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

Model Order Reduction for PDE-constrained Optimization

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

May 6, 2013

TU Delft Delft University of Technology

MSc THESIS APPLIED MATHEMATICS

"Model Order Reduction for PDE-constrained Optimization"

Manuel M. Baumann

Delft University of Technology

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

Committee members

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

May 6, 2013 Delft, The Netherlands

Contents

Acknowledgements

This Master thesis has been written within the Erasmus Mundus programme *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 the great people I met from all over the world.

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

Furthermore, I would like to thank the supervisors of my Master thesis, Martin van Gijzen and Marielba Rojas. Our weekly meetings, widely 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.

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] Find out more about COSSE: `http://www.kth.se/en/studies/programmes/em/cosse`

Chapter 1

Introduction

1.1 Motivation

1.2 Overview of the topic

1.3 Chapter outline

Chapter 2

Model order reduction for nonlinear dynamical systems

2.1 Proper orthogonal decomposition (POD)

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

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

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

which for example arise after spatial discretization of time-dependent nonlinear partial differential equations (PDEs). Therefore, we assume that the unknowns vector $\mathbf{y}$ 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.

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 $\mathcal{U}$ of dimension $\ell \ll n$ which 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 present that a singular value decomposition (SVD) of the snapshot matrix can be used to obtain an optimal projection space. In more detail, we present a proof of [10] that shows that the optimal ℓ-dimensional projection space is given by 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, [4]) *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 [4, Theorem 2.5.2]. □

Note that 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 YY^T with eigenvalues $\lambda_i = \sigma_i^2 > 0,\ i = 1, ..., d$.

The next theorem shows that the solution to the optimization problem (2.6) is given by the left singular vectors of the snapshot matrix (2.3). This is important since in (2.6) 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 (2.6).

Theorem 2.1.2. (POD basis, [10]) *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. Hereby, $\delta_{i,j}$ denotes the Kronecker delta.

Proof. The following proof is based on the ideas of [10]. Since (2.6) is an equality constrained optimization problem, we consider the corresponding Lagrangian function

$$\mathcal{L} : \mathbb{R}^n \times ... \times \mathbb{R}^n \times \mathbb{R}^{\ell\times\ell} \to \mathbb{R}$$
$$\mathcal{L}(\varphi_1, ..., \varphi_\ell, \Lambda) = \sum_{i=1}^{\ell}\sum_{j=1}^{n_s} |\langle \mathbf{y}_j, \varphi_i\rangle|^2 + \sum_{i,j=1}^{\ell} \lambda_{ij}(\delta_{ij} - \langle \varphi_i, \varphi_j\rangle),$$

where $\Lambda := (\lambda_{i,j})_{i,j=1}^{\ell}$. The first-order necessary optimality conditions of (2.6) are then given by

$$\frac{\partial \mathcal{L}}{\partial \varphi_k}(\varphi_1, ..., \varphi_\ell, \Lambda) = 0, \quad \text{for } k \in \{1, ..., \ell\},$$
$$\frac{\partial \mathcal{L}}{\partial \lambda_{i,j}}(\varphi_1, ..., \varphi_\ell, \Lambda) = 0, \quad \text{for } i, j \in \{1, ..., \ell\},$$

where the latter equation is equivalent to the constraint $\langle \varphi_i, \varphi_j \rangle = \delta_{i,j}$. In [10, p. 5], it is shown that the first condition leads to

$$YY^T \varphi_k = \frac{1}{2} \sum_{i=1}^{\ell} (\lambda_{i,k} + \lambda_{k,i}) \varphi_i, \quad \text{for all } k \in \{1, ..., \ell\}. \tag{2.7}$$

The actual proof that the optimal vectors $\varphi_1, ..., \varphi_\ell$ are given by the first ℓ left singular vectors of Y is done by induction on ℓ. First, let $\ell = 1$. From (2.7), it follows that $k = 1$ and we get:

$$YY^T \varphi_1 = \lambda_{1,1} \varphi_1.$$

From (2.5), we have that this eigenequation can be expressed as an SVD with φ_1 being the first left singular vector of Y.

Next, consider the case $\ell \geq 1$ and suppose

$$YY^T \varphi_k = \lambda_{k,k} \varphi_k, \quad \text{for all } k \in \{1, ..., \ell\}.$$

We want to show that the first-order necessary optimality conditions for a POD-basis of rank $\ell + 1$ are given by

$$YY^T \varphi_k = \lambda_{k,k} \varphi_k, \quad \text{for all } k \in \{1, ..., \ell + 1\},$$

and can, thus, be computed by an SVD of Y. Using the induction hypothesis, we only have to show that

$$YY^T \varphi_{\ell+1} = \lambda_{\ell+1,\ell+1} \varphi_{\ell+1}.$$

Since $\{\varphi_i\}_{i=1}^{\ell+1}$ is a POD-basis, it holds that $\langle \varphi_{\ell+1}, \varphi_i \rangle = 0$ for $1 \leq i \leq \ell$ and we get

$$\begin{aligned} 0 &= \lambda_{i,i} \langle \varphi_{\ell+1}, \varphi_i \rangle = \langle \varphi_{\ell+1}, YY^T \varphi_i \rangle = \langle YY^T \varphi_{\ell+1}, \varphi_i \rangle \\ &= \frac{1}{2} \sum_{j=1}^{\ell+1} (\lambda_{j,\ell+1} + \lambda_{\ell+1,j}) \langle \varphi_j, \varphi_i \rangle = (\lambda_{i,\ell+1} + \lambda_{\ell+1,i}) \quad \Rightarrow \quad \lambda_{\ell+1,i} = -\lambda_{i,\ell+1}, \ i \in \{1, ..., \ell\}, \end{aligned}$$

where the symmetry of YY^T guarantees the matrix to be self-adjoint in $\langle \cdot, \cdot \rangle_{\mathbb{R}^n}$.

Inserting the last relation into (2.7), we obtain

$$\begin{aligned} YY^T \varphi_{\ell+1} &= \frac{1}{2} \sum_{i=1}^{\ell+1} (\lambda_{i,\ell+1} + \lambda_{\ell+1,i}) \varphi_i = \frac{1}{2} \sum_{i=1}^{\ell} (\lambda_{i,\ell+1} + \lambda_{\ell+1,i}) \varphi_i + \lambda_{\ell+1,\ell+1} \varphi_{\ell+1} \\ &= \frac{1}{2} \sum_{i=1}^{\ell} \underbrace{(\lambda_{i,\ell+1} - \lambda_{i,\ell+1})}_{=0} \varphi_i + \lambda_{\ell+1,\ell+1} \varphi_{\ell+1} = \lambda_{\ell+1,\ell+1} \varphi_{\ell+1} \end{aligned}$$

We have shown that the first-order optimality conditions corresponding to (2.6) are given by the $n \times n$ eigenvalue problems

$$YY^T \mathbf{u}_i = \lambda_i \mathbf{u}_i, \quad i = 1, ..., \ell,$$

where the vectors $\{\mathbf{u}_i\}_{i=1}^{\ell}$ are given by the left singular vectors of Y due to (2.5). The proof that $\{\mathbf{u}_i\}_{i=1}^{\ell}$ are a solution of (2.6) can be found in [10]. □

For the practical usage of POD, we need to choose the dimension ℓ of the POD-basis. According to [10], 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,

$$\varepsilon(\ell) = \frac{\sum_{i=1}^{\ell} \sigma_i^2}{\sum_{i=1}^{n} \sigma_i^2},$$

gives a good estimate for the relation of the energy of the reduced system and the total energy.

2.1.2 The projected reduced-order model

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

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

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

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

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

The corresponding initial condition is given by

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

2.2 Discrete empirical interpolation method (DEIM)

In the previous sections, the POD-Galerkin approach (2.8) has been derived to be a reduced-order model of the full order dynamical system (2.1)-(2.2). The system (2.8) is of the (small) dimension ℓ and the unknown $\tilde{\mathbf{y}}$ is an ℓ-dimensional approximation of $\mathbf{y}$. However, the evaluation of the nonlinear term $\tilde{\mathbf{N}}(\tilde{\mathbf{y}})$ still has computational complexity

that depends on the original problem size n as the following consideration shows:

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

The DEIM method is an efficient way to overcome this dependence on n and is, therefore, an improvement of the POD algorithm, cf. [2]. For a more simple notation, 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 coefficient vector $\mathbf{c}(t) \in \mathbb{R}^m$, similar to the POD-approach. Indeed, the projection space $span\{\mathbf{w}_1, ..., \mathbf{w}_m\}$ is obtained by a 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 for $\mathbf{c}(t)$, we select m distinguished rows from both sides of the system. Therefore, we define the matrix $P = [\mathbf{e}_{\wp_1}, ..., \mathbf{e}_{\wp_m}] \in \mathbb{R}^{n\times m}$ with $\mathbf{e}_{\wp_i} = [0, ..., 0, 1, 0, ..., 0] \in \mathbb{R}^n$ having its non-zero entry at the i-th component. If $P^T W$ is nonsingular, the coefficient vector $\mathbf{c}(t)$ can uniquely be determined from

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

Altogether, we derive an approximation of $\mathbf{f}(t)$ of the following form

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

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

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

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

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 $\tilde{\mathbf{N}}(\tilde{\mathbf{y}})$ does not depend on the dimension n of the full order system and, moreover, we note that the matrix $U_\ell^T W(P^T W)^{-1}$ does not depend on t and can, thus, be pre-computed.

