Model order reduction with applications

Bošnjak, Domagoj

Model order reduction with applications

Bošnjak, Domagoj

UNIVERSITY OF ZAGREB

FACULTY OF SCIENCE

DEPARTMENT OF MATHEMATICS

Domagoj Bošnjak

MODEL ORDER REDUCTION WITH APPLICATIONS

Diploma Thesis

Thesis supervisor:
prof.dr.sc. Zlatko Drmač

Zagreb, 2021

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

To my sister Josipa for her endless support, to my brother Marko for his invaluable guidance, and to my parents for remembering every single test.

Contents

Introduction

The growing need for solving large-scale systems is not accompanied by a proportional development in quality and availability of appropriately powerful and affordable hardware. As complex as systems of partial and ordinary differential equations already are, they are made even more difficult to solve when introducing tunable parameters and nonlinearities. Neither of those circumstances can be avoided when developing methods for tackling real-life problems in applied sciences and engineering. Available approaches to such problems often end up using high-dimensional models in order to remain sufficiently accurate. This has inspired the development of *model order reduction*. It is an approach that seeks to convert a high-order (high-dimensional) system to a small-order one, by keeping the *most important* variables intact. The emphasis is on keeping an adequately accurate model whilst reducing both computation time and required resources. The systems we deal with are systems of ordinary and partial differential equations. Even though a plethora of methods and approaches exist, they may not perform well in the case of high dimensionality, especially in situations when limited hardware is available.

The rest of the thesis is organized as follows. The first chapter covers the topic of parametric differential equations, starting with a motivational example. A short overview of the weak formulation is given, as well as the method of solving such problems, namely the *proper orthogonal decomposition* (POD). The aforementioned example is then solved using this method. The chapter ends with an alternative - the *greedy basis generation*. Nonlinear parametric partial differential equations are tackled in the following chapter. The issue with using a POD-only approach is discussed in detail, as well as the way to solve it, i.e. the *discrete empirical interpolation method*. An extensive theoretical background is provided as well. Additionally, two potential improvements of the said method, called the Q-DEIM and K-DEIM, are presented next. The chapter ends with a synthetic example. The final chapter combines the POD and DEIM methods so as to solve a model example - a nonlinear PDE by the name of *Fisher-KPP equation*. Appendices at the end include MATLAB codes for the mentioned examples, as well as several general results which are omitted from the main chapters, so as not to draw attention away from the main theme.

1 Parametric Partial Differential Equations and POD

Many problems in applied sciences and engineering are modeled by partial differential equations. Often, such a mathematical model, expressed by partial differential equation, includes physical parameters (e.g. material properties) that take values in their respective domains, usually real segments. The task is, for example, to optimize the solution and obtain some desirable properties as function of the physical parameters. A model example is the heat conduction problem described by the following equation

$$\nabla(\kappa_\mu \nabla u(\mu)) = 0,$$

where

$$k_\mu(x,y) = \begin{cases} \mu_1, & (x,y) \text{ in } B(0,0.5) \\ 1, & (x,y) \text{ in } B(0,0.5)^c \end{cases},$$

where $B(0,0.5)$ denotes the ball centered around $(0,0)$ with the radius $r = 0.5$. The parameter μ_1 may, in this case, lie in the interval $[0.1, 10]$. As solving PDEs and systems of PDEs may be computationally taxing enough by itself, solving multiple times with particular choices of parameters is a time-consuming computational task. Moreover, this example, for now, depends only on a single parameter. In case of several different parameters, the computational task becomes even more difficult if not nearly untractable.

However, as much as variable parameters present a problem, the different solutions $u(\mu)$, $\mu \in \mathbb{P}$ (where $\mathbb{P}$ is the parameter space) are in fact connected because they represent variations of the same model. More specifically, we do not expect them to lie arbitrarily in an infinite-dimensional space, but rather in (or close to) a low-dimensional space or manifold. This assumption led to the development of *reduced basis methods*, where the goal is to find a basis that adequately describes that low-dimensional space. In case we manage to find such a basis, it is no longer necessary to solve the entire problem all over again when a solution for a different parameter choice is required. Instead, we would obtain the new wanted solution using the reduced basis.

Here we make use of the *Proper Orthogonal Decomposition* (POD), so as to compute the reduced basis of the space in which different solutions $u(\mu)$ lie. As usual

with PDEs, we start with the weak formulation after which we transition to the POD method and afterwards explain the aforementioned example in more detail and we solve it. An alternative to POD is presented at the end of the chapter, namely the *greedy basis generation*.

1.1 Weak Formulation of Parametric Partial Differential Equations

Let us introduce the abstract version of the problem, i.e. the weak formulation of the parametric partial differential equations. Most of the conclusions in this chapter originate from [1] and [4]. We consider an appropriate (regular, bounded) domain $\Omega \subset \mathbb{R}^n, n \in \{1, 2, 3\}$, with a suitably regular boundary $\partial\Omega$. If we consider scalar functions, the corresponding test functions reside in the Sobolev space $H^1(\Omega)$, which is slightly modified if we seek to impose Dirichlet boundary conditions on the partial differential equation at hand, so it becomes $H_0^1(\Omega)$. More on Sobolev spaces may be found in [15]. In the case of multivariable functions with domain dimension $d_v = 2, 3$, we divide the boundary $\partial\Omega$ into boundary measurable segments for Dirichlet boundary conditions, denoted by $\Gamma_i^D, i = 1, \ldots, d_v$. We then introduce the spaces:

$$\mathbb{V}_i = \{v \in H^1(\Omega) : v\big|_{\Gamma_i^D} = 0\} \subseteq H^1(\Omega),$$

for $i = 1, \ldots, d_v$. Finally, we make use of the Hilbert space:

$$\mathbb{V} = \mathbb{V}_1 \times \cdots \times \mathbb{V}_{d_v},$$

equipped with an inner product whose induced norm must be equivalent with the $(H^1(\Omega))^{d_v}$ norm, which additionally ensures that $\mathbb{V}$ is indeed a Hilbert space. The standard notation for the inner product and the corresponding induced norm is used:

$$\langle v, w\rangle_{\mathbb{V}}, \quad \|v\|_{\mathbb{V}} = \sqrt{\langle v, v\rangle_{\mathbb{V}}}.$$

Finally, we introduce a parameter space $\mathbb{P} \subset \mathbb{R}^p$. In that case, the standard variational equation

$$a(u, v) = f(v), \quad v \in \mathbb{V},$$

where $f(v) = (f, v)_{\mathbb{V}}$, becomes

$$a(u(\mu), v; \mu) = f(v; \mu), \quad v \in \mathbb{V},$$

where $a : \mathbb{V} \times \mathbb{V} \times \mathbb{P} \to \mathbb{R}$ denotes the bilinear form (with respect to the two first variables) and $f : \mathbb{V} \times \mathbb{P} \to \mathbb{R}$ denotes the right hand side, linear with respect to the first variable. Alongside the variational equation, the model may also contain an output function $s(\mu) = \ell(u(\mu); \mu)$.

In order to guarantee the well-posedness of the problem we introduce the standard assumptions for the use of the Lax-Milgram Lemma, i.e. we assume that the bilinear form a is coercive and continuous for all $\mu \in \mathbb{P}$. More precisely, we assume:

$$\forall \mu \in \mathbb{P}, \ \exists\, a(\mu) > 0, \quad a(u, u; \mu) \geq a(\mu)\|u\|_{\mathbb{V}}^2$$

and

$$\forall \mu \in \mathbb{P}, \ \exists\, \gamma(\mu) < \infty, \quad a(u, v; \mu) \leq \gamma(\mu)\|u\|_{\mathbb{V}}\|v\|_{\mathbb{V}}$$

which holds for all $u, v \in \mathbb{V}$. Moreover, we assume that f is continuous with respect to the first variable for all $\mu \in \mathbb{P}$, i.e.

$$\forall \mu \in \mathbb{P}, \quad \exists\, \epsilon(\mu) < \infty, \quad f(u; \mu) \leq \epsilon(\mu)\|u\|_{\mathbb{V}},$$

for all $u \in \mathbb{V}$. The coercivity assumption on a for all parameters μ implies that $a(\cdot, \cdot; \mu)$ introduces an inner product on $\mathbb{V}$, whose induced norm is equivalent to the $\|\cdot\|_{\mathbb{V}}$ norm. For a fixed parameter $\mu \in \mathbb{P}$, a standard finite element discretization technique can be performed through the use of a discrete (finite-dimensional) approximation space $\mathbb{V}_\delta$, an appropriate basis $\{\varphi_1, \ldots, \varphi_{N_\delta}\}$, and the enforcement of Galerkin orthogonality. The parameter δ depends on the discretization used to obtain $\mathbb{V}_\delta$. More on PDE discretization methods may be found in [16].

1.2 Proper Orthogonal Decomposition

We assume that we have obtained N snapshots, i.e. solutions $\{u(\mu_i)\}_{i=1}^N$ for different parameter values $\mu_1, \ldots, \mu_N$. Such snapshots may be obtained by solving the variational problem N times. However, solving N times for a large N and with high fidelity

discretization requires additional attention in order to be computationally tractable. Here we assume that there is an off-line phase, when we can afford time and computational resources to generate data snapshots. This is often the case in applications. Additionally, in the case of a discretization, we assume that all snapshots lie in $\mathbb{V}_\delta$, i.e. there is no need for additional refinement and all snapshots exist over the same mesh. The *Proper Orthogonal Decomposition* (POD) is a method of obtaining a *reduced basis* through the use of such snapshots. The end goal is to approximate future solutions of interest by obtaining M basis vectors from the N snapshots, where we naturally aim to achieve $M \ll N$.

We denote the snapshots with $y_1, \dots, y_N \in \mathbb{V}$. The goal is to find the reduced basis functions $\varphi_1, \dots, \varphi_M \in \mathbb{V}$ by solving the following minimization problem:

$$\begin{cases} \min\limits_{\varphi_1,\dots,\varphi_M \in \mathbb{V}} \sum_{n=1}^{N} \left\| y_n - \sum_{m=1}^{M} (y_n, \varphi_m)_{\mathbb{V}} \, \varphi_m \right\|_{\mathbb{V}}^2, \\ (\varphi_i, \varphi_j)_{\mathbb{V}} = \delta_{ij}, \quad \forall i, j \in \{1, \dots, M\}, \end{cases} \tag{1.1}$$

where M satisfies $1 \leq M \leq N$.

