Nonintrusive reduced order models

Florent Masmoudi

Université Fédérale
Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Florent MASMOUDI

le lundi 9 juillet 2018

Titre :

Nonintrusive Reduced Order Models

École doctorale et discipline ou spécialité :
ED MITT : Domaine Mathématiques : Mathématiques appliquées

Unité de recherche :
UMR 5219 IMT : Institut de Mathématiques de Toulouse

Directeur/trice(s) de Thèse :
Directeur : Jean-Pierre Raymond - Professeur - Université Toulouse III Paul Sabatier
CoDirecteur : Franck Plouraboué - Directeur de recherche CNRS - Institut de Mécanique des Fluides de Toulouse

Jury :
Rapporteur : Hélène Barucq - Directrice de Recherche - Inria Bordeaux Sud-Ouest
Rapporteur : Aziz Hamdouni - Professeur - Université de La Rochelle
Examinateur : Didier Auroux - Professeur - Université de Nice Sophia Antipolis
Examinateur : Manuel Bompard - Docteur - Entreprise Adagos

Remerciements,

Je tiens à remercier chaleureusement mes directeurs de thèse Jean-Pierre Raymond et Franck Plouraboué. Leurs conseils, connaissances et encouragements ont été déterminants pour l'aboutissement de cette thèse. Je tiens également à louer leur disponibilité et leur patience.

Manuel Bompard a assuré le suivi de cette thèse au sein de l'entreprise ADAGOS et je voudrais également l'en remercier.

Merci à Madame Hélène Barucq et Monsieur Aziz Hamdouni qui ont accepté d'évaluer ce travail.

En plus de l'apport de l'IMT, qui offre un environnement exceptionnel pour les doctorants, ADAGOS est une structure particulièrelent favorable à la création. Je voudrais remercier par ordre alphabétique : Houcine, Kateryna, Manuel, Mathieu, Mohamed et Victorien. Sans oublier nos complices et colocataires de Modartt.

Enfin, merci à Chloé de m'avoir soutenu et encouragé.

Contents

Introduction

Cette thèse a été réalisée dans le cadre de la Convention industrielle de formation par la recherche (CIFRE). Elle implique l'équipe MIP (Mathématiques pour l'Industrie et la Physique) de l'Institut de Mathématiques de Toulouse (IMT) et l'entreprise Adagos, à Ramonville-Saint-Agne.
Adagos est elle-même une jeune pousse de l'IMT qui a pour tutelles l'Université Paul Sabatier et le CNRS, et fut créée fin 2011. Elle s'est spécialisée dans la création de modèles réduits qui s'adaptent aux besoins des ingénieurs.
Le but de cette thèse est de construire des modèles réduits permettant de se substituer à des logiciels de simulation de systèmes physiques complexes. Ces modèles doivent être rapides à interroger.
L'état de l'art de la réduction de modèle est dominé par les méthodes intrusives qui nécessitent l'accès aux ressources internes d'un logiciel de simulation (modèle complet). Elles peuvent ainsi nécessiter un important travail d'intégration. Si elles permettent d'accélérer les calculs de manière significative leur dépendance au logiciel de simulation interdit de les embarquer sur une architecture informatique plus légère.
Pour pallier à ces inconvénients nous allons introduire dans cette thèse une gamme de modèles réduits basée sur des méthodes d'apprentissage non intrusives. L'apprentissage s'effectuera sur la base d'un nombre limité d'expériences issues du modèle complet. Une fois cette étape effectuée, le modèle réduit devra pouvoir être utilisé de manière autonome, sans avoir recours au modèle complet. Cette façon de faire entièrement non intrusive doit rendre possible un apprentissage utilisant des données mesurées plutôt que calculées.
Ce travail s'inscrit donc dans le monde des méthodes d'apprentissage. Celui-ci est en plein essor, et les algorithmes de "machine learning" et de "deep learning", toujours plus populaires, voient leur utilisation s'étendre à des domaines de plus en plus larges. S'il existe de nombreuses méthodes tournées vers la classification, la modélisation de systèmes n'est pas en reste. Les réseaux de

neurones, et notamment les réseaux profonds, sont très largement utilisés. Ils sont en quelque sorte un outil tout terrain. On peut les utiliser indifféremment en économie, en météorologie ou encore en médecine. D'après le théorème de Kolmogorv [1], ils ont l'avantage de pouvoir apprendre toute fonction continue à plusieurs variables. Toutefois, on verra que pour construire un modèle réduit d'un système physique en utilisant aussi peu de données d'apprentissage que possible, il faut incorporer dans la modélisation des propriétés physiques élémentaires.

Nous allons dans ce document développer deux types de modèles réduits, adaptées à deux classes de problèmes impliquant des physiques différentes.

Dans un premier chapitre [1], nous nous intéresserons à la mécanique linéaire. On cherchera alors à développer un modèle réduit qui nous permette d'apprendre la réponse harmonique d'un système en fonction de paramètres de conception comme sa géométrie ou les propriétés de ses matériaux. Il sera particulièrement intéressant d'observer ce qui se passe à proximité des fréquences de résonnance, qui se déplacent quand les paramètres du système changent. Pour cela nous modéliserons la réponse sous la forme d'une fraction dont nous détaillerons la nature et le processus d'identification.

Dans un second chapitre [2], nous étudierons la mécanique des fluides incompressibles. On considèrera un système modélisant un écoulement. Cette fois-ci le système ne sera pas modifié. La géométrie sera fixe, la nature du fluide également. Seules les excitations dynamiques, comme la vitesse du fluide à l'entrée ou la pression extérieure, changeront. On cherchera à identifier par apprentissage un modèle réduit dynamique permettant de prédire ce qui se passe pour toute nouvelle excitation dynamique. Le modèle réduit étant non linéaire et instationnaire, notre principal défi visera à définir une structure nous garantissant, par construction, la stabilité du schéma numérique mis en oeuvre.

Pour ces deux types de physique évoqués, nous vérifierons qu'incorporer des lois physiques élémentaires, caractérisant le phénomène en présence, nous permet d'acquérir une bonne capacité de prédiction en utilisant un nombre réduit de données d'apprentissage. On s'assurera également que les modèles ainsi développés resteront suffisamment flexibles et généraux pour pouvoir traiter une classe de problèmes assez large, ne nécessitant pas d'engager un travail d'adaptation conséquent à chaque nouvelle expérience.

Enfin, dans un troisième chapitre [3], nous nous pencherons sur une problématique quelque peu différente. On s'intéressera à la méthode des Sparse Grids, qui permet de réaliser efficacement l'approximation de fonctions qui ont un grand nombre de variables d'entrée en utilisant un nombre modéré de

points d'apprentissage. On introduira une stratégie permettant de les rendre plus économes encore.

Chapter 1

zROM: Reduced Order Models for Structural Analysis

1.1 Introduction

Structural analysis is a major field in engineering. Figuring out what will be the behavior of a mechanical component during its building phase is critical in many projects. As a matter of fact, resonant frequencies of mechanical systems can lead to safety issues or damages if they happen to be badly distributed.
Structural analysis is a strategic field in automotive, aeronautic or construction industries for example. Let us consider a car engine. It usually goes from 0 to 4 000 revolutions per minute (rpm). And other excitations like gear shifting bring in some higher frequencies. The engineers, in charge to design it, add some damping. But that does not make resonant frequencies disappear. They try to keep them far from ideal engine regime, let us say 1 500 rpm for example. Unfortunately when gears are shifted the resonant frequencies tend to move. That gives an insight of the complexity of the problem. Trying to find an optimal design of the engine to guarantee its good behavior is really computationally intensive. Computer simulations are usually performed using the finite elements method (FEM) [2]. A new computation is required for every single frequency value and every single design configuration. Exploring the space of frequencies and design configurations to find out what could be the best design becomes an almost impossible mission. The exploration must be refined as a slight perturbation of the design can result in a significant change in the behavior.
To address this problem we are going to build a Reduced Order Model (ROM) of

the harmonic response of a system based on a few computations for some design configurations and frequencies. The method is fully non-intrusive, it does not require any further information from the solver. In fact it could use measured data as well. It relies on the machine learning philosophy to which we add some simple but fundamental physical properties.
This method has been prototyped using a test case provided by Ansys that we are going to introduce right now.

1.2 Motivation: The industrial case

This work has been initiated in partnership with the Ansys company. They provided us with a qualitative test case to create a prototype. We consider a pipe made of steel and full of liquid (Figure 1.1). Its length may vary. This pipe is an element of a more global system from General Electrics. The pipe is clamped at one end. A longitudinal forced is applied to the other end.

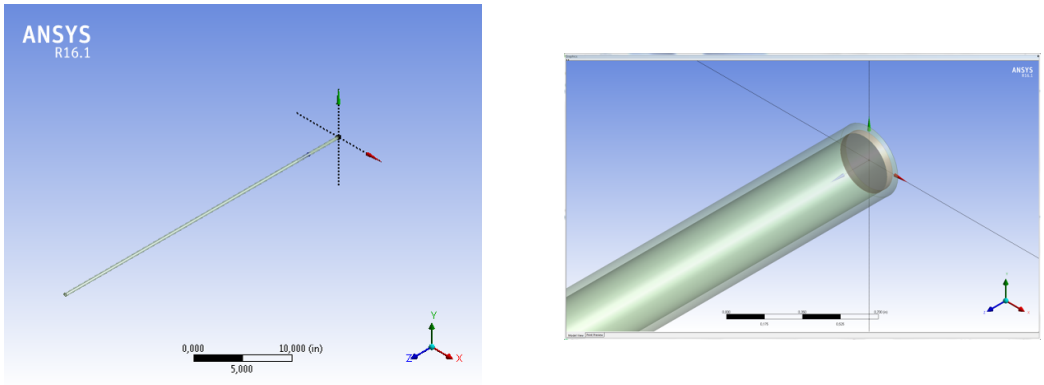


Figure 1.1 – Geometry

Our goal is to perform an harmonic analysis for any length of the pipe within a specified range. We call f and p_L respectively the frequency and the length parameter. Let us denote by x the vector of the longitudinal displacements at the mesh nodes of the FEM code. Let us note that the mesh is gently morphed when p_L changes. We aim at building a reduced order model of

$$(f, p_L) \rightarrow x(f, p_L), \quad \forall (f, p_L) \in [f_{min}, f_{max}] \times [p_{L_{min}}, p_{L_{max}}], \tag{1.2.1}$$

on the basis of n_l learning data $X_l = \left[x(f_1, p_{L_1}), \ldots, x(f_{n_l}, p_{L_{n_l}}) \right]$ computed via the ANSYS Mechanical solver. This method is fully nonintrusive. That means

that it does not require further information from the solver (to perform projections or any other operation). It only relies on outputs of the solver for some (f, p_L) configurations.
We will see that the structure of the proposed reduced order model is quite general and may be used to model the solution of parameterized linear systems. But it gets particularly interesting when the parameters lead to singularities. That is why we are going to introduce the model in the general field of linear elasticity.