It remains to present how the m entries of $\mathbf{f}(t)$ have to be selected such that the approximation $\hat{\mathbf{f}}(t)$ in formula (2.10) is optimal. In [2, Section 3.1], algorithm 1 is presented

which for a given input basis $\mathbf{w}_1, ..., \mathbf{w}_m$ successively builds the matrix P. In the initialization step, the first index $\wp_1 \in \{1, ..., n\}$ is selected corresponding to the largest component in magnitude of the first input basis vector $\mathbf{w}_1$. The loop over $i = 2$ to $i = m$ selects the next indices such that the largest component of the residual $\mathbf{r}$ between the input basis $\mathbf{w}_i$ and its approximation within the subspace $span\{\mathbf{w}_1, ..., \mathbf{w}_{i-1}\}$ compensated.

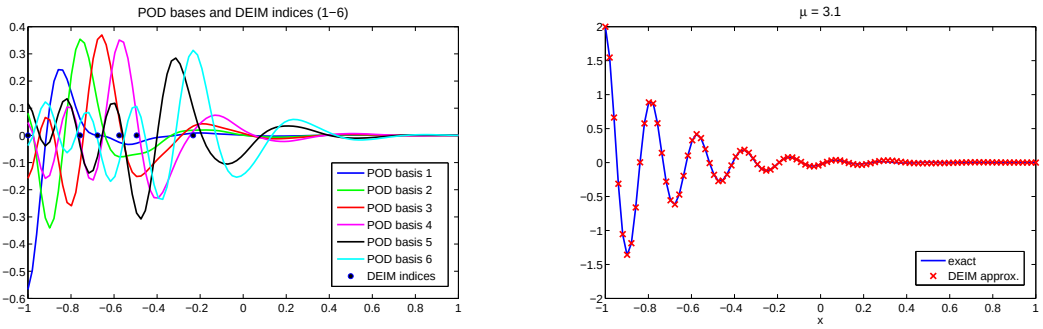
Algorithm 1 The DEIM algorithm, [2]

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

An illustrative example is given by the nonlinear parameterized function

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

where the DEIM algorithm has been applied based on 100 equidistantly spaced points $x_i \in [-1, 1]$ and 51 snapshots for $\mu_j \in [1, \pi]$. Figure 2.1 shows on the left that the first DEIM indices obtained by Algorithm 1 are chosen in a region where most of the dynamics of s occur. In the right plot of Figure 2.1, it can be seen that for $\mu = 3.1$ the approximation $\hat{s}$ based on DEIM 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 of example (2.11).

Theorem 2.2.1. (Error bound of DEIM approximation, [2]) *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*

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

where $W := [\mathbf{w}_1, ..., \mathbf{w}_m] \in \mathbb{R}^{n\times m}$ *and* $P := [\mathbf{e}_{\wp_1}, ..., \mathbf{e}_{\wp_m}] \in \mathbb{R}^{n\times m}$*, with* $\{\wp_1, ..., \wp_m\}$ *being obtained from Algorithm 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

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

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 [2, p. 2747-2749]. □

Theorem 2.2.1 not only gives an error bound on the approximation obtained by the DEIM algorithm, but the proof also points out that the choice of the DEIM indices in Algorithm 1 in fact minimizes the growth of $\|(P^T W)^{-1}\|_2$. Therefore, the algorithm is optimal in the sense that the approximation error is minimized.

2.3 Application: POD-DEIM for the unsteady Burger's equation

The following example is based on the considerations in [6] where the one-dimensional Burger's equations together with homogeneous Dirichlet boundary conditions and a step function $\Phi(x)$ as initial condition has been considered,

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

Burger's 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 to be the conservative form of Burger's equation. Note that the numerical solution highly depends on the viscosity parameter ν: For small ν, 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, B represents the convective term and, for simplicity, we assume the source term to be equal to zero, i.e. $f \equiv 0$. The system (2.13) can be integrated in time using the implicit Euler method, see Appendix A.2.

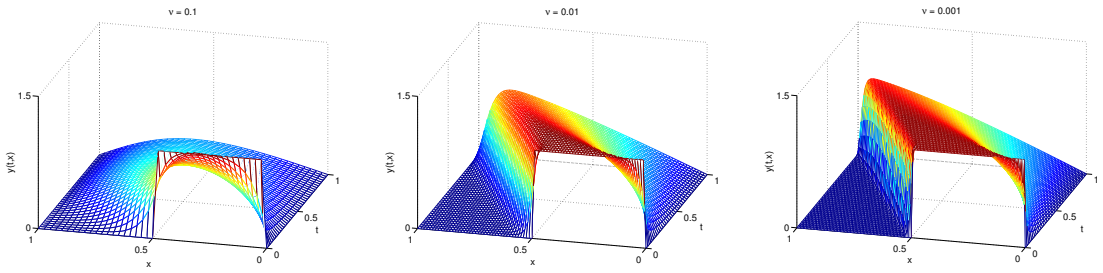


Figure 2.2: Numerical solution of the full-order Burger's equation for different viscosity parameters.

According to [6] the POD-basis $\{\varphi_i\}_{i=1}^{\ell}$ can be obtained by solving the eigenvalue problem

$$YY^T M\varphi = \sigma^2\varphi, \tag{2.14}$$

where Y is the matrix of snapshots as previously defined in (2.3).

Since the mass matrix M is symmetric and positive definite, there exists a Cholesky decomposition of the form $M = R^T R$ and the eigenequation (2.14) can instead be solved by an SVD of the matrix RY,

$$RY = U\Sigma V^T.$$

After choosing a suitable POD-dimension $\ell \ll n$, let us define the matrix of left singular vectors, $U_\ell := [u_1, ..., u_\ell]$. Then, the POD-basis is given by $\Phi_\ell := R^{-1}U_\ell$ and the following ansatz leads to a reduced system,

$$\mathbf{y}(t) \approx \mathbf{y}^{(\ell)}(t) = \Phi_\ell\tilde{\mathbf{y}}(t) \quad \text{with } \tilde{\mathbf{y}} \in \mathbb{R}^\ell. \tag{2.15}$$

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

$$\dot{\tilde{\mathbf{y}}}(t) = -\frac{1}{2}B_\ell(\Phi_\ell\tilde{\mathbf{y}})^2 - \nu C_\ell\tilde{\mathbf{y}}, \tag{2.16}$$

with the matrices

$$B_\ell := \Phi_\ell^T B \in \mathbb{R}^{\ell\times n}, \tag{2.17}$$

$$C_\ell := \Phi_\ell^T C\Phi_\ell \in \mathbb{R}^{\ell\times\ell}, \tag{2.18}$$

and the new mass matrix being equal to the $\ell \times \ell$ identity as the following calculation shows,

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

We will refer to (2.16) as the POD-reduced system. Note, that $\Phi\tilde{\mathbf{y}} \in \mathbb{R}^n$ is still of large dimension and, therefore, no dimension reduction for the nonlinearity has been obtained so far.

The initial condition of the POD-reduced system is given as follows,

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

and, therefore, the initial condition for (2.16) can be obtained by pre-multiplication of the full-size initial condition with the matrix product $\Phi_\ell^T M$ as shown below,

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

As it has already been pointed out, the nonlinear term in (2.16) is still of large dimension n. Therefore, we apply the DEIM method in a next step in order to reduce the computational effort 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 1. From algorithm 1, we obtain the projection matrix P and the nonlinear term becomes:

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

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

The fully reduced POD-DEIM system is then given by

$$\dot{\tilde{\mathbf{y}}} = -\frac{1}{2}\tilde{B}(\tilde{F}\tilde{\mathbf{y}})^2 - \nu \tilde{C}\tilde{\mathbf{y}} + \tilde{\mathbf{f}}, \tag{2.19}$$

where

$$\tilde{B} := \Phi_\ell^T B U_f (P^T U_f)^{-1} \in \mathbb{R}^{\ell \times m}, \tag{2.20}$$

$$\tilde{C} := \Phi_\ell^T C \Phi_\ell \in \mathbb{R}^{\ell \times \ell}, \tag{2.21}$$

$$\tilde{F} := P^T \Phi_\ell \in \mathbb{R}^{m \times \ell}. \tag{2.22}$$

Note that in a fast implementation, both matrices $\tilde{B}$ and $\tilde{F}$ can be pre-computed such that the system (2.19) is indeed of dimension $k \times k$ and no dependence on the original size n exists anymore. The reduced system (2.19) as well as the full-order system (2.13) have been solved by the implicit Euler method, see Appendix A.2.

