x) \; dx \; dt$$

has been neglected since it does not influence the position of the minimum with respect to u. This can lead to a negative value of the discrete objective function even though (3.32) is positive. Furthermore, we assume that the desired state z does not depend on time. The vector $\mathbf{z}$ tested against the hat functions ϕ_j is therefore given by,

$$\mathbf{z} = \begin{pmatrix} \int_0^L z(x)\phi_1(x)dx \\ \vdots \\ \int_0^L z(x)\phi_N(x)dx \end{pmatrix} \approx h \begin{pmatrix} z(x_1) \\ \vdots \\ z(x_N) \end{pmatrix}. \tag{3.36}$$

The discretization of Burgers' equation (3.33) with the control on the right-hand side has been obtained by a finite element approach in space and an implicit Euler method in time as described in Appendix A.1 and Appendix A.2, respectively. The resulting discrete constraint in the form $c(y, u) = 0$ is a vector-valued function with the components equal to,

$$c_{i+1}(\mathbf{y}_i, \mathbf{y}_{i+1}, \mathbf{u}_{i+1}) \equiv \frac{1}{\delta t} M\mathbf{y}_{i+1} - \frac{1}{\delta t} M\mathbf{y}_i + \frac{1}{2} B\mathbf{y}_{i+1}^2 + \nu C\mathbf{y}_{i+1} - \mathbf{f} - M\mathbf{u}_{i+1} = 0, \tag{3.37}$$

where $i = 0, ..., N_t - 1$ and the constraint is given by $c := [c_1, ..., c_{N_t}]^T$ which is a function of the state and the control at all discrete time instances.

Note, that the only nonlinearity that remains is derived from the discretization of the convective term of Burgers' equation. We will denote this nonlinearity by,

$$\mathcal{N}(\mathbf{y}_{i+1}) := \frac{1}{2} B\mathbf{y}_{i+1}^2, \tag{3.38}$$

and point out the special treatment of the nonlinearity in the following application of the Newton-type method using adjoint techniques for the derivative computation as introduced in Section 3.1.

Therefore, we first note that after discretization in space and time, the cost function (3.35) together with the constraint (3.37) fit the framework of (3.3). We can, thus, build the fully discretized Lagrangian function according to the definition (3.9). The discrete

Lagrangian of the full-order model is given by,

$$\begin{aligned}
&\mathcal{L}(\mathbf{y}_0, ..., \mathbf{y}_{N_t}, \mathbf{u}_0, ..., \mathbf{u}_{N_t}, \boldsymbol{\lambda}_1, ..., \boldsymbol{\lambda}_{N_t}) \\
&= \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2} \mathbf{y}_i^T M \mathbf{y}_i - \mathbf{z}^T \mathbf{y}_i + \frac{\omega}{2} \mathbf{u}_i^T M \mathbf{u}_i \right) \\
&\quad + \sum_{i=0}^{N_t-1} \boldsymbol{\lambda}_{i+1}^T \left(\frac{1}{\delta t} M \mathbf{y}_{i+1} - \frac{1}{\delta t} M \mathbf{y}_i + \frac{1}{2} B \mathbf{y}_{i+1}^2 + \nu C \mathbf{y}_{i+1} - \mathbf{f} - M \mathbf{u}_{i+1} \right), \qquad (3.39)
\end{aligned}$$

where the adjoint variable $\boldsymbol{\lambda}_i$ at each time instance is a vector of dimension N.

In all our numerical test calculations, adjoints were used for the computation of gradients and Hessian-vector products. Therefore, we will next present how the adjoint Algorithms 3.2 and 3.3 have been applied to the discretized Burgers' equation (3.35)-(3.37).

3.3.1 Numerical results for a Newton-type method using adjoints

We want to apply the optimization Algorithm 3.1 in order to solve (3.35) for $\mathbf{u}_0, ..., \mathbf{u}_{N_t}$, when $\mathbf{y}_0, ..., \mathbf{y}_{N_t}$ is the solution of (3.37). Therefore, we will need the corresponding Lagrangian function (3.39) as well as the gradient and the Hessian-times-vector product as described in the algorithms 3.2 and 3.3, respectively. We will concentrate on the solution of the two adjoint equations and refer to Appendix A for the numerical solution of Burgers' equation.

In the adjoint Algorithm 3.2 that computes the gradient of the cost functional (3.35), we mostly need to compute partial derivatives of the constraint (3.37) and the cost functional itself with respect to the control u and the state variable y. Since both the constraint and the cost functional depend on the control and the state at all time instances, the respective variables we need to consider are of the size $N \cdot N_t$,

$$\underline{\mathbf{u}} := \begin{pmatrix} \mathbf{u}_0 \\ \vdots \\ \mathbf{u}_{N_t} \end{pmatrix} \in \mathbb{R}^{(N \cdot N_t) \times 1}, \quad \underline{\mathbf{y}} := \begin{pmatrix} \mathbf{y}_0 \\ \vdots \\ \mathbf{y}_{N_t} \end{pmatrix} \in \mathbb{R}^{(N \cdot N_t) \times 1}.$$

Therefore, at every outer iteration of the optimization loop, we first need to solve Burgers' equation on the whole time interval. In Algorithm 3.6, we summarize the application of Algorithm 3.2 to the full-order discrete optimal control problem with Burgers' equation as an implicit constraint. Thereby, the adjoint equation (3.13) reduces to an ordinary differential equation in the adjoint variable. Note, that we first need to solve the terminal condition (3.40a) for $\boldsymbol{\lambda}_{N_t}$ and then solve the set of equations (3.40b) backwards in time. Given the solution of (3.40a)-(3.40b), the gradient of the cost function with respect to the control u can be obtained according to (3.15).

Algorithm 3.6 Algorithm 3.2 applied to the full-order discrete Burgers' equation

1: From the initial condition $\mathbf{y}_0$ and the current control $\mathbf{u}_1, ..., \mathbf{u}_{N_t}$, solve Burgers' equation for $\mathbf{y}_1, ..., \mathbf{y}_{N_t}$ as described in Appendix A

2: The adjoint equation (3.13) reads:

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_{N_t}) + \nu C\right)^T \boldsymbol{\lambda}_{N_t} = -\delta t(M\mathbf{y}_{N_t} - \mathbf{z}) \tag{3.40a}$$

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_i) + \nu C\right)^T \boldsymbol{\lambda}_i = -(-\frac{1}{\delta t}M)^T\boldsymbol{\lambda}_{i+1} - \delta t(M\mathbf{y}_i - \mathbf{z}), \quad i = N_t - 1, ..., 1 \tag{3.40b}$$

3: The gradient is computed according to formula (3.15):

$$\nabla_u \hat{\mathcal{J}}(\mathbf{u}_0, ..., \mathbf{u}_{N_t}) = \begin{pmatrix} \delta t\omega M\mathbf{u}_0 \\ \delta t\omega M\mathbf{u}_1 - M^T\boldsymbol{\lambda}_1 \\ \vdots \\ \delta t\omega M\mathbf{u}_{N_t} - M^T\boldsymbol{\lambda}_{N_t} \end{pmatrix} \tag{3.41}$$

In (3.40a) and (3.40b) it is necessary to compute the first derivative of the nonlinear term (3.38). For an arbitrary vector $\mathbf{y} = [y_1, ..., y_N]^T$ and a matrix $B \in \mathbb{R}^{N\times N}$, the first derivative of the nonlinear term $\mathcal{N}(\cdot)$ is given by,

$$\mathcal{N}'(\mathbf{y}) = \frac{d}{d\mathbf{y}}\left(\frac{1}{2}B\mathbf{y}^2\right) = \begin{pmatrix} B_{1,1}y_1 & \dots & B_{1,N}y_N \\ B_{2,1}y_1 & \dots & B_{2,N}y_N \\ \vdots & & \vdots \\ B_{N,1}y_1 & \dots & B_{N,N}y_N \end{pmatrix} \in \mathbb{R}^{N\times N},$$

which is again an $N \times N$ matrix.

In order to solve the Newton equation (3.6) we also need to compute the Hessian $\nabla^2\hat{\mathcal{J}}(u)$. Since this is a matrix of dimension $N \cdot N_t \times N \cdot N_t$, we solve the linear system (3.6) with the truncated CG method where we only need to compute the product of the Hessian times a vector and never need to store the whole Hessian matrix, see Algorithm B.1. In Algorithm 3.7, we present the application of the general Hessian-times-vector computation as derived in Section 3.1.2 to the optimization of Burgers' equation. Therefore, we define the arbitrary vector $\underline{\mathbf{v}} := (\mathbf{v}_0^T, ..., \mathbf{v}_{N_t}^T)^T$ and derive the equations (3.42a)-(3.42b) and (3.43a)-(3.43b) for the auxiliary variables w and p according to the respective general formulas (3.23) and (3.24). It is important to note that the initial condition (3.42a) and the terminal condition (3.43a) follow directly from the general equations (3.23) and (3.24) when the respective partial derivative is computed.

Algorithm 3.7 Algorithm 3.3 applied to the full-order discrete Burgers' equation

1: We assume that we have already computed $\mathbf{y}_0, ..., \mathbf{y}_{N_t}, \mathbf{u}_0, ..., \mathbf{u}_{N_t}, \boldsymbol{\lambda}_1, ..., \boldsymbol{\lambda}_{N_t}$ in Algorithm 3.6

2: Equation (3.23) reads:

$$\mathbf{w}_0 = 0 \tag{3.42a}$$

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_{i+1}) + \nu C\right)\mathbf{w}_{i+1} = -(-\frac{1}{\delta t}M)\mathbf{w}_i - M\mathbf{v}_{i+1}, \quad i = 0, ..., N_t - 1 \tag{3.42b}$$

3: Equation (3.24) reads:

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_{N_t}) + \nu C\right)^T \mathbf{p}_{N_t} = \delta t M\mathbf{w}_{N_t} + \mathrm{diag}(\boldsymbol{\lambda}_{N_t}^T b_1, ..., \boldsymbol{\lambda}_{N_t}^T b_N)\mathbf{w}_{N_t} \tag{3.43a}$$

$$\begin{aligned}\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_i) + \nu C\right)^T \mathbf{p}_i = &-(-\frac{1}{\delta t}M)^T\mathbf{p}_{i+1} + \delta t M\mathbf{w}_i \\ &+ \mathrm{diag}(\boldsymbol{\lambda}_i^T b_1, ..., \boldsymbol{\lambda}_i^T b_N)\mathbf{w}_i, \quad i = N_t - 1, ..., 1\end{aligned} \tag{3.43b}$$

4: The Hessian times a vector $\underline{\mathbf{v}}$ is computed according to formula (3.20):

$$\nabla^2\hat{\mathcal{J}}(\mathbf{u}_0, ..., \mathbf{u}_{N_t}) \cdot \underline{\mathbf{v}} = \begin{pmatrix} \delta t\omega M\mathbf{v}_0 \\ -M^T\mathbf{p}_1 + \delta t\omega M\mathbf{v}_1 \\ \vdots \\ -M^T\mathbf{p}_{N_t} + \delta t\omega M\mathbf{v}_{N_t} \end{pmatrix} \tag{3.44}$$

Note that in (3.43a)-(3.43b) as well as in (3.44) it is necessary to compute second partial derivatives of the Lagrangian function. Therefore, we first note that due to the definition of the Lagrangian, mixed second order derivatives vanish. Furthermore, we present the analytic computation of the second partial derivative of the quantity $\boldsymbol{\lambda}^T\mathcal{N}(\mathbf{y})$ with respect to the state variable $\mathbf{y}$,

$$\begin{aligned}\frac{d^2}{d\mathbf{y}^2}\left(\boldsymbol{\lambda}^T\mathcal{N}(\mathbf{y})\right)\mathbf{w} &= \frac{d^2}{d\mathbf{y}^2}\left(\boldsymbol{\lambda}^T(\frac{1}{2}B\mathbf{y}^2)\right)\mathbf{w} = \frac{d^2}{d\mathbf{y}^2}\left(\frac{1}{2}\sum_{k=1}^N \lambda_k \sum_{j=1}^N B_{k,j}y_j^2\right)\mathbf{w} \\ &= \frac{d}{d\mathbf{y}}\begin{pmatrix} \sum_{k=1}^N \lambda_k B_{k,1}y_1 \\ \vdots \\ \sum_{k=1}^N \lambda_k B_{k,N}y_N \end{pmatrix}\mathbf{w} = \begin{pmatrix} \sum_{k=1}^N \lambda_k B_{k,1} & & \\ & \ddots & \\ & & \sum_{k=1}^N \lambda_k B_{k,N} \end{pmatrix}\mathbf{w} \\ &= \mathrm{diag}(\boldsymbol{\lambda}^T b_1, ..., \boldsymbol{\lambda}^T b_N)\mathbf{w},\end{aligned}$$

where $\boldsymbol{\lambda} = (\lambda_1, ..., \lambda_N)^T$, $\mathbf{y} = (y_1, .., y_N)^T$,and $b_1, ..., b_N$ are the columns of the matrix B such that $B = (b_1|...|b_N)$.