1.3 Linear Elasticity

The equations of linear elasticity model materials as continua and define how solid objects deform and get internally stressed depending on loading conditions. The linearity assumption is legitimate if the deformations are small enough. In the special case of metals, the yield point (the limit of elastic behavior) is usually far enough to get a wide range of deformations in which the assumption remains appropriate.

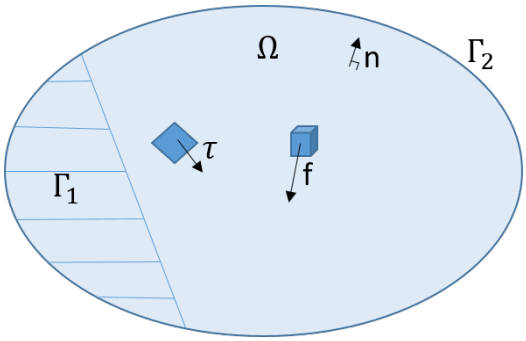


Figure 1.2 – Domain

Let us have a look at the governing equations for a system with no damping on a domain Ω

- The equation of displacement:

$$\nabla \cdot \sigma + f = \rho \ddot{u} \quad \text{in} \quad \Omega. \tag{1.3.1}$$

- The equation linking strain and displacement:

$$\varepsilon = \frac{1}{2}\left[\nabla u + (\nabla u)^T\right]. \tag{1.3.2}$$

- The Hooke's law that is a constitutive equation for elastic materials:

$$\sigma = D : \varepsilon \iff \sigma_{ij} = D_{ijkl}\varepsilon_{kl} \tag{1.3.3}$$

In the previous equations σ is the Cauchy stress tensor, ε is the strain tensor, u is the displacement vector, ρ is the mass density, D is the stiffness tensor, and f is the force per unit volume.
The stiffness sensor D has the classical following symmetry properties:

$$D_{ijkl} = D_{jikl} = D_{ijlk} = D_{klij} \tag{1.3.4}$$

When the material is homogeneous the coefficients of D are constant. And in case it is isotropic as well, we can show that D only depends on 2 coefficients: the Lamé parameters (λ, μ), or Young' modulus E and Poisson's coefficient ν

$$\sigma_{ij} = \frac{E}{1+\nu}\left(\varepsilon_{ij}(u) + \frac{\nu}{1-2\nu}\varepsilon_{ll}(u)\delta_{ij}\right). \tag{1.3.5}$$

The relation between (E, ν) and (λ, μ) is

$$E = \frac{\mu(\mu+3\lambda)}{\mu+\lambda}, \quad \nu = \frac{\lambda}{2(\mu+\lambda)} \ , \tag{1.3.6}$$

and

$$\mu = \frac{E}{2(1+\nu)}, \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \ . \tag{1.3.7}$$

The boundary conditions are usually in the form

$$\begin{cases} u = 0 \iff u_i = 0 \quad \text{on} \quad \Gamma_1 \\ \sigma.n = \tau \iff \sigma_{ij} n_j = \tau_i \quad \text{on} \quad \Gamma_2 \end{cases} \tag{1.3.8}$$

This whole set of equations can be reduced to one linear second order Partial Derivative Equation(PDE) of u

$$\rho\ddot{u} - \frac{1}{2}\nabla \cdot \left(D : \left(\nabla u + (\nabla u)^T\right)\right) = f, \tag{1.3.9}$$

with the boundary and initial conditions.

1.3.1 The weak form

Let us define the space V

$$V = \left\{ v = (v_1, v_2, v_3 | v_i \in H^1(\Omega); v_i = 0 \quad \text{on} \quad \Gamma_1 \right\}. \tag{1.3.10}$$

For all v in V, we have

$$m(\ddot{u}, v) + a(u, v) = l(v), \tag{1.3.11}$$

where

- the bilinear form m is
$$m(w, v) = \int_\Omega \rho w v, \tag{1.3.12}$$
where ρ is the density,
- the bilinear form a is
$$\begin{aligned} a(u, v) &= \int_\Omega \sigma_{ij} \varepsilon_{ij}(v) \\ &= \int_\Omega D_{ijkl} \varepsilon_{kl}(u) \varepsilon i j(v) \end{aligned} \tag{1.3.13}$$
- and the linear form l is
$$l(v) = \int_\Omega f_i v_i + \int_{\Gamma_2} \tau_i v_i. \tag{1.3.14}$$

1.3.2 The Finite Element Method

Now we assume that Ω is meshed. V^h is the finite dimension space associated to the shape functions of the mesh. In most cases the mesh is not time dependent.
Let N be the dimension of V^h. We consider a basis $\{w_1, \ldots, w_N\}$ in V^h. We are going to look for an approached solution u^h of 1.3.11 in the form

$$u^h(t, \xi) = \sum_{j=1}^{N} x_j(t) w_j(\xi). \tag{1.3.15}$$

Let $x \in \mathbb{R}^N$ be the vector whose components are x_j. x is solution of

$$M\ddot{x} + Kx = l, \tag{1.3.16}$$

where

$$M = \left[m_{ij}\right] = \left[m(w_i, w_j)\right], \tag{1.3.17}$$

$$K = \left[k_{ij}\right] = \left[a(w_i, w_j)\right], \tag{1.3.18}$$

$$l = [l_i] = [l(w_i)]. \tag{1.3.19}$$

$M \in \mathbb{R}^{N\times N}$ is called the mass matrix, which is symmetrical and positive-definite (usually lumped into a diagonal matrix).
$K \in \mathbb{R}^{N\times N}$ is called the stiffness matrix, which is symmetrical and positive-definite.

To this point we have not taking into account possible damping effects. They may come from the properties of the material constituting the solid itself or from external interactions such as energy dissipation due to interaction with a fluid flow. To model this we add a simple time derivative term:

$$M\ddot{x} + C\dot{x} + Kx = \ell. \tag{1.3.20}$$

C is called the matrix of the damping effects. It usually comes from a first order spatial differential operator. But $C \in \mathbb{R}^{N\times N}$ is sometimes set empirically in the form of a diagonal matrix.

To this point we have considered the general case of linear elasticity. But equation 1.3.20 is actually very general in the field of mechanics. The structure of the matrices involved may change but this general equation is common to

- linear elasticity
- plate models
- shell models
- etc.

1.4 The parameterized system

Equation 1.3.20 stands for a given design configuration. What happens when we change the design geometry or the material properties of the system ?

Let us call $p = (p^1, \dots, p^{n_p}) \in \mathbb{R}^{n_p}$ the vector that characterizes those changes. For example, if $p = (p^1, p^2)$, p^1 could stand for the length of the pipe test case 1.2 and p^2 could be Young's modulus.

- In a general manner, when p^k only stands for material properties, the matrices M, C and K gently depend on p^k. For example, when the density ρ is changed, only M is affected, alternatively when Young's modulus is tuned, only K is modified.
- Alternatively, if p^k depicts changes in the design geometry we need to be more cautious. First we need a mesh's structure that does not change with p^k. Second the morphing of the mesh depending on p^k needs to be smooth. In that case we can also say that M, C and K depend smoothly on p^k. In most cases we may assume that this dependency is analytic. Depending on how the applied force is affected by the design geometry changes, even ℓ may become a function of p^k. If the pipe from test case 1.2 is stimulated with force at one end of the beam uniformly spread over its surface, the magnitude per surface unit may change with the design for example.

Taking into account the vector of parameters p is the cornerstone for further work. Equation 1.3.20 changes into

$$M(p)\ddot{x}(t) + C(p)\dot{x}(t) + K(p)x(t) = \ell(t,p). \tag{1.4.1}$$

1.5 Frequency domain

Let us enter the frequency domain using the Fourier transform. We denote by $\hat{x}(\omega)$ and $\hat{\ell}(\omega, p)$ the Fourier transforms of $x(t)$ and $\ell(t,p)$ respectively.

From now on we will use indifferently the pulsation ω and the frequency f that are related via the equality $\omega = 2\pi f$.

Equation 1.4.1 becomes

$$\left(-\omega^2 M(p) + i\omega C(p) + K(p)\right)\hat{x}(\omega) = \hat{\ell}(\omega, p). \tag{1.5.1}$$

For convenience, we will drop the $\hat{\cdot}$ sign above frequency variables, as we will mainly work in the frequency domain.

Our goal being to perform an harmonic analysis, the right-hand side that is used usually does not depend on ω, and we get

$$\ell(\omega, p) = L(p). \tag{1.5.2}$$

Objective

Our main objective is to build an explicit model of

$$(\omega, p) \rightarrow x(\omega, p), \tag{1.5.3}$$

where $x(\omega, p)$ is solution to a linear system in the form

$$\left(-\omega^2 M(p) + i\omega C(p) + K(p)\right) x(\omega, p) = L(p). \tag{1.5.4}$$

where $p = (p^1, \ldots, p^n)$. The model will be implicit in the sense that it will **not** come from the solution of a reduced system that approximates 1.5.4.
We want this model to be valid in a given box, i.e. for

$$(\omega, p^1, \ldots, p^n) \in \mathscr{D} = \mathscr{D}_\omega \times \mathscr{D}_p \tag{1.5.5}$$

where $\mathscr{D}_\omega$ and $\mathscr{D}_p$ are defined as follows

$$\begin{aligned} \mathscr{D}_\omega &= [\omega_{\min}, \omega_{\max}] \\ \mathscr{D}_p &= \left[{p^1}_{\min}, {p^1}_{\max}\right] \times \cdots \times \left[{p^n}_{\min}, {p^n}_{\max}\right]. \end{aligned} \tag{1.5.6}$$

The model will be built in a fully non-intrusive manner. It does not need to get information about M, C or K. It does not even require to get access to L either. It only uses a set of n_l computed solutions of x at some learning configuration points $(f_i, p_i) \in D$

$$\text{Learning Set} = X_l = \left[x(\omega_1, p_1), \ldots, x(\omega_{n_l}, p_{n_l})\right]. \tag{1.5.7}$$

More generally, later in this chapter, we will be interested in systems in the form

$$\mathscr{A}(\omega, p)x(\omega, p) = L(p), \tag{1.5.8}$$

where ω represents the frequency, or even

$$\mathscr{A}(\lambda, p)x(\omega, p) = L(p). \tag{1.5.9}$$

where λ can be complex.