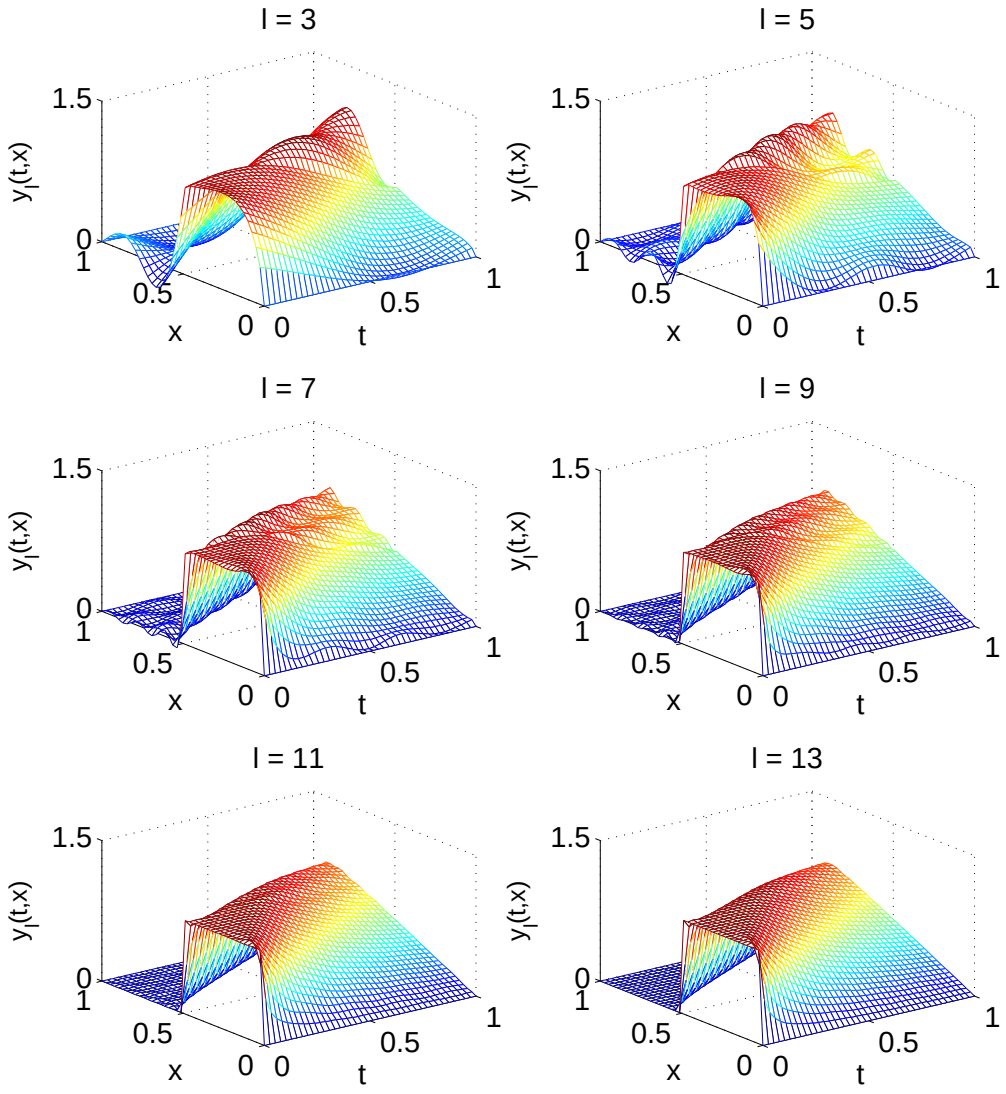


Figure 2.3: POD-DEIM approximations for different dimensions ℓ and $m = 15$, $\nu = 0.01$.

Figure 2.3 shows to convergence of the reduced model to the full-order solution: In each computation a fixed DEIM dimension of $m = 15$ has been used to approximate the nonlinear term of Burger's equation. If the dimension of the POD-basis ℓ and, thus, the size of the projected model is increased, we see from Figure 2.3 that the numerical solution of the reduced system converges towards a smooth solution that corresponds to the solution of the original model.

Chapter 3

Optimal control of partial differential equations

In optimal control, problems of the following form are considered,

$$\begin{aligned} &\min_u f(y,u), \\ &\text{subject to} \quad c(y,u)=0, \end{aligned} \tag{3.1}$$

where y is called the *state* and u is considered to be the *control* or the *input* of the problem (3.1). Furthermore, the scalar function f is usually called the *cost function* and in the following we assume that 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 f and c map as follows,

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

As it is stated in (3.1), we seek for an optimal control $u^* = \arg\ \min_u\ f(y,u)$ that minimizes the cost function f 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. A standard reference for optimal control of partial differential equations is given by [9].

A standard approach which converts the constrained optimization problem (3.1) into an unconstrained optimization problem is the introduction of the Lagrangian function $\mathcal{L}$ via

$$\begin{aligned} &\mathcal{L} : \mathbb{R}^{n_y} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_y} \to \mathbb{R} \\ &\mathcal{L}(y,u,\lambda) = f(y,u) + \lambda^T c(y,u), \end{aligned} \tag{3.2}$$

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 (3.1), there exists a λ^* such

that (y^*, u^*, λ^*) is a stationary point of the Lagrangian function (3.2), cf. [8]. Stationary points of (3.2) fulfill the first order optimality conditions (zero-gradient condition),

$$\nabla_y \mathcal{L}(y, u, \lambda) = 0, \tag{3.3}$$

$$\nabla_u \mathcal{L}(y, u, \lambda) = 0, \tag{3.4}$$

$$\nabla_\lambda \mathcal{L}(y, u, \lambda) = 0, \tag{3.5}$$

where the subscripts are used to denote partial derivatives with respect to the indicated variable. Note, that from (3.5) the original constraint $c(y, u) = 0$ follows directly. Furthermore, we will use (3.3) in order to derive the so-called *adjoint equation* which determines the auxiliary variable λ. In the following section, we will describe an efficient way to compute (3.4) numerically by solving a Newton equation for (3.4).

3.1 Newton-type methods using adjoint techniques for derivative computation

In this section, we will describe an efficient way to obtain stationary points of the Lagrangian function (3.2) numerically. Following the instructions of [5], we will us the conditions (3.3) and (3.5) in order to derive efficient algorithms to solve condition (3.4) by the Newton equation. For the simplicity of notation, we will introduce the function $\hat{\mathcal{L}}$ to be the Lagrangian as a function of the control u only, i.e.

$$\hat{\mathcal{L}}(u) = \mathcal{L}(y, u, \lambda)\big|_{y=y(u),\lambda=\lambda(u)}.$$

Consequently the corresponding gradient and Hessian can be denoted in the following compact way,

$$\nabla\hat{\mathcal{L}}(u) = \nabla_u \mathcal{L}(y, u, \lambda)\big|_{y=y(u),\lambda=\lambda(u)}, \quad \nabla^2\hat{\mathcal{L}}(u) = \nabla_u^2 \mathcal{L}(y, u, \lambda)\big|_{y=y(u),\lambda=\lambda(u)}.$$

Since (3.4) is in general a nonlinear equation, finding the roots of $\nabla\hat{\mathcal{L}}(\cdot)$ can be achieved iteratively by Newton's method which reads,

$$u_{k+1} = u_k - (\nabla^2\hat{\mathcal{L}}(u_k))^{-1}\nabla\hat{\mathcal{L}}(u_k). \tag{3.6}$$

Here, one usually starts with an initial guess for u_0 and iterates until convergence or a maximum number of iterations is obtained. When introducing the search direction $s_k := u_{k+1} - u_k$, we can reformulate (3.6) and derive the so-called Newton equation,

$$\nabla^2\hat{\mathcal{L}}(u_k)s_k = -\nabla\hat{\mathcal{L}}(u_k), \tag{3.7}$$

which is a system of linear equations for the unknown search direction s_k. The optimization algorithm presented in this section will solve the Newton equation (3.7) by a truncated version of the conjugate gradient method, see Appendix B.1. We will present how the gradient and the Hessian of the Lagrangian function $\hat{\mathcal{L}}(\cdot)$ can be computed efficiently by

so-called adjoint equations. Once the new search direction is obtained by solving (3.7), the optimal step size in the direction s_k is obtained by minimizing the cost function f for a fixed state along s_k. This can be done by a simple line-search algorithm, for instance by an implementation of the Armijo line-search algorithm, see Appendix B.2. An overview of the described optimization loop is presented in Algorithm 2.

