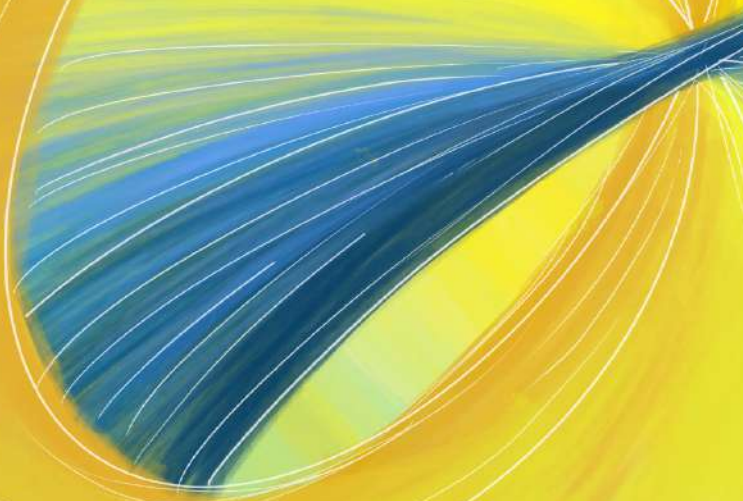
# First steps into Model Order Reduction



Alessandro Alla

impa

33º Colóquio Brasileiro de Matemática

# First steps into Model Order Reduction

**First steps into Model Order Reduction**
Segunda impressão, março de 2022

Publicado no Brasil / Published in Brazil.

**ISBN** 978-65-89124-33-7
**MSC** (2020) Primary: 62H25, Secondary: 65N06, 37M99, 37M05, 65D05

**Coordenação Geral** Carolina Araujo

**Produção** Books in Bytes **Capa** Izabella Freitas & Jack Salvador

**Realização da Editora do IMPA**
**IMPA**
Estrada Dona Castorina, 110
Jardim Botânico
22460-320 Rio de Janeiro RJ

www.impa.br
editora@impa.br

# Contents

# *Preface*

Model order reduction (MOR) methods are of growing importance in scientific computing as they provide a principled approach to approximate many modern mathematical models of real-life processes, replace high-dimensional PDEs, with low-dimensional models. The dimensionality reduction provided by MOR helps to reduce the computational complexity and time needed to solve large-scale engineering systems enabling simulation based scientific studies not possible even a decade ago.

Examples of real-time simulation settings include control systems in electronics and visualization of model results while examples for a many-query (parameterized) setting can include optimization problems and design exploration. In order to be applicable to real-world problems, often the requirements of a reduced order model are:

- a small approximation error compared to the full order model,
- conservation of the properties and characteristics of the full order model,
- computationally efficient and robust reduced order modelling techniques.

Mathematically, MOR constructs low-dimensional subspaces, typically generated by the Singular Value Decomposition (SVD), where the evolution dynamics is projected. Thus, a high-dimensional system of differential equations is replaced by a low-rank model in a systematic fashion. Three steps are required for this low-rank approximation: (i) snapshots of the dynamical system for some time instances, (ii) dimensionality-reduction of this solution data typically produced with an SVD,

and (iii) projection of the dynamics on the low-rank subspace. The first two steps are often called the *offline* stage of the MOR architecture whereas the third step is known as the *online* stage. Offline stages are exceptionally expensive, but enable the (cheap) online stage to potentially run in real time. This approach has been successfully applied to e.g. parametrized PDEs and optimal control problems.

A popular and well-established technique in MOR is Proper Orthogonal Decomposition (POD) which, in these notes, is introduced in a discrete setting. The notes can not replace a text book or research papers on the topic. Hopefully, they will be enough to get the reader excited and motivated to learn the topic more. Throughout the notes, we will discuss the method and its Matlab implementation. More information will be provided by the references[1] cited in the manuscript. At the end of the notes, we will list some possible applications of model order reduction methods.

The notes are structured as follows: In Chapter 1 we recall the finite difference method for a parabolic equation. In Chapter 2 we present the Proper Orthogonal Decomposition method and in Chapter 3 the Discrete Empirical Interpolation Method.

**Acknowledgments.** The author wishes to acknowledge IMPA for this opportunity, Carlos Tomei to support and revise this work. A deep gratitude to the colleagues who helped to discover, learn and appreciate this topic: Maurizio Falcone, Michael Hinze, J. Nathan Kutz and Stefan Volkwein.

Alessandro Alla
Università Ca' Foscari Venezia
March 2022
alessandro.alla@unive.it

[1] The list of references is by far not complete.

# 1 Finite Differences method

In this chapter we focus on the discretization of evolutive Partial Differential Equations (PDEs). We review some numerical schemes for PDEs, with emphasis on the finite difference method. We refer to the manuscripts Falcone and Ferretti (2013), Leveque (2002, 2007), and Quarteroni and Valli (1994), for finite differences, finite elements, semi-lagrangian and finite volume methods.

The semi-discretization of a PDE, say the spatial discretization, leads to a system of ordinary differential equations

$$\begin{aligned} M\dot{y}(t) &= Ay(t) + F(t, y(t)), \quad t \in (0, T], \\ y(0) &= y_0, \end{aligned} \tag{1.1}$$

where $y_0 \in \mathbb{R}^d$ is a given initial data, $M, A \in \mathbb{R}^{d\times d}$ given matrices and $F : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ a continuous function in both arguments and locally Lipschitz with respect to the second variable. It is well–known that under these assumptions there exists an unique solution for (1.1). Throughout these notes, we always assume that the model is given and known.

This wide class of problems arises in many applications, such as e.g. heat transfer or wave equations. In such cases, the dimension $d$ is the number of grid points in the spatial discretization of the PDE and can be very large. The solution

of system (1.1) may be computationally demanding and we will consider reduced order modeling techniques in the next chapters.

Let $y(t)$ be a smooth function of one variable. We approximate the time derivative $y_t(\hat{t})$ by a finite difference approximation based only on values of $y$ in a neighbourhood of $\hat{t}$. For $\Delta t > 0$, the standard one sided approximations are given by

$$y_t(\hat{t}) \approx \frac{y(\hat{t} + \Delta t) - y(\hat{t})}{\Delta t}, \tag{1.2}$$

$$y_t(\hat{t}) \approx \frac{y(\hat{t}) - y(\hat{t} - \Delta t)}{\Delta t}. \tag{1.3}$$

Approximations (1.2) and (1.3) are of first order, whereas the following centered approximation

$$y_t(\hat{t}) \approx \frac{y(\hat{t} + \Delta t) - y(\hat{t} - \Delta t)}{2\Delta t}$$

is of order two. The verification uses the Taylor expansion of $y$ at $\hat{t}$.

The centered approximation to the second derivative

$$y_{tt}(\hat{t}) \approx \frac{y(\hat{t} + \Delta t) - 2y(\hat{t}) + y(\hat{t} - \Delta t)}{\Delta t^2} \tag{1.4}$$

is also of order two.

**Exercise.** Compute approximations for the first and second derivative of $y(t) = e^t$ at $\hat{t} = 1$ for $\Delta t = \{0.1, 0.05, 0.025, 0.0125\}$. Verify the order of convergence.

The time discretization of (1.1) might be done in several ways, see Quarteroni, Sacco, and Saleri (2007). We begin by setting a temporal step size $\Delta t > 0$ and defining $t_k = k\Delta t \in [0, T]$, with $k = 0, \ldots, m$ and $t_m = T$. We will denote by $y(t_k)$ the continuous solution of (1.1) at time $t_k$, whereas by $y^k$ the numerical approximation at time $t_k$. If the method converges $y_k \to y(t_k)$ when $\Delta t \to 0$.

To build a numerical scheme for (1.1) one might use formula (1.2), say a Taylor expansion around $t_k$ for the time derivative and get the explicit Euler method:

$$M\frac{y^{k+1} - y^k}{\Delta t} = Ay^k + F(t_k, y^k), \quad y^0 = y_0, \qquad k = 0, \ldots, m-1. \tag{1.5}$$

This method is explicit: the unknown $y^{k+1}$ only depends on the solution at the previous step $y^k$:

$$My^{k+1} = y^k + \Delta t(Ay^k + F(t_k, y^k)), \quad y^0 = y_0, \qquad k = 0, \ldots, m-1.$$

If $M$ is not the identity matrix, this is a linear system at each iteration $k$.

The implicit Euler method is, on the contrary, built using a Taylor expansion around $t_{k+1}$. This leads to

$$M\frac{y^{k+1}-y^k}{\Delta t} = Ay^{k+1} + F(t_{k+1}, y^{k+1}), \quad y^0 = y_0, \qquad k = 0, \dots, m-1 \tag{1.6}$$

where it has been used (1.3) to discretize the time derivative.