Figures 3.2 and 3.3 show numerical results of Algorithm 3.1 applied to problem (3.35) with implicit constraint (3.37). In the considered setting, we chose $\nu = 0.01$ in Burgers' equation, $\omega = 0.005$ for the control penalty, and $N = N_t = 80$ grid points in time and space. In order to be able to perform the optimization Algorithm 3.1 using Armijo

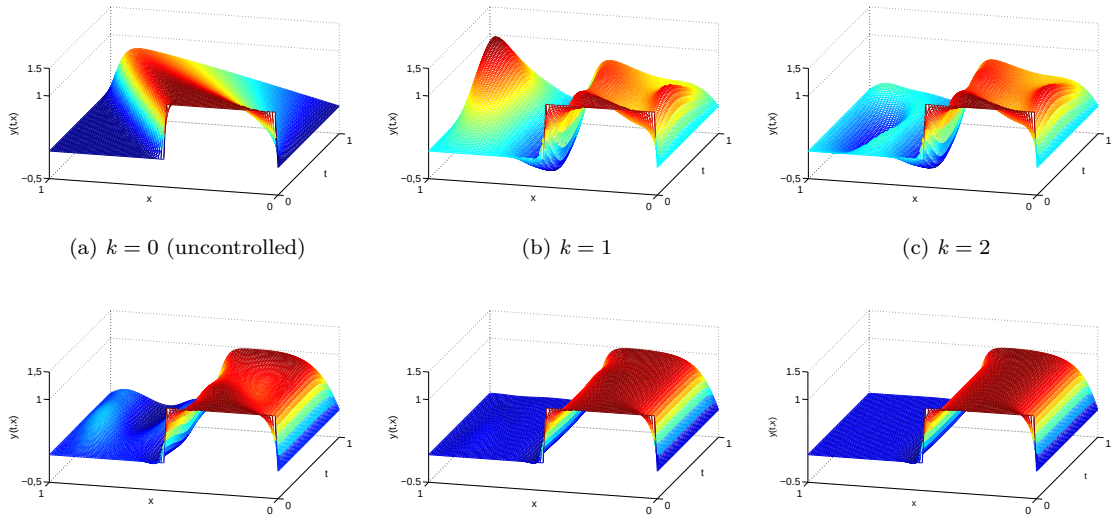
line search as described in Algorithm B.2 and the gradient and Hessian-vector product computation in Algorithms 3.6 and 3.7, respectively, we need to specify some settings which have been summarized in Table 3.1. Also the Newton iteration for the numerical solution of Burgers' equation requires to set some tolerances.

$\varepsilon_{\mathcal{J}}$	ε_{∇}	ε_{α}	α_{min}	η_k	ε_{Newton}	`max_newton`
`10e-8`	`10e-9`	`10e-4`	`10e-8`	`10e-2`	`10e-3`	20

Table 3.1: Choice of parameters for the numerical results in Figure 3.2 and 3.3.

The outer optimization loop of algorithm 3.1 stops when either the value of the objective function does not change anymore ($\varepsilon_{\mathcal{J}}$ in Algorithm 3.1) or the zero-gradient condition is fulfilled up to a certain precision (ε_{∇} in Algorithm 3.1). The tolerance of the Armijo line search is set by ε_{α} and a minimum step length is guaranteed by α_{min}, see Algorithm B.2. The truncated CG-algorithm B.1 is terminated by the choice of η_k, and ε_{Newton} and `max_newton` are, respectively, the tolerance and the maximum number of iterations of Newton's method as described in Algorithm A.1.

From the numerical results in Figure 3.2, we see that the state variable y converges to the desired state z as the number of optimization iterations k increases. For the presented setting, after 6 iterations a state has been reached such that the value of the objective function (3.35) does not decrease any further than `0.024`.

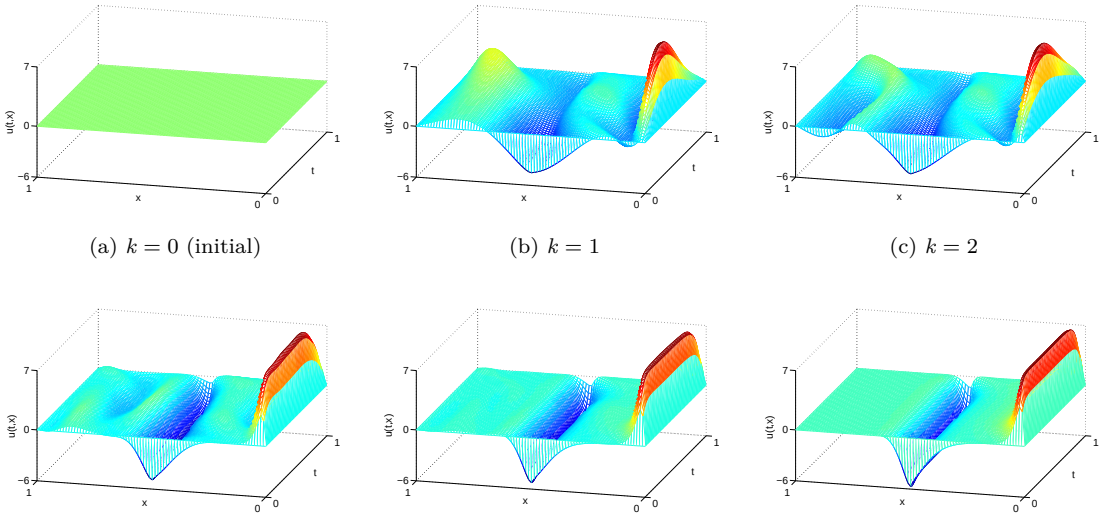


(a) $k = 0$ (uncontrolled) (b) $k = 1$ (c) $k = 2$

(d) $k = 3$ (e) $k = 4$ (f) $k = 5$

Figure 3.2: The state y at different stages k of the optimization iteration.

In Figure 3.3, we present the control that corresponds to the states presented before. After convergence, we see in the last plot of Figure 3.3 the desired optimal control u^* that drives the solution to Burgers' equation into the desired state z, i.e. minimizes the cost function.



(a) $k = 0$ (initial) (b) $k = 1$ (c) $k = 2$

(d) $k = 3$ (e) $k = 4$ (f) $k = 5$

Figure 3.3: The control u at different stages k of the optimization iteration.

A second numerical test using the same parameters as in Table 3.1 has been performed in order to show the dependence of the optimization algorithm on the control penalty ω. From (3.32) we see that the smaller we choose ω the more we put emphasis on driving the state y into the desired state and allow a large control.

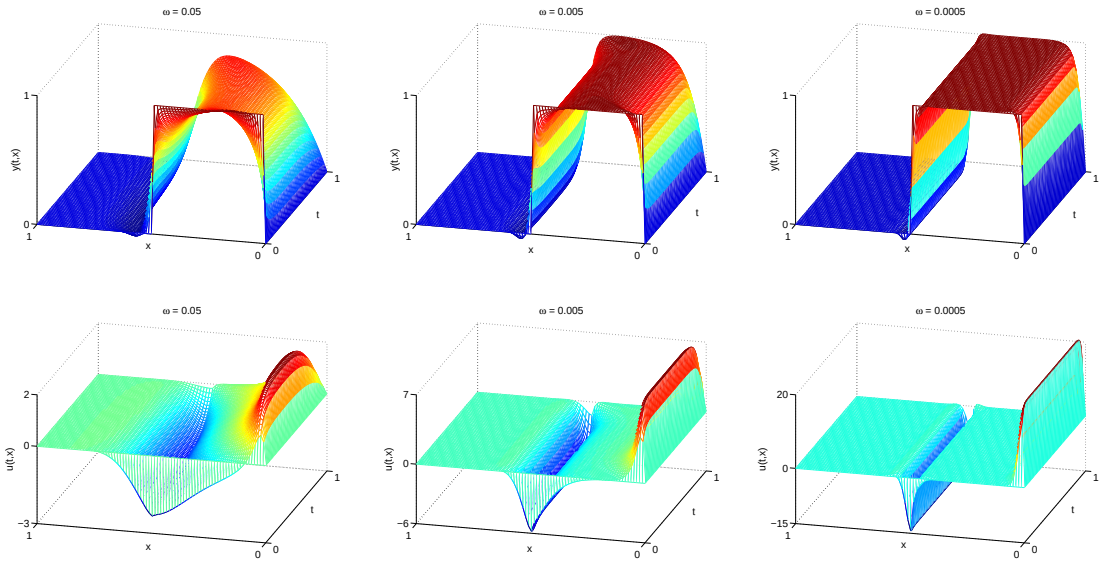


Figure 3.4: State y and corresponding optimal control u after convergence for different control parameters $\omega = \{0.05, 0.005, 0.0005\}$.

In Figure 3.4 we see the final state of the optimization for different values of ω as well as the corresponding optimal control. For the smallest value of ω, we see that the desired state is almost perfectly fitted and, therefore, the control has large values which leads to

a large contribution in the cost function but has been compensated by the small value of ω.

3.3.2 Numerical results for gradient-based optimization methods

In this section, we present results corresponding to the BFGS and SPG methods as described in Section 3.2.1 and 3.2.2, respectively. In our tests, we used publicly available MATLAB implementations that require the cost function and the gradient as input as well as some tolerances and settings. With formula (3.35) and Algorithm 3.6 we have already given a fully discretized version of the cost function and its gradient and, therefore, the application of both first-order optimization algorithms in straightforward. Here, we only present numerical results of the SPG method because this method allows us to impose bounds on the control $\mathbf{u}$ at all time instance. Accuracy and performance results for all approaches including the BFGS method are presented in Section 4.4. One can imagine that bounds on the control arise in many engineering application from physical constraints, cf. [43].

In Figure 3.5, we present the converged solution of the SPG method when the control is bounded between -2 and 2. Clearly, this affects the corresponding optimal state of Burgers' solution and we can see that the desired state can not be reached with the same accuracy as in the previous examples.

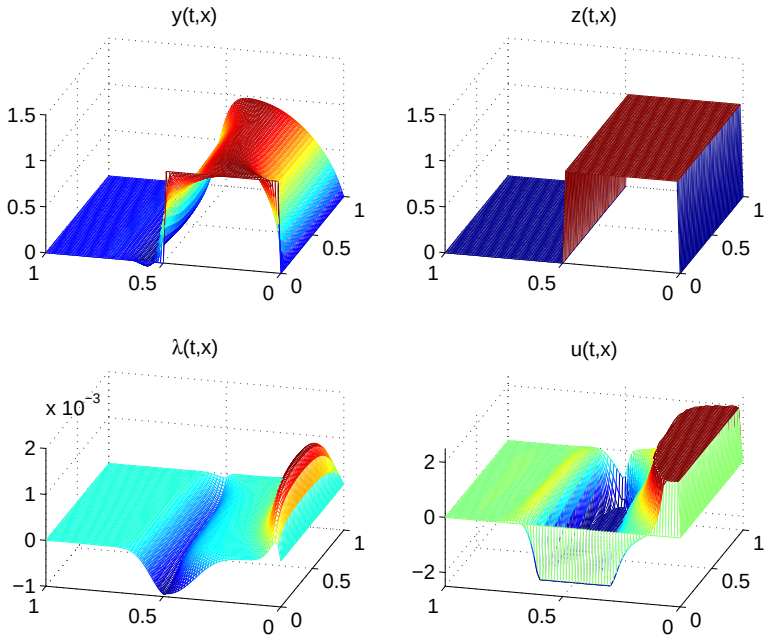


Figure 3.5: Optimal state (upper left), desired state (upper right), adjoint state (lower left) and optimal control (lower right) for a bounded control $-2 \leq u \leq 2$ using the SPG Algorithm 3.5.

Chapter 4

Implementation and analysis of a POD-DEIM reduced model for the optimal control of Burgers' equation

In [29], the authors used a POD-reduced model for the optimal control of Burgers' equation. In this Section, we extend this approach by applying model order reduction using POD-DEIM as described in Section 2 to an optimal control problem as described in Section 3 and compare the POD-DEIM reduced model to the POD model in terms of computational cost and accuracy. The reduced optimal control problem is in the implicitly-constrained form,

$$\begin{aligned} &\min_u \tilde{\mathcal{J}}(\tilde{y}(u), u), \\ &\text{with} \quad \tilde{c}(\tilde{y}, u) = 0, \end{aligned} \tag{4.1}$$

where we use a tilde to indicate that the considered quantity is of lower dimension. In (4.1), the constraining nonlinear PDE $\tilde{c}(\tilde{y}, u) = 0$ is solved for the reduced state variable $\tilde{y}(u)$. Since the evaluation of the cost function $\tilde{\mathcal{J}}$ requires the computation of $\tilde{y}$ first, this approach also leads to a faster evaluation of the cost function due to the lower dimension of the state variable. We will refer to the solution of (4.1) as the optimal control of the reduced system. It is important to note, that even though this optimal control is derived from the reduced system, we denote it without tilde in order to stress that the control is still of the large dimension of the full-order model. This is a natural approach at this point because u can be seen as an input variable because the state $\tilde{y}(u)$ directly depends on the control. A dimension reduction of the control u is not considered at this point. Similarly to the previous section, we denote by $\hat{\tilde{\mathcal{J}}}(u) := \tilde{\mathcal{J}}(\tilde{y}(u), u)$ the cost function as a function of the control only.