The solutions of the minimization problem can be obtained through the use of the eigenvalue decomposition of the *Gramian matrix* G of the snapshots, given elementwise by:

$$G_{ij} = (y_i, y_j)_{\mathbb{V}}.$$

The eigendecomposition of the matrix G yields eigenvalues $\lambda_1, \dots, \lambda_N$ and eingevectors $v^1, \dots, v^N$. Since the matrix G is Hermitian and positive semidefinite, it follows that $\lambda_i \in \mathbb{R}$, and $v_i \in \mathbb{R}^N$, for all $i = 1, \dots, N$. We denote by d the index of the last non-zero eigenvalue, where we assume the ordering

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d > 0 = \lambda_{d+1} = \cdots = \lambda_N.$$

Naturally, in applications, a tolerance ε relatively small in comparison to e.g. λ_1 or λ_d would be chosen as the threshold. Now M may be chosen in $\{1, \dots, d\}$.

Finally, the starting POD functions are now given by:

$$\varphi_k = \sum_{n=1}^{N} \frac{v_n^k}{\sqrt{\lambda_k}} \, y_n, \quad k = 1, \dots, M.$$

We provide the proof for the case $\mathbb{V} = \mathbb{R}^K$. Denote the snapshot matrix by

$$Y = \Big[y_1 \, y_2 \, \dots y_N \Big].$$

We consider the singular value decomposition of the matrix Y

$$Y = U\Sigma V^T = U \begin{bmatrix} \Sigma_d & 0 \\ 0 & 0 \end{bmatrix} V^T,$$

where $\Sigma_d \in \mathbb{R}^{d\times d}$ is the diagonal matrix of non-zero singular values $\sigma_1, \dots, \sigma_d$. Additionally, we denote by u_i and v_i the columns of U and V, respectively. Using the orthogonality of U and V it follows that

$$U^T Y V = \begin{bmatrix} \Sigma_d & 0 \\ 0 & 0 \end{bmatrix}.$$

Now we note that, for $i \in \{1, \dots, d\}$

$$Y v_i = \sigma_i v, \quad Y^T u_i = \sigma_i v_i.$$

Combining the two we obtain, again for $i \in \{1, \dots, d\}$

$$Y Y^T u_i = \sigma_i^2 u_i, \quad Y^T Y = \sigma_i^2 v_i.$$

Returning to the minimization problem 1.1, we introduce the appropriate Lagrange functional

$$L(\varphi_1, \dots, \varphi_M, \lambda_{11}, \lambda_{12}, \dots \lambda_{MM}) = \sum_{n=1}^{N} \Big\| y_n - \sum_{m=1}^{M} (y_n, \varphi_m)_{\mathbb{V}} \, \varphi_m \Big\|_{\mathbb{V}}^2 + \sum_{i,j=1}^{M} \lambda_{ij} ((\varphi_i, \varphi_j)_{\mathbb{V}} - \delta_{ij}),$$

and consider the optimality conditions

$$\frac{\partial}{\partial \varphi_i} L = 0 \iff \sum_{m=1}^{M} (y_m, \varphi_i) y_m = \lambda_{ii} \varphi_i \text{ and } \lambda_{ij} = 0, \quad i \neq j,$$

$$\frac{\partial}{\partial \lambda_{ij}} L = 0 \iff (\varphi_i, \varphi_j) = \delta_{ij}$$

This yields, with the notation $\lambda_i = \lambda_{ii}$:

$$YY^T\varphi_i = \lambda_i\varphi_i, \quad i = 1, \dots, M.$$

This means that the required solution consists of $u_1, \dots, u_M$, where $M \leq d = rank(Y)$. Since our assumption is that the dimension of the space is much larger than the number of snapshots, the snapshot matrix Y has significantly more rows than columns. That is why the computation starts with the eigendecomposition of the matrix Y^TY, yielding:

$$Y^TY = \sigma_i^2 v_i \text{ and } u_i = \frac{1}{\sigma_i} Y v_i,$$

which is exactly what the statement was, since the Gramian G is Y^TY, and $\sigma_i = \sqrt{\lambda_i}$.

Remark 1.2.1 *If the scalar product $\langle \cdot, \cdot \rangle_{\mathbb{V}}$ is replaced by the usual l^2 scalar product, the POD basis vectors can be obtained directly by taking the first M left singular vectors of the snapshot matrix, i.e. the singular vectors which correspond to the M largest singular values. Additionally, M is chosen such that $M \leq d$, where d denotes the largest index such that $\sigma_d \neq 0$, meaning we consider the singular vectors which correspond to non-zero singular values.*

$$Y = [y_1, \dots, y_N] \overset{SVD}{=} U\Sigma V,$$

and the denoting the first M columns of U with $U(:, 1 : M)$ we get:

$$[\varphi_1, \dots, \varphi_M] = U(:, 1 : M).$$

Subsequently, we seek to approximate the solution $u(\cdot\,; \mu)$, for a fixed parameter $\mu \in \mathbb{P}$ as:

$$u(\mu) = \sum_{m=1}^{M} p_m \varphi_m,$$

so the final step includes determining the POD coefficients $\{p_m(\mu) \colon m = 1, \dots, M\}$.
Let us denote the POD space as $\mathbb{V}_{POD} = \text{span}\{\varphi_1, \dots, \varphi_M\}$. A natural choice of coefficients stems from projecting $u(\mu)$ onto $\mathbb{V}_{POD}$:

$$P^M(u(\mu)) = \sum_{m=1}^{M} (\varphi_m, u)_{\mathbb{V}}\, \varphi_m,$$

for any $M = 1, \dots, d$. We now limit M from above with d, rather than N, since there

are d non-zero eigenvalues of the Gramian matrix G, similarly as in the Remark 1.2.1. The POD coefficients for a chosen function u are then given by:

$$p_m = (\varphi_m, u)_{\mathbb{V}}, \quad m = 1, \ldots, M.$$

This choice of POD coefficients is additionally motivated by the optimality of the projection in the following sense:

$$\|u - P^d(u)\|_{\mathbb{V}} = \inf_{v \in \mathbb{V}_{POD}} \|u - v\|_{\mathbb{V}}.$$

Finally, we apply the aforementioned procedure to obtain the final model of the form: for a given $\mu \in \mathbb{P}$, find $u(\mu) \in \mathbb{V}_{POD}$ which satisfies

$$a(u(\mu), v; \mu) = f(v; \mu), \quad v \in \mathbb{V}_{POD}.$$

Since $u(\mu)$ may be represented through the POD coefficients as $u(\mu) = \sum_{m=1}^{M} p_m \varphi_m$, it is enough to determine the POD coefficient vector $p = (p_1, \ldots, p_m)$. For a fixed μ, this can be achieved by testing the equation for POD functions, i.e.

$$\sum_{i=1}^{M} a(\varphi_i, \varphi_m; \mu)\, p_i = f(\varphi_m; \mu), \quad m = 1, \ldots, M,$$

where it is important to emphasize that p depends on μ, i.e. $p = p(\mu)$.

Remark 1.2.2 *A common application of POD involves using the SVD to obtain the reduced basis. However, since the resulting basis will naturally heavily depend on the choice of the inner product $\langle \cdot\,, \cdot \rangle$, it makes sense to modify it according to a specific problem. Inner product is related to the concept of energy, which is why it makes sense to use a* weighted *inner product instead, depending on the problem.*

1.3 Parametric PDE Example

In this section we consider a heat conduction problem modeled by a parametric partial differential equation. The example originates from [1]. The domain of the problem is $\Omega = \langle -1, 1 \rangle \times \langle -1, 1 \rangle$, with the boundary sides denoted by $\Gamma_{top}, \Gamma_{bottom}, \Gamma_{left}$ and

Γ_{right}. The partial differential equation together with boundary conditions is given by:

$$\begin{cases} \nabla(\kappa_\mu \nabla u(\mu)) = 0 & \text{in } \Omega, \\ u(\mu) = 0 & \text{in } \Gamma_{top}, \\ \kappa_\mu \nabla u(\mu) \cdot \vec{n} = 0, & \text{on } \Gamma_{left} \cup \Gamma_{right}, \\ \kappa_\mu \nabla u(\mu) \cdot \vec{n} = \mu_2, & \text{on } \Gamma_{bottom}. \end{cases} \tag{1.2}$$

There are two parameters in the model, μ_1 and μ_2. The dependence on the parameter μ_1 is given through the function k_μ which is interpreted as the thermal conductivity, in the following way:

$$k_\mu(x, y) = \begin{cases} \mu_1, & (x, y) \text{ in } B(0, 0.5) \\ 1, & (x, y) \text{ in } B(0, 0.5)^c \end{cases},$$

where $B(0, 0.5)$ denotes the ball centered around $(0, 0)$ with the radius $r = 0.5$. The dependence on the parameter μ_2 is given through the Neumann boundary condition on $\Gamma_{bottom} = \{-1\} \times \langle -1, 1 \rangle$. The value intervals for the parameters are given by $\mu_1 \in [0.1, 10]$ and $\mu_2 \in [-1, 1]$. An example of a solution together with the mesh is shown in the Figure 1.1. Note that the effect of different values of the parameter-dependent function k_μ inside and outside of $B(0, 0.5)$ is clearly visible.

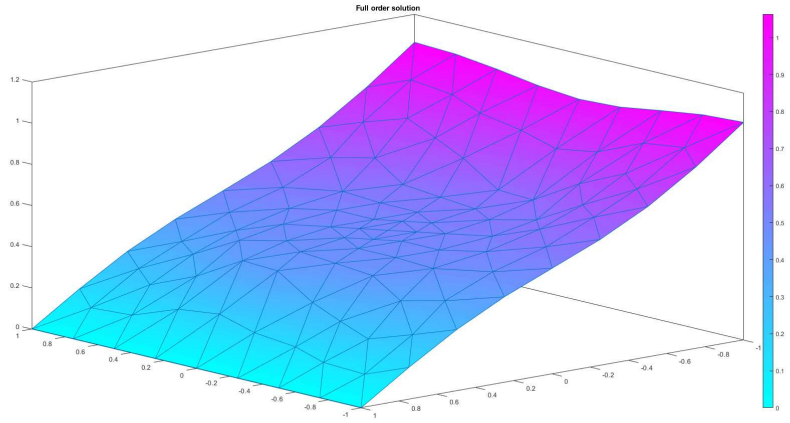

Figure 1.1: A solution of the parametrized PDE (1.2): areas of different thermal conductivity k_μ are clearly visible. The parameter values are $\mu_1 = 9$, $\mu_2 = 0.75$.

The snapshot generation for solving the problem by the POD method of snapshots is performed by choosing n equidistant parameter values in each parameter range,

yielding n^2 snapshots. The implementation related to assembling the finite element matrices is greatly supplemented both through code and conclusions from [2]. From there on, the algorithm essentially follows the procedure described in the previous subsection.