The solution (1.6) is defined implicitly and requires the solution of a nonlinear equation. If we define the function

$$\mathcal{F}(x) := M(x - y^k) - \Delta t\,(Ax + F(t_{k+1}, x))\,, \tag{1.7}$$

our approximation problem at time $t_{k+1}$ reads $\mathcal{F}(y^{k+1}) = 0$.

Due to the nonlinearity of the problem, we use Newton's method to compute $y^{k+1}$. Here, we recall the standard Newton's method, which makes use the computation of $J_{\mathcal{F}}(x)$ the full Jacobian of $\mathcal{F}(x)$. There is a large literature describing faster variants for inexact Newton's method (see e.g. Quarteroni, Sacco, and Saleri (ibid.)).

The Jacobian with respect to $x$ is

$$J_{\mathcal{F}}(x) := M - \Delta t\,(A + J_F(t_{k+1}, x))\,, \tag{1.8}$$

where $J_F$ is the Jacobian of the nonlinear term $F$ in (1.1).

Newton's method gives rise to the iteration below, with initial condition $x_0$,

$$J_{\mathcal{F}}(x_i)\delta_i = \mathcal{F}(x_i) \tag{1.9}$$

$$x_{i+1} = x_i - \delta_i\,. \tag{1.10}$$

We iterate until $\|x_{i+1} - x_i\| \leqslant \varepsilon$ for a prescribed tolerance $\varepsilon$. Each iteration requires the solution of a linear system of dimension $d \times d$. The choice of $x_0$ is crucial: it is well-known that the method converges quadratically if the initial condition is *close* to the solution, e.g. to compute $y^{k+1}$ one might set the initial condition $x_0 = y^k$.

The explicit Euler method (1.5) and the implicit Euler method (1.6) have order of convergence equal to one. However, in the rest of the paper we will work with the implicit scheme which is more stable than the explicit method.

## 1.1 Finite Differences for parabolic PDEs

Let us now consider a one dimensional two points boundary value problem:

$$\begin{aligned} \widetilde{y}_t(x,t) &= \alpha \widetilde{y}_{xx}(x,t) + f(t,\widetilde{y}(x,t)), \quad (x,t) \in (a,b)\times(0,T), \\ \widetilde{y}(x,0) &= \widetilde{y}_0(x) \\ \widetilde{y}(a,t) &= 0 = \widetilde{y}(b,t) \end{aligned} \tag{1.11}$$

where $\widetilde{y}(x,t) : [a,b]\times[0,T] \to \mathbb{R}$ is the unknown, satisfying zero-Dirichlet boundary conditions, $y_0(x) : [a,b] \to \mathbb{R}$ is the initial condition and $f : [0,T]\times\mathbb{R} \to \mathbb{R}$ is given.

**Semi-discretization.** Let us start with the spatial discretization of equation (1.11). We first choose a spatial step size $\Delta x > 0$ and set $x_i = a + (i-1)\Delta x$ for $i = 1,\dots n$ and $x_n = b$. We denote by $y_i(t)$ the semi-discrete approximation of the continuous solution $\widetilde{y}(x_i,t)$ at $x_i$ with $y(t) : [0,T] \to \mathbb{R}^n$, and approximate the second derivative by the centered finite difference (1.4)

$$\widetilde{y}_{xx}(x_i,t) \approx \frac{y_{i-1}(t) - 2y_i(t) + y_{i+1}(t)}{\Delta x^2}, \quad i = 2,\dots,n-1. \tag{1.12}$$

From the boundary conditions in (1.11), $y_1(t) = 0 = y_n(t)$. The semi-discretization in space of (1.11) leads to a system of ODEs as in equation (1.1), where

$$A = \frac{1}{\Delta x^2}\begin{pmatrix} -2 & 1 & & & & \\ 1 & -2 & 1 & & & \\ & 1 & -2 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -2 & 1 \\ & & & & 1 & -2 \end{pmatrix}, \quad F(t,y(t)) = \begin{pmatrix} f(t,y_2(t)) \\ f(t,y_3(t)) \\ \vdots \\ \vdots \\ f(t,y_{n-2}(t)) \\ f(t,y_{n-1}(t)) \end{pmatrix},$$

of dimension $d = n-2$, $A \in \mathbb{R}^{d\times d}$, $F(t,y(t)) \in \mathbb{R}^d$ and the matrix $M$ is the identity matrix of dimension $d \times d$ in this context.

**Exercise.** How does the matrix $A$ and the vector $F(t,y(t))$ look like in case of nonzero Dirichlet boundary conditions $\widetilde{y}(a,t) = \beta$, $\widetilde{y}(b,t) = \gamma$ and $\beta,\gamma \in \mathbb{R}$?

Let us now consider a two dimensional two points boundary value problem,

$$\begin{aligned} \widetilde{y}_t(\xi,t) &= \alpha \Delta \widetilde{y}(\xi,t) + f(t,\widetilde{y}(\xi,t)) \quad (\xi,t) \in \Omega \times [0,T], \\ \widetilde{y}(\xi,0) &= \widetilde{y}_0(x), \\ \widetilde{y}(\xi,t) &= 0 \qquad (\xi,t) \in \partial\Omega \times [0,T] \end{aligned} \tag{1.13}$$

where $\Omega \subset \mathbb{R}^2$ is an open set, $\xi = (\xi_1, \xi_2) \in \Omega$, $\widetilde{y}(\xi,t) : \Omega \times [0,T] \to \mathbb{R}$ is the unknown, $y_0(\xi) : \Omega \to \mathbb{R}$ is the initial condition and $f(t,\widetilde{y}(\xi,t)) : [0,T] \times \mathbb{R} \to \mathbb{R}$ is a given function. The Laplace operator is $\Delta y(\xi,t) = y_{\xi_1\xi_1} + y_{\xi_2\xi_2}$. We discretize the derivatives in space following the one dimensional example. We use the same step $\Delta\xi > 0$ for both $\xi_1$ and $\xi_2$, the notation $\xi_{ij} = ((\xi_1)_i, (\xi_2)_j)$ and approximate $y\left(\xi_{ij}, t_k\right) \approx y_{i,j}^k$. Then

$$\Delta y\left(\xi_{i,j}, t_k\right) \approx \frac{y_{i-1,j}^k - 2y_{i,j}^k + y_{i+1,j}^k}{\Delta\xi^2} + \frac{y_{i,j-1}^k - 2y_{i,j}^k + y_{i,j+1}^k}{\Delta\xi^2}.$$

Using compact notations, we obtain the matrix $A \in \mathbb{R}^{n^2 \times n^2}$

$$A = \frac{1}{\Delta\xi^2} \begin{pmatrix} T & I & & & & \\ I & T & I & & & \\ & I & T & I & & \\ & & \ddots & \ddots & \ddots & \\ & & & I & T & I \\ & & & & I & T \end{pmatrix}, \quad T = \begin{pmatrix} -4 & 1 & & & & \\ 1 & -4 & 1 & & & \\ & 1 & -4 & 1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 & -4 & 1 \\ & & & & 1 & -4 \end{pmatrix}, \tag{1.14}$$

with $I, T \in \mathbb{R}^{(n-2)\times(n-2)}$ and $I$ is the identity matrix. Now, the dimension of the problem is $d = (n-2)^2$. The order of the matrix $A$ follows the *natural row-wise ordering* where we take the unknowns along the bottom row from left to right, $\{y_{11}, y_{21}, y_{31}, \ldots, y_{n1}\}$ followed by the unknowns in the second row, $\{y_{12}, y_{22}, y_{32}, \ldots, y_{n2}\}$, and so on.

## 1.2 Matlab code

In this section we provide the Matlab code for (1.13) with

$$\begin{aligned} \alpha &= 0.05, \\ \widetilde{y}_0(\xi) &= \sin(\pi\xi_1)\sin(\pi\xi_2), \\ f(t,y(t)) &= \mu(y(t)^2 - y(t)^3), \\ \mu &= 10. \end{aligned} \tag{1.15}$$

We set $\Omega = [0,1]^2$, $T = 2$, $\Delta\xi = 0.0125$, $\Delta t = 0.05$. The solution at time $t = 0$ and $t = 2$ is given in the top of Figure 1.1, whereas the contour lines in the bottom of the same figure.