1.6 The eigenproblem

Before going into to the details of how the reduced model of $x(\omega, p)$ will be built, we are going to study the singularities of the matrix $\mathscr{A}(\lambda, p)$, where $\lambda \in \mathbb{C}$ is a complex scalar (which is more general that the real pulsation $\omega \in \mathbb{R}$). The so called singularities are the points (λ, p) where $\mathscr{A}(\lambda, p)$ is not invertible. This analysis is important as those singular points play a major role in the behavior of $x(\omega, p)$.

First, let us consider the case where there is no parameter p. We call eigenvalue problem the system

$$\mathscr{A}(\lambda)x = (-\lambda^2 M + i\lambda C + K)x = 0. \tag{1.6.1}$$

This is a quadratic eigenvalue problem. Rewriting the system, it is equivalent to a linear eigenvalue problem whose dimension is doubled.
We look for its non trivial solutions. If (λ_i, x_i) is a non-trivial solution to 1.6.1, we call λ_i an eigenvalue or a pole, and x_i the corresponding eigenvector or mode.

Now, let us introduce the more general multivariate case where we incorporate p. In the same manner we call multivariate eigenvalue problem the system

$$\mathscr{A}(\lambda, p)x = (-\lambda^2 M(p) + i\lambda C(p) + K(p))x = 0. \tag{1.6.2}$$

Once again the eigenelements are the non-trivial solutions.
If, for a given p, λ_i^p and x_i^p form a non-trivial solution to 1.6.1, we call λ_i^p an eigenvalue or a pole, and x_i^p the corresponding eigenvector or mode, at p.
Can we express those eigenelements as regular functions of p ?
Kato's book [3] gives us some answers at page 65:

Theorem 1 *Let us consider a classical linear eigenvalue problem of type*

$$T(p) - \lambda I_N = 0, \tag{1.6.3}$$

*where p is now a complex **scalar** and $p \to T(p)$ is an holomophic function from $D_0 \in \mathbb{C}$ to $\mathbb{C}^{N\times N}$. If p is restricted to a simply-connected domain D_0 (for the complex plan that implies there is no hole in it) containing no exceptional points (where the total number of distinct eigenvalues changes), we can write the eigenelements with respect to p as follows:*

- *The eigenvalue i*

$$p \to \lambda_i(p) \in \mathbb{C}^N \tag{1.6.4}$$

- *and the corresponding eigenmode*

$$p \to x_i(p) \in \mathbb{C}^N \tag{1.6.5}$$

that are both holomorphic functions of p.

This can be generalized to eigenvalue problems in the form $\mathscr{A}(\lambda, p)x = (-\lambda^2 M(p) + i\lambda C(p) + K(p))x$.

However the properties of the eigenvalues from theorem 1 are not really convenient as it only works on a domain that contains no exceptional points and that cannot even surround one of them (while still excluding it) either. This is indeed very limiting.
Furthermore, in a general manner the behavior of the eigenvalues with respect to p is more complicated than the one of T. For example, when $p \to T(p)$ is only polynomial the eigenvalues are not. In a general manner, when $p \to T(p)$ is a really gentle function, there is no reason that the behavior of the eigenvalues with respect to p is gentle too. We will observe that in example 1.6.1 where $p \to T(p)$ is only linear.
On the contrary, the characteristic polynomial $\det(T(p) - \lambda I)$ shares overall the same properties as $p \to T(p)$. If T is continuous, or polonymial, or holomorphic, so will be $(\lambda, p) \to \det(T(p) - \lambda I)$. We will see later on that we will mainly rely on this feature to build the reduced model.
Finally, if p is not a scalar but a vector we have no guarantee that we can extend the theorem.

For all these reasons, we will not try to model the eigenelements themselves as it is a really tricky job. We will see in section 1.7.3 that this will prevent us from trying to adapt methods that are the state of the art in the case where there is no parameter p.

1.6.1 A relevant toy problem

Let us illustrate the difficulty to parameterize the eigenelements of a parameterized matrix. We consider a matrix M depending on $p = (\mu, \nu)$ as follows:

$$M(\mu,\nu)=\begin{bmatrix}\mu & \nu\\ \nu & -\mu\end{bmatrix}. \tag{1.6.6}$$

We now address the usual eigenvalue problem stated as

$$\left(M(\mu,\nu)-\lambda I_2\right)x=0. \tag{1.6.7}$$

The eigenvalues are the roots of the characteristic polynomial

$$\chi_M(\lambda,\mu,\nu)=\det(M(\mu,\nu)-\lambda I_2)=\lambda^2-\mu^2-\nu^2. \tag{1.6.8}$$

As μ and ν parameterize M, it is relevant to observe the behavior of the eigenvalues when they move in the (μ,ν) plan.
The roots of polynomial 1.6.8 are $\lambda=\pm\sqrt{\mu^2+\nu^2}$. Figure 1.3 depicts the behavior of these eigenvalues with respect to ν for $\mu=0$ and $\mu=\varepsilon>0$. The eigenvalues are the two branches of an hyperbola.

Figure 1.3 – Eigenvalues trajectory along ν for $\mu=0$ (left) and $\mu=0.05$ (right)

The explicit mode parameterization is difficult and unstable.

1.7 Modeling x as a rational function

Now, let us focus on the modeling of $x(\omega,p)$ which is solution to

$$\mathscr{A}(\omega,p)x(\omega,p)=L(p). \tag{1.7.1}$$

We are going to see how important it is to use a rational function as the reduced order model of x.

1.7.1 An eigenelements-based rational model

First, let us only consider the case where there is no extra parameter p. If $\mathscr{A}$ and ℓ are holomorphic functions of ω and $\det(\mathscr{A}(\omega))$ is not identically zero, then $x(\omega)$ is a meromorphic function of ω, i.e. the ratio between an holomorphic function from $\mathscr{D}_\omega$ to $\mathbb{C}^n$ and a holomorphic function from $\mathscr{D}_\omega$ to the field $\mathbb{C}$.
An interesting representation of the behavior of x with respect to ω is:

$$x(\omega) = \sum_{i=0}^{s} \sum_{j=1}^{j(i)} \frac{S_{i,j}\ell(\omega)}{(\omega - \lambda_i)^j} + H(\omega)\ell(\omega), \tag{1.7.2}$$

where

- H is a $n \times n$ matrix-valued holomorphic functions,
- λ_i are the eigenvalues,
- $S_{i,j}$ are rank j matrices.

This representation is really useful to picture how x behaves with respect to ω, in relation with the eigenelements. But the fact that it explicitly uses the eigenelements of $\mathscr{A}$ makes it unconvenient when extra parameters p are introduced. However it shows how important it is that the model of x relies on rational functions.

1.7.2 A multivariate friendly rational model

To overcome the difficulties induced by extra parameters we are going to rewrite x in a different manner. It directly comes from Cramer's rule. And this time, p will not cause much difficulty. If we consider

$$\mathscr{A}(\omega, p)x(\omega, p) = L(p), \tag{1.7.3}$$

then

$$x_i(\omega, p) = \frac{\det(\mathscr{A}_i(\omega, p))}{\det(\mathscr{A}(\omega, p))}, \tag{1.7.4}$$

where $\mathscr{A}_i$ is the matrix formed by replacing the i^{th} column of $\mathscr{A}$ by the column vector L.
From this result, if $\mathscr{A}$ and L are polynomials of p, so are the numerator and the

denominator. On the contrary, even in the case where p is a scalar, the eigenvalues of $\mathscr{A}$ are generally not polynomials of p even on an appropriate subspace. More generally, when $(\omega, p) \to \mathscr{A}$ and $p \to L(p)$ are holomorphic functions (Definition 1), both the numerator and the denominator are holomorphic. In addition,

$$(\omega, p) \to x(\omega, p) \tag{1.7.5}$$

is a meromorphic function of (ω, p), as it comes from the ratio of a vector-valued holomorphic function and scalar-valued holomorphic function (that should not be identically zero).

Definition 1 *Let $\mathscr{D}$ denote an open subset of $\mathbb{C}^n$. Let $f : \mathscr{D} \to \mathbb{C}^n$. The function f is analytic at a point p in $\mathscr{D}$ if there exists an open neighborhood of p in which f is equal to a convergent power series in the n complex variables. We define f to be holomorphic if it is analytic at each point in its domain.*

In addition, Osgood's lemma [4] shows that, if f is a continuous function and is holomorphic in each variable separately, then f is holomorphic as a multivariate function.
The formulation 1.7.4 is also interesting as the denominator does not depend on the right-hand side and is shared by every element of x. It is the key to catch singularities.
We can see how the denominator of x plays a massive role in its behavior, especially when we are close to singularities of $\mathscr{A}$. This is the main challenge of building the ROM.

This is the corner stone for the creation of our reduced order model
We are going to build a reduced order model y of x in the form

$$y(\omega, p) = \frac{\alpha(\omega, p)}{\beta(\omega, p)} \approx x(\omega, p), \tag{1.7.6}$$

where α is a vector-valued function and β is a scalar function. A particular attention will be paid to keep α and β regular enough.
We are not going to look for the singularities (or poles) themselves (unlike the Vector Fitting method [7]) but we rather look directly for good α and β as combinations of simple basis functions (1.8.2). However, that does not mean that we will not be able to recover the eigenelements (cf. paragraph 1.8.9).