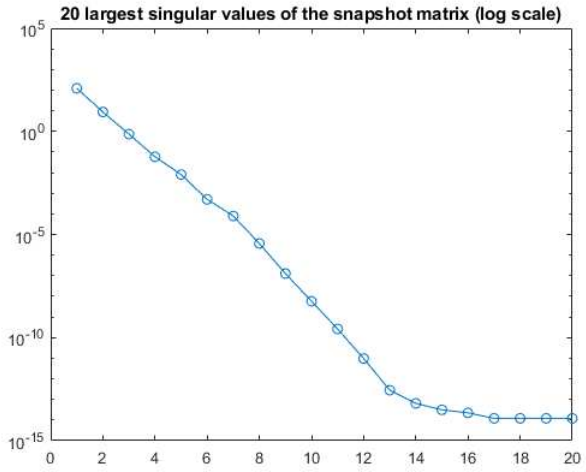

Figure 1.2: 20 largest singular values of the snapshot matrix in the log scale

The problem appears to be well-posed in terms of model reduction, as choosing several thousands of snapshots yields a reduced basis subspace of dimension 6. The error of the projection is of order 10^{-9}, meaning that a 6–dimensional function subspace is essentially good enough to describe an arbitrary solution of the partial differential equation, for the given parameter ranges. Observing the singular value drop-off (Figure 1.2) in the snapshot matrix, this result might not be so surprising. Obviously an even smaller subspace may be chosen, depending on the required accuracy.

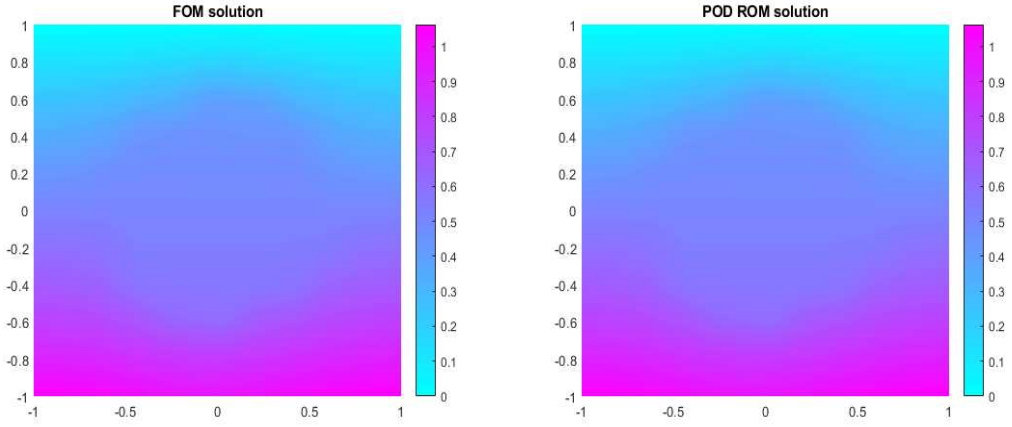

Figure 1.3: Original solution and the projection to the POD reduced basis space: the area of different thermal conductivity is again visible. The parameter values are $\mu_1 = 9$, $\mu_2 = 0.75$.

However, the process of generating a large enough set of snapshots in order to come to experimental conclusions is, by far, the most computationally taxing part of the aforementioned process. Therefore, this approach might not be ideal in case of a more ill-posed model, in which no snapshots are obtained in the off-line phase. Subsequently, for a larger problem, with e.g. 10 parameters and n samples per parameter, this procedure would require computing n^{10} snapshots, which is clearly not a viable option. That is why it is important to consider alternative methods of generating a reduced basis, perhaps without generating snapshots in this manner.

1.4 Greedy Basis Generation

In the previous section the reduced basis of size M was computed through the use of $N \ll M$ snapshots, which is cumbersome if they are to be manually computed. In this section we present an alternative: a "natural" *greedy* algorithm for the reduced basis generation. An important assumption needed in this entire section is that we possess an *error estimator* at hand, denoted by $e(\mu)$, which (we assume) adequately estimates the error caused by performing the model reduction, not including the discretization. The algorithm itself is rather intuitive. First, a finite dimensional subspace $\mathbb{P}_h \subset \mathbb{P}$ is chosen. The greedy process is iterative: consider the step k and the appropriate generated reduced basis space $\mathbb{V}_{rb}$ of dimension k. The goal is to determine the next parameter $\mu_{k+1} \in \mathbb{P}_h$ for which we compute the solution $u(\mu_{k+1})$, which we then choose as the next basis function. Naturally, there is no guarantee of linear independence yet – this issue will be addressed later. The choice of the parameter is done by minimizing the error estimator $e(\mu)$ over $\mathbb{P}_h$, with respect to the space $\mathbb{V}_{rb}$. For every parameter $\mu \in \mathbb{P}_h$, we compute the reduced basis approximation using the current space $\mathbb{V}_{rb}$, i.e. we solve a modified (finite) problem

$$a(u_{rb},\, v_{rb};\, \mu) = f(u_{rb};\, \mu), \quad \forall\, v_{rb} \in \mathbb{V}_{rb},$$

of dimension $k = dim(\mathbb{V}_{rb})$ or more specifically the dimension of the reduced basis of the current step. For each choice of parameter $\mu \in \mathbb{P}_h$, we compute the error estimate $e(\mu)$ and choose the parameter with the largest error value:

$$\mu_{k+1} = \arg\max_{\mu \in \mathbb{P}_h} e(\mu).$$

Then, the solution $u(\mu_{k+1})$ is computed and the reduced basis space is updated as

$$\mathbb{V}_{rb} \leftarrow \mathbb{V}_{rb} \cup \{u(\mu_{n+1})\}.$$

Finally, the process stops when the error estimator in the given step becomes small enough, i.e. smaller than an a priori set tolerance ϵ. The final greedy algorithm goes as follows. The expression $A \leftarrow B$ denotes that we replace whichever value the variable A held with the value in the variable B, whether A was defined before or not.

Algorithm 1: Greedy basis generation

Input: A tolerance ϵ, the first parameter μ_1 and set $n = 1$;

while $e(\mu_{n+1}) > tol$ ***and*** $n \leq dim(\mathbb{P}_h)$ **do**

  Compute the solution $u(\mu_n)$ for the parameter μ_n

  $\mathbb{V}_{rb} \leftarrow \text{span}\{u(\mu_1), \ldots, u(\mu_n)\}$

  for $\mu \in \mathbb{P}_h$ **do**

    Compute the **reduced basis** approximation $u_{rb}(\mu) \in \mathbb{V}_{rb}$

    Evaluate the error estimator $e(\mu)$

  end

  $\mu_{n+1} \leftarrow \arg\max_{\mu \in \mathbb{P}_h} e(\mu)$

  $n \leftarrow n + 1$

end

Output: The reduced basis $\{u(\mu_1), \ldots, u(\mu_n)\}$

The obvious advantage of this approach is that an M-sized reduced basis space requires the computation of only M snapshots. Another, maybe less obvious advantage, is that the computation may be freely continued, should the generated space of dimension M prove to not be of desired accuracy. The process of generating additional k reduced basis functions is completely the same as if we determined $M + k$ functions in the first place, i.e. the space $\mathbb{V}_{rb}$ is absolutely the same in both cases.

Another important issue to consider is the fact that the computed snapshots may be nearly linearly dependent, which is bound to cause problems with condition numbers. This is why orthogonalization should be applied, e.g. the Gram-Schmidt procedure. On the other hand, POD is usually performed using the singular value decomposition, meaning it already yields an orthogonal reduced basis. Ultimately, we note that the greedy algorithm heavily depends on the quality of the error estimator.

2 Discrete Empirical Interpolation Method (DEIM)

Consider the problem of solving nonlinear systems of partial or ordinary differential equations. Many interesting and relevant examples may be found in [11]. One such example is the *Kolmogorov–Petrovskii–Piskunov equation*, given in a general form by

$$\frac{\partial}{\partial t}u(t,x) - a\frac{\partial^2}{\partial x^2}u(t,x) = f\big(u(t,x)\big),$$

where a is a constant and $f = f(u)$ is a nonlinear function of the unknown variable u. The equation often appears in problems involving heat or mass transfer, as well as in many problems in biology, e.g. population growth. A notable example is the *Fisher-Kolmogorov–Petrovskii–Piskunov equation* which models both wave propagation as well as population growth, defined as

$$\frac{\partial}{\partial t}u(t,x) - D\frac{\partial^2}{\partial x^2}u(t,x) = r\,u(t,x)\,\big(1 - u(t,x)\big),$$

for constants $r, D > 0$. Naturally, this needs to be equipped with the proper boundary conditions.

An approach to solving this problem involves a discretization or a semi-discretization technique, both resulting in a large, nonlinear system. Here we focus on the semi-discretization method, yielding a dynamical system of the form

$$\frac{d}{dt}v(t) = Av(t) + F\big(v(t)\big) + b(t),$$

where the wanted solution v is now of dimension n. F corresponds to the nonlinearity, whereas $b(t)$ may come from boundary conditions of the original problem. The details on the semi-discretization, i.e. the *method of lines* may be found in the appendix. In order to obtain a relatively accurate solution, the chosen discretization may need to be rather fine, resulting in a large dimension n. The computational barrier is now obvious, especially taking into consideration that computing $F(v)$ many times may impose a heavy burden. We then turn to demonstrate why a POD approach is not enough to solve this problem both accurately and in a time/memory-effective manner. In combination with using the POD approach to obtain a reduced model, we show in detail why the nonlinear part of the problem presents an issue, and we

offer a method that deals with such issues even more generally. Namely, it may be applied to any nonlinear function approximation problem.

2.1 Basic Setting

In this chapter we focus on the *Discrete Empirical Interpolation Method* (DEIM), first presented in [5], which aims to improve the efficiency of the model reduction process. As discussed, let us consider a dynamical system, described as:

$$\frac{d}{dt}x(t) = Ax(t) + f(x(t)), \tag{2.1}$$

where $x(t) \in \mathbb{R}^n$ describes the state, $A \in \mathbb{R}^{n\times n}$ is a matrix with constant values and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a (generally nonlinear) function. A setting we work with is that n may be very large, for example of order 10^6 or higher.
Such systems model many situations, appearing directly or being the result of a semi-discretization of partial differential equations. Since many of these systems are very large-scale, they induce a significant computational burden. In numerous situations however, there are in fact very few ”internal degrees of freedom”, i.e. such a large dimension is not needed to essentially describe the behaviour of the state variable $x(t)$.

Remark 2.1.1 *The entire process is equally valid for systems which, unlike the observed one, aren't time-dependant, i.e. systems of the form*

$$Ay(x) + F(y(x)) = 0.$$

The idea of the DEIM method is to use projection in order to obtain a model of reduced order, which adequately approximates the starting model. To that extent, we denote by V_k the matrix of dimension $n \times k$, where $k \ll n$, whose columns are given by the reduced basis. This means that the reduced basis space $\mathbb{V}_k$ is then given by

$$\mathbb{V}_k = \text{range}(V_k).$$

Such a reduced basis may be obtained through the POD described in the previous chapter. An additional assumption is that the columns are orthonormalized. This is in accordance with the conclusions from the previous chapter, as the reduced basis is

computed by the POD, meaning that columns of V_k are columns of an orthogonal matrix U from the singular value decomposition. Alternatively, if the reduced basis was computed by the greedy basis algorithm we assume that the Gram Schmidt process has been performed during the basis generation itself, meaning V_k is orthonormal in that case as well.