4.1 A POD-DEIM reduced model for optimal control of Burgers' equation

In Section 3.3, we introduced the optimal control problem (3.32)-(3.33). We will now derive a POD-DEIM reduced model of the corresponding discrete optimization problem (3.35)-(3.37). Therefore, we first consider the discretized objective function (3.35) and recall that the reduced state variable has been derived in the following way,

$$\mathbf{y}(t) \approx \Phi_\ell \tilde{\mathbf{y}}(t),$$

where the columns of the matrix Φ_ℓ are the POD basis. When we plug in this approximation into (3.35), we obtain the reduced objective function,

$$\min_{\mathbf{u}_0,...,\mathbf{u}_{N_t}} \tilde{J}(\tilde{\mathbf{y}}_0,...,\tilde{\mathbf{y}}_{N_t},\mathbf{u}_0,...,\mathbf{u}_{N_t}) = \min_{\mathbf{u}_1,...,\mathbf{u}_{N_t}} \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2}\tilde{\mathbf{y}}_i^T\tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}^T\tilde{\mathbf{y}}_i + \frac{\omega}{2}\mathbf{u}_i^T M \mathbf{u}_i \right), \quad (4.2)$$

where the mass matrix in the first term vanishes due to the M-orthogonality of Φ_ℓ and the reduced desired state $\tilde{\mathbf{z}}$ is given by,

$$\tilde{\mathbf{z}} := \Phi_\ell^T \mathbf{z} \in \mathbb{R}^\ell. \quad (4.3)$$

Note that (4.3) can be pre-computed once the POD basis is obtained because the desired state of the full-order model is a given function, see (3.36). We further note problem (4.2) is still formulated in terms of the full-dimensional control $\mathbf{u}_i$ at the time instances t_i. In Section 4.3, we will show one way in which a lower-dimensional control can lead to a reduced model that is completely independent of the full-order dimension N.

In order to obtain a POD-DEIM reduced model for the constraining Burgers' equation, we refer to Section 3.3 and apply the POD-DEIM projection to (3.37). The fully discretized reduced constraint is, thus, given by

$$\tilde{c}_i(\tilde{\mathbf{y}}_i,\tilde{\mathbf{y}}_{i+1},\mathbf{u}_{i+1}) \equiv \frac{1}{\delta t}\tilde{\mathbf{y}}_{i+1} - \frac{1}{\delta t}\tilde{\mathbf{y}}_i + \frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}}_{i+1})^2 + \nu\tilde{C}\tilde{\mathbf{y}}_{i+1} - \tilde{\mathbf{f}} - \tilde{M}\mathbf{u}_{i+1} = 0, \quad (4.4)$$

where $i = 0,...,N_t - 1$ and a dimension reduction for the right-hand side as well as the projected mass matrix has been obtained,

$$\tilde{\mathbf{f}} := \Phi_\ell^T \mathbf{f} \in \mathbb{R}^\ell, \quad (4.5)$$

$$\tilde{M} := \Phi_\ell^T M \in \mathbb{R}^{\ell\times N}. \quad (4.6)$$

Note that $\tilde{M}$ still depends on N and the matrices $\tilde{B}, \tilde{C}, \tilde{F}$ have been defined in (2.26), (2.27), (2.28), respectively.

Furthermore, we note that the nonlinear term in (4.4) has to be considered as a function of the reduced variable $\tilde{\mathbf{y}}$ and has the slightly more complicated form,

$$\tilde{\mathcal{N}}(\tilde{\mathbf{y}}_{i+1}) := \frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}}_{i+1})^2. \quad (4.7)$$

The Lagrangian function of the reduced model can be obtained directly from (4.2) and (4.4) and is given by,

$$\begin{aligned}
&\tilde{\mathcal{L}}(\tilde{\mathbf{y}}_0, ..., \tilde{\mathbf{y}}_{N_t}, \mathbf{u}_0, ..., \mathbf{u}_{N_t}, \tilde{\boldsymbol{\lambda}}_1, ..., \tilde{\boldsymbol{\lambda}}_{N_t}) \\
&= \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2} \tilde{\mathbf{y}}_i^T \tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}^T \tilde{\mathbf{y}}_i + \frac{\omega}{2} \mathbf{u}_i^T M \mathbf{u}_i \right) \\
&\quad + \sum_{i=0}^{N_t-1} \tilde{\boldsymbol{\lambda}}_{i+1}^T \left(\frac{1}{\delta t} \tilde{\mathbf{y}}_{i+1} - \frac{1}{\delta t} \tilde{\mathbf{y}}_i + \frac{1}{2} \tilde{B}(\tilde{F} \tilde{\mathbf{y}}_{i+1})^2 + \nu \tilde{C} \tilde{\mathbf{y}}_{i+1} - \tilde{\mathbf{f}} - \tilde{M} \mathbf{u}_{i+1} \right)
\end{aligned} \tag{4.8}$$

In (4.8), we see that the adjoint variable $\tilde{\boldsymbol{\lambda}}_i \in \mathbb{R}^\ell$ is of the same dimension ℓ as the reduced state variable. This gives a good motivation for a faster solution of the adjoint equations (3.13) when applied to the POD-DEIM reduced model.

4.2 A Newton-type method for the POD-DEIM reduced model

In Section 3.3.1, we have already seen that the application of the adjoints for derivative computation according to Algorithm (3.2) and (3.3) is straightforward but the derivation of partial derivatives of the nonlinear term is still involved when applied to a concrete problem. In Algorithm 4.1, we present the computation of the gradient of the reduced objective function which follows from the application of Algorithm 3.2 to the reduced model (4.2)-(4.4).

Algorithm 4.1 Algorithm 3.2 applied to the reduced Burgers' model

1: From the initial condition $\tilde{\mathbf{y}}_0$ and the current control $\mathbf{u}_0, ..., \mathbf{u}_{N_t}$, solve the reduced Burgers' equation for $\tilde{\mathbf{y}}_1, ..., \tilde{\mathbf{y}}_{N_t}$ according to 4.4

2: The adjoint equation (3.13) reads:

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_{N_t}) + \nu \tilde{C} \right)^T \tilde{\boldsymbol{\lambda}}_{N_t} = -\delta t (\tilde{\mathbf{y}}_{N_t} - \tilde{\mathbf{z}}) \tag{4.9a}$$

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_i) + \nu \tilde{C} \right)^T \tilde{\boldsymbol{\lambda}}_i = -(-\frac{1}{\delta t} I_\ell)^T \tilde{\boldsymbol{\lambda}}_{i+1} - \delta t (\tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}), \quad i = N_t - 1, ..., 1 \tag{4.9b}$$

3: The gradient is computed according to formula (3.15):

$$\nabla \hat{\tilde{\mathcal{J}}}(\mathbf{u}_0, ..., \mathbf{u}_{N_t}) = \begin{pmatrix} \delta t \omega M \mathbf{u}_0 \\ \delta t \omega M \mathbf{u}_1 - \tilde{M}^T \tilde{\boldsymbol{\lambda}}_1 \\ \vdots \\ \delta t \omega M \mathbf{u}_{N_t} - \tilde{M}^T \tilde{\boldsymbol{\lambda}}_{N_t} \end{pmatrix} \tag{4.10}$$

Because of the small dimension of the adjoint variable $\tilde{\boldsymbol{\lambda}}_i$, we note that the solution of the linear systems (4.9a)-(4.9b) can be obtained much faster than in the full-order case. In (4.9a) and (4.9b) and the remainder of the paper, I_ℓ denotes the identity matrix of dimensions $\ell \times \ell$. It is important to stress that the Jacobian of the nonlinearity (4.7) has

to be computed in terms of the reduced variable $\tilde{\mathbf{y}}$ and is, hence, of dimension $\ell \times \ell$ as the the following computation shows:

$$\tilde{\mathcal{N}}'(\tilde{\mathbf{y}}) = \frac{d}{d\tilde{\mathbf{y}}}\left(\frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}})^2\right) = \begin{pmatrix} \tilde{B}_{1,1} & \dots & \tilde{B}_{1,m} \\ \vdots & & \vdots \\ \tilde{B}_{\ell,1} & \dots & \tilde{B}_{\ell,m} \end{pmatrix} \cdot \begin{pmatrix} \langle \tilde{\mathbf{y}}, \tilde{F}_1 \rangle \tilde{F}_{1,1} & \dots & \langle \tilde{\mathbf{y}}, \tilde{F}_1 \rangle \tilde{F}_{1,\ell} \\ \vdots & & \vdots \\ \langle \tilde{\mathbf{y}}, \tilde{F}_m \rangle \tilde{F}_{m,1} & \dots & \langle \tilde{\mathbf{y}}. \tilde{F}_m \rangle \tilde{F}_{m,\ell} \end{pmatrix}. \tag{4.11}$$

In (4.11), we denoted by $\tilde{F}_i$ the i-th row of the matrix $\tilde{F}$ and $\tilde{B}_{i,j}, \tilde{F}_{i,j}$ denotes the entry of the respective matrix at position i, j. Here, $\langle \cdot, \cdot \rangle$ stands for the standard scalar product in $\mathbb{R}^\ell$. Note that because of the full dimension of $\mathbf{u}_i$, the gradient in (4.10) is still of dimension $N \cdot N_t \times 1$. Algorithm 4.1 has also been used for the gradient computation that is required for the first-order optimization methods SPG and BFGS.

We next present the efficient computation of the Hessian-times-vector product $\nabla^2 \hat{\tilde{\mathcal{J}}} \cdot \underline{\mathbf{v}}$ using adjoints. Since the Hessian with respect to u is of dimension $N \cdot N_t \times N \cdot N_t$, the corresponding vector used in Algorithm 4.2 is of the following form, $\underline{\mathbf{v}} := (\mathbf{v}_0, ..., \mathbf{v}_{N_t})^T \in \mathbb{R}^{(N \cdot N_t) \times 1}$.

Algorithm 4.2 Algorithm 3.3 applied to the reduced Burgers' model

1: We assume that we have already computed $\tilde{\mathbf{y}}_0, ..., \tilde{\mathbf{y}}_{N_t}, \mathbf{u}_0, ..., \mathbf{u}_{N_t}, \tilde{\boldsymbol{\lambda}}_1, ..., \tilde{\boldsymbol{\lambda}}_{N_t}$ in Algorithm 4.1

2: Equation (3.23) reads:

$$\tilde{\mathbf{w}}_0 = 0 \tag{4.12a}$$

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_{i+1}) + \nu \tilde{C}\right) \tilde{\mathbf{w}}_{i+1} = -(-\frac{1}{\delta t} I_\ell)\tilde{\mathbf{w}}_i - \tilde{M}\mathbf{v}_{i+1}, \quad i = 0, ..., N_t - 1 \tag{4.12b}$$

3: Equation (3.24) reads:

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_{N_t}) + \nu \tilde{C}\right)^T \tilde{\mathbf{p}}_{N_t} = \delta t I_\ell \tilde{\mathbf{w}}_{N_t} + \left(\tilde{\boldsymbol{\lambda}}_{N_t}^T \tilde{\mathcal{N}}(\tilde{\mathbf{y}}_{N_t})\right)'' \tilde{\mathbf{w}}_{N_t} \tag{4.13a}$$

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_i) + \nu \tilde{C}\right)^T \tilde{\mathbf{p}}_i = -(-\frac{1}{\delta t} I_\ell)^T \tilde{\mathbf{p}}_{i+1} + \delta t I_\ell \tilde{\mathbf{w}}_i + \left(\tilde{\boldsymbol{\lambda}}_i^T \tilde{\mathcal{N}}(\tilde{\mathbf{y}}_i)\right)'' \tilde{\mathbf{w}}_i, \quad i = N_t - 1, ..., 1 \tag{4.13b}$$

4: The Hessian times a vector $\underline{\mathbf{v}}$ is computed according to formula (3.20):

$$\nabla^2 \hat{\tilde{\mathcal{J}}}(\mathbf{u}_0, ..., \mathbf{u}_{N_t})\underline{\mathbf{v}} = \begin{pmatrix} \delta t \omega M \mathbf{v}_0 \\ -\tilde{M}^T \tilde{\mathbf{p}}_1 + \delta t \omega M \mathbf{v}_1 \\ \vdots \\ -\tilde{M}^T \tilde{\mathbf{p}}_{N_t} + \delta t \omega M \mathbf{v}_{N_t} \end{pmatrix} \tag{4.14}$$