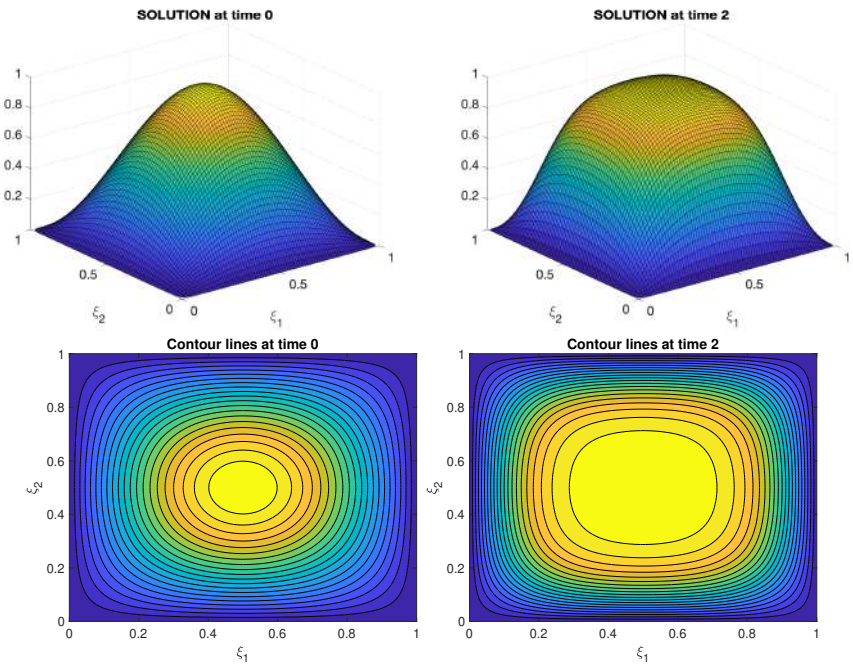



Figure 1.1: Top: Numerical approximation of (1.13) at time $t = 0$ (left) and $t = 2$ (right). Bottom: contour lines of (1.13) at time $t = 0$ (left) and $t = 2$ (right).

In the first part of the code we set the parameters used to define the problem.

```
clear
clc
close all

%Parameters
dx =0.0125;
PDE.mu = 10;
alpha = 0.05;
dt = 0.05;
```

```
x_tmp = 0:dx:1;
x = x_tmp(2:end-1);
y = x;
n = length(x);
t = 0:dt:2;
PDE.tol = 1e-5; %tolerance Newton's method

%Initial Condition
ic_cond_1 = sin(pi*x);
ic_cond = ic_cond_1'*ic_cond_1;
sol(:,1) = ic_cond(:);
```

Next, we discretize the Laplace operator. We note we used sparse matrices Matlab tools.

```
%Laplace discretization
e = ones(n,1);
A = spdiags([e -2*e e],-1:1,n,n);
A = kron(A,speye(n))+kron(speye(n),A);
PDE.A = alpha*A/dx/dx;
```

The functions $\mathcal{F}(x)$, $J_{\mathcal{F}}(x)$ in (1.7) and (1.8) are defined below. The Jacobian here is computed exactly, and defined as a sparse matrix due to the structure of the nonlinearity which is polynomial.

```
%Function and Jacobian for Newton's Method
full_sol =@(y,tmp,PDE)...
(tmp-y-dt*(PDE.A*tmp+PDE.mu*(tmp.^2-tmp.^3)));

df_full_sol = @(tmp,PDE)(speye(size(PDE.A))-...
dt*(PDE.A+spdiags(2*PDE.mu*tmp-3*PDE.mu*tmp.^2,0,n^2,n^2)));
```

Loop over time. We note that the initial condition in the Newton's method to compute $y^{k+1}$ is the solution at the previous time $y^k$. That is true for $k = 0, \ldots, m-1$.

```
%Loop over time
tic
for i =k:length(t)-1
```

```
    sol(:,k+1) = calc_newton(sol(:,k),PDE,full_sol,df_full_sol);
end
toc %CPU time for the online stage
```

A way to show the picture of the solution of (1.13) over time.

```
%% Video PDE
close all
fmin = min(min(sol));
fmax = max(max(sol));
nsteps = length(t);
fig = figure;
for i = 1:nsteps
    matrix = reshape(sol(:,i),length(x),length(y));
    matrix = [zeros(length(x),1) matrix zeros(length(x),1)];
    matrix = [zeros(1,length(x_tmp));...
                              matrix; zeros(1,length(x_tmp))];
  surf(x_tmp,x_tmp, matrix);
  zlim([fmin fmax]);
  archivo = strcat('FULL SOLUTION time = ',num2str(dt*(i-1)));
  title(archivo,'FontSize',16);
  set(gca,'FontSize',16)
  drawnow;
  pause(0.2)
end
```

To conclude the function for the Newton's method.

```
function xnew = calc_newton(x0,PDE,f,JF)

tol = PDE.tol;
err = 1;
count = 0;
tmp=x0;
xnew = x0;
while err>tol && count<500
    err_sol(count+1) = err;
    JF_c = JF(xnew,PDE);
```

```
    F_c =f(tmp,xnew,PDE);
    deltax = JF_c\F_c; %DO NOT INVERT MATRICES
    xnew = xnew - deltax;
    err = norm(deltax)
    count = count+1;
end

err_sol(count) = err;

end
```

# 2 The POD method

The aim of this chapter is to present a fast and accurate method to approximate (1.1) which reduces the dimension of the problem by means of orthogonal projections. In particular, we discuss the Proper Orthogonal Decomposition (POD), the key ingredient of these notes. More details can be found, e.g. in Benner, Gugercin, and Willcox (2015) and Gräßle, Hinze, and Volkwein (2020). We will only focus on the discrete version of POD. A continuous version is described in Gräßle, Hinze, and Volkwein (2020).

A numerical method to approximate a PDE already reduces the dimension of the problem: we pass from an infinite dimensional problem to a finite dimensional problem of the form (1.1). However, the dimension $d$ of the semi discretized problem is usually very large. The focus of model order reduction is to accurately approximate (1.1) reducing the dimension of the problem, say $\ell \ll d$. Clearly, the computational speed to approximate our problem will benefit from the reduction of the dimension. In these notes, $\ell$ will be the dimension of the reduced problem.

Let us consider a fixed matrix $\Psi \in \mathbb{R}^{d\times\ell}$ such that its columns are orthonormal vectors. Then, $\Psi^T\Psi = I \in \mathbb{R}^{\ell\times\ell}$ and the columns $\{\psi\}_{i=1}^{\ell}$ of $\Psi$ form a basis for an $\ell-$dimensional subspace $V^\ell = \text{span}\{\psi_1, \dots, \psi_\ell\} \subset \mathbb{R}^d$. An appropriate choice of $V^\ell$ would be such that the solution $y(t)$ of (1.1) can be approximated

by a linear combination of $\Psi$:

$$y(t) \approx \Psi y^{\ell}(t), \tag{2.1}$$

where $y^{\ell}(t)$ are functions from $[0, T]$ to $\mathbb{R}^{\ell}$.

If we plug assumption (2.1) into our reference problem (1.1) we obtain:

$$\begin{aligned} M\Psi \dot{y}^{\ell}(t) &= A\Psi y^{\ell}(t) + F(t, \Psi y^{\ell}(t)), \\ \Psi y^{\ell}(0) &= y_0. \end{aligned} \tag{2.2}$$

Multiplying (2.2) by $\Psi^T$ on the left,

$$\begin{aligned} \Psi^T M\Psi \dot{y}^{\ell}(t) &= \Psi^T A\Psi y^{\ell}(t) + \Psi^T F(t, \Psi y^{\ell}(t)), \\ \Psi^T \Psi y^{\ell}(0) &= \Psi^T y_0. \end{aligned} \tag{2.3}$$

This multiplication imposes the *Galerkin condition*, i.e. orthogonality with respect to residual of (2.2). In a compact notation, the reduced order model (ROM) takes the following form:

$$\begin{aligned} M^{\ell} \dot{y}^{\ell}(t) &= A^{\ell} y^{\ell}(t) + \Psi^T F(t, \Psi y^{\ell}(t)), \\ y^{\ell}(0) &= y_0^{\ell} \end{aligned} \tag{2.4}$$

where $M^{\ell} = \Psi^T M\Psi$, $A^{\ell} = \Psi^T A\Psi$, so that $M^{\ell}, A^{\ell} \in \mathbb{R}^{\ell \times \ell}$, and $y_0^{\ell} = \Psi^T y_0 \in \mathbb{R}^{\ell}$. If the dimension of the system is $\ell \ll d$, then a significant dimensionality reduction is accomplished. System (2.4) formally is the same problem of (1.1) with the huge gain of having reduced its dimension. The structure of the matrices in (2.4) is usually not preserved, e.g. if $A$ is large and sparse, its projection $A^{\ell}$ will be small but dense. The ROM (2.4) might be solved by an implicit Euler method as explained in Chapter 2.

**Exercise.** Write the Euler implicit scheme for (2.4) and define the projection of $\mathcal{F}$ in (1.7) and $J_{\mathcal{F}}$ in (1.8)

## 2.1 Singular Values Decomposition

We briefly recall the Singular Value Decomposition (SVD) of a matrix since it is strictly linked to the POD method as we will see in the next section. We refer to Golub and Loan (1996) for further details on SVD.