2.2 Algorithm and Theoretical Background

The order reduction goes as follows: after determining the matrix V_k, we perform the substitution $x(t) = V_k\, y(t) \in \mathbb{R}^k$ in the system 2.1 which yields:

$$\frac{d}{dt}(V_k\, y(t)) = AV_k\, y(t) + f(V_k y(t)).$$

Since V_k has orthonormal columns it follows:

$$\frac{d}{dt}y(t) = V_k^T A V_k\, y(t) + V_k^T f(V_k y(t)).$$

Now the unknown variable of the system is no longer of dimension n, but rather dimension k (recall that $k \ll n$). We now take a look at the dimensions currently present in the system, in comparison to the original system 2.1, again taking into consideration that $k \ll n$.

$$\begin{cases} \frac{d}{dt}\, x(t) = Ax(t) + f(x(t)), \\ x(t) \in \mathbb{R}^n, \\ A \in \mathbb{R}^{n\times n}, \\ f(x(t)) \in \mathbb{R}^n, \end{cases}$$

whereas the latter situation is as follows:

$$\begin{cases} \frac{d}{dt}\, y(t) = V_k^T A V_k\, y(t) + V_k^T f(y(t)), \\ y(t) \in \mathbb{R}^{\mathbf{k}}, \\ V_k^T A V_k \in \mathbb{R}^{\mathbf{k}\times\mathbf{k}}, \\ V_k^T f(y(t)) \to V_k \in \mathbb{R}^{k\times n},\ f(y(t)) \in \mathbb{R}^{n\times 1}. \end{cases}$$

We observe that the only computationally taxing part of the new reduced system is the evaluation of the nonlinear part of the system, i.e. $V_k^T f(y(t))$, which still directly depends on the *large* dimension n. This implies that the model has not yet been adequately reduced. The DEIM method aims to compute the nonlinear function f in a cost-effective manner. The goal is to project the function onto a space of dimension $m \ll n$ in order to yield a satisfactory approximation of the nonlinearity. Let us denote $F(t) = V_k^T f(y(t))$ and observe that the following procedure can be performed in order to approximate a general nonlinear function F, independently of the framework presented here.

Let the m-dimensional approximation space be denoted by $\text{span}\{u_1, \ldots, u_m\} \subset \mathbb{R}^n$, and the coefficient vector for a given t with $p = p(t) = (p_1(t), \ldots, p_m(t))$. The approximation space may be obtained by applying the POD method to a matrix of snapshots of F, namely $S = \left[F(t_{i_1}),\ F(t_{i_1}),\ \ldots\ F(t_{i_q})\right]$. The approximation of F is then given by

$$F(t) \approx [u_1 \ldots u_m]p = Up,$$

which is a system we need to solve for $p = p(t)$. Considering the dimensions of the system, we observe that $F(t) \in \mathbb{R}^{n\times 1}$, $U \in \mathbb{R}^{n\times m}$, but $p(t) \in \mathbb{R}^m$ and $m \ll n$. Therefore, the system is *overdetermined*. That is why it is sensible to somehow *pick* m rows of the system in order to obtain an $m \times m$ system for $p(t)$. This is equivalent to finding a permutation matrix $P \in R^{n\times m}$ and multiplying the system from the left side by P^T. The system then becomes

$$P^T F(t) = (P^T U)p(t), \tag{2.2}$$

where

$$P^T F(t) \in \mathbb{R}^m, \quad P^T U \in \mathbb{R}^{m\times m},$$

meaning that the system is no longer overdetermined, but quadratic. Since $P \in \mathbb{R}^{n\times m}$ is a permutation matrix we know it can be represented as

$$P = [e_{\varphi_1}, \ldots, e_{\varphi_m}],$$

where e_{φ_i} represents the canonical R^m basis vector

$$(e_{\varphi_i})_j = \begin{cases} 1, & \varphi_i = j, \\ 0, & \varphi_i \neq j. \end{cases}$$

Note that the new system depends on the assumption that $P^T U$ is invertible, which we take as is now, but we provide the required argumentation later.

The algorithm goes as follows. The first index φ_1 is chosen as the index of maximum value of u_1, in the sense of absolute value. In a general step k the matrices U and P from the system 2.2 are set to be matrices with columns $u_1, \ldots, u_{k-1}$ and $e_{\varphi_1}, \ldots, e_{\varphi_{k-1}}$, respectively, meaning that they are set to $U = [u_1]$ and $P = [e_{\varphi_1}]$ in the first step.

In order to compute the next $m - 1$ indices, in each step we compute p from the system $(P^T U)\, p = P^T u_k$ and we choose the index φ_k as the index of the largest (again in terms of absolute value) entry in the residual vector $r = u_k - Up$.

We denote by *arg max* the index of the maximum element of a vector.

Algorithm 2: DEIM

Input: An approximation space basis $\{u_1, \ldots, u_m\}$;

$\varphi_1 = \arg\max(|u_1|)$;

$U \leftarrow [u_1]$ $P \leftarrow [e_{\varphi_1}]$;

$\varphi \leftarrow [\varphi_1]$;

for $k = 2, \ldots, m$ **do**

  Compute p from the system $(P^T U)\, p = P^T u_k$;

  Compute the residual $r = u_k - Up$;

  $\varphi_k = \arg\max(|r|)$;

  $U \leftarrow [U \;\; u_k]$;

  $P \leftarrow [P \;\; e_{\varphi_k}]$;

  $\varphi \leftarrow (\varphi \,,\, \varphi_k)$

end

Output: Vector of row indices $\varphi = [\varphi_1, \ldots, \varphi_m]$

Formalizing the algorithm, we observe that DEIM is essentially an *oblique projector*. Figure 2.1 illustrates the intuition behind the DEIM oblique projector. As per [5, Definition 3.1], the DEIM approximation of order m of the nonlinear term $F(t)$ is

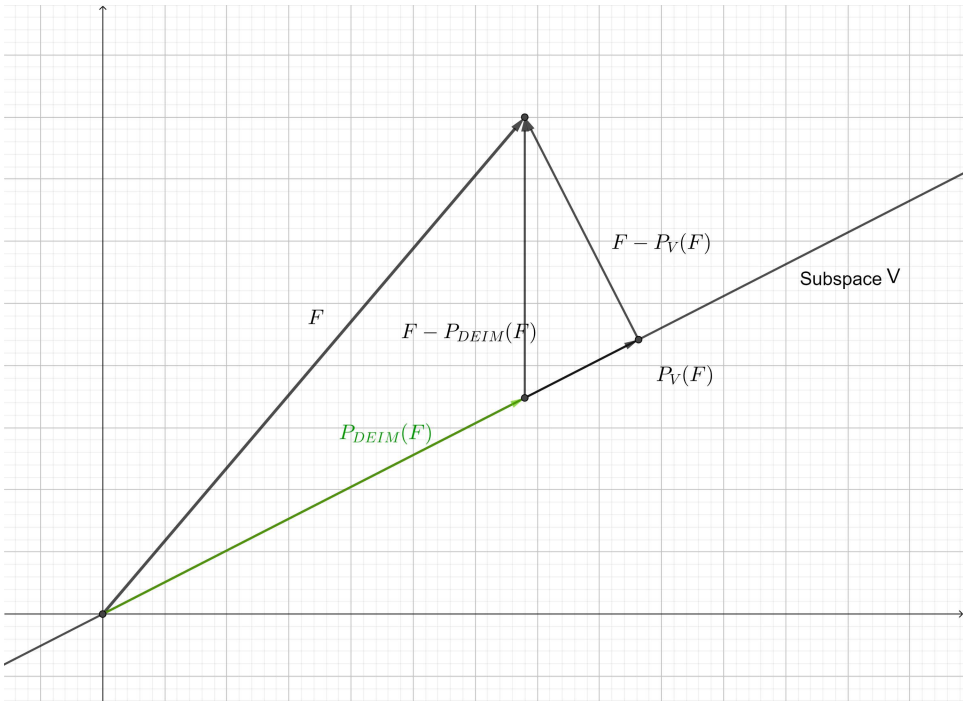

Figure 2.1: An example of the DEIM oblique projector onto the subspace V applied to the vector F, in comparison to the orthogonal projection.

given by:

$$\hat{F}(t) \approx U(P^T U)^{-1} P^T F(t),$$

where $U = [u_1, \ldots, u_m]$ and $P = [e_{\varphi_1}, \ldots, e_{\varphi_m}]$ are results of the Algorithm 2. Observe that applying P^T from the left side yields

$$P^T \hat{F}(t) = P^T F(t),$$

meaning that this projection, unlike the orthogonal projection, interpolates *exactly* at some rows. Again, the obvious issue is the regularity of the matrix $P^T U$ at each step of the DEIM algorithm. The following result [5, Lemma 3.2] together with its proof serves to prove that regularity follows directly from the construction of the matrix P. The step of choosing an index φ_i in the DEIM algorithm essentially looks like

$$[|\rho_i|,\ \varphi_i] = [\max(\,|r|\,), \text{ind}\max(\,|r|\,)].$$

The value $|\rho_i|$ is omitted in the algorithm as it is not used directly. However, the proof that $P^T U$ is in fact invertible will rely on proving that the value $|\rho_i|$ is non-zero at each step. Moreover, it will follow that the chosen indices are all unique, i.e. $\forall i, j,\ i \neq j \implies \varphi_i \neq \varphi_j$. Prior to the main statement we provide a helpful property.