Algorithm 2 Main optimization loop, [5]

1: Set initial control $u_0 = 0$, and $tol > 0$, `MAXIT` $\in \mathbb{N}$
2: **for** $k = 0$ to `MAXIT` **do**
3: Solve $c(y_k, u_k) = 0$ for y_k
4: Compute $\nabla\hat{\mathcal{L}}(u_k)$ via algorithm 3
5: **if** $\|\nabla\hat{\mathcal{L}}(u_k)\| < tol$ **then**
6: **return**
7: **end if**
8: Compute $\nabla^2\hat{\mathcal{L}}(u_k)$ via algorithm 4
9: Solve the Newton equation $\nabla^2\hat{\mathcal{L}}(u_k)s_k = -\nabla\hat{\mathcal{L}}(u_k)$ via the conjugate gradient method, see Appendix B.1
10: Obtain $\alpha_k = \min_{\alpha_k \in \mathbb{R}} f(y_k, u_k + \alpha s_k)$ via Armijo line-search, see Appendix B.2
11: Set $u_{k+1} = u_k + \alpha_k s_k$
12: **end for**

3.1.1 Using adjoint equations for gradient computation

In Algorithm 2, we need to compute the gradient with respect to u of the Lagrangian function (3.2). Therefore, we first recall condition (3.3) which is given by

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

If we apply the gradient with respect to y to the Lagrangian function (3.2), we derive

$$c_y(y(u), u)^T \lambda = -\nabla_y f(y(u), u), \tag{3.8}$$

which is called the adjoint equations because from (3.8) we are able to determine the adjoint variable λ.

The gradient in the right-hand side of (3.7) can, thus, be computed via

$$\nabla\hat{\mathcal{L}}(u) = \nabla_u \mathcal{L}(y, u, \lambda)\big|_{y=y(u),\lambda=\lambda(u)} = \nabla_u f(y(u), u) + c_u(y(u), u)^T \lambda(u), \tag{3.9}$$

where $\lambda(u)$ is the solution to (3.8). Note, that at that stage the state y and also the adjoint λ are assumed to be computed after a specific control u has been chosen. We emphasize this be the notation $y(u)$ and $\lambda(u)$, respectively. A summary of the computation of $\nabla\hat{\mathcal{L}}(u)$ is stated in Algorithm 3.

Algorithm 3 Computing $\nabla\hat{\mathcal{L}}(u)$ via adjoints, [5]

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

3.1.2 Using adjoint equations for Hessian computation

The Hessian of $\hat{\mathcal{L}}$ is given by

$$\nabla^2\hat{\mathcal{L}}(u) = \nabla_{uy}\mathcal{L}(y(u),u,\lambda(u))y_u(u) + \nabla_{uu}\mathcal{L}(y(u),u,\lambda(u)) + \nabla_{u\lambda}\mathcal{L}(y(u),u,\lambda(u))\lambda_u(u), \tag{3.10}$$

where inner derivatives exist due to the dependence of y and λ on u. In order to compute (3.10), we need to determine the derivatives $y_u(\cdot)$ and $\lambda_u(\cdot)$.

Therefore, we first note that the condition (3.5) is equivalent to the constraint of the original problem, $c(y(u),u) = 0$, and we can conclude that the Jacobian of $y(\cdot)$ is the solution of,

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

The last equation can be rearranged to,

$$y_u(u) = -c_y(y(u),u)^{-1}c_u(y(u),u). \tag{3.11}$$

Secondly, we conclude from differentiating the condition $\nabla_y\mathcal{L}(y(u),u,\lambda(u)) = 0$ that,

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

and, therefore,

$$\begin{aligned}\lambda_u(u) &= (\nabla_{y\lambda}\mathcal{L}(y(u),u,\lambda))^{-1}\left[-\nabla_{yy}\mathcal{L}(y(u),u,\lambda)y_u(u) - \nabla_{yu}\mathcal{L}(y,u,\lambda)\right] \\ &= (\nabla_{y\lambda}\mathcal{L}(y(u),u,\lambda))^{-1}\left[\nabla_{yy}\mathcal{L}(y(u),u,\lambda)c_y(y(u),u)^{-1}c_u(y(u),u) - \nabla_{yu}\mathcal{L}(y,u,\lambda)\right],\end{aligned} \tag{3.12}$$

where (3.11) has been used to substitute $y_u(u)$.

From the definition (3.2), we see that $\nabla_\lambda\mathcal{L}(y,u,\lambda) = c(y,u)^T$ and, therefore the second mixed derivatives are simply given by,

$$\nabla_{y\lambda}\mathcal{L}(y,u,\lambda) = c_y(y,u)^T, \quad \nabla_{u\lambda}\mathcal{L}(y,u,\lambda) = c_u(y,u)^T. \tag{3.13}$$

If we plug-in (3.11) and (3.12) into (3.10) and as well use relation (3.13), we end up with,

$$\begin{aligned}\nabla^2\hat{\mathcal{L}}(u) = {} & c_u(y(u),u)^T c_y(y(u),u)^{-T}\nabla_{yy}\mathcal{L}(y(u),u,\lambda(u))c_y(y(u),u)^{-1}c_u(y(u),u) \\ & - c_u(y(u),u)^T c_y(y(u),u)^{-T}\nabla_{yu}\mathcal{L}(y(u),u,\lambda(u)) \\ & - \nabla_{uy}\mathcal{L}(y(u),u,\lambda(u))c_y(y(u),u)^{-1}c_u(y(u),u) + \nabla_{uu}\mathcal{L}(y(u),u,\lambda(u)),\end{aligned} \tag{3.14}$$

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

$$w := c_y(y(u),u)^{-1}c_u(y(u),u), \tag{3.15}$$

$$\begin{aligned}p &:= c_y(y(u),u)^{-T}\nabla_{yy}\mathcal{L}(y(u),u,\lambda(u))c_y(y(u),u)^{-1}c_u(y(u),u) \\ &\quad - c_y(y(u),u)^{-T}\nabla_{yu}\mathcal{L}(y(u),u,\lambda(u)) \\ &= c_y(y(u),u)^{-T}\left(\nabla_{yy}\mathcal{L}(y(u),u,\lambda(u))w - \nabla_{yu}\mathcal{L}(y(u),u,\lambda(u))\right)\end{aligned} \tag{3.16}$$

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

$$c_y(y(u), u)w = c_u(y(u), u), \tag{3.17}$$

$$c_y(y(u), u)^T p = \nabla_{yy}\mathcal{L}(y(u), u, \lambda(u))w - \nabla_{yu}\mathcal{L}(y(u), u, \lambda(u)). \tag{3.18}$$

The computation of the Hessian can, thus, be obtained in three steps as shown in Algorithm 4. Note, that in Algorithm 4 we don't compute the whole Hessian matrix but instead the Hessian times a given vector directly. This is primarily done because in the truncated CG algorithm of Appendix B.1 the entire Hessian in not required but only the product of the Hessian times a vector.

Algorithm 4 Computing the product $\nabla^2\hat{\mathcal{L}}(u) \cdot v$ via adjoints, [5]

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

Note, that in most applications, mixed derivatives of the Lagrangian function vanish.

3.2 Gradient-based optimization techniques

3.3 Application: Optimal control of Burger's equation

In order to apply the optimization algorithm described in Section 3.1, we need to specify the cost function f as well as the nonlinear PDE c that is constraining the optimization problem (3.1). In this section, we will consider a test problem that has been widely used in many different articles and can, thus, be considered as a standard test problem, cf. [5, 6]. We want to minimize the following cost functional,

$$\min_u \frac{1}{2}\int_0^T \int_0^L [y(x,t) - z(x,t)]^2 + \omega u^2(x,t)\ dx\ dt, \tag{3.19}$$

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

$$\begin{aligned} y_t + \left(\frac{1}{2}y^2 - \nu y_x\right)_x &= f + u, \quad (x,t) \in (0,L) \times (0,T), \\ y(t,0) = y(t,L) &= 0, \quad t \in (0,T), \\ y(x,0) &= \Phi(x), \quad x \in (0,L). \end{aligned} \tag{3.20}$$

Hereby, 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 (3.19)-(3.20) because if we are able to control the solution of Burger's 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 (3.19) 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 Burger's equation and can be chosen arbitrary as long as the initial and boundary conditions on y are not violated.