Note that the second partial derivative of the nonlinear term $\tilde{\boldsymbol{\lambda}}^T\tilde{\mathcal{N}}(\tilde{\mathbf{y}})$ with respect to the reduced variable $\tilde{\mathbf{y}}$ in step 3 of the above algorithm is given by:

$$\begin{aligned}\frac{d^2}{d\tilde{\mathbf{y}}^2}\left(\tilde{\boldsymbol{\lambda}}^T\tilde{\mathcal{N}}(\tilde{\mathbf{y}})\right)\tilde{\mathbf{w}} &= \frac{d^2}{d\tilde{\mathbf{y}}^2}\left(\tilde{\boldsymbol{\lambda}}^T(\frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}})^2)\right)\tilde{\mathbf{w}} = \frac{d^2}{d\tilde{\mathbf{y}}^2}\left(\sum_{k=1}^{\ell}\lambda_k\frac{1}{2}\sum_{j=1}^{m}\tilde{B}_{k,j}\left(\sum_{i=1}^{\ell}\tilde{F}_{j,i}\tilde{y}_i\right)^2\right)\tilde{\mathbf{w}} \\ &= \frac{d}{d\tilde{\mathbf{y}}}\begin{pmatrix}\sum_{k=1}^{\ell}\lambda_k\sum_{j=1}^{m}\tilde{B}_{k,j}\sum_{i=1}^{\ell}\tilde{F}_{j,i}\tilde{y}_i\cdot\tilde{F}_{j,1}\\ \vdots\\ \sum_{k=1}^{\ell}\lambda_k\sum_{j=1}^{m}\tilde{B}_{k,j}\sum_{i=1}^{\ell}\tilde{F}_{j,i}\tilde{y}_i\cdot\tilde{F}_{j,\ell}\end{pmatrix}\tilde{\mathbf{w}} \\ &= \underbrace{\begin{pmatrix}\sum_{k=1}^{\ell}\lambda_k\sum_{j=1}^{m}\tilde{B}_{k,j}\tilde{F}_{j,1}\tilde{F}_{j,1} & \dots & \sum_{k=1}^{\ell}\lambda_k\sum_{j=1}^{m}\tilde{B}_{k,j}\tilde{F}_{j,\ell}\tilde{F}_{j,1}\\ \vdots & & \vdots\\ \sum_{k=1}^{\ell}\lambda_k\sum_{j=1}^{m}\tilde{B}_{k,j}\tilde{F}_{j,1}\tilde{F}_{j,\ell} & \dots & \sum_{k=1}^{\ell}\lambda_k\sum_{j=1}^{m}\tilde{B}_{k,j}\tilde{F}_{j,\ell}\tilde{F}_{j,\ell}\end{pmatrix}}_{\in\mathbb{R}^{\ell\times\ell}}\tilde{\mathbf{w}}\end{aligned}$$

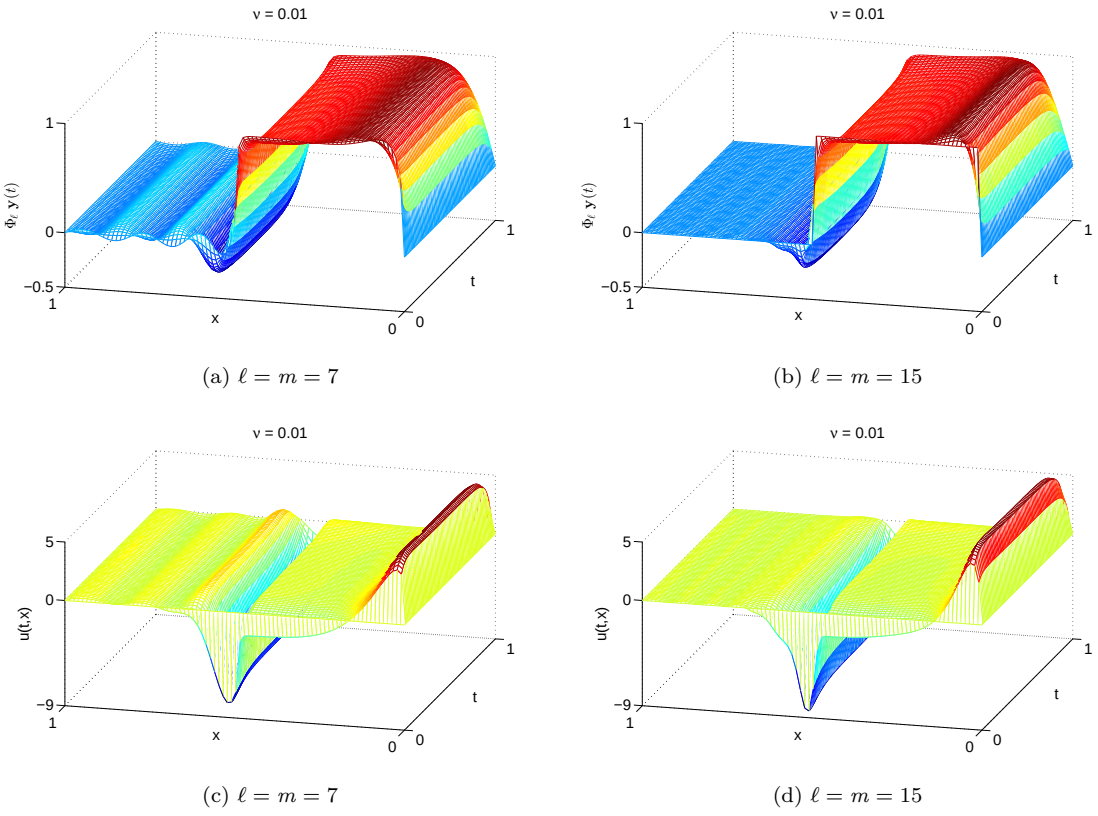
Due to the dimensions of the reduced variables, it is natural that the above matrix-times-vector product only depends on the dimension ℓ. The matrix is not constant in time because the adjoint variable depends on time and can, thus, not be pre-computed. An efficient implementation is therefore necessary in order to obtain a computational speedup if $\ell \ll N$.

Algorithm 4.3 gives an overview of the optimization algorithm that has been developed for the optimal control of the POD-DEIM reduced model of Burgers' equation. Algorithm 4.3 is an iterative procedure to update the reduced model aiming to improve the approximation to the full-order model. The procedure is as follows: In order to obtain snapshots, it is necessary to solve the full-order model corresponding to an initial control once. From snapshots of the full-order solution, we can derive the reduced model and solve the *reduced* optimization problem for an optimal control. The reduced model can be improved in an iterative process by obtaining new snapshots of the state corresponding to the current optimal control. Then, a new POD-DEIM model is derived from the new snapshots and the same optimization algorithm is applied to the improved reduced model. This can be repeated until a certain stopping criterion is satisfied. In Algorithm 4.3, we improve the reduced model as long as the obtained optimal state is close enough to the desired state z measured in the L_2-norm. Another approach for stopping could be that the reduced objective function reaches the value of the objective function of the full-order model evaluated at the optimal state. This, however, is only possible if the full-order optimal control has already been solved which we want to avoid in our implementation.

Algorithm 4.3 Optimal control: Iterative improvement of the reduced model

1: Set initial control $u^{(0)} = 0$, and $K = 0$, $\varepsilon_z \in \mathbb{R}_+$, `max_iter` $\in \mathbb{N}$
2: Solve the full-order Burgers' equation $c(y^{(0)}, u^{(0)})$ for the uncontrolled state $y^{(0)}$ via (3.37)
3: Choose POD dimension ℓ and DEIM-dimension m
4: **while** $\|y^{(K)} - z\|_{L^2([0,L]\times[0,T])} > \varepsilon_z$ **and** $K <$`max_iter` **do**
5: Obtain the POD-DEIM reduced model from snapshots of $y^{(K)}$, i.e. compute Φ_ℓ via (2.18) and $\mathcal{P}$ via Algorithm 2.1
6: Calculate reduced optimal control, $u^{(K+1)} = \operatorname{argmin}_u \tilde{\mathcal{J}}(\tilde{y}(u), u)$ by solving the reduced optimization problem (4.2) with implicit constraint (4.4). Choose one of the optimization algorithms 3.1, 3.4 or 3.5.
7: Solve the reduced Burgers' equation $\tilde{c}(\tilde{y}^{(K+1)}, u^{(K+1)}) = 0$ for $\tilde{y}^{(K+1)}$ via (4.4)
8: Expand $y^{(K+1)} = \Phi_\ell \tilde{y}^{(K+1)}$
9: $K = K + 1$
10: **end while**

In Figure 4.1, we show numerical results for the optimal control of Burgers' equation using a POD-DEIM reduced model of different dimensions ℓ and m and using the Newton-type method of Algorithm 3.1 for solving the optimization problem. We see that due to the low-dimensional state variable, the desired state is not reached with the same accuracy as in the full-order case. This leads to a larger optimal value of the objective function (4.2) as the results in Section 4.4 will show.



(a) $\ell = m = 7$ (b) $\ell = m = 15$

(c) $\ell = m = 7$ (d) $\ell = m = 15$

Figure 4.1: Optimal control of the POD-DEIM reduced Burgers' model using different dimensions.

4.3 Low-dimensional control using individual control points

In order to overcome the dependence of the control on the dimension of the full-order model, we will restrict the control to certain discrete *control points*. In Figure 4.2, we have indicated that we allow to control the system at only three different points in $[0, L]$. This approach requires, of course, some physical inside of the problem in the sense that we need to know at which points it is important to control the state.

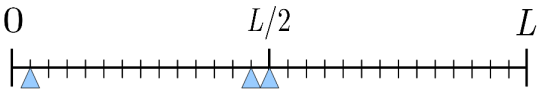


Figure 4.2: Discretization of the interval $[0, L]$ indicating only 3 discrete positions for control.

We can model this approach by writing the control in the following way:

$$\mathbf{u}(t) = \Psi_{n_c}\tilde{\mathbf{u}}(t), \quad \text{with } \tilde{\mathbf{u}}(t) \in \mathbb{R}^{n_c}, \tag{4.15}$$

where $\tilde{\mathbf{u}}(t)$ is low-dimensional and contains only non-zero values of the control at the n_c considered points. The matrix Ψ indicates which entries of the control we want to consider. For the case in Figure 4.2 with $n_c = 3$, we would have:

$$\Psi_3 = \begin{pmatrix} 1 & 0 & 0 \\ & \vdots & \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ & \vdots & \end{pmatrix} \in \mathbb{R}^{N\times 3}.$$

Note that this decomposition corresponds to an ansatz:

$$u(t, x) \approx \sum_{j\in\mathcal{I}_c} u_j(t)\phi_j(x),$$

with an index set $\mathcal{I}_c$ containing n_c indices of the control points. This way, the mass matrix for the control can be derived in a straightforward way but the approach of (4.15) shows the gain in dimension reduction better: When we plug in (4.15) into the discretized objective function (4.2), we obtain:

$$\min_{\tilde{\mathbf{u}}_0,\dots,\tilde{\mathbf{u}}_{N_t}} \tilde{J}(\tilde{\mathbf{y}}_0, \dots, \tilde{\mathbf{y}}_{N_t}, \tilde{\mathbf{u}}_0, \dots, \tilde{\mathbf{u}}_{N_t}) = \min_{\tilde{\mathbf{u}}_1,\dots,\tilde{\mathbf{u}}_{N_t}} \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2}\tilde{\mathbf{y}}_i^T\tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}^T\tilde{\mathbf{y}}_i + \frac{\omega}{2}\tilde{\mathbf{u}}_i^T \underbrace{\Psi^T M\Psi}_{\text{pre-compute}} \tilde{\mathbf{u}}_i \right), \tag{4.16}$$

which does not depend on the full-order dimension N at all because the matrix $\Psi^T M\Psi \in \mathbb{R}^{n_c\times n_c}$. Note also that this matrix can be pre-computed. In the same way, we have for

the discretized reduced Burgers' equation (4.4) the following:

$$\tilde{c}_i(\tilde{\mathbf{y}}_i, \tilde{\mathbf{y}}_{i+1}, \tilde{\mathbf{u}}_{i+1}) \equiv \frac{1}{\delta t}\tilde{\mathbf{y}}_{i+1} - \frac{1}{\delta t}\tilde{\mathbf{y}}_i + \frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}}_{i+1})^2 + \nu\tilde{C}\tilde{\mathbf{y}}_{i+1} - \tilde{\mathbf{f}} - \underbrace{\Phi_\ell^T M \Psi}_{\text{pre-compute}} \tilde{\mathbf{u}}_{i+1} = 0, \tag{4.17}$$

where $i = 0, ..., N_t - 1$ and $\Phi_\ell^T M \Psi \in \mathbb{R}^{\ell \times n_c}$.

In Figure 4.3, we show the numerical result for $n_c = 3$. We see that even a control at only three different control points leads to a reasonable approximation to the desired state. We also present the corresponding control that drives the solution to the POD-DEIM reduced Burgers' equation into the desired state.