Lemma 2.2.1 *Let $x \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times k}$ a matrix with linearly independent columns. The best approximation of x in terms of Range(A) is given by*

$$A(A^TA)^{-1}A^Tx.$$

Proof. Let $x = y + z$, where $y \in \text{Range}(A)$ and $z \in \text{Ker}(A^T) = \text{Range}(A)^\perp$. This implies that

$$A(A^TA)^{-1}A^Tx = A(A^TA)^{-1}A^T(y+z) = A(A^TA)^{-1}A^Ty.$$

Since $y \in \text{Range}(A)$ there exists a vector q such that $y = Aq$. This yields:

$$A(A^TA)^{-1}A^Tx = A(A^TA)^{-1}A^Ty = A(A^TA)^{-1}A^T(Aq) = Aq = y.$$

□

Remark 2.2.2 *Since the matrix $A \in \mathbb{R}^{n\times m}$ is a matrix with linearly independent columns, the Moore-Penrose inverse of A is given by $(A^TA)^{-1}A^T$, meaning that the expression from the statement above is $AA^\dagger$.*

Theorem 2.2.3 *Let $f \in \mathbb{R}^n$ be a vector and $\{u_1, \dots, u_m\} \subset \mathbb{R}^n$ a set of orthonormal vectors. Assume we have computed the DEIM approximation of order m, where $m \leq n$, as per the Algorithm 2, yielding $U = [u_1, \dots, u_m]$, $P = [e_{\varphi_1}, \dots, e_{\varphi_m}]$ and the approximation*

$$\hat{f} = U(P^TU)^{-1}P^Tf.$$

We denote by $C = \|(P^TU)^{-1}\|_2$ and $\epsilon_*(f) = \|(I - UU^T)f\|_2$, where I represents the identity matrix. The interpretation of ϵ_* is the error of the best 2-norm approximation for f, from the space Range(U). An error bound is then given by

$$\|f - \hat{f}\|_2 \leq C\,\epsilon_*(f),$$

Moreover, a bound for C is given by:

$$C \leq (1 + \sqrt{2n})^{m-1}\|u_1\|_\infty^{-1}.$$

Proof. Let $\hat{f}$ be the DEIM approximation of order m for f. We denote by f_U the best

approximation of f with respect to the space Range(U). Lemma 2.2.1 now yields

$$f_U = U(U^TU)^{-1}U^Tf \overset{U^TU=I}{=} UU^Tf.$$

We denote the DEIM projector by $\mathbb{P}_D = U(P^TU)^{-1}P^T \implies \hat{f} = \mathbb{P}_D f$. Additionally, we represent $f = (f - f_U) + f_U$ and denote and $w = f - f_U$, meaning $f = w + f_U$. Now it follows that

$$\hat{f} = \mathbb{P}_D f = \mathbb{P}_D(w + f_U) = \mathbb{P}_D w + f_U.$$

The term $f - \hat{f}$, for which we are trying to determine the bound, is then given by

$$f - \hat{f} = (w + f_U) - (\mathbb{P}_D w + f_U) = w - \mathbb{P}_D w = (I - \mathbb{P}_D)w,$$

which implies

$$||f - \hat{f}||_2 = ||(I - \mathbb{P}_D)w||_2 \leq ||I - \mathbb{P}_D||_2||w||_2.$$

Observe that

$$||w||_2 = ||f - f_U||_2 = ||f - UU^Tf||_2 = ||(I - UU^T)f||_2 = \epsilon_*(f),$$

which is the first term from the statement of the theorem. In the case of the second term, we first prove that $\mathbb{P}_D$ is a projector:

$$\mathbb{P}_D^2 = [U(P^TU)^{-1}P^T][U(P^TU)^{-1}P^T] = U(P^TU)^{-1}P^T = \mathbb{P}_D,$$

meaning that $I - \mathbb{P}_D$ is a projector as well. Assuming $\mathbb{P}_D \neq 0, I$, and taking into consideration that (due to column orthonormality) $||U||_2 = ||P||_2 = 1$, it follows that

$$||I - \mathbb{P}_D||_2 = ||\mathbb{P}_D|| \leq ||U||_2||(P^TU)^{-1}||_2||P^T||_2 = ||(P^TU)^{-1}||_2 = C,$$

which completes the proof of the first statement of the theorem, i.e.

$$||f - \hat{f}||_2 \leq ||I - \mathbb{P}_D||_2||w||_2 \leq ||I - \mathbb{P}_D||_2\,\epsilon_*(f) \leq C\epsilon_*(f).$$

The next step of the proof is to establish the mentioned upper bound on $C = ||(P^TU)^{-1}||_2$. This is further justified by the fact that the term $\epsilon_*(f)$ in the bound does not depend

on the DEIM process. The analysis of each step of the DEIM algorithm is performed, so as to prove the bound.

For each step of the algorithm $k = 2, \dots, m$ we introduce the following notation

$$U_{k-1} = [u_1, \dots, u_{k-1}], \quad , P_{k-1} = [e_{\varphi_1}, \dots, e_{\varphi_{k-1}}],$$

$$U_k = [U_{k-1}\ u_k], \quad , P_k = [P_{k-1}\ e_{\varphi_k}].$$

Additionally, let $M_k = P_k^T U_k$, meaning that in the first step of the algorithm, i.e. for $k = 1$ we have

$$M_1 = P_1^T U_1 = e_{\varphi_1}^T u_1 \implies \|M_1^{-1}\|_2 = \frac{1}{|e_{\varphi_1}^T u_1|} \overset{DEIM}{=} \frac{1}{\|u_1\|_\infty} \geq 1,$$

meaning the norm $\|M_1^{-1}\|_2$ is minimized by the DEIM algorithm. From here on, we proceed with analysis for the algorithm steps $k = 2, \dots, m$. For, starters, we introduce some additional notation :

$$a_k = p_k^T U_{k-1}, \quad c_k = M_{k-1}^{-1} P_{k-1}^T u_k, \quad \rho_k = p_k^T u_k - a_k^T c_k,$$

and observe that $|\rho_k| = \|r\|_\infty$, defined as the residual in the DEIM algorithm. We now rewrite M_k as:

$$M_k = P_k^T U_k = \begin{bmatrix} M_{k-1} & P_{k-1}^T u_k \\ p_k^T U_{k-1} & p_k^T u_k \end{bmatrix} = \begin{bmatrix} M_{k-1} & 0 \\ a^T & \rho \end{bmatrix} \begin{bmatrix} I & c \\ 0 & 1 \end{bmatrix},$$

Finally, we rewrite the main matrix of interest M_k^{-1} as

$$M_k^{-1} = \begin{bmatrix} I & -c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} M_{k-1}^{-1} & 0 \\ -\frac{1}{\rho_k} a_k^T & \frac{1}{\rho_k} \end{bmatrix} = \begin{bmatrix} I & -c \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \\ -\frac{1}{\rho_k} a_k^T & \frac{1}{\rho_k} \end{bmatrix} \begin{bmatrix} M_{k-1}^{-1} & 0 \\ 0 & 1 \end{bmatrix}$$

$$\implies M_k^{-1} = \left(\begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} + \frac{1}{|\rho_k|} \begin{bmatrix} I & -c \\ 0 & 1 \end{bmatrix} \right) \begin{bmatrix} M_{k-1}^{-1} & 0 \\ 0 & 1 \end{bmatrix}.$$

This yields the bound

$$\|M_k^{-1}\| \leq \left(\left\| \begin{bmatrix} I & 0 \\ 0 & 0 \end{bmatrix} \right\|_2 + \frac{1}{|\rho_k|} \left\| \begin{bmatrix} I & -c \\ 0 & 1 \end{bmatrix} \right\|_2 \right) \left\| \begin{bmatrix} M_{k-1}^{-1} & 0 \\ 0 & 1 \end{bmatrix} \right\|_2.$$

As established earlier, the matrix $U_k = [U_{k-1}\ u_k]$ is a matrix with orthonormal columns, meaning that $\|x\|_2 = \|U_k x\|_2$. Applying this to the second term in the bracket we obtain

$$\left\| \begin{bmatrix} c_k \\ -1 \end{bmatrix} \begin{bmatrix} a_k^T & -1 \end{bmatrix} \right\|_2 = \left\| [U_{k-1}\ u_k] \begin{bmatrix} c \\ -1 \end{bmatrix} \begin{bmatrix} a_k^T & -1 \end{bmatrix} \right\|_2 \leq \|U_{k-1}c_k - u\|_2 \, \| [a_k^T \ \ 1] \|_2.$$

Taking into consideration that $\| [a_k^T \ 1] \|_2 = \sqrt{1 + \|a_k\|_2^2}$ and applying the general inequality $\|x\|_2 \leq \sqrt{n}\|x\|_\infty$ yields

$$\left\| \begin{bmatrix} c_k \\ -1 \end{bmatrix} \begin{bmatrix} a_k^T & -1 \end{bmatrix} \right\|_2 \leq \sqrt{2n}\, \|U_{k-1}c_k - u_k\|_\infty = \sqrt{2n}\, |\rho_k|.$$

We now obtain the boundary for the norm of the matrix M_k^{-1} as follows:

$$\|M_k^{-1}\|_2 \leq \left(1 + \frac{1}{|\rho_k|} * \sqrt{2n}\, |\rho_k|\right) \|M_{k-1}^{-1}\|_2 \overset{\text{recursively}}{\leq} (1 + \sqrt{2n})^{k-1} \|M_1^{-1}\|_2.$$

Ultimately, as previously established, $\|M_1^{-1}\|_2 = \frac{1}{\|u_1\|_\infty}$ which implies

$$\|M_k^{-1}\|_2 \leq (1 + \sqrt{2n})^{k-1} \frac{1}{\|u_1\|_\infty},$$

which is exactly the boundary from the second statement of the theorem. □

Per [5] however, the bound in the theorem is rarely used as an a priori estimate since it grows rather fast and it is therefore observed to be far too pessimistic in practice. Finally, an important issue to consider is that the DEIM algorithm will depend on the ordering of the columns in the snapshot matrix. These are some of the things which determine the way forward for the DEIM approach.

2.3 Example

We demonstrate how the POD-DEIM method combination works on an artificial example, and we later move on to the example from the beginning of the chapter. For starters we consider an ODE system, created artificially to demonstrate the DEIM method. The function $F : \mathbb{R}^n \to \mathbb{R}^n$ is given component-wise as

$$(F(y_1, \ldots, y_n))_i = \begin{cases} e^{\sin(x_{i+1})} + \sqrt[3]{x_{i+1}}, & i < n, \\ e^{\sin(x_1)} + \sqrt[3]{x_1}, & i = n \end{cases}.$$

Now we observe the system $\frac{\partial}{\partial t} y(t) = F(y(t))$. The function is selected as such, so that the components depend on each other in order to better mimic the behaviour of an actual ODE system. The dimension is set to $n = 5 \cdot 10^4$ and the number of snapshots is set to $m = 10^3$. Numerical rank is set to be chosen as the number of singular values greater than $tol = 10^{-10}$. The snapshot matrix rank turned out to be only 62. The snapshots are chosen as the solutions of the system $\frac{d}{dt} y = F(y(t))$, in the time interval $[1, 5]$.

Solutions and snapshots are calculated using the Runge-Kutta method. The final error is evaluated as the infinity norm of the exact solution matrix $x \in \mathbb{R}^{n \times n_{time}}$ and the reduced model solution multiplied by the matrix V_k, as the original substitution was $x(t) = V_k\, y(t)$:

$$\|x(t) - V_k\, y(t)\|_\infty = 1.0923 \cdot e^{-9}.$$

We plot the reduced model errors depending on the selected POD truncation rank in the DEIM algorithm (the matrix U) values in the set $\{1, \ldots, 25\}$, as shown in Figure 2.2.