**Definition 1** (Singular values and vectors)**.** *Let $Y \in \mathbb{R}^{d\times m}$ be a matrix of rank $r \leqslant m$ with $d > m$. Let $\Psi := \{\psi_i\}_{i=1}^{d} \subset \mathbb{R}^d$ and $V := \{v_i\}_{i=1}^{m} \subset \mathbb{R}^m$ be the set of orthonormal vectors such that:*

$$Y v_i = \sigma_i \psi_i \quad \text{and} \quad Y^T \psi_i = \sigma_i v_i \quad \text{for } i = 1, \ldots, r. \tag{2.5}$$

*Then, $\sigma_1, \ldots, \sigma_r$ are called singular values, and the vectors $\psi \in \Psi, v \in V$ are called: right and left singular vectors respectively.*

**Theorem 1** (Existence of SVD)**.** *Let $Y = [y_1, \ldots, y_m]$ be a given matrix with real value $d \times m$ of rank $r \leqslant \min\{d, m\}$. Then, there exists a singular value decomposition of $Y$, with real numbers $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_r > 0$ and orthogonal matrices $\Psi = [\psi_1, \ldots, \psi_d] \in \mathbb{R}^{d\times d}$ and $V = [v_1, \ldots, v_m] \in \mathbb{R}^{m\times m}$ such that:*

$$Y = \Psi \Sigma V^T, \qquad \Sigma = \begin{pmatrix} D & 0 \\ 0 & 0 \end{pmatrix} \in \mathbb{R}^{d\times m}, \tag{2.6}$$

*where $D$ is the diagonal matrix with singular values.*

The following Lemma provides a uniqueness result for SVD.

**Lemma 1.** *For any matrix $Y \in \mathbb{R}^{d\times m}$ the singular values are uniquely defined. The singular vectors corresponding to the singular values greater than zero of multiplicity one are unique up to a change of the sign.*

**Observation 1.** *SVD may be interpreted as a pair of diagonalizations. If we insert the first equation in* (2.5) *into the second and vice versa we see the right singular vectors $\{\psi_i\}_{i=1}^{r}$ are eigenvectors of $YY^T$ with eigenvalues $\lambda_i = \sigma_i^2$ and the left singular vectors $\{v_i\}_{i=1}^{r}$ are eigenvectors of $Y^TY$ with eigenvalues $\lambda_i = \sigma_i^2$. Then the following equalities hold:*

$$YY^T\psi_i = \sigma_i^2\psi_i, \ \text{ and } \ Y^TYv_i = \sigma_i^2 v_i, \text{ for } i = 1, \ldots, r$$

*and for $i > r$ we have $YY^T\psi_i = 0 = Y^TYv_i$.*

Another important result for SVD concerns the optimal approximation in the Frobenius norm, defined for a matrix $A \in \mathbb{R}^{n\times m}$ as

$$\|A\|_F = \sqrt{\sum_{i=1}^{d}\sum_{j=1}^{m} |a_{ij}|^2}.$$

**Theorem 2.** *Let $Y \in \mathbb{R}^{d\times m}$ be a matrix of rank $r \leqslant m$, with $d \geqslant m$, Let $Y = \Psi\Sigma V^T$ be the SVD with singular values $\sigma_1, \ldots, \sigma_m$. We define $Y^\ell$ of rank $\ell$, obtained by setting $\sigma_{\ell+1} = \sigma_{\ell+2} = \ldots = \sigma_m = 0$ in the matrix $\Sigma$.*
*Then, $Y^\ell$ is the best approximation with respect to the Frobenius-norm of $Y$ among the matrices of rank $\ell$:*

$$\left|\left|Y - Y^\ell\right|\right|_F = \min_{\text{rank(B)}=\ell} ||Y - B||_F = \sqrt{\left(\sum_{i=\ell+1}^{r} \sigma_i^2\right)}.$$

## 2.2 Proper Orthogonal Decomposition

In this section, we explain one popular method proposed by Sirovich (1987), known as the Proper Orthogonal Decomposition (POD), to obtain an appropriate matrix $\Psi$ in (2.1).

We first need to collect data from (1.1), say the solution or approximate solution $y(t_j)$ for some time instances $\{t_0, \ldots, t_m\}$. This set of data is usually called *snapshots*. To determine the columns of $\Psi$ we solve the following minimization problem:

$$\min_{\boldsymbol{\psi}_1,\ldots,\boldsymbol{\psi}_\ell \in \mathbb{R}^d} \sum_{j=0}^{m} \left\| y(t_j) - \sum_{i=1}^{\ell} \langle y(t_j), \psi_i\rangle \psi_i \right\|_2^2 \quad \text{such that } \langle \psi_i, \psi_j\rangle = \delta_{ij}, \tag{2.7}$$

with $\delta_{ij}$ being the Kronecker delta. The vector norm, here and in the sequel, is $\langle u, v\rangle = u^T v$ and $\|u\|^2 = \langle u, u\rangle$.

The optimization problem (2.7) looks for orthonormal basis $\{\psi_i\}_{i=1}^{\ell}$ for the subspace of $\mathbb{R}^d$ which minimizes the distance between the snapshots and their projection onto these unknown basis. In other words, we search for the best approximation between our dataset and a set of linearly independent vectors $\{\psi_i\}_{i=1}^{\ell}$ with $\ell$ elements.

Let $Y = [y(t_0), \ldots, y(t_m)] \in \mathbb{R}^{d\times(m+1)}$ consists of columns given by the snapshots. The solution of (2.7) is given by the left singular vectors of the (reduced) singular value decomposition of $Y = \Psi\Sigma V^T$, where $\Psi \in \mathbb{R}^{d\times\ell}$, $\Sigma \in \mathbb{R}^{\ell\times\ell}$, $V \in \mathbb{R}^{K\times\ell}$. Thus, we will call *POD basis* of rank $\ell$ the columns $\{\psi_1, \ldots, \psi_\ell\}$ of the matrix $\Psi$ solution of (2.7).

To apply the POD method in concrete problems, the choice of the truncation parameter $\ell$ plays a crucial role. There are no a priori estimates which guarantee the ability to build a coherent reduced order model (ROM), but one can focus on heuristic considerations, introduced by Sirovich (1987), and ask for the following ratio to be close to one:

$$\mathcal{E}(\ell) = \frac{\sum_{i=1}^{\ell} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2}, \tag{2.8}$$

where $r$ is the rank of the snapshots matrix $Y$, and $\sigma_i$'s are the singular values of $Y$. This indicator is motivated by the fact that the error in (2.7) is given by the singular values we neglect:

$$\sum_{j=0}^{m} \left\| y(t_j) - \sum_{i=1}^{\ell} \langle y(t_j), \psi_i \rangle \psi_i \right\|_F^2 = \sum_{i=\ell+1}^{r} \sigma_i^2. \tag{2.9}$$

Usually, one requires $\mathcal{E}(\ell) \approx 0.999$.

We note that the error (2.9) is strictly related to the computation of the snapshots and it is not linked to the reduced dynamical system. We refer to Kunisch and Volkwein (2001) for a study of the POD error in the context of parabolic equations. In Section 2.2.2 we will show and comment on the relevance of the snapshots in model order reduction.

### 2.2.1 Offline–online decomposition

In this section we resume and comment all the steps needed to get and solve a reduced model. Four steps are required for this low dimensional approximation:

1. Computation of the snapshots $y(t_j)$. It might happen that those data are already available or they can be computed by a numerical approximation of the original high-dimensional system (1.1).

2. Computation of the POD basis $\{\psi_1, \dots, \psi_\ell\}$ of rank $\ell$. This is usually done by a SVD or eigenvalue decomposition of the snapshots matrix. The orthonormal vectors $\{\psi_i\}_{i=1}^{\ell}$ will be a base for the $\ell-$dimensional subspace of $\mathbb{R}^d$.

3. Projection of the dynamics on the low-rank subspace, i.e. compute $A^\ell$, $M^\ell$, $y_0^\ell$ in (2.4).

4. Solution of the ROM (2.4) by a numerical scheme for ODEs.

The first three steps are often called the *offline stage* of the ROM architecture, whereas the fourth step is known as the *online stage*. Offline stages are exceptionally expensive, but enable the (cheap) online stage to potentially run in real time. Randomized numerical linear algebra methods help to reduce the computational cost of the offline cost as done in Alla and Kutz (2019). Recently, a matrix-oriented approach that reduces the offline stage has been introduced in Kirsten and Simoncini (2020).

The offline part for the POD method is summarized in Algorithm 1. We first compute snapshots from the high dimensional problem (1.1). Then, we either compute the eigenvalues of $Y^TY$ or the singular value decomposition of $Y$ to obtain the POD basis. The choice of the critical parameter $\ell$ depends on the dimension of the problem $d$ and the number of snapshots $m+1$. For instance, if $m \ll d$ then it is faster to solve the eigenvalue problem for $Y^TY$. We note that we only compute the reduced SVD for a given $\ell$, that is also in general efficient if we do not consider $d \approx m$. Finally, we store the projected quantities $A^\ell$, $y_0^\ell$, $M^\ell$ to set (2.4).