We now present a discretization of the cost functional (3.19) as well as Burger's equation (3.20) which is again in conservative form as 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,

$$u(t,x) \approx \sum_{j=1}^{N} u_j(t)\phi_j(x), \tag{3.21}$$

where u_j are the respective coefficients of ϕ_j that only depend on time, see for example [3]. Note, that this ansatz also implies that the control is zero at the boundary of Ω and, therefore, does not change the bahavior 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 (3.19), the outer 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,

$$\min_{\mathbf{u}_0,\dots,\mathbf{u}_{N_t}} f(\mathbf{y}_0,\dots,\mathbf{y}_{N_t},\mathbf{u}_0,\dots,\mathbf{u}_{N_t}) = \min_{\mathbf{u}_0,\dots,\mathbf{u}_{N_t}} \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2}\mathbf{y}_i^T M \mathbf{y}_i - \mathbf{z}^T \mathbf{y}_i + \frac{\omega}{2}\mathbf{u}_i^T M \mathbf{u}_i \right), \tag{3.22}$$

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 (3.22) the constant term $\mathbf{z}^T\mathbf{z}$ can be neglected since it does not influence the position of the minimum with respect to u. Furthermore, the missing subscript indicates that we have assumed that the desired state z does not depent on time. The vector $\mathbf{z}$ tested against the hat functions ϕ_j is therefore given by,

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

The discretization of Burger's equation (3.20) 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 descrite constraint in the form $c(y,u)=0$ is a vector-valued function with the components equal to,

$$c_{i+1}(\mathbf{y}_i,\mathbf{y}_{i+1},\mathbf{u}_{i+1}) \equiv \frac{1}{\delta t}M\mathbf{y}_{i+1} - \frac{1}{\delta t}M\mathbf{y}_i + \frac{1}{2}B\mathbf{y}_{i+1}^2 + \nu C\mathbf{y}_{i+1} - \mathbf{f} - M\mathbf{u}_{i+1} = 0, \quad (3.23)$$

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 Burger's equation. We will denote this nonlinearity by,

$$\mathcal{N}(\mathbf{y}_{i+1}) := \frac{1}{2}B\mathbf{y}_{i+1}^2, \quad (3.24)$$

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 (3.22) together with the constraint (3.23) fit the framework of (3.1). We can, thus, build the fully discretized Lagrangian function according to the definition (3.2). The discrete Lagrangian of the full-order model is given by,

$$\begin{aligned}
&\mathcal{L}(\mathbf{y}_0,...,\mathbf{y}_{N_t},\mathbf{u}_0,...,\mathbf{u}_{N_t},\boldsymbol{\lambda}_1,...,\boldsymbol{\lambda}_{N_t}) \\
&= \sum_{i=0}^{N_t}\delta t\left(\frac{1}{2}\mathbf{y}_i^T M\mathbf{y}_i - \mathbf{z}^T\mathbf{y}_i + \frac{\omega}{2}\mathbf{u}_i^T M\mathbf{u}_i\right) \\
&\quad + \sum_{i=0}^{N_t-1}\boldsymbol{\lambda}_{i+1}^T\left(\frac{1}{\delta t}M\mathbf{y}_{i+1} - \frac{1}{\delta t}M\mathbf{y}_i + \frac{1}{2}B\mathbf{y}_{i+1}^2 + \nu C\mathbf{y}_{i+1} - \mathbf{f} - M\mathbf{u}_{i+1}\right), \quad (3.25)
\end{aligned}$$

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

We next want to apply the optimization algorithm 2 in order to minimize (3.22) subject to (3.23). Therefore, we will need the corresponding Lagrangian function (3.25) as well as the gradient and the Hessian-times-vector product as described in the algorithms 3 and 4, respectively. We will concentrate on the solution of the two adjoint equations and refer to Appendix A for the numerical solution of Burger's equation.

In the adjoint algorithm 3 that computes the gradient of the Lagrangian (3.25), we mostly need to compute partial derivatives of the constraint (3.23) and the cost functional (3.22) 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 Burger's equation on the whole time interval. In Algorithm 5, we summarize the concrete application of Algoritm 3 to the full-order discrete optimal control problem with Burger's equation as a constraint. Thereby, the adjoint equation (3.8) reduces to an ordinary differential equation for the adjoint variable. Note, that we first need to solve the terminal condition (3.26a) for $\boldsymbol{\lambda}_{N_t}$ and then solve the set of equations (3.26b) backwards in time. Given the solution of (3.26a)-(3.26b), the gradient of the Lagrangian function with respect to the control u can be obtained according to (3.9).

Algorithm 5 Algorithm 3 applied to the full-order discrete Burger's equation

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

2: The adjoint equation (3.8) reads:

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_{N_t}) + \nu C\right)^T \boldsymbol{\lambda}_{N_t} = -\delta t(M\mathbf{y}_{N_t} - \mathbf{z}) \tag{3.26a}$$

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_i) + \nu C\right)^T \boldsymbol{\lambda}_i = -(-\frac{1}{\delta t}M)^T\boldsymbol{\lambda}_{i+1} - \delta t(M\mathbf{y}_i - \mathbf{z}), \quad i = N_t - 1, ..., 1 \tag{3.26b}$$

3: The gradient is computed according to formula (3.9):

$$\nabla_u \hat{\mathcal{L}}(\mathbf{u}_0, ..., \mathbf{u}_{N_t}) = \begin{pmatrix} \delta t\omega M\mathbf{u}_0 \\ \delta t\omega M\mathbf{u}_1 - M^T\boldsymbol{\lambda}_1 \\ \vdots \\ \delta t\omega M\mathbf{u}_{N_t} - M^T\boldsymbol{\lambda}_{N_t} \end{pmatrix} \tag{3.27}$$

In (3.26a) and (3.26b) it is necessary to compute the first derivative of the nonlinear term (3.24). 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 (3.7) we also need to compute the Hessian $\nabla^2\hat{\mathcal{L}}(u)$. Since this is a matrix of dimension $N \cdot N_t \times N \cdot N_t$, we solve the linear system (3.7) with the truncated CG method where we only need to compute the product of the Hessian times a vector, see Algorithm 10. In Algorithm 6, we present the application of the general Hessian-times-vector computation as derived in Section 3.1.2 to the optimization of Burger's equation. Therefore, we define the arbitrary vector $\underline{\mathbf{v}} := (\mathbf{v}_0^T, ..., \mathbf{v}_{N_t}^T)^T$ and derive the equations (3.28a)-(3.28b) and (3.29a)-(3.29b) for the auxiliary variables w and

p according to the respective general formulas (3.17) and (3.18). It is important to note that the initial condition (3.28a) and the terminal condition (**??**) follow directly from the genral equations (3.17) and (3.18) when the respective partial derivative is computed.

Algorithm 6 Algorithm 4 applied to the full-order discrete Burger's 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 5

2: Equation (3.17) reads:

$$\mathbf{w}_0 = 0 \tag{3.28a}$$

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_{i+1}) + \nu C\right)\mathbf{w}_{i+1} = -(-\frac{1}{\delta t}M)\mathbf{w}_i - M\mathbf{v}_{i+1}, \quad i = 0, ..., N_t - 1 \tag{3.28b}$$

3: Equation (3.18) reads:

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_{N_t}) + \nu C\right)^T \mathbf{p}_{N_t} = \delta t M\mathbf{w}_{N_t} + \text{diag}(\boldsymbol{\lambda}_{N_t}^T b_1, ..., \boldsymbol{\lambda}_{N_t}^T b_N)\mathbf{w}_{N_t} \tag{3.29a}$$

$$\left(\frac{1}{\delta t}M + \mathcal{N}'(\mathbf{y}_i) + \nu C\right)^T \mathbf{p}_i = -(-\frac{1}{\delta t}M)^T\mathbf{p}_{i+1} + \delta t M\mathbf{w}_i + \text{diag}(\boldsymbol{\lambda}_i^T b_1, ..., \boldsymbol{\lambda}_i^T b_N)\mathbf{w}_i, \quad i = N_t - 1, ..., 1 \tag{3.29b}$$