There are essentially two POD processes to perform. The first one is needed to obtain the matrix V_k which translates the high-order ODE system to a lower dimensional system. The second one should yield the matrix U, needed to carry out the DEIM algorithm for the nonlinear part of the system. In order to make calculations more efficient, the snapshots are computed all at once. The solutions of the equation are obtained by the Runge-Kutta method. Seeing as the method requires the evaluation of $f(t, y)$ at every step, the results are simply saved separately as the snapshots of f. This naturally saves a lot on both memory and time.

2.4 Q-DEIM

A newer version of the DEIM algorithm given in [6] yields a significant improvement in both the implementation and error bound. This algorithm, called *Q-DEIM*, essentially consists of utilizing a QR decomposition of the snapshot matrix in order to potentially improve the index choices. The goal of this subsection is to describe this

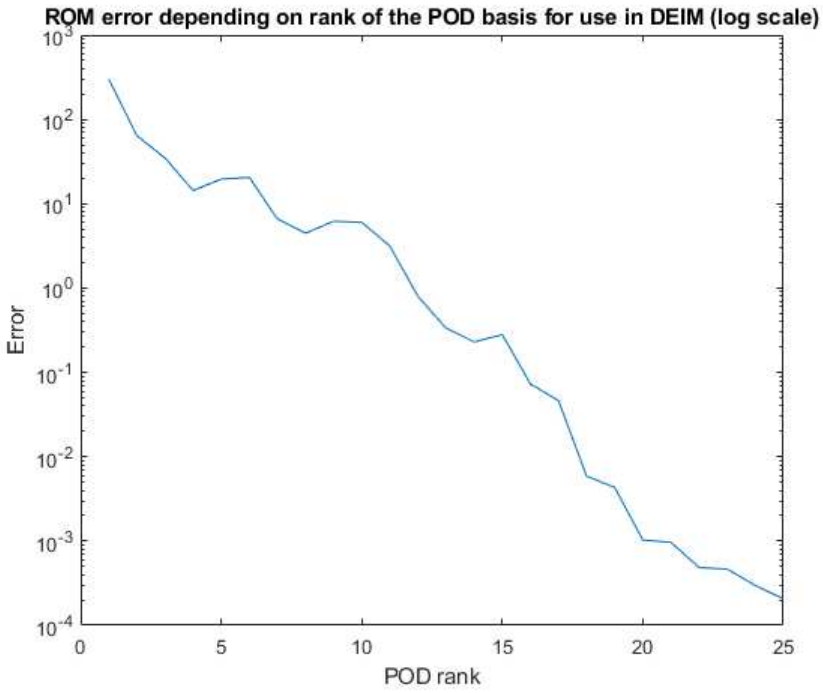

Figure 2.2: The errors when performing the DEIM process, depending on the POD rank used to obtain the input matrix U in the DEIM process, i.e. the rank of the reduced basis matrix used for DEIM (log scale)

method and provide a comparison with the DEIM method on the previously shown example.

When the DEIM points $\{\varphi_1, \ldots, \varphi_m\}$ are selected, the approximation does not depend on the basis for Range(U). Let $V = [v_1, \ldots, v_m]$ be an alternative basis, which implies that there exists a regular change-of-basis matrix $Z \in \mathbb{R}^{m \times m}$ such that $U = VZ$. Now it follows

$$\begin{aligned} U(P^T U)^{-1} P^T F(t) &= (VZ)(P^T VZ)^{-1} P^T F(t) = \\ &= VZZ^{-1}(P^T Z)^{-1} P^T F(t) = \\ &= V(P^T V)^{-1} P^T F(t). \end{aligned} \tag{2.3}$$

However, the entire process does depend on the matrix U. This may obviously present issues in the case of bad conditions due to e.g. clustered singular values. These issues were the main driving force behind the development of the Q-DEIM method. The main theorem [6, Theorem 2.1.] is presented below.

Theorem 2.4.1 *Let $m \ll n$ and $U = [u_1, \ldots, u_m] \in \mathbb{C}^{n \times m}$ be a matrix with orthonormal columns, meaning $U^* U = I_m$. Let $f \in \mathbb{C}^n$ be an arbitrary vector. There exists an*

algorithm to compute a selection operator P with complexity $O(nm^2)$ such that

$$C = \|(P^T U)^{-1}\| \leq \sqrt{n-m+1}\,\frac{\sqrt{4^m + 6m - 1}}{3},$$

whereas a full rank matrix U yields a bound

$$C = \|(P^T U)^{-1}\| \leq \frac{\sqrt{n-m+1}}{\sigma_{min}(U)}\,\frac{\sqrt{4^m + 6m - 1}}{3},$$

Furthermore, the approximation error is bound as follows

$$\|f - U(P^T U)^{-1} P^T f\|_2 \leq \sqrt{n}\, O(2^m) \|f - UU^* f\|_2.$$

Moreover, there exists a selection operator P_* such that the approximation error is bound as

$$\|f - U(P^T U)^{-1} P^T f\|_2 \leq \sqrt{n}\,\sqrt{1 + m(n-m)}\|f - UU^* f\|_2.$$

Finally, the selection operators do not change if we replace U by $U\Omega$, for an arbitrary unitary matrix Ω.

A constructive proof of this theorem may be found in [6]. In order to apply the method in this framework, the selection matrix P used in the DEIM algorithm is simply obtained in a different manner. By performing `[Q,R,P] = qr(U')` and then selecting m columns as $P_{QDEIM} = P(:, 1 : m)$ we keep the entire process intact, but with a different selection matrix P. The performance of this improvement will be demonstrated in the following chapter.

2.5 K-DEIM

The focal point of both DEIM and Q-DEIM was the way of choosing indices of rows $\varphi_1, \ldots, \varphi_m$ which we retain, for use in model reduction. A different approach to this process is suggested in [12]. Let S denote the snapshot matrix for the nonlinear function F. Observe now that the rows of S may be interpreted as trajectories of a state with respect to time. This implies potentially similar behaviour to one another, i.e. trajectories may be naturally grouped together. The method therefore consists of applying *clustering* methods to the matrix S row-wise. More specifically, *k-means* algorithm (going also by Lloyd's algorithm, [13]) is suggested to be used in this

problem. This algorithm is then called *K-DEIM*. As k-means yields not only cluster labels but also cluster *centroids*, a natural selection of the *representative* from each cluster is provided as the member of the cluster closest to the centroid. Moreover, the number of clusters is easily controlled, as opposed to using e.g. *hierarchical* clustering algorithms.

A potential flaw of the k-means algorithm is the fact that it heavily depends on the initial setting, i.e. it may yield a different clustering each time, if a somewhat random starting position is used (the usual approach). A potential solution to this problem is *k-means++*, given in [14], already implemented in MATLAB. It consists of a seeding heuristic, for better control of the starting centroids. As per [14], the k-means++ enjoys better speed and accuracy, while introducing it results in an algorithm just as fast as the original k-means. The example in the next chapter will be additionally used to demonstrate how K-DEIM works compared to the previous methods.

3 Numerical Example

To sum up the conclusion from the previous chapters, we take a look at the motivational example for chapter 2, i.e. the Kolmogorov–Petrovsky–Piskunov equation. The equation describes reaction-diffusion systems, namely population growth. We will present the full model solution, as well as reduced model solutions using POD and DEIM, together with its' suggested improvements.

$$\frac{\partial}{\partial t}u(t,x) - D\frac{\partial^2}{\partial x^2}u(t,x) = r\,u(t,x)\,(1-u(t,x)).$$

The constants are set to $r = D = 1$. Fixing a domain $\Omega = [a,b]$ and a time interval $[0,T]$, the problem equipped with Neumann boundary conditions is then given by

$$\begin{cases} \frac{\partial}{\partial t}u - \frac{\partial^2}{\partial x^2}u = ru(1-u), \quad \text{in } \Omega \\ \frac{\partial}{\partial x}u(t,a) = \frac{\partial}{\partial x}u(t,b) = 0, \\ u(0,x) = \mathrm{e}^{\frac{-x+10}{10}} + 0.1. \end{cases} \tag{3.4}$$

We now select n equidistant points such that $x_0 = a < x_1 < x_2 < \cdots < x_n < x_{n+1} = b$ and denote the step by δx. In this specific example, n is chosen to be 200. Applying the semi-discretization, i.e. the *method of lines* (described in more detail in the Appendix) we obtain the system of ordinary differential equations

$$\frac{d}{dt}u = \frac{1}{(\delta x)^2}Au + F(u), \tag{3.5}$$

where

$$A = \begin{bmatrix} -2 & \mathbf{2} & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & 0 & 0 & \mathbf{2} & 2 \end{bmatrix}, \text{ and } F = (F(u_1), \dots, F(u_n)).$$

The highlighted elements of the matrix A are changed due to the Neumann boundary conditions. The notation is left the same for simplicity, but u now represents

$$u(t) = (u_1(t), u_2(t), \dots, u_n(t)),$$

because of the discretization. The ODE system may now be solved with any number of methods, here we choose the Runge-Kutta method. An example of a solution is given in Figure 3.3. Moreover, in a standard method of lines for a linear PDE a

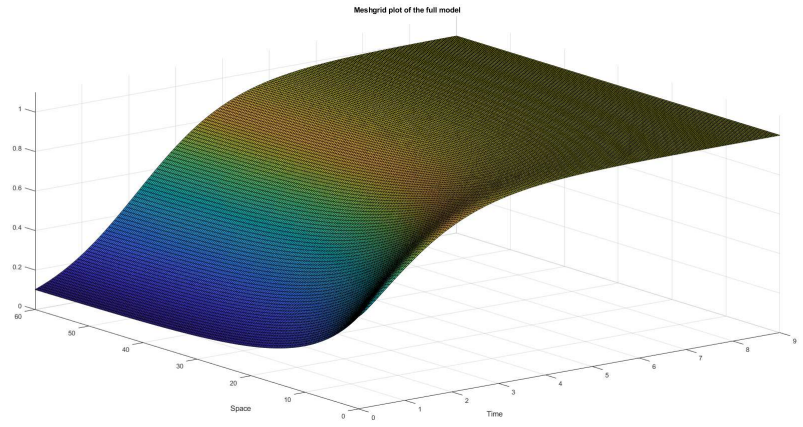

Figure 3.3: A full order solution of the system (3.5): observe that for any non-negative initial conditions, the function eventually ends up being at the same value - this may be interpreted as a guaranteed population growth, if there are any number of non-negative individuals in the beginning

relevant value is the *Courant's* number, whose value has implications on convergence. Regarding the results for linear PDEs, it would be ideal if it was smaller than $\frac{1}{2}$. In this case, we have

$$\mathbf{c} = \frac{\delta t}{\delta x^2} = 0.6644.$$

We compute the matrix of snapshots for dimension reduction, as well as the snapshots of the nonlinear function for the DEIM process. Observe that the singular values of both matrices drop extremely fast, seen in Figure 3.4.