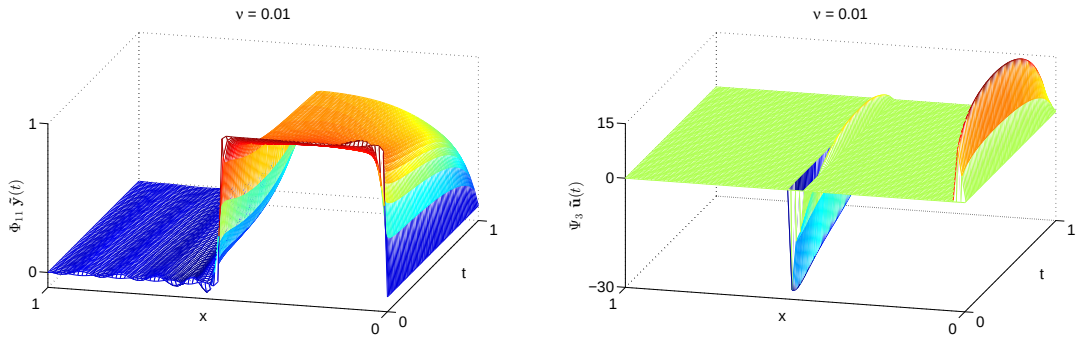


Figure 4.3: Optimal control (right) and corresponding state (left) for $\ell = m = 11$ and $n_c = 3$ control points.

4.4 Performance and error analysis

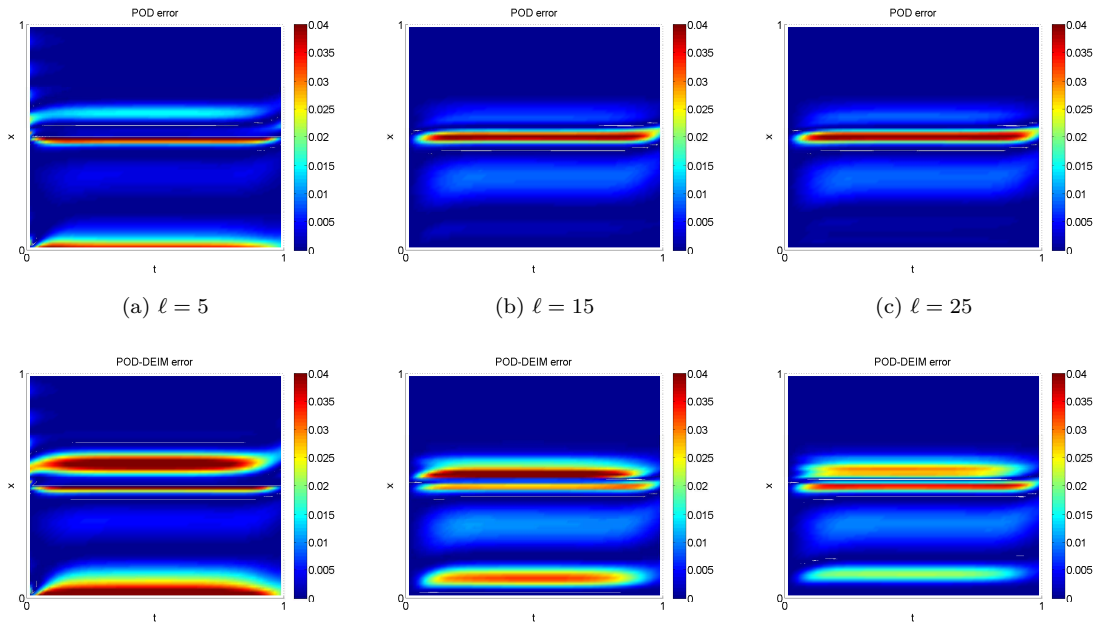
The main goal of this thesis work is to evaluate the performance of POD-DEIM when applied to optimal control of Burgers' equation. In order to compare the results of the optimization using a POD-DEIM reduced model with the optimization based on a reduced model obtained from a pure POD reduction as suggested in [29], it is important to specify identical stopping criteria and tolerances for the respective numerical optimization algorithm. In this section, we present numerical results for three different viscosity parameters, $\nu = \{0.01, 0.001, 0.0001\}$, and for the three different optimization algorithms Newton-type, BFGS and SPG as presented in Section 3.1, 3.2.1 and 3.2.2, respectively. Since all three optimization algorithms are iterative methods, we are able to define a stopping of the optimization when either the change in the objective function is smaller than the tolerance $\varepsilon_{\mathcal{J}}$ or the zero-gradient condition is fulfilled upto a numeric tolerance specified by ε_∇. For the presented results of this section, we used $\varepsilon_{\mathcal{J}} =$ `10e-8` and $\varepsilon_\nabla =$ `10e-9` in the respective Algorithms 3.1, 3.4 or 3.5. In general, it is difficult to design a *fair* comparison between different algorithms. For example, the stopping criterion for the gradient is implemented differently for the SPG method, see line 5 of Algorithm 3.5. Therefore, we also used a criterion which is called *targeting* and which provides comparable results by choosing all parameters in such a way that the final value of the reduced

objective function and the objective function of the full-order model are close. Moreover, we have required the error of the optimal state in the relative L_2-norm to be small.

In order to compare the optimal control of the POD and the POD-DEIM reduced Burgers' model, we are first interested in the *accuracy* of the optimal state obtained by the two reduced models in comparison to the full-order optimal control as described in Section 3.3. Therefore, we choose the viscosity parameter $\nu = 0.01$ which requires a dimension of $N = 80$ for the full-order model, cf. Section 3.3. For a comparison of the optimal state obtained by a POD and POD-DEIM, we choose the DEIM-dimension constant, $m = 15$, and increase the POD dimension ℓ from 3 to 25. In Figure 4.4, we present the distribution of the error which is defined as the squared difference of the final state of the full-order optimization and the final state of the respective optimization of the reduced model,

$$e(t, x) := [y(t, x) - \Phi_\ell \tilde{y}(t)]^2.$$

Figure 4.4 shows the error as a function of time and space for three different POD dimensions, $\ell = \{5, 15, 25\}$. We see that the error is large where the desired state is not smooth, i.e. at the boundary of the step function z. Furthermore, the plot in Figure 4.4 shows that the error decreases as the POD dimension increases. For all three cases considered, the error is generally larger for the POD-DEIM reduced model but it is always of the same order of magnitude.



(a) $\ell = 5$ (b) $\ell = 15$ (c) $\ell = 25$

(d) $\ell = 5$ (e) $\ell = 15$ (f) $\ell = 25$

Figure 4.4: Error distribution of the optimal state obtained by the Newton-type method and a POD and POD-DEIM reduced model.

In Figure 4.5, we present the error in the two reduced optimal sates with respect to the full optimal state not as a function of time and space but calculated in the corresponding L_2-norm. Since the POD reduced model only depends on ℓ, we have again chosen a fixed DEIM-dimension $m = 15$ in order to compare the error in L_2 as a function of ℓ. In Figure 4.5, we see that, as expected, the POD-DEIM error is larger for all considered ℓ. It is also interesting to note that for $\ell > m$, the error of the POD-DEIM optimal state is dominated by the DEIM approximation error and does not decrease further whereas the error of the optimal state obtained from the POD reduced model still decreases.

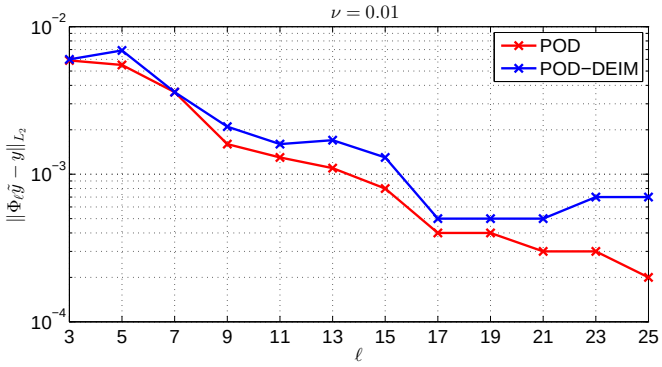


Figure 4.5: Comparison of the L_2-error of the POD and the POD-DEIM approximation when the projection dimensions are increased.

The results presented in Figure 4.4 and Figure 4.5 show that the POD-DEIM reduced optimal control problem (4.2)-(4.4) yields to a comparable optimal solution to a POD-reduced model and the full-order optimal control problem (3.35)-(3.37). We will next consider the application of the Newton-type optimization method of Algorithm 3.1 to the three different models with respect to computational time. Therefore, we have calculated the optimal solution for three different values of ν in Table 4.1. The measurements reported in Table 4.1 are:

- $N/\ell/m$ - The (spatial) dimension of the full-order, POD and POD-DEIM reduced model, respectively. In the case of POD-DEIM we present both reduced dimensions as the tuple (ℓ, m)
- $t_{opt}[s]$ - This is the time needed for the considered optimization algorithm. In case of the reduced optimal control presented in Algorithm 4.3, this also includes the time which is necessary for building the reduced model, i.e. the pre-computation of the POD basis and DEIM-indices.
- $\mathcal{J}(y^*, u^*)$ - The value of the (reduced) objective function after convergence of the optimization iteration.
- $\bar{e}$ - The relative error as defined in (2.29), computed for the optimal state.
- S_P - The speedup in computation time.

	$\nu = 0.01$			$\nu = 0.001$			$\nu = 0.0001$		
	Full	POD	DEIM	Full	POD	DEIM	Full	POD	DEIM
$N/\ell/m$	80	9	$(9, 25)$	200	11	$(11, 25)$	800	15	$(15, 25)$
$t_{opt}[s]$	4.83	4.13	3.22	23.1	6.38	5.25	1,865.8	23.61	18.42
$\mathcal{J}(y^*, u^*)$	0.0241	0.0262	0.0233	0.0206	0.0198	0.0200	0.0202	0.0191	0.0233
$\bar{e}$	-	0.0097	0.0141	-	0.0194	0.0192	-	0.0255	0.0238
S_P	-	1.35	1.9	-	3.7	4.4	-	**79.0**	**101.3**

Table 4.1: Results of the Newton-type optimization method 3.1 for $\nu = \{0.01, 0.001, 0.0001\}$.

The results in Table 4.1 show that for all values of ν, the POD as well as the POD-DEIM reduced optimal control problem lead to an optimal solution such that the value of the objective function $\mathcal{J}(y^*, u^*)$ is similar. We further note that the optimal state of the reduced model is a good approximation for the optimal state of the full-order optimization problem which can be seen by a small relative error, $\bar{e} \in \mathcal{O}(10^{-2})$, for all considered settings. The most important conclusion from Table 4.1 is that for all three values of ν, the speedup of the POD-DEIM reduced model is larger than the speedup of the POD-reduced model. In the case of $\nu = 0.0001$, the large full-order dimension N that is required for numerical stability even leads to a computational speedup of ~ 80 for the POD-reduced optimization compared to a speedup of more than 100 for the POD-DEIM reduced optimal control problem.

In the next numerical test we consider the same setting as before but we use the approach of a low-dimensional control as described in Section 4.3. For the results presented in Table 4.2, we used $n_c = 3$ control points at the positions as indicated in Figure 4.2. Since for the time discretization of the interval $[0, 1]$ we needed to choose $N_t = 80$ equidistant time-steps, the choice of $n_c = 3$ has the consequence that the optimization in (4.16) can be formulated in the unknown $\underline{\tilde{\mathbf{u}}} := [\tilde{\mathbf{u}}_1, ..., \tilde{\mathbf{u}}_{N_t}]^T$ which is a vector of dimension 240. In Table 4.2, we first note that the value of the objective function is higher for all models and all values of ν compared to Table 4.1. This is expected since we only allow the control to be different from zero at three discrete positions and, therefore, it is not possible to drive the solution of Burgers' equation into the desired state with the same accuracy as before. At the same time, we see that the computational cost for all presented simulations is much less than in Table 4.1 due to the lower dimension of the respective optimization problem. Again, the approximation of the optimal state obtained when using the reduced order optimization is of good quality as the small relative error indicates, $\bar{e} \in \mathcal{O}(10^{-2})$. The speedup obtained by the dimension reduction of POD and POD-DEIM is generally smaller in this case compared to the previous experiment but, again, we can see an improvement obtained by the application of DEIM.

	$\nu = 0.01$			$\nu = 0.001$			$\nu = 0.0001$		
	Full	POD	DEIM	Full	POD	DEIM	Full	POD	DEIM
$N/\ell/m$	80	9	$(9, 25)$	200	11	$(11, 25)$	800	15	$(15, 25)$
$t_{opt}[s]$	3.68	2.41	1.90	5.44	2.6	1.89	117.32	8.17	6.10
$\mathcal{J}(y^*, u^*)$	0.0300	0.0394	0.0396	0.0348	0.0439	0.0371	0.0763	0.0865	0.0893
$\bar{e}$	-	0.0132	0.0142	-	0.0127	0.0187	-	0.0204	0.0304
S_P	-	1.6	1.9	-	2.0	2.9	-	14.4	19.2

Table 4.2: Results of the Newton-type optimization method 3.1 using a low-dimensional control with $n_c = 3$ and $\nu = \{0.01, 0.001, 0.0001\}$.