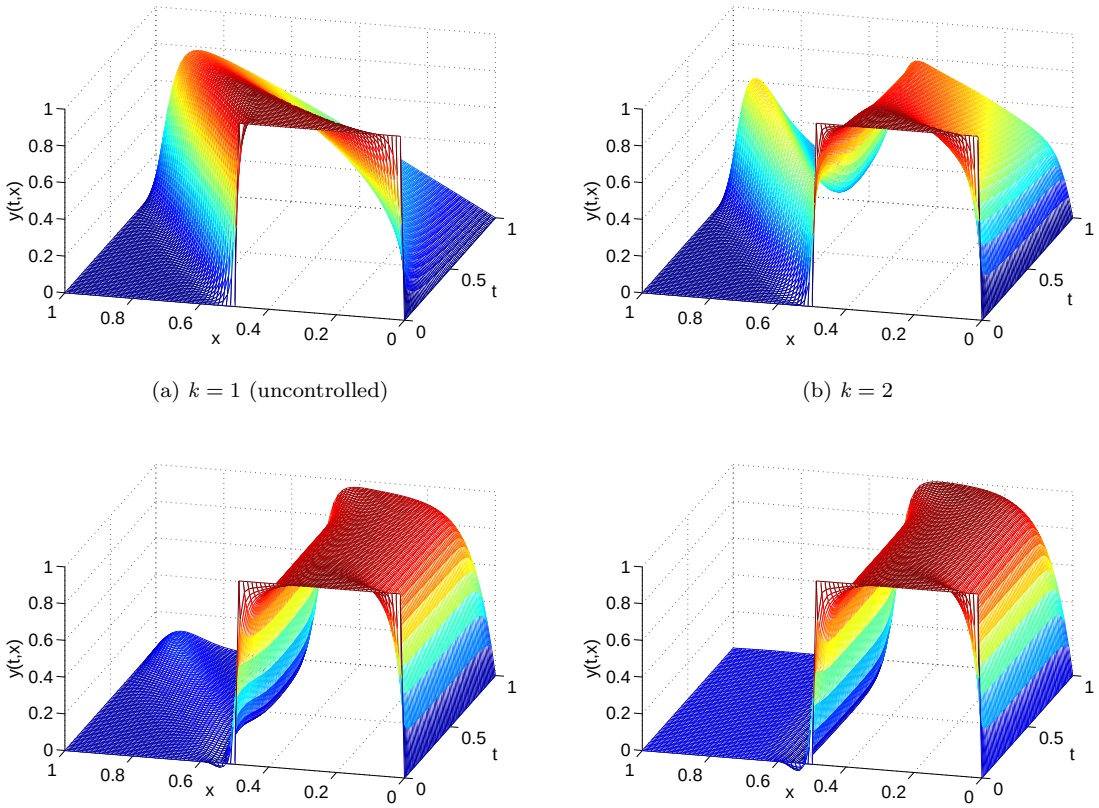
4: The Hessian times a vector $\underline{\mathbf{v}}$ is computed according to formula (3.14):

$$\nabla^2\hat{\mathcal{L}}(\mathbf{u}_0, ..., \mathbf{u}_{N_t}) \cdot \underline{\mathbf{v}} = \begin{pmatrix} \delta t\omega M\mathbf{v}_0 \\ -M^T\mathbf{p}_1 + \delta t\omega M\mathbf{v}_1 \\ \vdots \\ -M^T\mathbf{p}_{N_t} + \delta t\omega M\mathbf{v}_{N_t} \end{pmatrix} \tag{3.30}$$

Note that in (3.29a)-(3.29b) as well as in (3.30) 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 $B = (b_1|...|b_N)$.



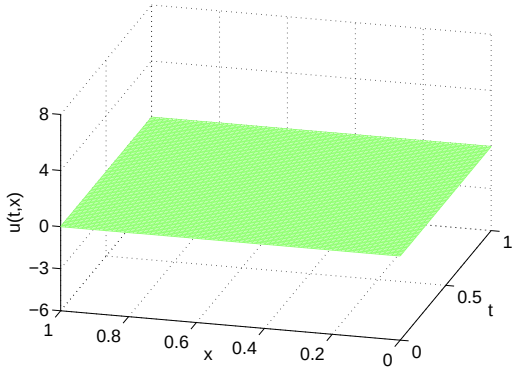
(a) $k = 1$ (uncontrolled)

(b) $k = 2$

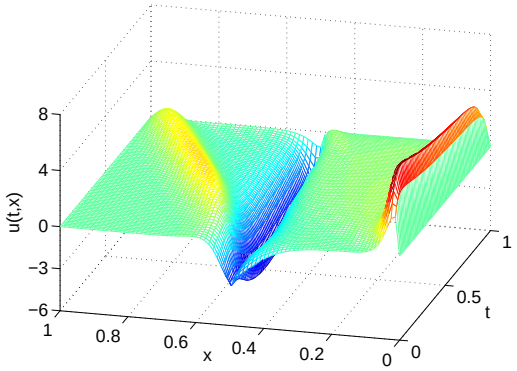
(c) $k = 5$

(d) $k = 10$

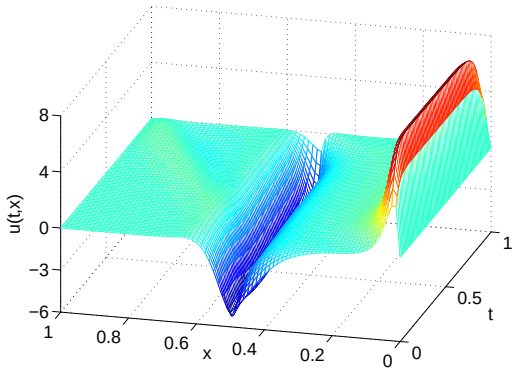
Figure 3.1: Full-order optimization.



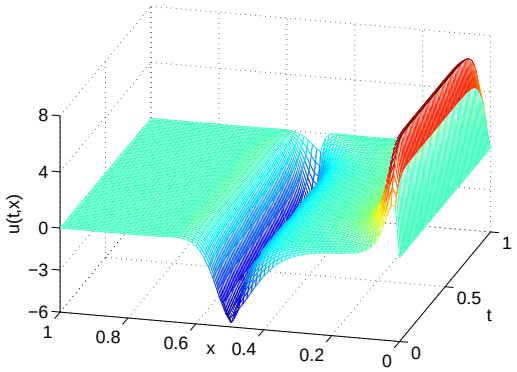
(a) $k = 1$ (initial)



(b) $k = 2$



(c) $k = 5$



(d) $k = 10$

Figure 3.2: Full-order control.

Chapter 4

Advanced model order reduction techniques for PDE-constrained optimization

text

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

We want to apply POD-DEIM to the optimization problem (3.19)-(3.20)

$$\mathbf{y}(t) \approx \Phi_\ell \tilde{\mathbf{y}}(t), \quad \tilde{\mathbf{y}} \in \mathbb{R}^\ell$$
$$\mathbf{u}(t) \approx \Psi_\ell \tilde{\mathbf{u}}(t), \quad \tilde{\mathbf{u}} \in \mathbb{R}^\ell$$

Cost function becomes

$$\min_{\tilde{\mathbf{u}}_0,...,\tilde{\mathbf{u}}_{N_t}} \tilde{f}(\tilde{\mathbf{y}}_0,...,\tilde{\mathbf{y}}_{N_t},\tilde{\mathbf{u}}_0,...,\tilde{\mathbf{u}}_{N_t}) = \min_{\tilde{\mathbf{u}}_1,...,\tilde{\mathbf{u}}_{N_t}} \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2} \tilde{\mathbf{y}}_i^T \tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}^T \tilde{\mathbf{y}}_i + \frac{\omega}{2} \tilde{\mathbf{u}}_i^T \tilde{M} \tilde{\mathbf{u}}_i \right), \quad (4.1)$$

where

$$\tilde{\mathbf{z}} := \Phi_\ell^T \mathbf{z} \in \mathbb{R}^\ell, \quad (4.2)$$
$$\tilde{M} := \Psi_\ell^T M \Psi_\ell \in \mathbb{R}^{\ell \times \ell}. \quad (4.3)$$

The constraint becomes

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

where

$$\tilde{\tilde{M}} := \Phi_\ell^T M \Psi_\ell \in \mathbb{R}^{\ell \times \ell}, \quad (4.5)$$

and $\tilde{B}, \tilde{C}, \tilde{F}$ as defined in (2.20), (2.21), (2.22), respectively.

nonlinearity

$$\tilde{\mathcal{N}}(\tilde{\mathbf{y}}_{i+1}) := \frac{1}{2} \tilde{B}(\tilde{F} \tilde{\mathbf{y}}_{i+1})^2 \quad (4.6)$$

The Lagrangian function of the reduced model is, thus, given by

$$\begin{aligned}&\tilde{\mathcal{L}}(\tilde{\mathbf{y}}_0, ..., \tilde{\mathbf{y}}_{N_t}, \tilde{\mathbf{u}}_0, ..., \tilde{\mathbf{u}}_{N_t}, \tilde{\boldsymbol{\lambda}}_1, ..., \tilde{\boldsymbol{\lambda}}_{N_t}) \\ &= \sum_{i=0}^{N_t} \delta t \left(\frac{1}{2} \tilde{\mathbf{y}}_i^T \tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}^T \tilde{\mathbf{y}}_i + \frac{\omega}{2} \tilde{\mathbf{u}}_i^T \tilde{M} \tilde{\mathbf{u}}_i \right) \\ &\quad + \sum_{i=0}^{N_t - 1} \tilde{\boldsymbol{\lambda}}_{i+1}^T \left(\frac{1}{\delta t} \tilde{\mathbf{y}}_{i+1} - \frac{1}{\delta t} \tilde{\mathbf{y}}_i + \frac{1}{2} \tilde{B}(\tilde{F} \tilde{\mathbf{y}}_{i+1})^2 + \nu \tilde{C} \tilde{\mathbf{y}}_{i+1} - \tilde{\mathbf{f}} - \tilde{\tilde{M}} \tilde{\mathbf{u}}_{i+1} \right)\end{aligned} \tag{4.7}$$

Note, that even $\tilde{\boldsymbol{\lambda}}_i \in \mathbb{R}^\ell$. If we choose $k = \ell$, then (4.7) only depends on ℓ.