We then set $k = 5$ and compute the matrix V_k and reduce the system using the POD approach. In the next step we set $m = 5$ for DEIM reduction and compute the reduced model solution. The Figure 3.5 shows the POD-DEIM and the POD-QDEIM solutions.

The errors when raising the solution back to the full dimension for $k = m = 5$ turned out to be

$$e(u_{DEIM}) = 0.0408, \quad e(u_{DEIM}) = 0.0397, \quad e(u_{DEIM}) = 0.0478,$$

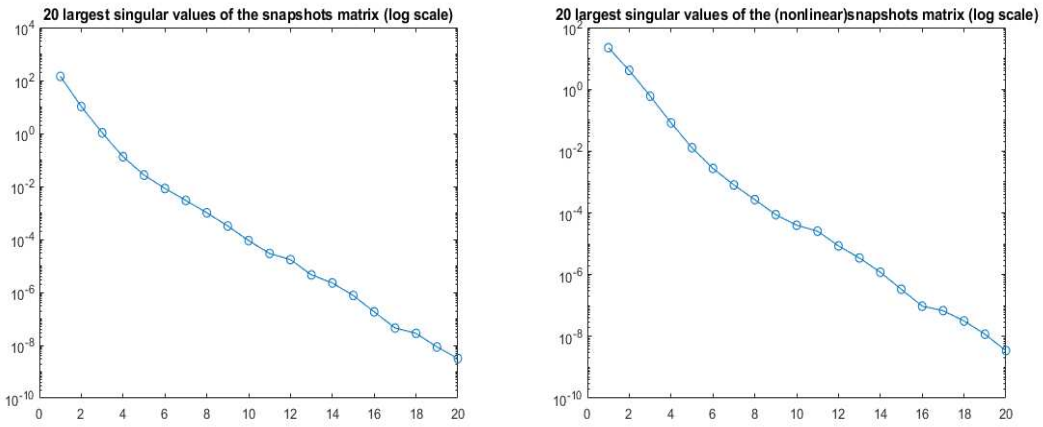

Figure 3.4: 20 largest singular values of the snapshot matrices: left side for the main model, right side for the snapshots of the nonlinear function for use in DEIM methods (log scale)

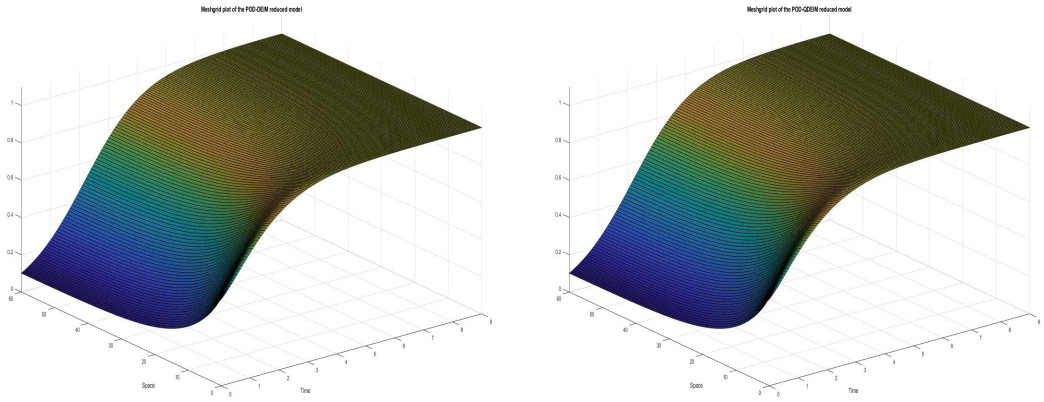

Figure 3.5: Solutions when using DEIM(left) and QDEIM(right): virtually no difference is visible in regard to the FOM solution; outcome is the same for K-DEIM, which is why it is omitted

and the errors for $k = m = 6$:

$$e(u_{DEIM}) = 0.0220 \quad e(u_{DEIM}) = 0.0220, \quad e(u_{DEIM}) = 0.0265,$$

The results of the clustering (i.e. how many points ended up in each cluster during the K-DEIM algorithm) may also provide interesting conclusions:

Cluster	Count	Percent
1	20	10.00%
2	99	49.50%
3	36	18.00%
4	20	10.00%
5	25	12.50%

The second cluster contains almost half the trajectories, whereas the rest of them are somewhat uniform.

It is worth noting that the problem was rather unstable, and small differences in discretization parameters rendered it completely unsolvable, both by Runge-Kutta method and the implemented functions in MATLAB.

Overall, all three approaches performed well and the solutions are quite satisfactory, especially taking into consideration that the dimension of the system was reduced from 200 to 5 or 6.

Appendices

Appendix A General Results

This appendix will contain several general results which the reader may already be familiar with, but they are provided so as to achieve self-sufficiency.

A.1 Weak Formulation

Let H denote a Hilbert space, $\langle \cdot, \cdot \rangle$ the scalar product and $\| \cdot \|$ the induced norm. **Theorem 1. (Lax-Milgram)** Let $B : H \times H \in \mathbb{R}$ be a bilinear form, which is continuous and coercive, i.e.:

$$\exists \mu > 0, \quad \forall u, v \in H, \quad |B(u, v)| \leq \mu \|u\| \, \|v\|,$$

$$\exists \alpha > 0 \quad \forall u \in H, \quad B(u, u) \geq \alpha \|u\|^2.$$

Then, for each element of the dual space $f \in H^*$, there exists a unique element $u \in H$ such that

$$B(u, v) = f(v), \quad v \in H.$$

Theorem 2. (Cea's Lemma) Let H be a real Hilbert space and let $A : H \times H \in \mathbb{R}$ be a bilinear form, which is continuous and coercive. Let $f : H \to \mathbb{R}$ be a bounded linear operator, i.e. $f \in H^*$. The problem

$$\text{find } u \in H, \quad B(u, v) = f(v), \quad v \in H$$

has a unique solution as per the Lax-Milgram theorem. Let $H_h \leq H$ be a finite dimensional subspace. Observe that the finite dimensional problem

$$\text{find } u_h \in H_h, \quad B(u_h, v) = f(v), \quad v \in H_h$$

has a unique solution as per the same theorem. Then it follows that u_h is the best approximation of u in the finite dimensional space V_h, up to the continuity and coercivity constants, i.e.

$$\|u - u_h\| \leq \frac{\gamma}{\alpha} \|u - v\|, \quad v \in H_h$$

A.2 Method of Lines

The *method of lines* is a numerical method for semi-discretization of partial differential equations, based on the notion of discretizing all variables except for a single one. Afterwards, an appropriate ODE solving method may be used. The best way to demonstrate the method is an example, so we consider the homogeneous version of the problem studied in Chapter 3:

$$\frac{\partial}{\partial t}u(t,x) - \frac{\partial^2}{\partial x^2}u(t,x) = 0, \quad x \in [a,b].$$

The discretization with respect to x is performed by choosing $a = x_0 < x_1 < \cdots < x_n = b$ (e.g. an δx-equidistant grid) followed by the finite difference choice:

$$\frac{\partial^2}{\partial x^2}u(t,x_i) = \frac{u(t,x_{i-1}) - 2u(t,x_i) + u(t,x_{i+1})}{(\delta x)^2}.$$

We introduce the notation $u_i(t) = u(t,x_i)$. In order to obtain a final system, we need to specify the boundary conditions. In the case of a Dirichlet boundary condition at $x = a$:

$$u(t,a) = b_a(t)$$

we observe that $u_0 = u(t,x_0) = u(t,a) = b_a(t)$ is no longer an unknown variable. This means that the first equation

$$\frac{\partial}{\partial t}u(t,x_1) = \frac{u_0(t) - 2u_1(t) + u_2(t)}{(\delta x)^2}$$

becomes

$$\frac{\partial}{\partial t}u(t,x_i) = \frac{2u_1(t) + u_2(t)}{(\delta x)^2} + \frac{b_a(t)}{(\delta x)^2}.$$

On the other hand, in case of a Neumann condition like in the example from Chapter 3:

$$\frac{\partial}{\partial x}u(t,a) = 0.$$

This means that the variable $u_0 = u(t,x_0) = u(t,a)$ is no longer specified, meaning it requires its' own equation:

$$\frac{\partial}{\partial t}u(t,x_0) = \frac{u_{-1}(t) - 2u_0(t) + u_1(t)}{(\delta x)^2}.$$

The obvious issue is the point x_{-1} which lies outside of the domain $[a, b]$. Considering a symmetric approach we obtain:

$$0 = \frac{\partial}{\partial x}u(t, 0) \approx \frac{u_1(t) - u_{-1}(t)}{2\delta x} + O((\delta x)^2)$$

which motivates the choice $u(t, y_{-1}) \approx u(t, y_1)$. This means that the equation becomes

$$\frac{\partial}{\partial t}u(t, x_0) = \frac{u_{-1}(t) - 2u_0(t) + u_1(t)}{(\delta x)^2} = \frac{-2u_0(t) + 2u_1(t)}{(\delta x)^2}.$$

Finally, we observe the system as it was in the example before, meaning that both boundary conditions are Neumann type, yielding the following system

$$\begin{bmatrix} \frac{\partial}{\partial t}u_0 \\ \frac{\partial}{\partial t}u_1 \\ \frac{\partial}{\partial t}u_2 \\ \vdots \\ \frac{\partial}{\partial t}u_{n-1} \\ \frac{\partial}{\partial t}u_n \end{bmatrix} = \frac{1}{(\delta x)^2}\begin{bmatrix} -2 & \mathbf{2} & 0 & 0 & \dots & 0 \\ 1 & -2 & 1 & 0 & \dots & 0 \\ 0 & 1 & -2 & 1 & \dots & 0 \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & 1 & -2 & 1 \\ 0 & \dots & 0 & 0 & \mathbf{2} & 2 \end{bmatrix}\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix}$$

As discussed, this system may be solved by any available methods for solving systems of ODEs. However, the system may be especially problematic as it depends on δx not only in terms of multiplication by $\frac{1}{(\delta x)^2}$, but in terms of the number of variables. Therefore, it makes sense to do additional analysis when solving it.