1.7.2.1 What about the infinite dimensional problem?

When we deal with systems whose size can get really large it is always meaningful to have an insight into the infinite dimensional case. If good properties are preserved in the infinite dimensional case, that usually is a good sign to address a large dicretized problem. Here the question is the following: Is the rational function still a good approach?
To answer that question we use the multivariate Steinberg theorem [9]. It depicts the generalization of the Steinberg Theorem when there are more than one parameter that perturbs an operator.

Theorem 2 *Let X be a complex Banach space. $\mathscr{B}(X)$ denotes the set of bounded linear operators on X.*
$\mathscr{B}(X)$ is a Banach algebra (with operator composition as multiplication) and we denote its unit by I.
Let $\mathscr{D} \in \mathbb{C}^d$ be a connected open set.
Suppose that the mapping

$$\begin{array}{rcl} T\mathscr{D} & \rightarrow & \mathscr{B}(X) \\ z & \rightarrow & T(z) \end{array} \tag{1.7.7}$$

is holomorphic in $\mathscr{D}$, and that $T(z)$ is compact for all $z \in \mathscr{D}$.
Then we have the following alternative:

(i) either $\forall z \in \mathscr{D}$, $I - T(z)$ is not invertible,

(ii) or $z \rightarrow [I - T(z)]^{-1}$ is meromorphic in $\mathscr{D}$.

In our case, $z = (\omega, p) \in \mathscr{D} = \mathscr{D}_\omega \times \mathscr{D}_p$.
Let us remind equation 1.3.9 from section 1.3:

$$\rho \ddot{u} - \frac{1}{2}\nabla \cdot \left(D : \left(\nabla u + (\nabla u)^T\right)\right) = f, \tag{1.7.8}$$

To make things simpler we only consider Dirichlet boundary conditions, therefore

$$u \in V = H_0^1(\Omega). \tag{1.7.9}$$

We can now introduce parameters to the system and consider the frequency domain

$$-\omega^2 \rho(p)\hat{u} - \frac{1}{2}\nabla \cdot \left(D(p) : \left(\nabla \hat{u} + (\nabla \hat{u})^T\right)\right) = \hat{f}(p). \tag{1.7.10}$$

which is the continuous equivalent of the finite dimension problem we consider in this chapter.
We assume that the mappings $p \to D(p)$ and $p \to \rho(p)$ are holomorphic, and that $D(p)$ is invertible for all $p \in \mathscr{D}_p$.
We set $\mathscr{A}(\omega, p)$ such that

$$\begin{aligned} \mathscr{A}(\omega, p)\hat{u} &= -\omega^2 \rho(p)\hat{u} - \frac{1}{2}\nabla \cdot \left(D(p) : \left(\nabla \hat{u} + (\nabla \hat{u})^T\right)\right) \\ &= (-\omega^2 M(p) + K(p))\hat{u} \end{aligned} \tag{1.7.11}$$

where $M(p)$ and $K(p)$ are linear operators in $\mathscr{B}(V \cap H^2(\Omega), L^2(\Omega))$ such that

$$M(p)\hat{u} = \rho(p)\hat{u}, \tag{1.7.12}$$

and

$$K(p)\hat{u} = -\frac{1}{2}\nabla \cdot \left(D(p) : \left(\nabla \hat{u} + (\nabla \hat{u})^T\right)\right). \tag{1.7.13}$$

Therefore the mappings $p \to M(p)$ and $p \to K(p)$ are holomorphic. Furthermore, $M(p) \in \mathscr{B}(V \cap H^2(\Omega), L^2(\Omega))$ is a bounded operator.
If Ω is regular enough, $K(p) \in \mathscr{B}(V \cap H^2(\Omega), L^2(\Omega))$ is an isomorphism for all $p \in \mathscr{D}_p$. In that case, from lemma 6 in [9], the mapping $p \to K(p)^{-1}$ is holomorphic as well.
For all $(\omega, p) \in \mathscr{D}$, let us define $G(\omega, p) \in \mathscr{B}(V \cap H^2(\Omega), V \cap H^2(\Omega))$ as

$$G(\omega, p) = \left(I - \omega^2 K(p)^{-1} M(p)\right), \quad \forall (\omega, p) \in \mathscr{D}. \tag{1.7.14}$$

We can now rewrite $\mathscr{A}(\omega, p)$ as

$$\mathscr{A}(\omega, p) = K(p)G(\omega, p), \quad \forall (\omega, p) \in \mathscr{D}. \tag{1.7.15}$$

We are interested in

$$\mathscr{A}(\omega, p)^{-1} = G(\omega, p)^{-1} K(p)^{-1}. \tag{1.7.16}$$

We want to prove that $(\omega, p) \to G(\omega, p)^{-1}$ is meromorphic.
The restriction of $M(p)$ to $V \cap H^2(\Omega)$ is a bounded operator in $\mathscr{B}(V \cap H^2(\Omega), V \cap H^2(\Omega))$.
In addition the restriction of $K(p)^{-1}$ to $V \cap H^2(\Omega)$ is a compact operator in $\mathscr{B}(V \cap H^2(\Omega), V \cap H^2(\Omega))$.
As a consequence $(-\omega^2 K(p)^{-1} M(p))$ is a compact operator in $\mathscr{B}(V \cap H^2(\Omega), V \cap H^2(\Omega))$ as the result of the composition of a bounded operator and a compact

operator .
Hence, from theorem 2, the mapping

$$(\omega, p) \rightarrow G(\omega, p)^{-1} = \left(I - \omega^2 K(p)^{-1} M(p)\right)^{-1} \tag{1.7.17}$$

is meromorphic.
Thus, by composition, the mapping

$$(\omega, p) \rightarrow \mathscr{A}(\omega, p)^{-1} \tag{1.7.18}$$

is meromorphic as well.
Now, assuming that the mapping $p \rightarrow \hat{f}(p)$ is holomorphic, we can define x be such that

$$x(\omega, p) = \mathscr{A}(\omega, p)^{-1} \hat{f}(p). \tag{1.7.19}$$

We can state that $(\omega, p) \rightarrow x(\omega, p)$ is meromorphic. Hence from Weierstrass factorization theorem it can be expressed as the ratio of two holomorphic mappings. The numerator is an holomorphic mapping from $\mathscr{D}$ to $V \cap H^2(\Omega)$ and the denominator is an holomorphic mapping from $\mathscr{D}$ to the field $\mathbb{C}$.

The damping operator To this point we have omitted any damping effect $C(p)$ so that

$$\mathscr{A}(\omega, p) = -\omega^2 M(p) + i\omega C(p) + K(p) \tag{1.7.20}$$

In practice, this operator is generally at most a first order spatial differential operator. That leads to the property that $K^{-1}C(p)$ is compact as well. Hence, writing

$$\mathscr{A}(\omega, p) = K(p) \left(I + \underbrace{K(p)^{-1}(-\omega^2 M(p) + i\omega C(p))}_{\text{compact}} \right) \tag{1.7.21}$$

we can extend in a straightforward way the previous results.

1.7.3 About state-of-the-art methods to build rational models

ROMs for rational functions are quite well mastered when there is only one parameter (usually the frequency). In that case, the reference in the industry must be the Vector Fitting method [7].

1.7.3.1 Vector Fitting

The Vector Fitting method [7] is a powerful tool to address the question of modeling frequency domain response of a system based on a set of computed or measured data (for different frequency values). It is generally used to approximate a scalar function depending on frequency ω in the form of a **rational function**. The key feature is to set an adequate number of poles (the zeros of the denominator) and to locate them properly. These poles are moved iteratively to fit learning data. This method is really efficient to achieve a transfer function approximation.
Let us call g the targeted function

$$g(\omega) = \frac{\alpha_0 + \alpha_1\omega + \alpha_2\omega^2 + \cdots + \alpha^N\omega^N}{\beta_0 + \beta_1\omega + \beta_2\omega^2 + \cdots + \beta^N\omega^N} \tag{1.7.22}$$

The Vector Fitting method aims at approximating g by G in the form

$$G(\omega) = \sum_{n=1}^{N} \frac{c_n}{\omega - a_n} + d + h\omega. \tag{1.7.23}$$

The method requires evaluations of g for some ω values as learning data. The strength of the Vector Fitting Method relies on a smart way to address the pole location problem by solving a sequence of linear systems.

This is a reliable method when the only variable is the frequency and consequently the poles are scalars. When there are additional parameters p, the unknowns and in particular the poles become functions of p. Modeling those poles is complex and may not be a good option. We have seen through the toy example in 1.6.1 how the behavior of the poles with respect to p can really be complicated.
In a general manner, we have to avoid modeling the eigenelements themselves if we want a reliable way to build a reduced order model.

1.7.3.2 Multivariate Padé approximant

Padé approximant is another way to perform a frequency domain response fitting. The $[m/n]$ Padé approximant of an holomorphic scalar function f is

$$g(\omega) = \frac{\sum_{j=0}^{m} a_j\omega^j}{1 + \sum_{k=1}^{n} b_k\omega^k}, \tag{1.7.24}$$

such that its first $m+n$ Taylor series terms at 0 are the same as those of f.
The Padé approximant is likely to be closer to f than the truncated Taylor series at x far from 0.

When extra parameters are introduced, we can use multivariate Padé approximant method [8].
It is based on the calculation of successive derivatives of g at one particular (ω_0, p_0). Because of that, the method is known to be unstable. In addition, it is intrusive and needs to access the solver. Hence it cannot be generalized to learning from measured data.

1.7.4 Back to the toy problem

Let us go back to the toy problem from 1.6.1. We are going to see that modeling the solution itself can be much easier. In this toy problem, we now consider a vector $b \in \mathbb{C}^2$, and look for v such that

$$(M - \lambda I_2)\, v = b. \tag{1.7.25}$$

The solution $v = (v_1, v_2)^T$ can be written

$$v_1(\lambda, \mu, \nu) = -\frac{(\mu+\lambda)b_1 - \nu b_2}{\lambda^2 - \mu^2 - \nu^2}, \tag{1.7.26}$$

and

$$v_2(\lambda, \mu, \nu) = \frac{(\mu-\lambda)b_2 - \nu b_1}{\lambda^2 - \mu^2 - \nu^2}. \tag{1.7.27}$$

We can observe that both the numerators and the denominator are simple polynomial function much easier to fit than the stiff eigenvalues presented earlier.