As a last numerical test, we want to compare the second-order Newton-type method 3.1 to the first-order methods BFGS and SPG, as presented in Section 3.2.1 and 3.2.2, respectively. In our experiments, the same stopping criteria have been used for all three iterative methods. In Table 4.3, we present the results of the three optimization algorithms for the optimal control of Burgers' equation and the respective results for the POD/DEIM reduced models when the viscosity parameter is $\nu = 0.0001$ and a control is only possible at $n_c = 3$ control points. We have chosen the smallest ν because for this case we have previously seen the largest speedup. Table 4.3 shows almost the same speedup for all three optimization algorithms when compared to the full-order solution. Note that the results in Table 4.3 show that the value of the objective function was slightly lower for the full-order model in all three cases which indicates that the reduced models do not reach entirely the optimal state of the full model.

	Newton-type			BFGS			SPG		
	Full	POD	DEIM	Full	POD	DEIM	Full	POD	DEIM
$N/\ell/m$	800	15	$(15, 25)$	800	15	$(15, 25)$	800	15	$(15, 25)$
$t_{opt}[s]$	117.32	8.17	6.10	294.90	15.64	14.38	123.00	7.61	16.11
$\mathcal{J}(y^*, u^*)$	0.0763	0.0865	0.0893	0.0840	0.0879	0.0857	0.0763	0.0861	0.0878
$\bar{e}$	-	0.0204	0.0304	-	0.0226	0.0247	-	0.0218	0.0355
S_P	-	14.4	19.2	-	18.3	20.5	-	16.3	7.6

Table 4.3: Results of three different optimization algorithms and $\nu = 0.0001, n_c = 3$.

We note in Table 4.3 that the speedup for the POD-DEIM reduced model is small when the SPG method is applied. This can be explained when taken into account the number of evaluations of the cost function (4.2) and its gradient in Table 4.4. We see that in order to converge within the same precision, the SPG method needs 52 evaluations of the objective function when POD-DEIM has been applied. This analysis explains the relatively poor speedup of 7.6 for POD-DEIM compared to a speedup of 16.3 of POD. The much larger number of required function evaluations overtops the speedup of a single iteration.

	BFGS			SPG		
	Full	POD	DEIM	Full	POD	DEIM
$\#\mathcal{J}$	65	38	40	17	13	52
$\#\nabla\mathcal{J}$	65	38	40	16	12	12

Table 4.4: Number of evaluations of the cost function and the gradient for the first-order methods in the setting of Table 4.3.

Moreover, the SPG method has been considered for three different values of ν and the full-dimensional control that has been constrained in the following way, $-2 \leq u \leq 2$. In Section 3.2.2, we have defined a projector that allows to include this bound constraint in an easy way. The numerical results presented in Table 4.5 show a larger value of the objective function at the optimal state compared to the results obtained by the Newton-type method presented in Table 4.1. This shows that the restriction of the control to be within a certain range influences the quality of the obtained optimal state. Again, the performed numerical tests show that the reduced models approximate the optimal state of the full-order model well since $\bar{e} \in \mathcal{O}(10^{-2})$. For the smallest value of $\nu = 0.0001$ which corresponds to a full-order dimension of $N = 800$, we observe a speedup of the SPG method of 3.6 when POD is used to solve the constraining Burgers' equation and a speedup of 8.8 in the case of a model order reduction by POD-DEIM while the value of the objective function for both reduced models has been almost the same.

	$\nu = 0.01$			$\nu = 0.001$			$\nu = 0.0001$		
	Full	POD	DEIM	Full	POD	DEIM	Full	POD	DEIM
$N/\ell/m$	80	9	$(9, 25)$	200	11	$(11, 25)$	800	15	$(15, 25)$
$t_{opt}[s]$	6.61	4.02	3.77	12.55	6.61	5.99	193.40	53.16	22.05
$\mathcal{J}(y^*, u^*)$	0.0367	0.0399	0.0332	0.0347	0.0397	0.0378	0.0339	0.0357	0.0367
$\bar{e}$	-	0.0193	0.0250	-	0.0255	0.0249	-	0.0238	0.0248
S_P	-	1.6	1.8	-	1.9	2.1	-	3.6	8.8

Table 4.5: Results of the SPG method and a bounded control $-2 \leq u \leq 2$.

Chapter 5

Summary and Future research

5.1 Overview and main results

In this project, we have applied the nonlinear MOR methods POD and DEIM to the optimal control of Burgers' equation. Therefore, the optimal control problem (3.32)-(3.33) has been discretized in space using a finite element approach and integrated in time using an implicit Euler scheme. The resulting discretized equations are an implicitly constrained optimization problem that has been solved using the three different optimization algorithms Newton-type method, BFGS and SPG. The main theoretical contribution of this work is the derivation of the adjoints equation in Algorithm 3.2 and Algorithm 3.3and its application to the full-order and the POD-DEIM reduced Burgers' model in the Algorithms 3.6-3.7 and the Algorithms 4.1-4.2, respectively. The discretized optimal control problem for Burgers' equation has been implemented in MATLAB and the three different optimization algorithms have been tested with the full-order model, the POD-reduced model and the POD-DEIM-reduced model.

One main goal of this thesis was to apply DEIM to a POD-reduced model of Burgers' equation and compare the results of both reduced models with respect to accuracy of the dynamical behavior and computational speedup. The theoretical derivations in Section 2.2 have shown that the POD-DEIM reduced Burgers' model is completely independent of the size of the full-order model N whereas in the case of a purely POD-reduced model, the nonlinear term of Burgers' equation still depends on N. The numerical tests in Section 2.3.3 have shown that the larger the original dimension N is, the more speedup we obtain from DEIM. For $N = 800$ and a viscosity parameter of $\nu = 0.0001$ we have shown that the speedup of POD-DEIM is about 52 times the full-order model while a POD-reduced model only leads to a speedup of 16. Further numerical tests have proven the independence of the POD-DEIM model of N.

In Section 3 and 4, the DEIM method has been used for the optimal control of Burgers' equation. In this thesis, we have compared the optimal control of Burgers' equation using

the three different optimization algorithms BFGS, SPG and a Newton-type method. All of them have been applied to the optimal control of the full-order model as well as a POD and a POD-DEIM reduced model. Numerical tests have been shown that the optimal state of the reduced models is close to the optimal state of the full-order model measured in the L_2-norm. At the same time we have seen for $\nu = 0.0001$ and $N = 800$ that a computational speedup of more than 100 times for the POD-DEIM reduced optimal control is possible whereas the POD-reduced model only let to a speedup of ~ 80. Additionally, the SPG method has been used to introduce so-called bound constraints on the control of the full-order and the respective reduced models. Numerical tests in Section 4.4 have shown that also for this case, the optimization of the POD-DEIM reduced model leads to a significant speedup and an optimal state that is close to the optimal state of the full-order model.

5.2 Outlook on future research questions

At this point, we would like to present some directions for future research that might build up on this thesis work.

A priori dimension reduction of the control variable

In Section 4.3, we have derived a POD-DEIM reduced optimal control problem that uses only a low-dimensional control in order to drive the solution of Burgers' equation into the desired state. This has been done by defining discrete control points at which the control is allowed to be different from zero. Numerical tests in Section 4.4 have shown that this approach leads to a tremendous reduction in computational time even for the optimization of the full-order Burgers' model. It has been shown as well that the choice of the position of the control points is crucial and requires some *physical inside* of the considered optimization problem. Since it is in general not clear which positions are optimal for the control points, it would be desirable to develop an algorithmic approach that reduces the dimension of the control and, hence, the dimension of the optimization problem and at the same time leads to an optimal state close to the desired state. A different and more general approach for the dimension reduction of the control variable might be to use the POD basis of the state variable. For the dimension reduction of the state variable we have used an approximation of the form,

$$\mathbf{y}(t) \approx \Phi_\ell \tilde{\mathbf{y}}(t),$$

where $\Phi_\ell = [\varphi_1, ..., \varphi_\ell]$ consists columnwise of the POD basis and, thus, the state variable can be expressed as $\mathbf{y}(t) \approx \sum_{i=1}^{\ell} \varphi_i \tilde{y}_i(t)$. In a future work, one might consider a similar approach also for the control variable,

$$\mathbf{u}(t) \approx \Phi_\ell \tilde{\mathbf{u}}(t) = \sum_{i=1}^{\ell} \varphi_i \tilde{u}_i(t), \tag{5.1}$$

where $\tilde{\mathbf{u}}(t) = [\tilde{u}_1, ..., \tilde{u}_\ell]^T$ is the reduced control. Note that we suggest to use the same basis $\{\varphi_i\}_{i=1}^{\ell}$ for the low-dimension expression of the control variable. In Section 2 we have argued that the POD basis captures well the dynamical behavior of the state variable. Therefore, the quality of the approximation (5.1) might be of interest in future research.

Optimal control of Burgers' equation in 2D/3D

In order to obtain a larger factor for the computational speedup of DEIM, the work of [11, 45] has shown that a problem defined for the physical domain $\Omega \subset \mathbb{R}^d$, for $d = 2, 3$, shows in general more potential for model order reduction. In [11], the authors present the application of DEIM to the numerical simulation of a two-dimensional model for miscible fingering in porous media. Therein, it is presented that the POD-DEIM reduced model leads to a reduction of the computational time by a factor of $\mathcal{O}(1000)$. Since it has already been shown in [19, 42] that the optimal control for 2 or 3-dimensional models is possible, a further improvement of this work would be to extend the optimal control of Burgers' equation to higher dimensions and evaluate the computational gain of POD-DEIM. It is expected that this leads to a larger speedup for the reduced model.

Educated choice of the reduction dimensions ℓ and m

During the numerical tests in Chapter 4, we have seen that it is in general not trivial to choose suitable values for the reduced dimensions ℓ and m. Especially when the SPG method has been used for the solution of the optimal control problem, we have seen in Section 4.4 that it might happen that more function and gradient evaluations for the POD-DEIM reduced model are necessary which has a bad influence on the computational speedup even though a single optimization iteration is much faster. For the test calculations presented in Table 4.3, we have not been able to choose the DEIM dimension in such a way that this behavior does not appear. It is therefore desirable to evaluate the choice of ℓ and m in more detail and derive an a priori estimate for both dimensions such that the convergence behavior of the SPG method is optimal.

Development of a reduced model for optimal flow control

Burgers' equation is an important model equation in the field of computational fluid dynamics (CFD) because the structure of the nonlinear term is similar to the nonlinearity in the Navier-Stokes equations. Since this thesis has proven that POD-DEIM leads to a good approximation of the dynamical behavior of Burgers' equation, one might conclude that the same mathematical methods can be applied in order to derive a reduced model of the Navier-Stokes equations. In [24, 42], the Navier-Stokes equations have been considered for the optimal control of two-dimensional flow. The implementation of MOR techniques to the software used in [24, 42] might be a major step in order to reduce the huge computational work that is required in CFD applications. The theoretical considerations for

Burgers' equation presented in this thesis might be very helpful in order to derive a POD-DEIM reduced model for the Navier-Stokes equations due to the similar nonlinear terms of both models.

Bibliography

[1] A. C. Antoulas. *Approximation of Large-Scale Dynamical Systems (Advances in Design and Control)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005.

[2] J. Atwell. *Proper Orthogonal Decomposition for Reduced Order Control of Partial Differential Equations*. PhD thesis, Virginia Polytechnic Institute and State University, 2000.

[3] Z. Bai and D. Skoogh. Krylov Subspace Techniques for Reduced-Order Modeling of Nonlinear Dynamical Systems. *Appl. Numer. Math*, 43:9–44, 2002.

[4] G. Bärwolff. *Numerik für Ingenieure, Physiker und Informatiker: für Bachelor und Diplom*. Für Bachelor und Diplom. Spektrum Akademischer Verlag, 2006.

[5] M. Baumann. Model order reduction for nonlinear dynamical systems. *Literature Study at TU Delft*, 2013.

[6] A. D. Belegundu and T. R. Chandrupatla. *Optimization Concepts and applications in Engineering*. Prentice Hall, 2003.

[7] E. G. Birgin, J. M. Martinez, and M. Raydan. Nonmonotone spectral projected gradient methods on convex sets. *SIAM J. Optim.*, 10:1196–1211, 2000.

[8] E. G. Birgin, J. M. Martínez, and M. Raydan. Spectral projected gradient methods: Review and perspectives. *Journal of Statistical Software*, 2012.

[9] A.E. Bryson and Y.C. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Halsted Press book'. Hemisphere Publishing Company, 1975.

[10] S. Chaturantabut. *Nonlinear Model Reduction via Discrete Empirical Interpolation*. PhD thesis, Rice University, 2011.

[11] S. Chaturantabut and D. Sorensen. Application of POD and DEIM on dimension reduction of non-linear miscible viscous fingering in porous media. *Mathematical and Computer Modelling of Dynamical Systems*, 17:337–353, 2010.

[12] S. Chaturantabut and D. Sorensen. Nonlinear Model Reduction via Discrete Empirical Interpolation. *SIAM J. Sci. Comput.*, 32:2737–2764, 2010.

[13] Y. Chen and J. White. A Qaudratic Method for Nonlinear Model Order Reduction. URL: `http://www.physics.purdue.edu/quantum/files/chen_mor_icmsm2000.pdf`.