Appendix B MATLAB Code

The function kpp envelopes the entire example presented in chapter 3. The model is solved for the cases of the full model, POD-only reduced order model as well as POD-DEIM and POD-QDEIM reduced order models.

```
function kpp(a, b, T, n, n_time, m, r, g)
% Function for solving the F-KPP equation
%% Input:
% a, b   = space interval [a,b]
% T      = time interval [0,T]
% n      = number of points for x-discretization
% n_time = number of points for t-discretization
% m      = POD system reduction dimension
% r      = DEIM/QDEIM reduction dimension

% g      = initial condition function

%% Space domain
x_vec = linspace(a,b,n);
dx = x_vec(2) - x_vec(1);

%% Time domain
time = linspace(0, T, n_time);
dt = time(2)-time(1);

c_number = dt/(dx^2)

%% Initial and Dirichlet conditions
u_0 = zeros(n,1);
for i = 1:n
    u_0(i) = g(x_vec(i));
end
```

```
%% ODE System matrices
A = (-2 * eye(n) + diag(ones(n-1,1), -1) + diag(ones(n-1,1), 1));
A(1,2) = 2;
A(n,n-1)=2;
A = A * (1/(dx*dx));

b_temp = zeros(n,1);
b_fun = @(t) b_temp;

F_fun = @(t,x) f_nonlin(x);

%% Right-hand-side of the system
RHS = @(t,x) A*x + F_fun(t,x) + b_fun(t);

%% Solving ODE
u = rk4(RHS, time, u_0);

%% POD
Snapshots = u; % column wise snapshot vectors
[U_pod, S_pod, V_pod] = svd(Snapshots, 0);

V_k = U_pod(:, 1:r);  % V_k dimension reduction matrix!

%% DEIM
Snapshots_nonlin = zeros(size(Snapshots));

% Compute nonlinear part snapshots
for i = 1:max(size(time))
    Snapshots_nonlin(:, i) = F_fun(time(i), Snapshots(:,i));
end

[U_PD, S_PD, V_PD] = svd(Snapshots_nonlin, 0);
pod_basis_f = U_PD(:, 1:m);
```

```
phi = zeros(m, 1); %output

%[|rho|, phi_1] = max{|u|}
% phi = [phi_1, ..., phi_n]
[~, phi(1)] = max(abs(pod_basis_f(:,1)));

%U = [u_1]
U = zeros(n, 1);
U(:, 1) = pod_basis_f(:, 1);

%P = [e_{phi_1}]
P = zeros(n, 1);
P(phi(1) ,1) = 1;

%for loop
for L = 2:m
    u_l = pod_basis_f(:, L);

    % solve for c
    c = (P'*U)\(P'*u_l);

    % residual r = u_L - Uc
    res = u_l - U*c;

    %[|rho|, phi_L] = max{|r|}
    [~, phi(L)] = max(abs(res));

    % Save U and P
    U(:, L) = u_l;
    P(phi(L), L) = 1;
end
```

```
F_precomp_out   = ((V_k)' * U) / ((P)' * U);    % size = (r x m)
F_precomp_in    = (P') * V_k;                    % size = (m x r)
A_precomp       = (V_k)' * A * V_k;              % size = (r x r)

% Define reduced model right-hand-side function
RHS_poddeim = @(t,x) A_precomp      * x                          + ...
                     F_precomp_out  * F_fun(t, F_precomp_in * x) + ...
                     (V_k') * b_fun(t);
u_0_poddeim = (V_k)' * u_0;

% DEIM SOLUTION:
u_poddeim = rk4(RHS_poddeim, time, u_0_poddeim);

%% QDEIM
[Qs, Rs, Ps] = qr(Snapshots_nonlin');
P_qdeim = Ps(:, 1:m);

%Solving by qdeim
F_precomp_out_q   = ((V_k)' * U) / ((P_qdeim)' * U);    % size = (r x m)
F_precomp_in_q    = (P_qdeim') * V_k;                    % size = (m x r)
A_precomp_q       = (V_k)' * A * V_k;              % size = (r x r)

% Define reduced model right-hand-side function
RHS_qdeim   = @(t,x) A_precomp_q      * x                              + ...
                     F_precomp_out_q  * F_fun(t, F_precomp_in_q  * x) + ...
                     (V_k') * b_fun(t);
u_0_qdeim = (V_k)' * u_0;

%Q-DEIM SOLUTION:
u_qdeim = rk4(RHS_qdeim, time, u_0_qdeim);

%% K-DEIM
[idx, ~, ~, D] = kmeans(Snapshots_nonlin, m, 'OnlinePhase','on');
```

```
P_kdeim = zeros(n, m);
for i = 1:m
    [~, ind_min] = min(D(:,i));
    P_kdeim(ind_min, i) = 1;
end

F_precomp_out_k   = ((V_k)' * U)/((P_kdeim)' * U);    % size = (r x m)
F_precomp_in_k    = (P_kdeim') * V_k;                 % size = (m x r)
A_precomp_k       = (V_k)' * A * V_k;                 % size = (r x r)

% Define reduced model right-hand-side function
RHS_kdeim = @(t,x)  A_precomp_k      * x                              + ...
                    F_precomp_out_k  * F_fun(t, F_precomp_in_k * x) + ...
                    (V_k') * b_fun(t);
u_0_kdeim = (V_k)' * u_0;

% K-DEIM SOLUTION:
u_kdeim = rk4(RHS_kdeim, time, u_0_kdeim);
```

Bibliography

[1] Benjamin Stamm, Gianluigi Rozza, Jan S. Hesthaven: *Certified Reduced Basis Methods for Parametrized Partial Differential Equations*, Springer, 2015

[2] Fredrik Bengzon, Mats G. Larson: *The Finite Element Method: Theory, Implementation, and Applications*, Springer, 2013

[3] Vladimir Buljak: *Inverse Analyses with Model Reduction: Proper Orthogonal Decomposition in Structural Mechanics*, Springer, 2011

[4] Sebastian Ullmann, Marko Rotkvic, Jens Lang: *POD-Galerkin reduced-order modeling with adaptive finite element snapshots*, Journal of Computational Physics Volume 325, Pages 244-258, 2016

[5] Saifon Chaturantabut, Danny C. Sorensen: *Nonlinear Model Reduction via Discrete Empirical Interpolation*, SIAM J. Sci. Comput., 32(5), 2737–2764, 2010

[6] Zlatko Drmač, Serkan Gugercin: *A New Selection Operator for the Discrete Empirical Interpolation Method – improved a priori error bound and extensions*, SIAM Journal on Scientific Computing, Vol. 38, No. 2, pp. A631-A648, 2016

[7] Zlatko Drmač, Arvind K. Saibaba: *The Discrete Empirical Interpolation Method: Canonical Structure and Formulation in Weighted Inner Product Spaces*, SIAM J. Matrix Anal. Appl., 39(3), 1152–1180, 2018

[8] J. Nathan Kutz, Steven L. Brunton: *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*, Cambridge University Press, 2019

[9] J. Nathan Kutz, Steven L. Brunton: http://databookuw.com/

[10] Younes Chahlaoui, Paul Michel Van Dooren *Benchmark Examples for Model Reduction of Linear Time-Invariant Dynamical Systems*, Lecture Notes in Computational Science and Engineering, vol 45., Springer, 2005

[11] Andrei D. Polyanin, Valentin F. Zaitsev: *Handbook of Nonlinear Partial Differential Equations, Second Edition*, Chapman & Hall/CRC Press, 2012

[12] Benjamin Peherstorfer, Zlatko Drmač, Serkan Gugercin: *Stabilizing discrete empirical interpolation via randomized and deterministic oversampling*, 2018

[13] Lloyd, Stuart P.: *Least Squares Quantization in PCM*, IEEE Trans. Inf. Theory, vol 28.,129-136, 1982

[14] David Arthur, Sergei Vassilvitskii: *k-means++: The Advantages of Careful Seeding*, SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms, 1027–1035, 2007

[15] Evans, Lawrence C.: *Partial Differential Equations*, Graduate Studies in Mathematics. 19 (2nd ed.). American Mathematical Society, 2010

[16] Quarteroni, Alfio, Valli, Alberto: *Numerical Approximation of Partial Differential Equations* Springer, 1994

[17] Muruhan Rathinam and Linda Petzold: *A New Look at Proper Orthogonal Decomposition* SIAM Journal on Numerical Analysis, 2003

[18] Stefan Volkwein: *Model Reduction Using Proper Orthogonal Decomposition* Lecture Notes, 2011

Sažetak

Redukcija dimenzije modela je ključan pristup rješavanju visokodimenzionalnih sustava običnih i parcijalnih diferencijalnih jednadžbi. Kako potrebe za procesiranjem velikih količina podataka koji rezultiraju u modelima visoke dimenzije rastu, ovaj pristup, razumljivo, ostaje relevantan.

U ovom radu prezentirane su metode dobivanja reduciranih baza, s fokusom na POD metodu. Pokazana je i njihova primjena na parametrizirane parcijalne diferencijalne jednažbe. Problem na koji se nailazi u primjeni POD-a uključuje nelinearne parcijalne diferencijalne jednažbe, kojima se pristupa kombinacijom POD-a i diskretne empirijske interpolacijske metode(DEIM). Pored njih, dane su i novije verzije DEIM algoritma zvane Q-DEIM i K-DEIM, unutar kojih se, redom, koriste QR dekompozicija i metode klasteriranja. Numerički eksperimenti pokazali su učikovitost ovih pristupa na sintetičke i stvarne probleme, posebno na FKPP jednadžbu, vrlo važnu u biologiji. Potencijalno neočekivani rezultati ukazuju na to da se modeli čija je dimenzija u stotinama ili tisućama mogu dovoljno točno reprezentirati sistemima od svega nekoliko jednadžbi. Ipak, metode prezentirane u ovom radu obuhvaćaju samo maleni dio ogromnog spektra metoda za redukciju dimenzije modela.

Summary

Model order reduction is an integral approach to solving high-dimensional systems of ordinary and partial differential equations. As the needs for processing large amounts of data, resulting in high-order models, are only increasing, this approach will understandably stay relevant.

Reduced basis methods are discussed, with the focus being on the proper orthogonal decomposition (POD). Their applications to parametric PDEs are presented as well. The encountered issue in this approach involves nonlinear systems, which are instead tackled by a combination of POD and the discrete empirical interpolation method (DEIM). Alongside them, newer versions of the DEIM algorithm are presented, namely the Q-DEIM and K-DEIM, which consist of utilizing the QR decomposition and clustering methods, respectively. Numerical experiments have shown the effectiveness of these approaches, to both synthetic and real-life problems, notably the FKPP equation, which is very important in biology. The potentially surprising results indicate that models whose dimensions are in hundreds or thousands may be accurately represented by systems of only several equations. However, the methods presented here encompass only a small part of the plethora of model order reduction methods.

Curriculum Vitae

Domagoj Bošnjak was born on March 17th, 1998 in Mostar, Bosnia and Herzegovina. He attended both elementary and high school in Široki Brijeg. In 2016 he began his undergraduate studies at the Department of Mathematics, Faculty of Science at the University of Zagreb, finishing in 2019. Immediately afterwards, he started graduate studies in Applied Mathematics at the same department.