1.8 The Reduced Order Model

1.8.1 The learning data

To build the reduced order model, we need learning data:

$$X_l = \left[x(\omega_1, p_1), \ldots, x(\omega_{n_l}, p_{n_l})\right]. \tag{1.8.1}$$

The vectors $x(\omega_k, p_k)$ are solutions to $\mathscr{A}(\omega_k, p_k)x(\omega_k, p_k) = L(p)$ at n_l learning points $(\omega_k, p_k)_{1\le k\le n_l}$ in the domain of interest. We denote by n_x the length of x.

1.8.2 Structure of the Reduced Order Model

Based on 1.7.2 we are building a reduced order model $y(\omega, p)$ that aims at approaching $x(\omega, p)$ in a box domain. We look for $y(\omega, p)$ in the form

$$y(\omega, p) = \frac{\alpha(\omega, p)}{\beta(\omega, p)} \approx x(\omega, p), \tag{1.8.2}$$

where α is a vector-valued function and β is a scalar function. Those functions are supposed to be smooth.
In practice, the model is more precisely

$$y(\omega, p) = \frac{\alpha(\omega, p)}{\beta(\omega, p) + 1} \approx x(\omega, p). \tag{1.8.3}$$

where β is chosen to be orthogonal to the constant function 1 in the domain. As a result, there is uniqueness of α and β for a given model y.
$\alpha = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_{n_x} \end{pmatrix}$ and β are built using a set of n_{bf} basis functions

$$\alpha_i = \sum_{j=1}^{n_{bf}} a_{i,j} g_j(\omega, p), \tag{1.8.4}$$

$$\beta = \sum_{j=1}^{n_{bf}-1} b_j g_{j+1}(\omega, p). \tag{1.8.5}$$

where $a = (a_{i,j})_{\substack{1 \le i \le n_x \\ 1 \le j \le n_{bf}}}$ and $b = (b_1, \ldots, b_{n_{bf}-1})$.
We will see later in 1.4 that $g_1 = 1$, and that g_j, for $j > 1$ is L^2-orthogonal to 1 over the domain of interest (therefore its average is zero).
The whole problem is then to identify a and b solutions to the following optimization problem:

$$\min_{a,b} \sum_{k=1}^{n_l} \| \frac{\alpha(\omega_k, p_k, a)}{\beta(\omega_k, p_k, b) + 1} - x(\omega_k, p_k) \|_2^2. \tag{1.8.6}$$

1.8.3 The basis functions

To this point we have not yet mentioned the choice of the functions used to build α and β, i.e. the functions $(g_j)_{1 \le j \le n_{bf}}$. Their shape and how we pick them

are two crucial points. We are going to describe how we proceed and what is our motivation.
The most sensitive aspect of the model creation is to avoid that the denominator is equal to 0 where it should not be. To prevent this the denominator has to be relatively smooth. The role of the basis functions is crucial in this respect. At the same time the numerator needs to be smooth as well. If not, it tends to work by itself and prevent the denominator from doing the job during the learning phase. Which is unsuitable as getting a good denominator is the key to build a qualitative reduced order model.
In this section we are going to study

- the benefit of choosing an appropriate number of basis functions n_{bf},
- the creation of a partially ordered set of multivariate functions from the smoothest to the sharpest,
- the selection process of the basis functions accordingly to the learning points.

1.8.3.1 The number of basis functions

Assuming that the learning dataset X_l is a $n_x \times n_l$ matrix, what number of basis functions n_{bf} should we pick? Because of the denominator we have $n_{bf} \times (n_x + 1)$ unknowns and only $n_l \times n_x$ equations. These $n_l \times n_x$ equations can even be almost linearly dependent (we will see in section 1.8.4 that we will project X_l in a smaller space).
Therefore, working with $n_{bf} = n_l$ would imply that we need to regularize. However we have found out that this is not suitable. Let us illustrate the reason with the following cases:

- L^q **norm of the model output** y: This is not pertinent as $(\omega, p) \rightarrow x(\omega, p)$ is not even supposed to be in L^q, $q \geq 1$.
- L^2 **norm of the numerator** α **and the denominator** β: We try to insure that $\sum_i^n \|\alpha_i(\omega, p)\|_2^2$ and $\|\beta(\omega, p) + 1\|_2^2$ remain small. Let us remind that the denominator is set in the form $\beta + 1$ where β is orthogonal to 1. The L^2 regularization of both the numerator and the denominator leads to small numerator and denominator. This choice causes an important issue. The most simple case to illustrate this is when there is no damping in which case the solution is real. The poles are real and the denominator needs to

cross the zero plane. But because of the regularization β gets close to -1 (the denominator then gets close to zero) when needed and goes back to zero (denominator back to 1). The denominator never crosses zero and is not smooth at all. The results have never been satisfactory using this method.

- H^k **norm of the** α **and** β: To ensure that the numerator and the denominator are smooth we have tried to use the H^k norm. The problem gets really unstable, in the sense that depending on how strong we want the regularization to be the results vary a lot. Many issues occur with this type of regularization. Sometimes the denominator is lazy and let the numerator do the job. Some other time the denominator is smooth in a region and locally hectic in another region. We could try to handle norms from Sobolev spaces $W^{k,q}$ with q greater than 2 in order to make sure that the derivatives are small everywhere. But anyway the behavior of the model in this form is highly unpredictable.

At the end we have found out that the most effective way to address the problem is by simply taking a reduced number of naturally smooth basis functions. This is called regularization by discretization. To put it simply, we sort candidate basis functions following a smoothness criterion and we only pick the first r ones where

$$r < n_l \frac{n_x}{n_x + 1}. \tag{1.8.7}$$

This method has proven to be the most efficient.

1.8.3.2 Shape of the basis functions

As the domain on which we build the model is a box, we have made the choice to build multivariate basis functions from 1D ones using tensor product. Let us note that in this section the frequency parameter ω is treated as any other parameter.

Shape of the 1D functions: We aim at building an orthonormal set of 1D functions $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ that can naturally be sorted from the "smoothest" to the most complicated one. Those properties will allow us to select carefully a limited number of appropriate multivariate basis functions in section 1.8.3.3. To match those properties, we first choose to work with Legendre polynomials.

The orhtonormality criterion is by nature satisfied and we have

$$\deg(\varphi_i) = i - 1. \tag{1.8.8}$$

where *deg* gives the degree of the polynomial. That gives us a naturally sorted set of functions from the smoothest to the most complicated. This choice was also driven by the need to approach analytic functions (i.e. the numerators and the denominator) with a limited number of functions.
However, in terms of fitting a function using a set of learning data points this choice has drawbacks under certain circumstances. For example, even in the simplest case we want to approach a function using a polynomial model (i.e. there is no denominator), Runge's phenomenon [26] depicts a problem of oscillation of the model at the edges of the interval that occurs when using uppermost high degree polynomials to interpolate a set of equally spaced points. In that case a solution exists and is given by taking the Chebyshev nodes as interpolation points. But we cannot always monitor the set of learning points. This phenomenon still stands when we do not interpolate but rather use a polynomial regression.
Carrying out experiments we have observed that polynomials are efficient in a small domain where the model does not require a large number of basis functions to be accurate. Problems come when the domain gets larger and requires more functions. In essence we still need regular functions so that the model remains global: we need to take benefit from the learning points in a large region to predict at one point. But we do not want a learning point very faraway from a prediction point to play a major role in the model at this very prediction point. For this reason we choose to use splines [11], that are by the way a well known alternative to overcome Runge's phenomenon.

Splines are piecewise polynomials. We first tried working with cubic splines (piecewise cubic polynomials) that are the most common ones. There were quite efficient, but their lack of regularity made them perform less well than polynomials in many cases (especially when the degree of the polynomials did not need to grow too much). In order to get the best of both worlds we chose to work with higher order splines. A k-order spline is a piecewise polynomial of degree k whose global smoothness is C^{k-1} . Junctions between to pieces are called breaks. Therefore, if there are n pieces there are $n-1$ breaks. At piece

number j the spline formula is

$$Y_j(x) = \sum_{i=0}^{k} c_j^i x^i, \quad \forall j \in \{1,\dots,n\}. \tag{1.8.9}$$

There are $(k+1)n = kn+n$ unknowns. At each break the spline has to be continuous, as well as its $k-1$ first order derivatives. That gives us $k(n-1) = kn-k$ equations. Thus we need $n+k$ additional equations so the spline is correctly defined.

In our case the additional equations come naturally from the hierarchical orthonormality criterion. It works as follow:

- We choose k, the maximum degree of the standard polynomials we agree to use.
- For $i \leq k+1$ the 1D function g_i in our set is the usual i^{th} Legendre Polynomial.
- Whenever $i > k+1$, g_i is a k-order spline with $n = i-k$ pieces whose breaks are equally distributed on the interval. Therefore it needs $n+k = i$ additional equations to be set. They come naturally from the fact that we want g_i to be orthogonal to the $i-1$ preceding functions in the set and that its norm has to be 1. In practice that means that once we get to $i = k+2$ we start with a first break in the middle of the interval. Then for every increment the number of breaks increases just by one.

Experiments have shown that $k = 6$ is an efficient option. In that case the 10 first basis functions are depicted in Figure 1.4.