**Algorithm 1** offline part for POD

**Require:** snapshots matrix $Y \in \mathbb{R}^{d\times m}$, $\ell$, flag,$A$, $M$, $y_0$.
1: **if** flag ==0 **then**
2: Compute the reduced SVD of rank $\ell$, $Y = \Psi\Sigma\Phi$
3: **else if** flag == 1 **then**
4: Determine $R = YY^T \in \mathbb{R}^{d\times d}$
5: Compute $R = \Psi\Lambda\Psi^{-1}$
6: **else**
7: Determine $K = Y^TY \in \mathbb{R}^{k\times k}$
8: Compute $K = \Phi\Lambda\Phi^{-1}$
9: $\Psi = Y\Phi\Lambda^{-1/2}$
10: **end if**
11: Compute $A^\ell = \Psi^T A\Psi$, $y_0^\ell = \Psi^T y_0$, $M^\ell = \Psi^T M\Psi$.

### 2.2.2 Matlab code and comments

We provide a Matlab code for the computation of the POD basis. The function `svds` computes a reduced version of rank $\ell$[1] of the SVD. Then, we project the matrix $A$ and the initial condition $y_0$.

```
[Psi,Sigma,V]=svds(sol,ell);
PDE.Psi = Psi(:,1:ell);
PDE.A_ell = PDE.Psi'*PDE.A*PDE.Psi;
coeff_pod(:,1) = PDE.Psi'*sol(:,1);
```

Later, we compute the projection of the function $\mathcal{F}$ in (1.7) and its jacobian (1.8) to set an implicit scheme.

```
pod_sol =@(y,tmp,PDE)(tmp-y-dt*(PDE.A_ell*tmp+...
     PDE.Psi'*PDE.mu*((PDE.Psi*tmp).^2-(PDE.Psi*tmp).^3)));

df_pod_sol = @(tmp,PDE)(speye(size(PDE.A_ell))-...
     dt*(PDE.A_ell+PDE.mu*PDE.Psi'*(spdiags(2*PDE.Psi*tmp-...
     3*PDE.Psi*tmp.^2,0,n^2,n^2)*PDE.Psi)));
```

We switch finally to the online stage. Note that Newton's method uses the essentially same function of Chapter 1, for the projection of $\mathcal{F}$ and its projected Jacobian.

```
for i =1:length(t)-1
      coeff_pod(:,i+1) = ...
          calc_newton(coeff_pod(:,i),PDE,pod_sol,df_pod_sol);
end
sol_pod = PDE.Psi*coeff_pod; %high dimensional solution
```

In Figure 2.1 we provide the POD solution, contour lines for equation (1.13) and absolute difference between the POD solution and the *exact* solution[2]. The snapshots are computed with $\Delta t = 0.05$ whereas, for the POD solution, we set $\Delta t = 0.025$. The quality of the POD approximation for $\ell \in \{1, 2, 6\}$ is clear, if compared with Figure 1.1. In the bottom line of Figure 2.1 the $z-$ scale is different.

[1] In the Matlab script we use the notation `ell`

[2] We assume that the exact solution is the approximate solution of (1.1)

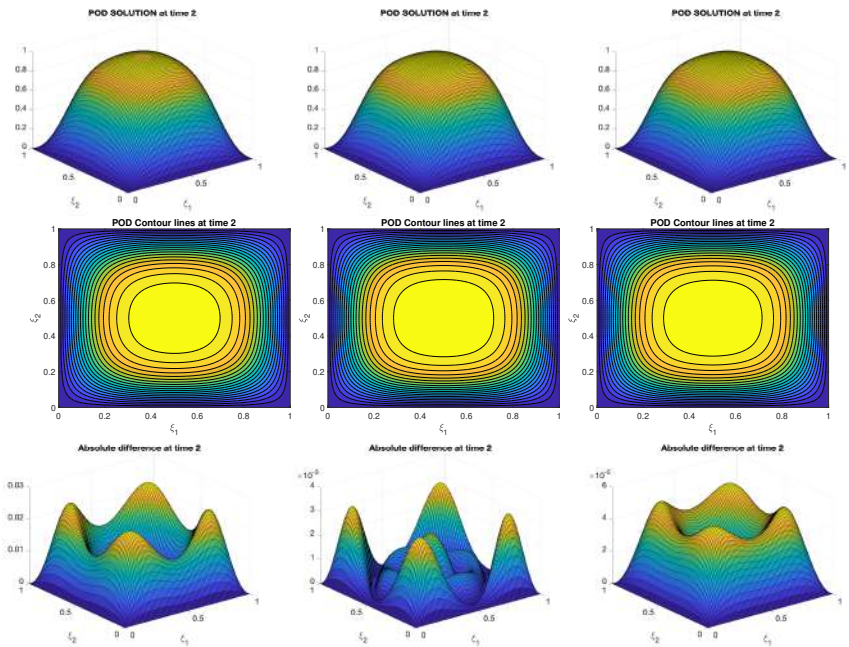



Figure 2.1: Top: POD Numerical approximation of (1.13) at time $t = 2$ for $\ell \in \{1, 2, 6\}$ (left to right). Middle: POD contour lines of (1.13) at time $t = 2$ for $\ell \in \{1, 2, 6\}$ (left to right). Bottom: Absolute difference (1.13) at time $t = 2$ for $\ell \in \{1, 2, 6\}$ (left to right).

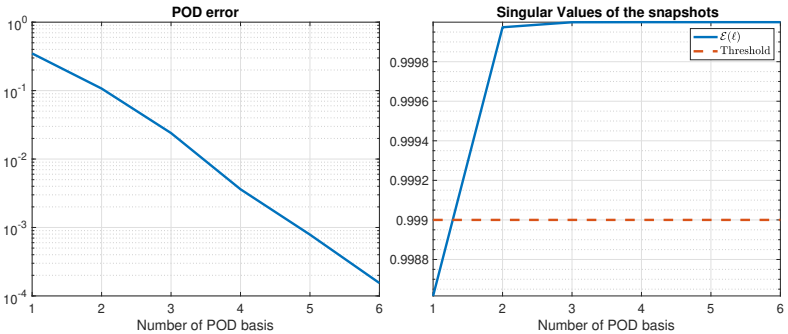



Figure 2.2: Error analysis of the POD solution for the approximation of (1.13)(left), indicator (2.8) and a possible threshold (right).

For completeness, we show the reduced matrix $A^\ell$ for $\ell = 5$.

$$A^\ell = \begin{pmatrix} 1.0123 & 0.0000 & -0.0000 & -0.0000 & -0.0000 \\ 0.0000 & 16.8132 & 0.0502 & 0.0420 & -0.0954 \\ 0.0000 & 0.0502 & 16.9314 & -0.0482 & 0.1405 \\ -0.0000 & 0.0420 & -0.0482 & 17.1884 & -0.0496 \\ -0.0000 & -0.0954 & 0.1405 & -0.0496 & 17.0287 \end{pmatrix}$$

The matrix $A$ in (1.14) is sparse, but this is not the case for its projected counterpart, e.g. the POD projection.

The quality of our POD approximation is shown in the left panel of Figure 2.2. We measure it using the infinity-norm, $\max_{t\in[0,T]} \|y(t) - \Psi y^\ell(t)\|_\infty$. As expected, the error decreases monotonically. On the right panel, we show the behaviour of the quantity of interest $\mathcal{E}(\ell)$, introduced in (2.8) which converges to 1 as $\ell$ increases. The threshold value $\mathcal{E}(\ell) = 0.999$ for the choice of basis dimension $\ell$ is indicated in red. In this case, $\ell = 2$ would suffice. As already mentioned, this indicator is very heuristic and is more related to the dataset than to the approximation of the dynamical system.

The $\ell-$dimensional model (2.4) has to be solved in real time and should allow for faster computations with respect to the high dimensional problem (1.1). Actually, the right panel of Figure 2.3 shows that the online stage in the POD approximation is slower than approximating the high dimensional problem. On the other hand, we see that when $f(t, y(t)) = 0$, i.e. for the linear heat equation ($\mu = 0$ in (1.15)), POD is way faster than the high dimensional problem. Is something

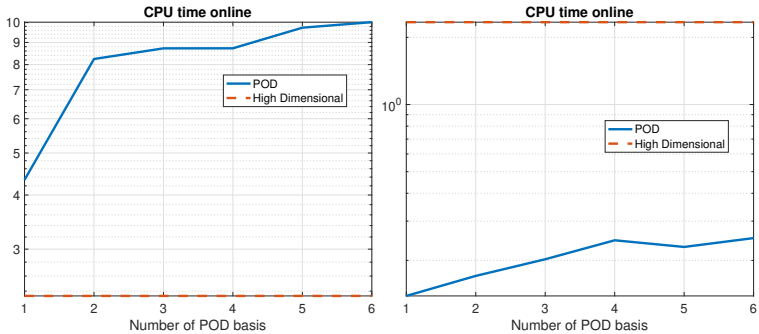



Figure 2.3: CPU time of the POD approximation for (1.13) with $\mu = 10$ (left) and $\mu = 0$ (right).

going wrong? The issue is discussed in the next chapter: why is the simulation with $f(t, y(t)) \neq 0$ in (1.13) so expensive?