Algorithm 7 Algorithm 3 applied to the reduced Burger's model

1: From the initial condition $\tilde{\mathbf{y}}_0$ and the current control $\tilde{\mathbf{u}}_0, ..., \tilde{\mathbf{u}}_{N_t}$, solve the reduced Burger's equation for $\tilde{\mathbf{y}}_1, ..., \tilde{\mathbf{y}}_{N_t}$ as described in Section 2.3

2: The adjoint equation (3.8) reads:

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_{N_t}) + \nu \tilde{C} \right)^T \tilde{\boldsymbol{\lambda}}_{N_t} = -\delta t (\tilde{\mathbf{y}}_{N_t} - \tilde{\mathbf{z}}) \tag{4.8a}$$

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_i) + \nu \tilde{C} \right)^T \tilde{\boldsymbol{\lambda}}_i = -(-\frac{1}{\delta t} I_\ell)^T \tilde{\boldsymbol{\lambda}}_{i+1} - \delta t (\tilde{\mathbf{y}}_i - \tilde{\mathbf{z}}), \quad i = N_t - 1, ..., 1 \tag{4.8b}$$

3: The gradient is computed according to formula (3.9):

$$\nabla_{\tilde{u}} \hat{\tilde{\mathcal{L}}}(\tilde{\mathbf{u}}_0, ..., \tilde{\mathbf{u}}_{N_t}) = \begin{pmatrix} \delta t \omega \tilde{M} \tilde{\mathbf{u}}_0 \\ \delta t \omega \tilde{M} \tilde{\mathbf{u}}_1 - \tilde{\tilde{M}}^T \tilde{\boldsymbol{\lambda}}_1 \\ \vdots \\ \delta t \omega \tilde{M} \tilde{\mathbf{u}}_{N_t} - \tilde{\tilde{M}}^T \tilde{\boldsymbol{\lambda}}_{N_t} \end{pmatrix} \tag{4.9}$$

Here, I_ℓ is the $\ell \times \ell$ identity matrix and,

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

where $\tilde{F}_i$ denotes the ith row of the matrix $\tilde{F}$ and $\tilde{F}_{i,j}$ denotes the respective entry of the matrix.

Next, Hessian times vector $\nabla^2 \hat{\tilde{\mathcal{L}}} \cdot \underline{\tilde{\mathbf{v}}}$, where

$$\underline{\tilde{\mathbf{v}}} := \begin{pmatrix} \tilde{\mathbf{v}}_0 \\ \vdots \\ \tilde{\mathbf{v}}_{N_t} \end{pmatrix} \in \mathbb{R}^{(\ell_2 \cdot N_t) \times 1} \tag{4.10}$$

Algorithm 8 Algorithm 4 applied to the reduced Burger's model

1: We assume that we have already computed $\tilde{\mathbf{y}}_0, ..., \tilde{\mathbf{y}}_{N_t}, \tilde{\mathbf{u}}_0, ..., \tilde{\mathbf{u}}_{N_t}, \tilde{\boldsymbol{\lambda}}_1, ..., \tilde{\boldsymbol{\lambda}}_{N_t}$ in Algorithm 7

2: Equation (3.17) reads:

$$\tilde{\mathbf{w}}_0 = 0 \tag{4.11a}$$

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_{i+1}) + \nu \tilde{C}\right) \tilde{\mathbf{w}}_{i+1} = -(-\frac{1}{\delta t} I_\ell)\tilde{\mathbf{w}}_i - \tilde{\tilde{M}}\tilde{\mathbf{v}}_{i+1}, \quad i = 0, ..., N_t - 1 \tag{4.11b}$$

3: Equation (3.18) reads:

$$\left(\frac{1}{\delta t} I_\ell + \tilde{\mathcal{N}}'(\tilde{\mathbf{y}}_{N_t}) + \nu \tilde{C}\right)^T \tilde{\mathbf{p}}_{N_t} = \delta t \tilde{M}\tilde{\mathbf{w}}_{N_t} + \left(\tilde{\boldsymbol{\lambda}}_{N_t}^T \tilde{\mathcal{N}}(\tilde{\mathbf{y}}_{N_t})\right)'' \tilde{\mathbf{w}}_{N_t} \tag{4.12a}$$

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

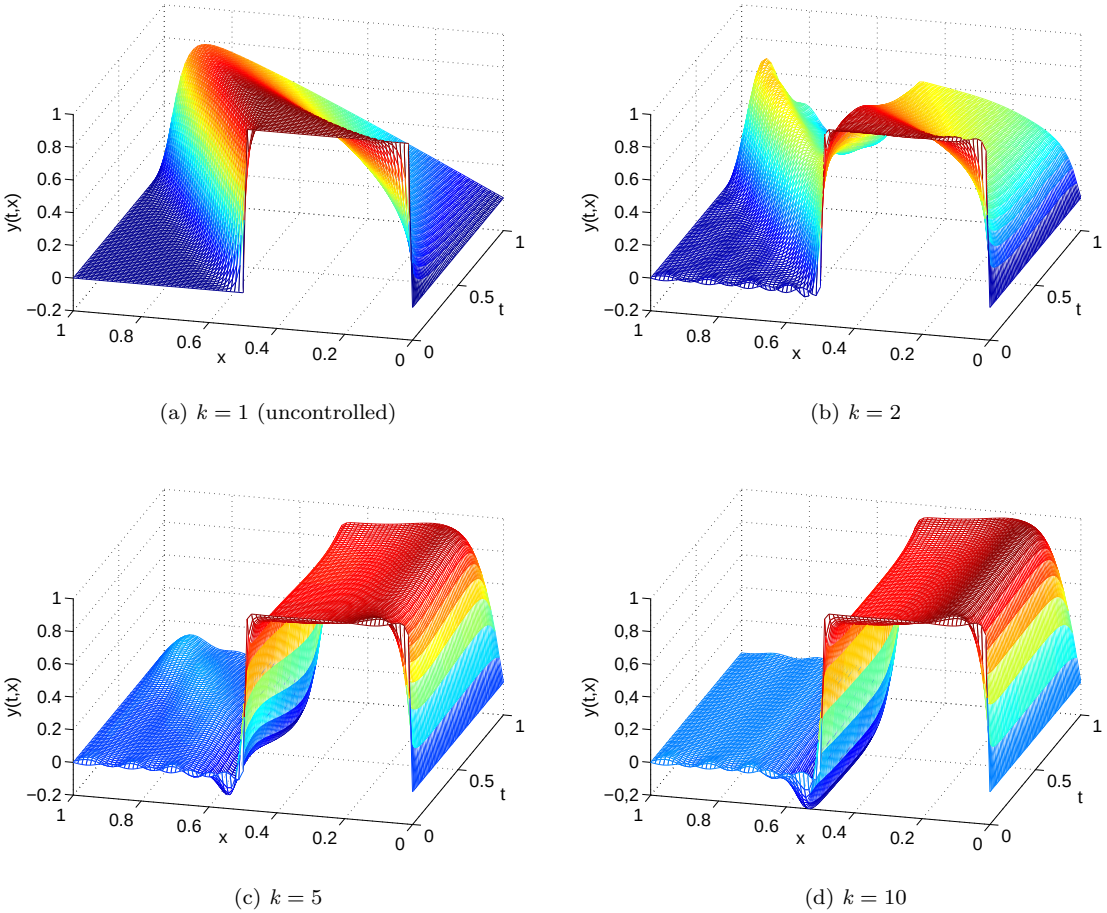
4: The Hessian times a vector $\underline{\tilde{\mathbf{v}}}$ is computed according to formula (3.14):

$$\nabla^2 \hat{\tilde{\mathcal{L}}}(\tilde{\mathbf{u}}_0, ..., \tilde{\mathbf{u}}_{N_t})\underline{\tilde{\mathbf{v}}} = \begin{pmatrix} \delta t \omega \tilde{M}\tilde{\mathbf{v}}_0 \\ -\tilde{\tilde{M}}^T \tilde{\mathbf{p}}_1 + \delta t \omega \tilde{M}\tilde{\mathbf{v}}_1 \\ \vdots \\ -\tilde{\tilde{M}}^T \tilde{\mathbf{p}}_{N_t} + \delta t \omega \tilde{M}\tilde{\mathbf{v}}_{N_t} \end{pmatrix} \tag{4.13}$$

Note that second partial derivative of $\tilde{\mathcal{L}}$ with respect to $\tilde{\mathbf{y}}$ in line 3 of the above algorithm is given by

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

This matrix is not constant in time because the adjoint variable depends on time...