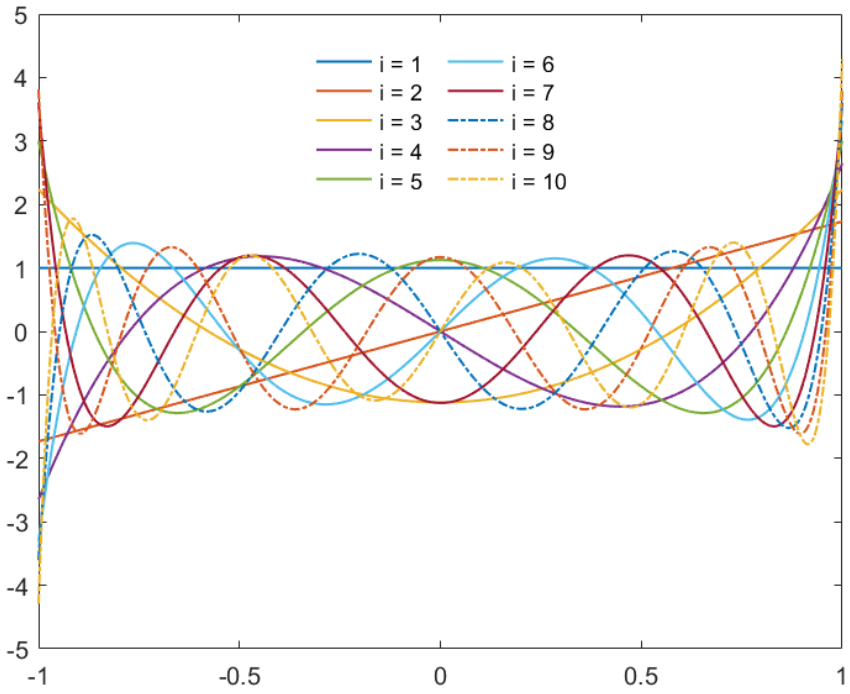


Figure 1.4 – The first 10 basis functions for k=6.

This gives us a nice set of 1D functions that are well ordered from the smoothest to most complicated in order to build a set of multivariate functions. Let us have an insight into the 2D case.

The 2D case: We change the box domain we consider into $\mathscr{D} = [-1,1] \times [-1,1]$ without any loss of generality. The set of 1D functions is $\Phi = \{\varphi_1, \ldots, \varphi_m\}$ as introduced above. Then we can define the basis functions $g_{i,j}$ from 1.8.2 as

$$g_{i,j}(x,y) = \varphi_i(x) \times \varphi_j(y). \tag{1.8.10}$$

As Φ is set of orthonormal functions

$$\int_{-1}^{1} \varphi_i(x)\varphi_j(x)dx = \delta_{i,j}, \tag{1.8.11}$$

then, it naturally leads to

$$\begin{aligned}\int_{\mathscr{D}_p} g_{i,j}(X)g_{k,l}(X)dX &= \int_{-1}^{1}\int_{-1}^{1}\left(\varphi_i(x)\varphi_j(y)\right)\left(\varphi_k(x)\varphi_l(y)\right)dxdy \\ &= \left(\int_{-1}^{1}\varphi_i(x)\varphi_k(x)dx\right)\left(\int_{-1}^{1}\varphi_j(y)\varphi_l(y)dy\right) \\ &= \delta_{i,j,k,l}.\end{aligned} \tag{1.8.12}$$

The new ND set of vectors is orthonormal as well. And it is partially ordered.

1.8.3.3 Selection of the basis functions

We need to make a choice among the available basis functions in order to build a reliable model. From previous section we know that any mutlivariate basis function comes from the multiplication of 1D ones.
The 1D functions are totally ordered, but alternatively comparing the smoothness of the ND functions is not straightforward.
However we need to find a way to totally order them for the purpose of being able to introduce them iteratively, one by one, during the learning process. To do so we are going to rely on the design of experiment (i.e. the set of learning points). We only consider the case we have been given a set of learning points that is set once and for all. Our goal is to chose the right functions for a given set of learning points. To illustrate the strategy we can omit the rational model and only consider the more general and simpler case where we want to build a simple function approximation thanks to those basis functions.
Let $P = \{p^1, \ldots, p^l\}$ be the set of the n_l learning points in a space of dimension d. To make things even simpler we are now considering the particular case where $d = 2$. Everything will be easily generalized to any greater value for d.
We denote by x^k and y^k the two coordinates of p^k:

$$p^k = \left[x^k, y^k\right]^T. \tag{1.8.13}$$

The set of 1D basis functions is $\Phi = \{\varphi_1, \ldots, \varphi_m\}$. Let us remind ourselves that the φ_i are orthonormal and sorted according to their smoothness.
For any pair (i, j) we have

$$g_{i,j}(x, y) = \varphi_i(x)\varphi_j(y). \tag{1.8.14}$$

Let us denote by

$$m_{i,j} = \begin{pmatrix} g_{i,j}(x^1, y^1) \\ \vdots \\ g_{i,j}(x^k, y^k) \\ \vdots \\ g_{i,j}(x_l^n, y_l^n) \end{pmatrix} \tag{1.8.15}$$

the vector of the evaluation of $g_{i,j}$ at the learning points. To pick the basis functions we are going to build step by step a matrix M composed of $m_{i,j}$ vectors. To initialize the process the first basis function we pick is $g_{1,1}$. Hence, at first

$$M = \left(m_{1,1}\right). \tag{1.8.16}$$

Then we wonder if we should add first $g_{1,2}$ or $g_{2,1}$. To decide which is the best candidate we test the two possible ways to make the M matrix evolve

$$M^1 = \left[M, m_{1,2}\right], \tag{1.8.17}$$

and

$$M^2 = \left[M, m_{2,1}\right]. \tag{1.8.18}$$

We extend the notion of condition number (cond) to a rectangular matrix as the ratio of its largest and its lowest singular value, and we compute

$$c^1 = \text{cond}(M^1), \tag{1.8.19}$$

and

$$c^2 = \text{cond}(M^2). \tag{1.8.20}$$

At that point we look for the smallest value between c^1 and c^2. This gives us a notion of which basis function will least deteriorate the stability of the inverse problem once added, according to the learning points.
If $c^1 < c^2$ we add $g_{2,1}$, conversely if $c^2 < c^1$ we pick $g_{1,2}$. In case of equality we add both of them (this may happen usually because of symmetry properties in the learning set).
Repeating this, we iterate the process and make the set of basis functions evolve. At each step, we have to know what are the functions that are candidates to be added. A function $g_{i,j}$ is candidate if and only if

- if $i > 1$ and $j > 1$, $g_{i-1,j}$ and $g_{i,j-1}$ are already in the set

- if $i = 1$, $g_{1,j-1}$ is already in the set

- if $j = 1$, $g_{i-1,1}$ is already in the set

Once the candidates have been identified we test them individually by adding them in the M matrix and only keep the one(s) that induce(s) the smallest condition number(s).
Figures 1.5 to 1.8 show in which order the functions are added for several 2D design of experiment. In the spirit of the Sparse Grids method 3, we call *levels* the complexity order of the 1D functions. Hence g_i is the $level - i$ 1D function. In the same manner we say that $g_{i,j}$ is a function whose level is $i \times j$. That means that it has been built as the product of the $level - i$ 1D function (i.e. g_i) for the first variable and the $level - j$ 1D function (i.e. g_j) for the second variable. The corresponding added functions are marked with a blue circle and the order in which they have been added is written nearby. In each case we add at most as many functions as learning points (as the condition number is infinite beyond that limit), although in practice we should stop picking functions earlier.