[14] A. J. Chorin and J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Springer New York/Berlin/Heidelberg/Tokyo, 1979.

[15] A.J. Chorin and J.E. Marsden. *A Mathematical Introduction to Fluid Mechanics*. Texts in Applied Mathematics Series. Springer London, Limited, 2012.

[16] J.E. Dennis and R.B. Schnabel. *Numerical Methods for Nonlinear Equations and Unconstrained Optimization*. SIAM, Philadelphia, 1996.

[17] K. Eriksson, D. Estep, P. Hansbo, and C. Johnson. *Computational Differential Equations*. Studentlitteratur, 2008.

[18] P.M. Fitzsimons and C. Rui. Determining low dimensional models of distributed systems. *Advances in Robust and Nonlinear Control Systems*, 53, 1993.

[19] J. Ghiglieri and S. Ulbrich. Optimal Flow Control Based on POD and MPC and an Application to the Cancellation of Tollmien-Schlichting Waves. *Optimization Methods and Software*, 00(00):1–34, January 2012.

[20] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996.

[21] J. Haslinger and R. Mäkinen. *Introduction to Shape Optimization: Theory, Approximation and Computation*. Society for Industrial and Applied Mathematic, 2003.

[22] S.B. Hazra. *Large-scale PDE-constrained Optimization in Applications*. Lecture Notes in Applied and Computational Mechanics, 49. Springer Berlin Heidelberg, 2010.

[23] M. Heinkenschloss. Numerical solution of implicitly constrained optimization problems. Technical report, Department of Computational and Applied Mathematics, Rice University, 2008.

[24] L. Henning, D. Kuzmin, V. Mehrmann, M. Schmidt, A. Sokolov, and S. Turek. Flow Control on the Basis of a FEATFLOW-MATLAB Coupling. *Notes on Numerical Fluid Mechanics and Multidisciplinary Design*, 95:325–338, 2007.

[25] Y. Jaluria and K.K.E. Torrance. *Computational heat transfer*. Series in Computational and Physical Processes in Mechanics and Thermal Sciences Series. Taylor & Francis Group, 2003.

[26] I.T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, 2002.

[27] C.T. Kelley. *Iterative Methods for Optimization*. SIAM, Philadelphia, 1999.

[28] G. Kerschen, J.-C. Golinval, A. Vakakis, and L. Bergman. The Method of Proper Orthogonal Decomposition for Dynamical Characterization and Order Reduction of Mechanical Systems: An Overview. *Nonlinear Dynamics*, 41:147–169, 2005.

[29] K. Kunisch and S. Volkwein. Control of the Burgers Equation by a Reduced-Order Approach Using Proper Orthogonal Decomposition. *Journal of Optimization Theory and Applications*, 102:345–371, 1999.

[30] S. Lall, J.E. Marsden, and S. Glavaški. A subspace approach to balanced truncation for model order reduction of nonlinear control systems. *International Journal of Robust and Nonlinear Control*, 12:519–535, 2002.

[31] G. Leugering, S. Engell, and A. Griewank. *Constrained optimization and optimal control for partial differential equations*. International series of numerical mathematics. Springer Basel, 2012.

[32] J.L. Lions. *Optimal control of systems governed by partial differential equations*. Grundlehren der mathematischen Wissenschaften. Springer-Verlag, 1971.

[33] V. Mehrmann. Kontrolltheorie. *Lecture Notes, Technical University of Berlin*, 2004.

[34] B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Oxford University Press, 2001.

[35] H. B. Nielsen. IMMOPTIBOX. A Matlab toolbox for optimization and data fitting. Informatics and Mathematical Modelling, Technical University of Denmark, DTU, 2005. `http://www.imm.dtu.dk/~hbn/immoptibox`.

[36] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer Verlag, Berlin, Heidelberg, New York, second edition, 2006.

[37] A.F. Peterson, S.L. Ray, and R. Mittra. *Computational Methods for Electromagnetics*. IEEE Press Series on Electromagnetic Wave Theory. Wiley, 1998.

[38] J.W. Polderman and J.C. Willems. *Introduction to Mathematical Systems Theory: A Behavioral Approach*. Texts in Applied Mathematics Series. Springer-Verlag, 1998.

[39] S. S. Rao. *Engineering Optimization: Theory and Practice*. Wiley, fourth edition, 2009.

[40] T. Reis. Model order reduction. *Lecture notes, University Hamburg*, 2012.

[41] J. C. De Los Reyes and K. Kunisch. A comparison of algorithms for control constrained optimal control of the Burgers equation. *CALCOLO*, 41:203–225, 2004.

[42] J. C. De Los Reyes and F. Tröltzsch. Optimal control of the stationary Navier-Stokes equations with mixed control-state constraints. *SIAM Journal on Control and Optimization*, 46:604–629, 2007.

[43] P. Sarma, W.H. Chen, L.J. Durlofsky, and K. Aziz. Production Optimization with Adjoint Models under Nonlinear Control-State Path Inequality Constraints. *SPE Reservoir Evaluation & Engineering*, 11(2):326–339, April 2008.

[44] W. H. A. Schilders, H.A. van der Vorst, and J. Rommes. *Model Order Reduction*. Springer, 2008.

[45] R. Stefanescu and I. M. Navon. POD/DEIM nonlinear model order reduction of an ADI implicit shallow water equations model. *Journal of Computational Physics*, 237:95–114, 2013.

[46] G. Strang. *Introduction to Applied Mathematics*. Wellesley-Cambridge Press, 1986.

[47] A. Taflove and S.C. Hagness. *Computational Electrodynamics: The Finite- Difference Time- Domain Method*. The Artech House antenna and propagation library. Artech House, Incorporated, 2005.

[48] L.N. Trefethen and III David Bau. *Numerical Linear Algebra*. Miscellaneous Bks. Society for Industrial and Applied Mathematics, 1997.

[49] H.L. Trentelman, A.A. Stoorvogel, and M. Hautus. *Control theory for linear systems*. Springer London, 2001.

[50] F. Tröltzsch. *Optimal Control of Partial Differential Equations: Theory, Methods, and Applications*. Graduate Studies in Mathematics. American Mathematical Society, 2010.

[51] T. van den Boom and B. De Schutter. *Optimization in Systems and Control*. Lecture Notes for the Course SC4091, TU Delft, 2012.

[52] S. Volkwein. Proper orthogonal decomposition: Applications in optimization and control. Lecture notes, University of Konstanz, 2008. URL : `http://www.math.uni-konstanz.de/numerik/personen/volkwein/teaching/Lecture-Notes-Volkwein.pdf`.

[53] S. Volkwein. Model reduction using proper orthogonal decomposition. Lecture notes, University of Konstanz, 2011. URL: `http://www.math.uni-konstanz.de/numerik/personen/volkwein/teaching/POD-Vorlesung.pdf`.

[54] P. Wesseling. *Principles of Computational Fluid Dynamics.* Springer Berlin/Heidelberg/New York, 2001.

Appendices

Appendix A

Numerical solution of Burgers' equation

A.1 Spacial discretization via the finite element method

We consider Burgers' equation (A.1) together with homogeneous Dirichlet boundary conditions (A.2) and initial value (A.3) given by the function $y_0(x)$,

$$y_t + \left(\frac{1}{2}y^2 - \nu y_x\right)_x = f, \tag{A.1}$$

$$y(t,0) = y(t,L) = 0, \tag{A.2}$$

$$y(0,x) = y_0(x). \tag{A.3}$$

When we define the spatial grid as $\{0 = x_0, ..., x_{N+1} = L\}$ with constant step size h, the following FEM ansatz,

$$y(t,x) \approx \sum_{i=1}^{N} y_i(t)\phi_i(x), \tag{A.4}$$

implicitly fulfills the boundary conditions (A.2). As test functions, the following *hat functions* as proposed, for instance, in [17] have been used:

$$\phi_i(x) = \begin{cases} \frac{x - x_{i-1}}{h}, & \text{for } x \in [x_{i-1}, x_i], \\ \frac{x_{i+1} - x}{h}, & \text{for } x \in [x_i, x_{i+1}], \\ 0, & \text{otherwise.} \end{cases}$$

In order to derive the weak form of (A.1), let us first assume that a source function $f \neq 0$ is given. In the FEM-Galerkin method we then multiply (A.1) by the test function ϕ_j and integrate over the spatial domain $[0, L]$:

$$\begin{aligned} \int_0^L y_t(t,x)\phi_j(x)\,dx &= -\frac{1}{2}\int_0^L \left(y^2(t,x)\right)_x \phi_j(x)\,dx + \nu\int_0^L \left(y(t,x)\right)_{xx}\phi_j(x)\,dx + \int_0^L f(x)\phi_j(x)\,dx \\ &= -\frac{1}{2}\int_0^L \left(y^2(t,x)\right)_x \phi_j(x)\,dx - \nu\int_0^L y_x(t,x)\left(\phi_j(x)\right)_x dx + \int_0^L f(x)\phi_j(x)\,dx, \end{aligned}$$

using integration by parts and the homogeneous Dirichlet boundary conditions. We now plug in the approximation (A.4) and assume further $y^2(t,x) \approx \sum_{i=1}^N y_i^2(t)\phi_i(x)$:

$$\int_0^L \sum_{i=1}^N \dot{y}_i(t)\phi_i(x)\ \phi_j(x)dx = -\frac{1}{2}\int_0^L \sum_{i=1}^N y_i^2(t)(\phi_i(x))_x\ \phi_j(x)dx \\ -\nu\int_0^L \sum_{i=1}^N y_i(t)(\phi_i(x))_x\ (\phi_j(x))_x dx + \int_0^L f(x)\phi_j(x)dx,$$

which is equivalent to

$$\sum_{i=1}^N \dot{y}_i(t)\underbrace{\int_0^L \phi_i(x)\phi_j(x)dx}_{=:M_{i,j}} = -\frac{1}{2}\sum_{i=1}^N y_i^2(t)\underbrace{\int_0^L (\phi_i(x))_x\ \phi_j(x)dx}_{=:B_{i,j}} - \\ \nu\sum_{i=1}^N y_i(t)\underbrace{\int_0^L (\phi_i(x))_x\ (\phi_j(x))_x dx}_{=:C_{i,j}} + \int_0^L f(x)\phi_j(x)dx.$$

It is important to note that the matrices M, B, C are constant in time and their entries consist of polynomials which due to the fact that they are defined as hat function mostly cancel out. The matrices can be pre-computed and are tridiagonal:

$$M = \frac{h}{6}\begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 4 & 1 \\ & & & 1 & 4 \end{bmatrix}, B = \begin{bmatrix} 0 & \frac{1}{2} & & & \\ -\frac{1}{2} & 0 & \frac{1}{2} & & \\ & \ddots & \ddots & \ddots & \\ & & -\frac{1}{2} & 0 & \frac{1}{2} \\ & & & -\frac{1}{2} & 0 \end{bmatrix}, C = \frac{1}{h}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}.$$

For the source term, we assumed a function $f(x)$ that does not depend on time and, thus, this vector can be pre-computed as well. Taking into account that the linear ansatz function are only non-zero at two cells, the Trapezoidal rule [4] yields:

$$\int_0^L f(x)\phi_j(x)dx = \int_{x_{j-1}}^{x_j} f(x)\phi_j(x)dx + \int_{x_j}^{x_{j+1}} f(x)\phi_j(x)dx \approx \frac{h}{2}f(x_j) + \frac{h}{2}f(x_j) = hf(x_j).$$

In order to formulate the discretization in vector notation, we define $\mathbf{y}(t) := [y_1(t), ..., y_N(t)]^T$ and obtain the following system of ODEs:

$$M\dot{\mathbf{y}}(t) = -\frac{1}{2}B\mathbf{y}^2(t) - \nu C\mathbf{y}(t) + \mathbf{f}, \tag{A.5}$$

where the source vector is given by

$$\mathbf{f} = \begin{bmatrix} \int_0^L f(x)\phi_1(x)dx \\ \vdots \\ \int_0^L f(x)\phi_N(x)dx \end{bmatrix} \approx h\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_N) \end{bmatrix}.$$

Suitable initial conditions when $y_0(x)$ is equal to a step function can be derived straight forward since the test functions are equal to 1 at the grid points:

$$y_0(x) = \begin{cases} 1, & \text{if } 0 \le x \le \frac{L}{2} \\ 0, & \text{if } \frac{L}{2} < x \le L \end{cases} \quad \Rightarrow \quad y_i(0) = \begin{cases} 1, & \text{if } x_i \in [0, \frac{L}{2}] \\ 0, & \text{if } x_i \in (\frac{L}{2}, L] \end{cases}.$$

A.2 Time integration with the implicit Euler method

Since the ODE (A.5) is nonlinear, the application of the implicit Euler methods requires to solve for the root of a nonlinear equation using Newton's method at each time step.

The implicit Euler method applied to (A.5) reads

$$M\frac{\mathbf{y}^{(n+1)}-\mathbf{y}^{(n)}}{\tau} = -\frac{1}{2}B(\mathbf{y}^{(n+1)})^2 - \nu C\mathbf{y}^{(n+1)} + \mathbf{f}, \quad n = 1, ..., N_t,$$

where $\mathbf{y}^{(n)} = \mathbf{y}(t_n)$, τ is the time step, and N_t is the total number of time steps.