It is highly suggested to compute a few number of snapshots, e.g. $m \ll d$ which is how information on the system gives rise to the the ROM. How crucial are the snapshots? Let us try to approximate (1.13) with $\mu = 10$ in (1.15), by computing the snapshots with $\mu = 0$. In other words, we compute snapshots from the linear problem and we want to see how POD works with snapshots not directly related to the problem we want to deal with. First we can compare in the left and middle panel of Figure 2.4 the POD solution with $\ell = 6$ at time $t = 2$. It is visually clear that this solution is way different from the pictures provided in Figure 1.1. Furthermore, we make a study of the error in the infinity norm which is around 0.47. We do not state the CPU time since the POD approximation is not accurate. As already mentioned, this is a snapshots based approach where the computation

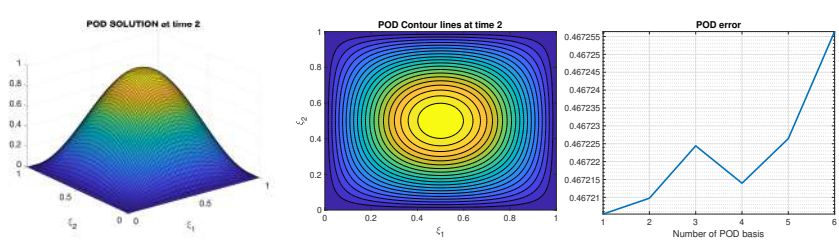



Figure 2.4: POD solution at time $t = 2$ with $\ell = 6$ (left), contour lines (middle), erro decay (right) with *wrong snapshots*.

of the snapshots is the state of the art of this approach. This example should not surprise the reader but should be considered as a confirmation that the snapshots are the key fact in this approach.

**Exercise.** Look for extrapolation properties of the POD basis. In other words, compute snapshots in $[0, T]$ and then evaluate the POD approximation in $[0, T+\gamma]$ with $\gamma > 0$. How does the POD changes with $\gamma$?

**Exercise.** Consider equation (1.13) where the parameters in (1.15) considers $\alpha \in [\alpha_{min}, \alpha_{max}]$ and $\mu \in [\mu_{min}, \mu_{max}]$. Thus, we are interested in approximating the problem for a set of parameters. How could one extend the optimization problem (2.7)?

# 3 Discrete Empirical Interpolation Method

The focus of this chapter is an efficient and accurate approximation of the nonlinear function $F : [0, T] \times \mathbb{R}^d \to \mathbb{R}^d$ in (1.1). In principle, computations are performed in dimension dimension $d$ of the problem, and the reduced order model (ROM) introduced in (2.4) does not reduce the complexity of these evaluations. Indeed, consider the nonlinearity in (2.4):

$$F^\ell(t, y^\ell(t)) := \Psi^T F(t, \Psi y^\ell(t)).$$

The variable $y^\ell(t) \in \mathbb{R}^\ell$ is first multiplied by $\Psi$ obtaining a $d-$dimensional vector $\Psi y^\ell(t) \in \mathbb{R}^d$, then $F(t, \Psi y^\ell(t))$ is evaluated and, at the end, we return back to low dimension, multiplying by $\Psi^T$. Thus the evaluation of the nonlinear term requires computing the full, high-dimensional model: the reduced model is not independent of the full dimension $d$.

To circumvent this inconvenience, the *Empirical Interpolation Method* (EIM, Barrault et al. (2004)) and its discrete counterpart, the *Discrete Empirical Interpolation Method* (DEIM, Chatarantabut and Sorensen (2010)), were introduced. We note that DEIM is built upon EIM: the two methods are essentially equivalent and

are based on a POD approach combined with a greedy algorithm. In this chapter we consider the DEIM. The interested reader is referred to Chatarantabut and Sorensen (2010) for further details. We want to approximate the nonlinear function by interpolating onto the empirical[1] basis $\{\phi_1, \ldots, \phi_k\}$ for the $k-$dimensional subspace of $\mathbb{R}^d$ as follows:

$$F(t, \Psi y^\ell) \approx \Phi c(t) = \sum_{i=1}^{k} c_i(t)\phi_i \tag{3.1}$$

with unknown coefficients $c(t) : [0, T] \to \mathbb{R}^k$ and $\Phi = [\phi_1, \ldots, \phi_k] \in \mathbb{R}^{d\times k}$. Typically, the dimension $k$ is much smaller than the full dimension $d$.

To compute the empirical basis $\{\phi_1, \ldots, \phi_k\} \subset \mathbb{R}^d$, we first define the snapshots matrix of the nonlinear terms:

$$N = [F(t_0, y(t_0)), \ldots, F(t_m, y(t_m))] \in \mathbb{R}^{d\times(m+1)} \tag{3.2}$$

where the (pre-computed) snapshots $y(t_j)$ from (1.1) are evaluated in the nonlinear part $F(t_j, y(t_j))$. Then, we compute the POD basis from $N$ as for Algorithm 1 in Section 2.2.1. We emphasize that the empirical bases $\phi_1, \ldots, \phi_k$ are computed by taking information from the nonlinear part in (1.1).

Let us define a matrix $P \in \mathbb{R}^{d\times k}$ by taking $k$ columns of a $d \times d$ *permutation matrix*[2]. Then, to compute $c(t)$, we multiply (3.1) by $P^T$ such that

$$P^T F(t, \Psi y^\ell) \approx P^T \Phi c(t) \tag{3.3}$$

and if the matrix $P^T\Phi \in \mathbb{R}^{k\times k}$ is non-singular, we obtain

$$c(t) \approx (P^T\Phi)^{-1} P^T F(t, \Psi y^\ell).$$

As $P$ is a truncated permutation matrix, the product $P^T F(t, \Psi y^\ell)$ selects $k$ rows of the nonlinear term. This selection is made by LU decomposition algorithm with pivoting (see Chatarantabut and Sorensen (ibid.)), or following the QR decomposition with pivoting (see Drmac and Gugercin (2016)) of the matrix $\Phi^T$. The latter is called Q-DEIM and it is shown to be more stable and accurate (see Drmac and Gugercin (ibid.)) than the method provided in Chatarantabut and Sorensen (2010).

[1] The term empirical means the basis will be computed by some data as will show in this section.

[2] A permutation matrix is a reordering by rows (or columns) of the identity matrix.

DEIM is particularly powerful when the function $F$ can be evaluated componentwise, so that $P^T F(t, \Psi y^\ell(t)) = F(t, \Psi P^T y^\ell(t))$, which is directly evaluated only at $k$ components.

The DEIM approximation of $F(t, y(t))$ reads

$$F(t, y(t)) \approx F^{\ell,k}(t, y^\ell(t)) := \Phi(P^T\Phi)^{-1} F(t, P^T\Psi y^\ell(t)).$$

The matrices

$$P^T\Psi \in \mathbb{R}^{k\times\ell},\ (P^T\Phi)^{-1} \in \mathbb{R}^{k\times k} \text{ and } \Psi^T\Phi(P^T\Phi)^{-1} \in \mathbb{R}^{\ell\times k}, \tag{3.4}$$

can be pre-computed independently of the full dimension $d$. This allows the reduced-order model to be completely independent of the full dimension as follows:

$$\begin{aligned} M^\ell \dot{y}^\ell(t) &= A^\ell y^\ell(t) + \Psi^T\Phi(P^T\Phi)^{-1} F(t, P^T\Psi y^\ell(t)) \\ y^\ell(0) &= y_0^\ell. \end{aligned} \tag{3.5}$$

We note that the only difference with respect to (2.4) is the low-rank approximation of the nonlinear term, i.e. we replace $F^\ell$ in (2.4) by $F^{\ell,k}$ in (3.5). Whenever we mention the POD-DEIM approach, we refer to the system (3.5) wherre it has been used $\ell$ POD basis for the projection and $k$ DEIM basis in the approximation of the nonlinearity.

At time $t \in [0, T]$, the error between $F(t, y(t))$ and its DEIM approximation $F^{\ell,k}(t, y^\ell(t))$ is given by

$$\|F(t, y(t)) - F^{\ell,k}(t, y(t))\|_2 \leqslant \bar{C}\,\|(I - \Phi\Phi^T)F(t, y(t))\|_2$$

with

$$\bar{C} = \|(P^T\Phi)^{-1}\|_2,$$

where different error performances are achieved depending on the choice of the matrix $P$ as shown in Drmac and Gugercin (2016) and the term $(I - \Phi\Phi^T)$ measures the error in the projection over the basis $\{\phi_1, \ldots, \phi_k\}$ for the subspace of $\mathbb{R}^d$.