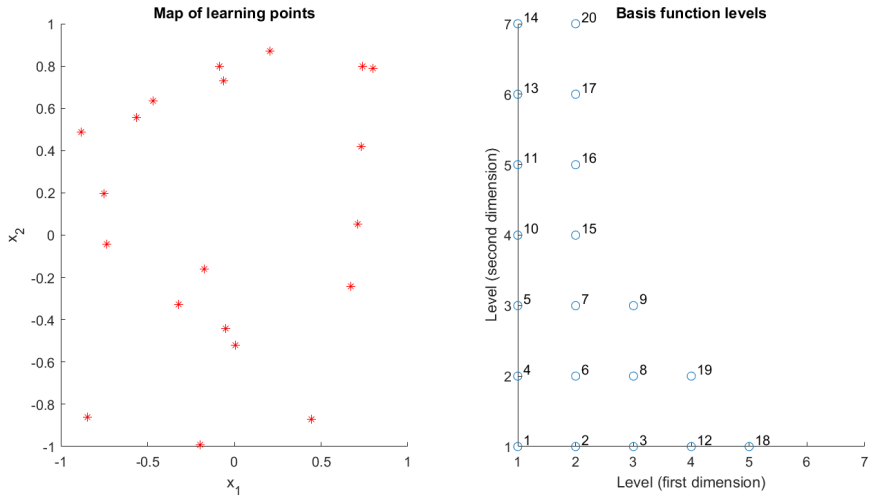


Figure 1.5 – Random grid

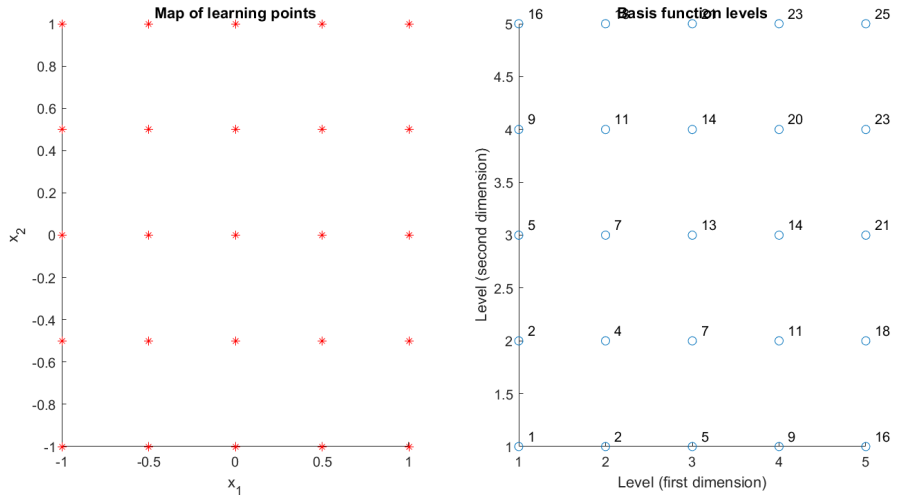


Figure 1.6 – Cartesian grid

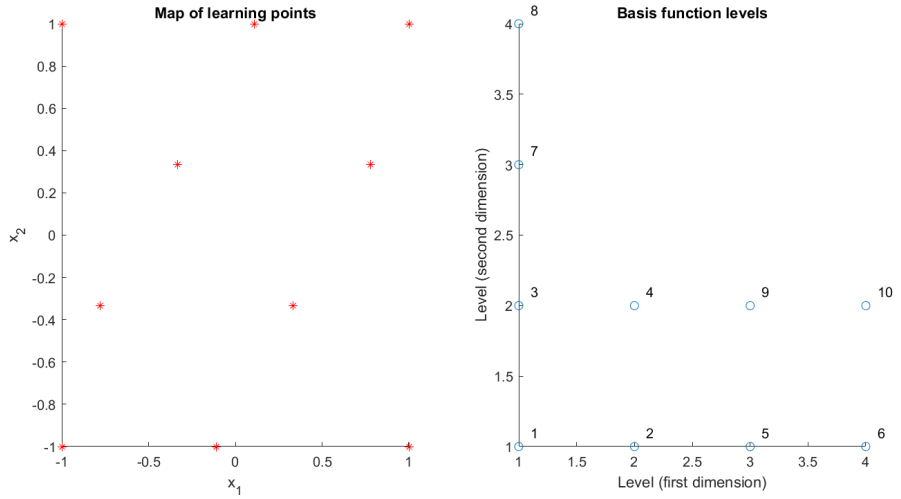


Figure 1.7 – Another grid

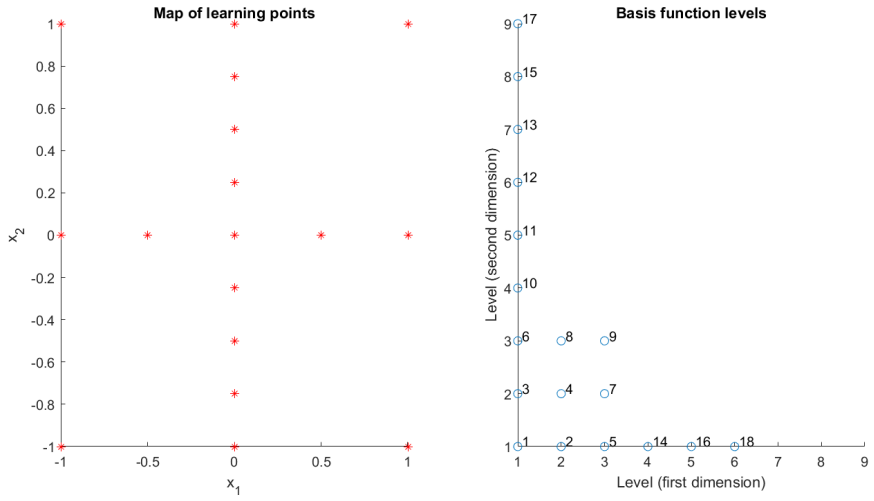


Figure 1.8 – Sparse grid

We generalize this process to higher dimension numbers.

1.8.4 Preprocessing of the learning data: Enhancing the process using the singular value decomposition (SVD)

The number of elements n_x of vector x can be really large. In the industrial test case presented in introduction 1.2 $n_x = 13564$ for example. The most important feature to get a good model for x is to correctly set the denominator β monitored by the coefficient vector b. Indeed, it is common to every element in x and it has a huge role in their behavior. But we do not need to consider every element of x to correctly identify b. We only need a few representative coefficients that depict well the field x. So we are going to preprocess the learning data in order to make the identification of the model easier.
Let us remind the Cramer's rule 1.7.4 in the case of a problem in the form $\mathscr{A}(\omega,p)x(\omega,p) = L(p)$

$$x_i(\omega,p) = \frac{\det(\mathscr{A}_i(\omega,p))}{\det(\mathscr{A}(\omega,p))}. \tag{1.8.21}$$

Let us consider a change of basis defined by a P matrix

$$\mathscr{A}(\omega,p)x(\omega,p) = b(\omega,p) \iff \hat{\mathscr{A}}(\omega,p)\hat{x}(\omega,p) = \hat{b}(\omega,p). \tag{1.8.22}$$

where $\hat{\mathscr{A}}(\omega,p) = P^{-1}\mathscr{A}(\omega,p)P$, $\hat{x}(\omega,p) = P^{-1}x(\omega,p)$ and $\hat{b}(\omega,p) = P^{-1}b(\omega,p)$. As the determinant is a similarity invariant, we get

$$\hat{x}_i(\omega,p) = \frac{\det(\hat{\mathscr{A}}_i(\omega,p))}{\det(\mathscr{A}(\omega,p))}. \tag{1.8.23}$$

Writing both $x(\omega,p)$ and $\hat{x}(\omega,p)$ using the Cramer's rule, we observe that they share the same denominator function of (ω,p).
From this we can infer that if we manage to find a good β function working with a given basis, it will still be a good one when the basis changes.
Now we can look for a nice basis to work with. One way to proceed is to perform a singular value decomposition (SVD) on the **learning data** X_l. We then get (U,S,V) such that $X_l = USV^*$ where U is a $n_x \times n_x$ unitary matrix, V is a $n_l \times n_l$ unitary matrix and S is a $n_x \times n_l$ rectangular diagonal matrix with non-negative real numbers on the diagonal. The singular values of X_l are the terms $S(i,i)$ sorted from the greatest to smallest. That means that the first columns of U are the most appropriate to project X_l on a smaller space without losing too much information. We call U_{red} the matrix of the first n_{red} columns of U, and U_{rm} the matrix of the remaining columns

$$U = \begin{bmatrix} U_{\text{red}} & U_{\text{rm}} \end{bmatrix} \tag{1.8.24}$$

We consider the following orthogonal projection

$$\hat{x} = U_{\text{red}}^* x. \tag{1.8.25}$$

If we define $\hat{X}_l = [\hat{x}(\omega_1,p_1),\cdots,\hat{x}_l(\omega_{n_l},p_{n_l})]$, we get

$$\hat{X}_l = U_{\text{red}}^* X_l. \tag{1.8.26}$$

The lines of $\hat{X}_l$ are orthogonal, the Euclidian norm of its i^{th} line being $S(i,i)$.
We now want to build a model $\tilde{y}$

$$\tilde{y}(\omega,p,a,b) = \frac{\alpha(\omega,p,a)}{\beta(\omega,p,b)+1} \approx \hat{x}_{red}(\omega,p). \tag{1.8.27}$$

Let us define $\hat{e}$ the model error in the projection space as

$$\hat{e}(\omega,p,a,b) = \|\hat{x}(\omega,p) - \tilde{y}(\omega,p,a,b)\|_2^2 \tag{1.8.28}$$

One simple way to get a model in the original space is to define $y = U_{\text{red}}\tilde{y}$ (we will see later that is not the only option). In that case, the model error in this space is

$$e(\omega, p) = \|x(\omega, p) - y\|_2^2. \tag{1.8.29}$$

Thus, using the fact that U is a unitary matrix, we have

$$\begin{aligned} e(\omega, p, a, b) &= \|x(\omega, p) - U_{\text{red}}\tilde{y}(\omega, p, a, b)\|_2^2 \\ &= \|x(\omega, p) - U\begin{pmatrix} \tilde{y}(\omega, p, a, b) \\ 0_{\mathbb{C}^{n_x - n_{\text{red}}}} \end{pmatrix}\|_2^2 \\ &= \hat{e}(\omega, p, a, b) + \|U_{\text{rm}}^* x\|_2^2 \end{aligned} \tag{1.8.30}$$

As U_{rm}^* is the matrix of the remaining columns of U, we expect that for n_{red} large enough the norm of the projection $U_{\text{rm}}^* x$ is small.
For the purpose of primarily setting a good denominator β, this method allows to reduce the size of the problem by projection using a convenient space. It is more efficient and healthy than choosing n_{red} elements from x randomly.
Once β is set properly we will see in 1.8.6 how we recover y, the model of the entire field x.
Algorithm 1 sum up the preprocessing of the learning data X_l.

Algorithm 1 Preprocessing

1: **procedure** PREPROCESSING($X_l, n_{bf}^0, n_{\text{red}}$)
2: $(U, S, V) \leftarrow \text{svd}(X_l)$;
3: $U_{\text{red}} = U(:, 1 : n_{\text{red}})$;
4: $\hat{X}_l \leftarrow U_{\text{red}}^* X_l$;
5: return $\hat{X}_l$ and U_{red};
6: **end procedure**

The new learning data set is $\hat{X}_l$.

1.8.5 Identification Process

The general optimization problem is the following

$$\min_{a,b} \sum_{k=1}^{n_l} \|F_k(a, b)\|_2^2 \tag{1.8.31}$$

where the residual F_k is

$$\begin{aligned} F_k(a,b) &= \hat{y}(\omega_k, p_k, a, b) - \hat{x}_l(\omega_k, p_k) \\ &= \frac{\alpha(\omega, p, a)}{\beta(\omega, p, b) + 1} - \hat{x}_l(\omega_k, p_k) \end{aligned} \tag{1.8.32}$$

with

$$\alpha_i(\omega, p, a) = \sum_{j=1}^{n_{bf}} a_{i,j} g_j(\omega, p), \tag{1.8.33}$$

and

$$\beta(\omega, p, b) = \sum_{j=1}^{n_{bf}-1} b_j g_{j+1}(\omega, p). \tag{1.8.34}$$

We need to find an efficient way to address this nonlinear optimization problem. In the following paragraphs, we will study two different ways to solve it that have both strengths and weaknesses. Eventually we will define a hierarchical approach that aims at taking the best from both methods.

1.8.5.1 The linear least squares problem

We may attempt to solve a simple linear least squares problem by multiplying the equations by the denominator $\beta + 1$:

$$\begin{aligned} &\min_{a,b} \sum_{k=1}^{n_l} \| \left(\hat{y}(\omega_k, p_k, a, b) - \hat{x}_l(\omega_k, p_k)\right) \left(\beta(\omega_k, p_k, b) + 1\right) \|_2^2 \\ \iff &\min_{a,b} \sum_{j=1}^{n_l} \| \left(\alpha(\omega_k, p_k, a) - \hat{x}_l(\omega_k, p_k)\beta(\omega_k, p_k, b)\right) - \hat{x}_l(\omega_k, p_k) \|_2^2 \end{aligned} \tag{1.8.35}$$

which is indeed a linear least squares problem as α and β depends linearly on a and b. This system is easy to solve.
But it is by nature an ill-posed problem and its solution has no guarantee to provide a small residual for the original problem 1.8.31.