A re-formulation leads to

$$\mathbf{F}(\mathbf{y}^{(n+1)}) \equiv \frac{1}{\tau}M\mathbf{y}^{(n+1)} - \frac{1}{\tau}M\mathbf{y}^{(n)} + \frac{1}{2}B(\mathbf{y}^{(n+1)})^2 + \nu C\mathbf{y}^{(n+1)} - \mathbf{f} = 0,$$

where Newton's method can be applied such that the root of the nonlinear function $\mathbf{F}$ is equal to the solution at the next time step $\mathbf{y}^{(n+1)}$ (see Algorithm A.1). In order to solve the linear system at line 6, we also need to derive the Jacobian of $\mathbf{F}$ which can be computed analytically by

$$J_F(\mathbf{y}^{(n+1)}) = \frac{1}{\tau}M + B.*\mathbf{y}^{(n+1)} + \nu C,$$

where $B.*\mathbf{y}^{(n)}$ means that every row of the matrix B is multiplied pointwise with the vector $\mathbf{y}^{(n)}$ such that the overall product is again a matrix of the appropriate dimension. The stopping criterium can be specified via a tolerance ε_{Eul} for the relative error of the Newton iteration (see line 7).

Algorithm A.1 Euler implicit with Newton's method

1: Initialize $\mathbf{y}^{(1)} = \mathbf{y}_0, \varepsilon_{Newton} \in \mathbb{R}_+$, `max_newton` $\in \mathbb{N}$
2: **for** $n = 1$ to N_t **do**
3: $\quad \mathbf{y}_{tmp,1} = \mathbf{y}^{(n)}$ *% Educated guess*
4: $\quad$ Set $err = 1$, $iter = 0$
5: $\quad$ **while** $err > \varepsilon_{Newton}$ **and** $iter <$ `max_newton` **do**
6: $\quad\quad$ Solve $\mathbf{y}_{tmp,2} = \mathbf{y}_{tmp,1} - J_F^{-1}\mathbf{F}(\mathbf{y}_{tmp,1})$
7: $\quad\quad$ $err = \|\mathbf{y}_{tmp,2} - \mathbf{y}_{tmp,1}\|_2 / \|\mathbf{y}_{tmp,2}\|_2$
8: $\quad\quad$ Update iterate, $\mathbf{u}_{tmp,1} = \mathbf{u}_{tmp,2}$
9: $\quad\quad$ $iter = iter + 1$
10: $\quad$ **end while**
11: $\quad \mathbf{y}^{(n+1)} = \mathbf{y}_{tmp,2}$ *% Assign update after convergence*
12: **end for**

Appendix B

Implementation issues

B.1 The truncated conjugant gradient (CG) method

In order to obtain a new search direction in the k-th iteration of the Newton-based method discussed in Section 3.1, we need to solve the Newton equation,

$$\nabla^2 \hat{\mathcal{J}}(u_k) s_k = -\nabla \hat{\mathcal{J}}(u_k), \tag{B.1}$$

for the search direction s_k. Note that (B.1) is a linear system of equations and the Hessian $\nabla^2 \hat{\mathcal{J}}(u_k)$ is by definition a symmetric matrix. For simplicity, let us denote $A := \nabla^2 \hat{\mathcal{J}}(u_k)$, $x := s_k$ and $b := -\nabla \hat{\mathcal{J}}(u_k)$ such that (B.1) is equal to $Ax = b$. In order to motivate the usage of the CG algorithm to solve (B.1), we use the theorem [48, Theorem 38.2] that states that in the i-th iteration of CG, the error $e^{(i)} = x^* - x^{(i)}$ is minimized in the A-norm, i.e.

$$x^{(i)} = \underset{x \in \mathcal{K}_i}{\operatorname{argmin}} \, \| e^{(i)} \|_A = \underset{x \in \mathcal{K}_i}{\operatorname{argmin}} \, \| x^* - x \|_A,$$

where $x^{(i)}$ is the i-th iterate of the CG algorithm and x^* is the (unknown) exact solution of linear system, i.e. $Ax^* = b$. Furthermore, the CG method minimizes in step i over the so-called Krylov space of dimension i,

$$\mathcal{K}_i := \text{span } \{b, Ab, ..., A^{i-1} b\}.$$

The following short calculation shows that minimizing the error $e^{(i)}$ in the A-norm is equivalent to minimizing the second-order Taylor expansion of the cost function $\hat{\mathcal{J}}$,

$$\begin{aligned} \| e^{(i)} \|_A^2 &= (e^{(i)})^T A e^{(i)} = (x^* - x^{(i)})^T A (x^* - x^{(i)}) \\ &= (x^{(i)})^T A x^{(i)} - 2 (x^{(i)})^T A x^* + (x^*)^T A x^* \\ &= (x^{(i)})^T A x^{(i)} - 2 (x^{(i)})^T b + (x^*)^T b =: 2\, T_{\hat{\mathcal{J}}}(x^{(i)}) + (x^*)^T b, \end{aligned}$$

where $(x^*)^T b$ is a constant term and after back-substitution of A, x, b we see that $T_{\hat{\mathcal{J}}}(x) = \frac{1}{2} x^T A x - x^T b$ is except for a constant term equal to the second-order Taylor expansion of

the objective function $\hat{\mathcal{J}}$. Therefore, the CG algorithm can be interpreted as an iterative process for minimizing the quadratic approximation of the cost function which motivates its usage in optimization, cf. [48]. This is also the reason why the update formula in line 19 of Algorithm B.1 reminds of a line search algorithm. In fact, it is derived in [48] that $T_{\hat{\mathcal{J}}}$ is minimized in step i over the Krylov space $\mathcal{K}_i$.

Furthermore, the version of the CG algorithm we present in Algorithm B.1 is truncated when the following condition is fulfilled,

$$\|\nabla^2\hat{\mathcal{J}}(u_k)s_k + \nabla\hat{\mathcal{J}}(u_k)\|_2 \leq \eta_k\|\nabla\hat{\mathcal{J}}(u_k)\|_2, \tag{B.2}$$

where $\eta_k \in (0,1)$ and the left-hand side is the residual of the Newton equation (B.1). For the numerical test in Section 3.3.1 and 4.2, we use $\eta_k = 0.01$ or, as suggested in [23], $\eta_k = \min\{0.01, \|\nabla\hat{\mathcal{J}}(u_k)\|_2\}$.

Note that in Algorithm B.1, we choose the method of steepest descent in the case that the Hessian is not positive definite (line 12).

Algorithm B.1 The truncated CG algorithm for solving the Newton equation $\nabla^2\hat{\mathcal{J}}(u_k)s_k = -\nabla\hat{\mathcal{J}}(u_k)$, [23]

1: **INPUT:** Function handle that evaluates the matrix-vector product $\nabla^2\hat{\mathcal{J}}(u_k)\cdot v$, right-hand side $-\nabla\hat{\mathcal{J}}(u_k)$, `max_cg`$\in\mathbb{N}$, truncation tolerance $\eta_k \in (0,1)$
2: **OUTPUT:** Solution of Newton's equation s_k
3: Set $s_k = 0, p_k^{(0)} = r_k^{(0)} = -\nabla\hat{\mathcal{J}}(u_k)$
4: **for** $i = 0, 1, 2, ...,$`max_cg` **do**
5: **if** $\|r_k^{(i)}\|_2 < \eta_k\|r_k^{(0)}\|_2$ **then**
6: **if** $i = 0$ **then**
7: $s_k = -\nabla\hat{\mathcal{J}}(u_k)$ *% Steepest descent direction*
8: **return**
9: **end if**
10: **end if**
11: Compute $q_k^{(i)} = \nabla^2\hat{\mathcal{J}}(u_k)\cdot p_k^{(i)}$
12: **if** $(p_k^{(i)})^T q_k^{(i)} < 0$ **then**
13: **if** $i = 0$ **then**
14: $s_k = -\nabla\hat{\mathcal{J}}(u_k)$ *% Steepest descent direction*
15: **return**
16: **end if**
17: **end if**
18: Compute $\gamma_k^{(i)} = \|r_k^{(i)}\|_2^2/(p_k^{(i)})^T q_k^{(i)}$
19: Update solution, $s_k = s_k + \gamma_k^{(i)} q_k^{(i)}$
20: Compute $r_k^{(i+1)} = r_k^{(i)} - \gamma_k^{(i)} q_k^{(i)}$
21: Compute $\beta_k^{(i)} = \|r_k^{(i+1)}\|_2^2/\|r_k^{(i)}\|_2^2$
22: Compute $p_k^{(i+1)} = r_k^{(i+1)} + \beta_k^{(i)} p_k^{(i)}$
23: **end for**

B.2 Armijo line search

Suppose the search direction s is known (we omit the index k since the situation is the same at each iteration) we need to minimize the objective function $\mathcal{J}(y(u), u)$ along the descent direction s. This means we are looking for an optimal step length α^* in the direction of s, i.e.

$$\alpha^* = \underset{\alpha \in \mathbb{R}_+}{\operatorname{argmin}} \mathcal{J}(y(u + \alpha \cdot s), u + \alpha \cdot s). \tag{B.3}$$

An update of u can then be computed according to formula (3.7). Since $\mathcal{J}(y(\cdot), \cdot)$ is a multi-dimensional function, we will only find a local minimum along the direction of s. The Armijo algorithm B.2 belongs to the family of line search algorithms that iteratively solve the optimization problem (B.3) along the search direction s. An overview of other line search algorithms can for instance be found in [39, 51].

Algorithm B.2 Armijo line search algorithm, [39]

1: **INPUT:** initial point u_0, search direction s, tolerance $\varepsilon_\alpha \in \mathbb{R}_+$, safeguard $\alpha_{min} \in (0, 1)$
2: **OUTPUT:** optimal step size α^* in direction s
3: Solve $c(y_0, u_0)$ for y_0
4: Compute $\mathcal{J}_0 = \mathcal{J}(y_0, u_0)$
5: Set $\alpha = 1$ and set $u = u_0 + \alpha \cdot s$
6: Solve $c(y, u)$ for y
7: **while** $\mathcal{J}(y, u) > \mathcal{J}_0 + \varepsilon_\alpha \cdot \alpha \cdot s^T \nabla_u \mathcal{J}(y_0, u_0)$ **and** $\alpha > \alpha_{min}$ **do**
8: Set $\alpha := \alpha/2$
9: Set $u = u_0 + \alpha \cdot s$
10: Solve $c(y, u)$ for y
11: Compute $\mathcal{J}(y, u)$
12: **end while**
13: We have found $\alpha^* = \alpha$

The stopping condition in line 7 does not require a re-calculation of the gradient of the objective function since, in practice, we used the respective value of the previous iterate as initial tuple (u_0, y_0) for which the gradient has already been computed. The tolerance that has been used is $\varepsilon_\alpha = 10^{-4}$. In line 8, the step size is devided by 2 and the control is updated. It is important to note that once the control is updated in line 9, we need to solve the constraining PDE c in order to obtain the state $y(u)$. Afterwards, the cost function $\mathcal{J}(y, u)$ can be evaluated again. Therefore, it is crucial to note that it is necessary to solve $c(y, u)$ in line 3 and line 10. In case of optimal control of the reduced model, this has been replaced by the solution of $\tilde{c}(\tilde{y}, u)$ which leads to a computational gain within the Armijo line search algorithm.

B.3 Matlab code

The numerical test calculations for the POD-DEIM model of Burgers' equation presented in Section 2.3 as well as the optimal control algorithm discussed in Section 3.3 and Chapter 4 have been implemented in MATLAB. The code is freely accessible via

`https://github.com/ManuelMBaumann/MasterThesis`

and can be used for further improvement or demonstration at any time.

Appendix C

Notation

Whenever I read a scientific publication I keep asking myself *What is f ?* Therefore, the following list gives an overview of the names of variables that are used in this Master thesis.

y	State variable
u	Control variable
λ	Adjoint variable
ν	Viscosity parameter in Burgers' equation
$[0, T]$	Time domain
N	Spatial dimension of the full model
ℓ	POD-dimension
m	DEIM-dimension
Ω	Spatial domain
N_t	Time discretization size
$\mathcal{P}$	DEIM projection matrix
$\mathcal{L}$	Lagrangian function
$\mathcal{J}$	Objective function
$\hat{\mathcal{J}}$	The same objective function depending on the control only
ω	Control penalty, we choose $\omega \in (0, 1)$
M, B, C	Constant matrices derived from the spatial discretization of Burgers' equation
I_N	Identity matrix of dimension $N \times N$
S_P	The speedup is defined as the ratio of computation times
$\varepsilon_{\mathcal{J}}$	Tolerance for change in objective function
ε_{∇}	Tolerance for zero-gradient condition
ε_{α}	Tolerance of the Armijo line-search
ε_{Newton}	Tolerance of the Newton-iteration for implicit Euler
η_k	Truncation tolerance for CG-algorithm