To summarize, the main ingredients for the DEIM approach are $\{\phi_1, \ldots, \phi_k\}$ and the matrix $P$. The algorithm is below.

**Algorithm 2** DEIM algorithm

**Require:** snapshots matrix $N \in \mathbb{R}^{d\times(m+1)}$, $k$, flag

**Output:** $\Phi$, P,

1: Compute the POD basis $\Phi$ of rank $k$ from Algorithm 1,
2: Compute the QR with pivoting such that $\widetilde{P}\Phi^T = QR$,
3: Store the first $k$ columns of $\widetilde{P}$ in $P$.

## 3.1 Matlab code and comments

Equation (2.4) can be, again, approximated with an Implicit Euler scheme. To set the Newton's method, we solve the following nonlinear equation at time $t_j$ :

$$\mathcal{F}^{\ell,k}(x) := M^{\ell}(x - y^j) - \Delta t \left(A^{\ell}x + \Psi^T\Phi(P^T\Phi)^{-1}F(t, P^T\Psi x)\right) = 0, \tag{3.6}$$

with Jacobian

$$J_{\mathcal{F}}^{\ell,k}(x) = M^{\ell} - \Delta t \left(A^{\ell} + \Psi^T\Phi(P^T\Phi)^{-1}J_F(t, P^T\Psi x)P^T\Psi\right), \tag{3.7}$$

We stress again that $F(t, P^T\Psi x) \in \mathbb{R}^k$ and $J_F(t, P^T\Psi x) \in \mathbb{R}^{k\times k}$.

We then provide a Matlab code for the computation of the POD-DEIM approach detailed in Algorithm 2 for our reference problem (1.13).

```
N = PDE.mu*(sol.^2-sol.^3); %evaluation of the nonlinearity
how_many = 2; %we consider half as many snapshots
[PDE.Psi,Sigma,~]=svds(sol(:,1:how_many:end),ell); %POD basis
[PDE.Phi,Sigma2,~] = svd(N(:,1:how_many:end),k); %DEIM basis
[~,~,P] = qr(Phi','vector'); %Selection Operator
P = P(1:k);
```

The Matlab function `qr` provides $P$ as a vector, which means that $P^T\Psi$ selects $k$ rows of $\Psi$ associated to the values of $P$, i.e. with Matlab notations $\Phi(P,:)$. The quantities in (3.4) are precomputed.

```
PDE.A_ell = PDE.Psi'*PDE.A*PDE.Psi;
ic_ell = PDE.Psi'*sol(:,1);
PDE.deim_basis1 = PDE.Psi(P,:);
```

```
PDE.deim_basis2 = PDE.Psi'*(PDE.Phi*pinv(PDE.Phi(P,:)));
```

Then, we provide the Matlab handle functions for (3.6) and (3.7).

```
poddeim_sol =@(y,tmp,PDE)(tmp-y-dt*(PDE.A_ell*tmp+...
PDE.deim_basis2*PDE.mu*((PDE.deim_basis1*tmp).^2...
-(PDE.deim_basis1*tmp).^3)));

df_poddeim_sol = @(tmp,PDE)(speye(size(PDE.A_ell))-...
    dt*(PDE.A_ell+...
PDE.mu*PDE.deim_basis2*(spdiags(2*PDE.deim_basis1*tmp-...
    3*(PDE.deim_basis1*tmp).^2,0,k,k)*PDE.deim_basis1)));
```

Finally, the online stage will be given as in the previous cases just updating the nonlinear functions with (3.6) and (3.7).

```
for i =1:length(t)-1
    coeff_pod(:,i+1) = calc_newton(coeff_pod(:,i),...
    PDE,poddeim_sol,df_poddeim_sol);
end
sol_pod = PDE.Psi*coeff_pod;
```

In Figure 3.1 we provide the POD-DEIM solutions and contour lines for equation (1.13), together with the absolute difference between the POD-DEIM solution and the *exact* solution. DEIM is applied for $k = 8$, i.e., we only evaluate the nonlinear term in 8 points out of 10201. The snapshots are computed with $\Delta t = 0.05$ whereas, for the POD solution, we set $\Delta t = 0.025$. The quality of the POD approximation for $\ell \in \{1, 2, 6\}$ is clear, if compared with Figure 1.1. In the case of $\ell = 1$ and $k = 8$, the POD-DEIM solution is completely wrong. Then, we see how the approximation improves by increasing the number $\ell$. This is confirmed in the bottom panels of Figure 3.1 ( the $z-$axis scale is different).

In Figure 3.2, we show the error decay for $1 \leqslant \ell \leqslant 8$ and $k = 8$. We can see on the left panel that the POD approximation reaches its plateau for $\ell \geqslant 6$ which means the basis have extrapolated all the information. On the other hand it is also possible to see that, as expected, the POD error is a lower-bound for the POD-DEIM approach. On the right panel, we then show the CPU time in second. It is clear the huge computational gain given by the POD-DEIM method. It is

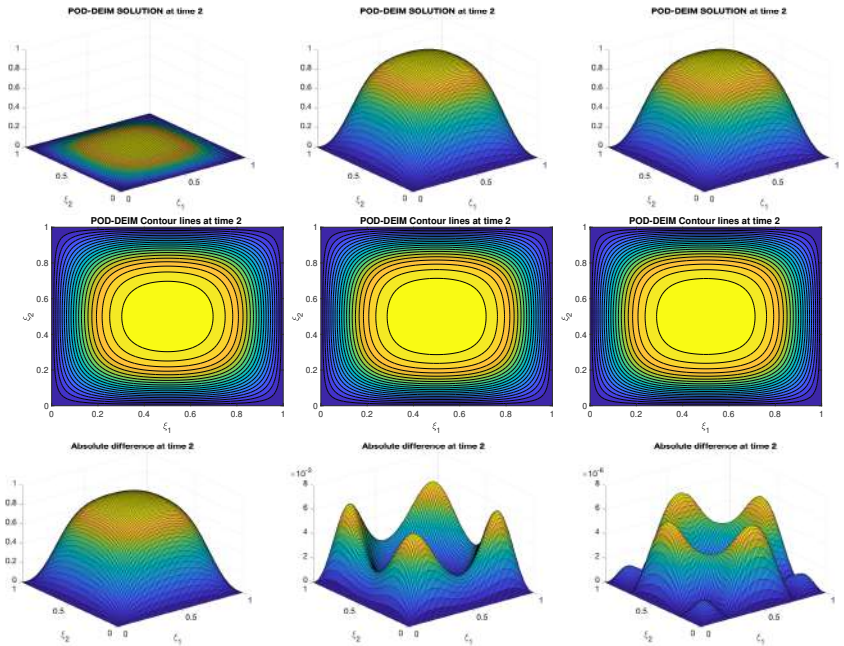



Figure 3.1: Top: POD-DEIM Numerical approximation of (1.13) at time $t = 2$ for $k = 8$, $\ell \in \{1, 2, 6\}$ (left to right). MIDDLE: POD contour lines of (1.13) at time $t = 2$ for $\ell \in \{1, 2, 6\}$ (left to right). BOTTOM: Absolute difference (1.13) at time $t = 2$ for $\ell \in \{1, 2, 6\}$ (left to right).

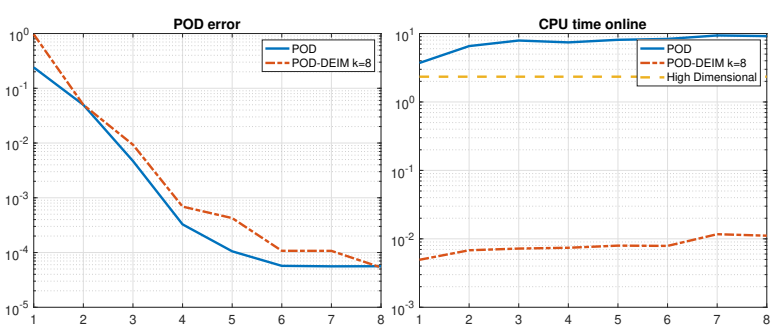



Figure 3.2: Error analysis of the POD solution for the approximation of (1.13)(left), indicator (2.8) and a possible threshold (right).