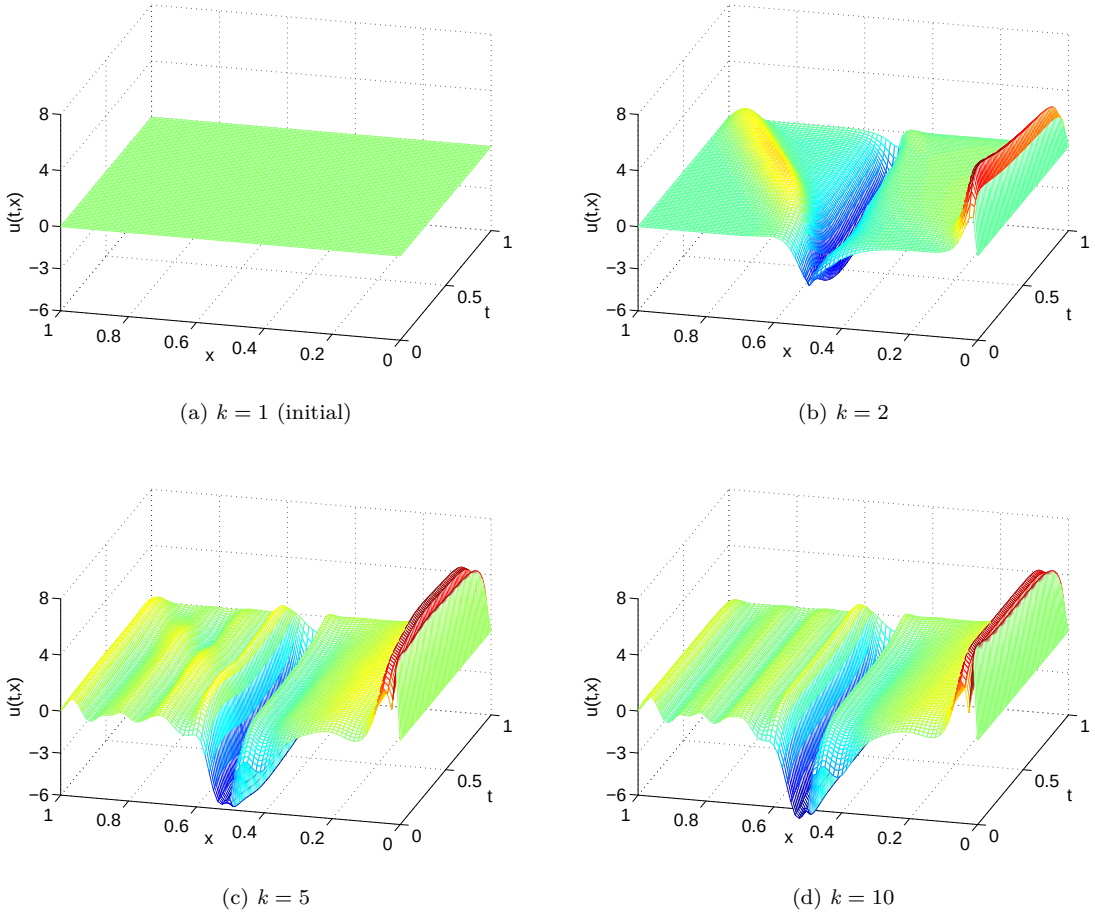
(a) $k = 1$ (uncontrolled)

(b) $k = 2$

(c) $k = 5$

(d) $k = 10$

Figure 4.1: Reduced-order optimization.



(a) $k = 1$ (initial)

(b) $k = 2$

(c) $k = 5$

(d) $k = 10$

Figure 4.2: Reduced-order control.

4.2 Parameter study of the reduced model

4.3 Performance and error analysis

Chapter 5

Summary and Outlook

bla

Bibliography

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

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

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

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

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

[6] 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.

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

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

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

[10] 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/scripts.php`.

Appendices

Appendix A

Numerical solution of Burger's equation

A.1 Spacial discretization via the finite element method

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

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

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

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

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 [3] 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 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 part 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 tat the linear ansatz function are only non-zero at two cells, the Trapezoidal rule [1] 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 ODE system:

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

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 $\Phi(x)$ is equal to a step function can be derived straight forward since the test functions are equal to 1 at the grid points:

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

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

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

where $\mathbf{y}^{(n)} = \mathbf{y}(t_n)$ and τ is the time step.

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 9). 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

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

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 of the relative error of the Newton iteration (see line 7).

Algorithm 9 Euler implicit with Newton's method

1: Initialize $\mathbf{y}^{(1)} = \mathbf{y}_0$, TOL
2: **for** $n = 1$ to T **do**
3: $\quad\mathbf{u}_{old} = \mathbf{y}^{(n)}$
4: $\quad err = 1$
5: $\quad$**while** $err > TOL$ **do**
6: $\quad\quad$Solve $\mathbf{u}_{new} = \mathbf{u}_{old} - J_F^{-1}\mathbf{F}(\mathbf{u}_{old})$
7: $\quad\quad err = \|\mathbf{u}_{new} - \mathbf{u}_{old}\|_2/\|\mathbf{u}_{new}\|_2$
8: $\quad$**end while**
9: $\quad\mathbf{y}^{(n+1)} = \mathbf{u}_{new}$
10: **end for**

Appendix B

Implementation issues

B.1 The truncated conjugant gradient (CG) method

$$\|\nabla^2 \hat{\mathcal{L}}(u_k) + \nabla \hat{\mathcal{L}}(u_k)\|_2 \leq \eta_k \|\nabla \hat{\mathcal{L}}(u_k)\|_2, \tag{B.1}$$

$\eta_k \in (0, 1)$

We assume outer iteration k

Algorithm 10 Function evaluation based version of the truncated CG algorithm in order to solve the Newton equation $\nabla^2\hat{\mathcal{L}}(u_k)s_k = -\nabla\hat{\mathcal{L}}(u_k)$, [5]

INPUT: Function handle that evaluates the matrix-vector product $\nabla^2\hat{\mathcal{L}}(u_k)\cdot v$, right-hand side $\nabla\hat{\mathcal{L}}(u_k)$, `MAXITER`, tolerance η_k
OUTPUT: Solution of Newton's equation s_k
Set $s_k = 0, p_k^{(0)} = r_k^{(0)} = -\nabla\hat{\mathcal{L}}(u_k)$
for $i = 0, 1, 2, ...,$`MAXITER` **do**
  if $\|r_k^{(i)}\|_2 < \eta_k\|r_k^{(0)}\|_2$ **then**
    if $i = 0$ **then**
      $s_k = -\nabla\hat{\mathcal{L}}(u_k)$ *% Steepest descent direction*
      return
    end if
  end if
  Compute $q_k^{(i)} = \nabla^2\hat{\mathcal{L}}(u_k)\cdot p_k^{(i)}$
  if $(p_k^{(i)})^T q_k^{(i)} < 0$ **then**
    if $i = 0$ **then**
      $s_k = -\nabla\hat{\mathcal{L}}(u_k)$ *% Steepest descent direction*
      return
    end if
  end if
  Compute $\gamma_k^{(i)} = \|r_k^{(i)}\|_2^2/(p_k^{(i)})^T q_k^{(i)}$
  Update solution, $s_k = s_k + \gamma_k^{(i)} q_k^{(i)}$
  Compute $r_k^{(i+1)} = r_k^{(i)} - \gamma_k^{(i)} q_k^{(i)}$
  Compute $\beta_k^{(i)} = \|r_k^{(i+1)}\|_2^2/\|r_k^{(i)}\|_2^2$
  Compute $p_k^{(i+1)} = r_k^{(i+1)} + \beta_k^{(i)} p_k^{(i)}$
end for

B.2 Armijo line-search

Algorithm 11 Armijo line-search algorithm, [7]

1: **INPUT:** initial point u_0, search direction s, $tol > 0$
2: **OUTPUT:** optimal step size α in direction s
3: Set $\alpha = 1$ and compute $f(u_0 + \alpha s)$
4: **while** $f(u_0 + \alpha s) > f(u_0) + tol \cdot \alpha \cdot s^T\nabla f(u_0)$ **do**
5: Set $\alpha := \alpha/2$
6: Compute $f(u_0 + \alpha s)$
7: **end while**

B.3 Matlab code

The numerical test calculations for the POD-DEIM model of Burger's equation presented in Section 2.3 as well as the optimal control algorithm discussed in Section 3.3 and Section 4.1 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

Nomenclature

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

y	State variable
u	Control variable
λ	Adjoint variable
ν	Viscosity parameter in Burger's equation
ℓ	POD-dimension
m	DEIM-dimension
Ω	Spatial domain
$[0, T]$	Time domain
N	Spatial discretiation size of the full model
N_t	Time discretization size
$\mathcal{P}$	DEIM projection matrix
$\mathcal{L}$	Lagrangian function
$\hat{\mathcal{L}}$	The same Lagrangian as a function of the control only
ω	Control penalty, we choose $\omega \in (0, 1)$
M, B, C	Constant matrices derived from the spatial discretization of Burger's equation