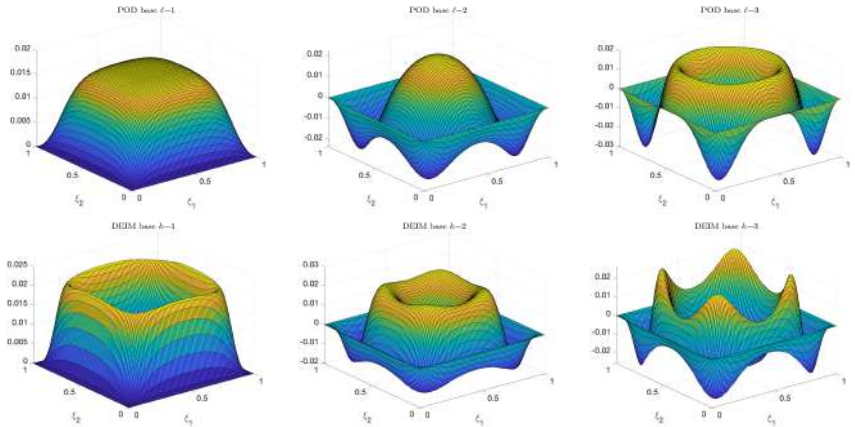


Figure 3.3: 3D plot for the first 3 POD basis (top) and DEIM basis (bottom).

important to note that the accuracy of the method is kept although its computational efficiency.

For completeness we also show in the first row of Figure 3.3 the first three POD basis for (1.13) with parameters taken in (1.15). In the second row we also show the first 3 DEIM basis.

# *Miscellaneous*

We conclude these notes with a list of other model order reduction techniques and possible applications of the methods.

**Model order reduction methods**

- Reduced Basis – RB, Quarteroni, Manzoni, and Negri (2015)
- Balance Truncation – BT, Antoulas (2005)
- Iterative Rational Krylov Algorithm – IRKA, Antoulas, Beattie, and Gügercin (2020)
- Dynamic Mode Decomposition – DMD, Kutz et al. (2016)
- Gauss–Newton with approximated tensors – GNAT, Carlberg et al. (2013)
- Proper Generalized Decomposition – PGD, Chinesta, Ladeveze, and Cueto (2011)

**Applications of model order reduction**

- Parametrized problems
- Optimal control problem (open-loop and closed-loop)
- Shape optimization
- Bifurcation problems

# *Bibliography*


A. Alla and J. N. Kutz (2019). "Randomized model order reduction." *Advances in Computational Mathematics* 45, pp. 1251–1271. Zbl: 1418.65193 (cit. on p. 17).

A. C. Antoulas (2005). *Approximation of Large-Scale Dynamical Systems*. SIAM. Zbl: 1112.93002 (cit. on p. 30).

A. C. Antoulas, C. A. Beattie, and S. Gügercin (2020). *Interpolatory Methods for Model Reduction,* SIAM (cit. on p. 30).

M. Barrault, Y. Maday, N. C. Nguyen, and A. T. Patera (2004). "An empirical interpolation method: application to efficient reduced-basis discretization of partial differential equations." *Comptes Rendus Mathematique* 339, pp. 667–672. MR: 2103208. Zbl: 1061.65118 (cit. on p. 23).

P. Benner, S. Gugercin, and K. Willcox (2015). "A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems." *SIAM Rev.* 57, pp. 483–531. MR: 3419868. Zbl: 1339.37089 (cit. on p. 12).

K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem (2013). "The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows." *Journal of Computational Physics* 242, pp. 623–647. MR: 3062051. Zbl: 1299.76180 (cit. on p. 30).

S. Chatarantabut and D. Sorensen (2010). "Nonlinear Model Reduction via Discrete Empirical Interpolation." SIAM J. Sci. Comput 32, pp. 2737–2764. Zbl: 1217.65169 (cit. on pp. 23, 24).

F. Chinesta, P. Ladeveze, and E. Cueto (2011). "A Short Review on Model Order Reduction Based on Proper Generalized Decomposition." *Arch Computat Methods Eng* 18, pp. 395–404 (cit. on p. 30).

Z. Drmac and S. Gugercin (2016). "A new selection operator for the discrete empirical interpolation method - improved a priori error bound and extensions." *SIAM J. Sci. Comput.* 38, A631–A648. MR: 3465424 (cit. on pp. 24, 25).

M. Falcone and R. Ferretti (2013). *Semi-Lagrangian Approximation Schemes for Linear and Hamilton—Jacobi Equations,* SIAM. MR: 3341715 (cit. on p. 3).

G. H. Golub and C. F. V. Loan (1996). *Matrix computations*. The Johns Hopkins University Press. MR: 1417720. Zbl: 0865.65009 (cit. on p. 13).

C. Gräßle, M. Hinze, and S. Volkwein (2020). *Snapshot-Based Methods and Algorithms, Volume 2 of Model Order Reduction*. De Gruyter (cit. on p. 12).

G. Kirsten and V. Simoncini (2020). "A matrix-oriented POD-DEIM algorithm applied to nonlinear differential matrix equations." arXiv: 2006.13289 (cit. on p. 17).

K. Kunisch and S. Volkwein (2001). "Galerkin proper orthogonal decomposition methodsfor parabolic problems." *Numer. Math.* 90, pp. 117–148. MR: 1868765. Zbl: 1005.65112 (cit. on p. 16).

J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor (2016). *Dynamic Mode Decomposition: Data-Driven Modeling of Complex Systems,* SIAM. MR: 3602007. Zbl: 1365.65009 (cit. on p. 30).

R. J. Leveque (2002). *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press. MR: 1925043. Zbl: 1010.65040 (cit. on p. 3).

— (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations, Steady State and Time Dependent Problems,* SIAM. MR: 2378550. Zbl: 1127.65080 (cit. on p. 3).

A. Quarteroni, A. Manzoni, and F. Negri (2015). *Reduced Basis Methods for Partial Differential Equations,* Springer, Cham. MR: 3379913 (cit. on p. 30).

A. Quarteroni, R. Sacco, and F. Saleri (2007). *Numerical Mathematics,* Springer. MR: 2265914. Zbl: 0913.65002 (cit. on pp. 4, 5).

A. Quarteroni and A. Valli (1994). *Numerical Approximation of Partial Differential Equations,* Springer Series in Computational Mathematics. MR: 1299729. Zbl: 0803.65088 (cit. on p. 3).

L. Sirovich (1987). "Turbulence and the dynamics of coherent structures. Parts I-II." *Quarterly of Applied Mathematics*, pp. 561–59. MR: 0910462. Zbl: 0676.76047 (cit. on pp. 15, 16).

## Títulos Publicados — 33º Colóquio Brasileiro de Matemática

**Geometria Lipschitz das singularidades** – *Lev Birbrair e Edvalter Sena*

**Combinatória** – *Fábio Botler, Maurício Collares, Taísa Martins, Walner Mendonça, Rob Morris e Guilherme Mota*

**Códigos Geométricos** – *Gilberto Brito de Almeida Filho e Saeed Tafazolian*

**Topologia e geometria de 3-variedades** – *André Salles de Carvalho e Rafał Marian Siejakowski*

**Ciência de Dados: Algoritmos e Aplicações** – *Luerbio Faria, Fabiano de Souza Oliveira, Paulo Eustáquio Duarte Pinto e Jayme Luiz Szwarcfiter*

**Discovering Euclidean Phenomena in Poncelet Families** – *Ronaldo A. Garcia e Dan S. Reznik*

**Introdução à geometria e topologia dos sistemas dinâmicos em superfícies e além** – *Víctor León e Bruno Scárdua*

**Equações diferenciais e modelos epidemiológicos** – *Marlon M. López-Flores, Dan Marchesin, Vítor Matos e Stephen Schecter*

**Differential Equation Models in Epidemiology** – *Marlon M. López-Flores, Dan Marchesin, Vítor Matos e Stephen Schecter*

**A friendly invitation to Fourier analysis on polytopes** – *Sinai Robins*

**PI-álgebras: uma introdução à PI-teoria** – *Rafael Bezerra dos Santos e Ana Cristina Vieira*

**First steps into Model Order Reduction** – *Alessandro Alla*

**The Einstein Constraint Equations** – *Rodrigo Avalos e Jorge H. Lira*

**Dynamics of Circle Mappings** – *Edson de Faria e Pablo Guarino*

**Statistical model selection for stochastic systems** – *Antonio Galves, Florencia Leonardi e Guilherme Ost*

**Transfer Operators in Hyperbolic Dynamics** – *Mark F. Demers, Niloofar Kiamari e Carlangelo Liverani*

**A Course in Hodge Theory Periods of Algebraic Cycles** – *Hossein Movasati e Roberto Villaflor Loyola*

**A dynamical system approach for Lane–Emden type problems** – *Liliane Maia, Gabrielle Nornberg e Filomena Pacella*

**Visualizing Thurston's Geometries** – *Tiago Novello, Vinícius da Silva e Luiz Velho*

**Scaling Problems, Algorithms and Applications to Computer Science and Statistics** – *Rafael Oliveira e Akshay Ramachandran*

**An Introduction to Characteristic Classes** – *Jean-Paul Brasselet*

impa

Instituto de
Matemática
Pura e Aplicada

ISBN 978-65-89124-33-7