1.8.5.2 The natural nonlinear least squares problem

Alternatively, we can choose to directly solve the system under its natural form 1.8.31 using the Gauss-Newton method. However, there is a major drawback. The function is far from being convex. The Gauss-Newton method leads to local minima. The reason for these local minima is that when the zeros of the

denominator are misplaced, they may have no way to get to the right position without increasing the cost function. This happens especially in the case their path necessarily requires to get close to a learning point. This is particularly noticeable in the case the data are real (no damping) and there is only one variable (for example the frequency). Hence we need a way to tend towards the global minimum. We will not do so by changing the optimization method, we will still be using Gauss-Newton method. What we are changing is the way the problem is posed using a hierarchical method.

1.8.5.3 The hierarchical method

Contrary to the natural nonlinear system 1.8.5.2, the quadratic least squares system 1.8.5.1 is convex. While it is unstable and does not provide a reliable model, it has the benefit of properly positioning the zeros of the denominator in the first place. Thus we can start with a valuable first guess that locates a and b in a good region for the global optimization.
We look for a continuous way to go from 1.8.5.1 to 1.8.5.2.
To do so we introduce a new residual parameterized with $\gamma \in [0,1]$

$$F_k^\gamma(a,b) = \big(\alpha(\omega_k,p_k,a) - \hat{x}_l(\omega_k,p_k)\big(\beta(\omega_k,p_k,b)+1\big)\big)\,|\beta(\omega_k,p_k,b)+1|^{-\gamma} \tag{1.8.36}$$

where $|\cdot|$ is the element-wise modulus.
Let us consider two particular cases

- For $\gamma = 0$,

$$F_k^0(a,b) = \big(\alpha(\omega_k,p_k,a) - \hat{x}_l(\omega_k,p_k)\beta(\omega_k,p_k,b)\big) - \hat{x}_l(\omega_k,p_k). \tag{1.8.37}$$

  Hence, we have

$$\begin{gathered}\min_{a,b}\sum_{k=1}^{n_{bf}}\left\|F_k^0\right\|_2^2 \\ \iff \min_{a,b}\sum_{k=1}^{n_{bf}}\left\|\big(\alpha(\omega_k,p_k,a) - \hat{x}_l(\omega_k,p_k)\beta(\omega_k,p_k,b)\big) - \hat{x}_l(\omega_k,p_k)\right\|_2^2\end{gathered} \tag{1.8.38}$$

  which is equivalent to 1.8.5.1.

- For $\gamma = 1$, the minimization problem is

$$\begin{gathered}\min_{a,b}\sum_{k=1}^{n_{bf}}\left\|F_k^1\right\|_2^2 \\ \iff \min_{a,b}\sum_{k=1}^{n_{bf}}\left\|\big(\alpha(\omega_k,p_k,a) - \hat{x}_l(\omega_k,p_k)\big(\beta(\omega_k,p_k,b)+1\big)\big)\,|\beta(\omega_k,p_k,b)+1|^{-1}\right\|_2^2.\end{gathered} \tag{1.8.39}$$

What is interesting here is that

$$\begin{aligned}\|F_k^1(a,b)\|_2^2 &= \| \left(\alpha(\omega_k,p_k,a) - \hat{x}_l(\omega_k,p_k)\left(\beta(\omega_k,p_k,b)+1\right)\right)|\beta(\omega_k,p_k,b)+1|^{-1}\|_2^2 \\ &= \| \left(\alpha(\omega_k,p_k,a) - \hat{x}_l(\omega_k,p_k)\left(\beta(\omega_k,p_k,b)+1\right)\right)\left(\beta(\omega_k,p_k,b)+1\right)^{-1}\|_2^2 \\ &= \| \frac{\alpha(\omega_k,p_k,a)}{\beta(\omega_k,p_k,b} - \hat{x}_l(\omega_k,p_k) - \hat{X}_l\|_2^2 \\ &= \|F_k(a,b)\|_2^2. \end{aligned} \tag{1.8.40}$$

Therefore it is equivalent to 1.8.5.2.

From those observations, we are going to identify a and b starting with $\gamma = 0$ (particular case where the problem is linear) and iteratively going up to $\gamma = 1$, solving at each step the optimization problem with the Gauss-Newton method. This is a continuation technique in the spirit of what is done in [12]. That should allow us to progressively reach the global minimum of F.
Figure 1.9 is interesting to illustrate the interest of this hierarchical method. It represents how $\|F^\gamma(a,b)\|_2^2$ behaves when we move away from the optimal solution $(a_{\text{opt}}, b_{\text{opt}})$ in a given direction (da, db). This movement is monitored by a weight w so that $(a,b) = (a_{\text{opt}}, b_{\text{opt}}) + w * (da, db)$. The figure illustrates more precisely what happens in the case 1.9.4 for $\gamma = 0$, 0.5 or 1. We can see that for $\gamma = 1$ the problem is not convex and really stiff. Although, it might look like the minimum is the same for every value of γ, but they are not. And it matters, as the final validation relative error is 10 times greater when we only use $\gamma = 0$ (i.e. we solve a linear system), rising up from 0.0217 to 0.214.

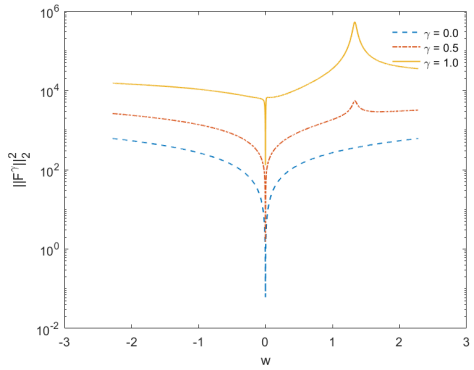


Figure 1.9 – Behavior of $\|F^\gamma(a,b)\|_2^2$ (logarithm scale)

1.8.5.4 The identification algorithm

Considering that the data have been preprocessed with algorithm 1, we can now introduce the full identification algorithm that aims at determining a and b. Let us define $\omega_{\text{set}} = (\omega_1, \cdots, \omega_{n_l})$ the vector of the learning frequencies, and $p_{\text{set}} = [p_1, \ldots, p_{n_l}]$ the table of the learning parameters. To get started we need to choose an initial number of basis functions n_{bf}^0.

Algorithm 2 zROM identification algorithm

procedure IDENTIFICATION($\omega_{\text{set}}, p_{\text{set}}, \hat{X}_l, n_{bf}^0$, tol)
  $n_{bf} = n_{bf}^0$;
  $\gamma^0 = 0$;
  $a^0 = \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}$; ▷ The length of a^0 is $n_{red} \times n_{bf}$
  $b^0 = (0, \ldots, 0)$; ▷ The length of b^0 is n_{bf}
  while $n_{bf} \leq n_l \frac{n_{\text{red}}}{n_{\text{red}}+1}$ **do**
    Select the n_{bf} first basis functions according to the method introduced in 1.8.3.3 that involves ω_{set} and p_{set}.
    for $\gamma = \gamma^0; \gamma \leq 1; \gamma = \gamma + 0.1$ **do**
      $(a, b, \text{err}) \leftarrow \text{optimize}(F^\gamma(\cdot, \cdot, \hat{X}_l), a^0, b^0)$;
      if textrmerr<tol **then**
        $a^0 \leftarrow a$;
        $b^0 \leftarrow b$;
      else
        $\gamma^0 = \gamma$;
        break;
      end if
    end for
    if err $< tol$ and $\gamma^0 = 1$ **then**
      break;
    else
      $n_{bf} \leftarrow n_{bf} + 1$;
      $a^0 \leftarrow \left[a^0, \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \right]$;
      $b^0 \leftarrow \left[b^0, 0 \right]$;
    end if
  end while
  return a, b;
end procedure

What we call *err* is the following relative error:

$$\text{err} = \frac{\sum_{k=1}^{n_{bf}} \|F_k^\gamma(a,b)\|_2^2}{\sum_{k=1}^{n_{bf}} \|\hat{x}_l(\omega_k, p_k)\|_2^2} \tag{1.8.41}$$

1.8.6 Optimization: the Gauss-Newton method

Here we are focusing on the *optimize* function called in algorithm 2. As already mentioned, we are using the Gauss-Newton method at each step of the hierarchical method and proceed accordingly to Appendix A.
First, let us define the full matrix of residuals F^γ as

$$F^\gamma = (F_{i,k}^\gamma)_{\substack{1\le i\le n_{\text{red}} \\ 1\le k\le n_l}} \tag{1.8.42}$$

where $F_{i,k}^\gamma$ is the i^{th} element of what was denoted F_k^γ earlier (i.e. the part of the residual corresponding to the i^{th} element of $\hat{x}_l$ and the k^{th} learning point).
Furthermore,

$$F_{i,k}^\gamma(a,b) = F_{i,k}^\gamma(a_{i,\cdot},b) \tag{1.8.43}$$

where $a_{i,\cdot}$ is the $i^{\textbf{th}}$ line of a.
Now, we need to calculate the forward and backward mode differentials for F^γ. To do so, let us define the following matrices:

$$A = \begin{pmatrix} g_1(\omega_1,p_1) & \dots & g_{n_{\text{bf}}}(\omega_1,p_1) \\ \vdots & \ddots & \vdots \\ g_1(\omega_{n_l},p_{n_l}) & \dots & g_{n_{\text{bf}}}(\omega_{n_l},p_{n_l}) \end{pmatrix}, \tag{1.8.44}$$

and

$$B = \begin{pmatrix} g_2(\omega_1,p_1) & \dots & g_{n_{\text{bf}}}(\omega_1,p_1) \\ \vdots & \ddots & \vdots \\ g_2(\omega_{n_l},p_{n_l}) & \dots & g_{n_{\text{bf}}}(\omega_{n_l},p_{n_l}) \end{pmatrix}, \tag{1.8.45}$$

For convenience, let us also denote by a_i the vector $a_i = a_{i,\cdot}^T$.

1.8.6.1 Forward mode

The derivative of $F_{i,\cdot}^\gamma$ in direction (da_i, db) is given by

$$\begin{aligned} dF_{i,\cdot}^\gamma(da_i,db,a_i,b)^T = &(Ada_i - X_l\cdot Bdb)\cdot|Bb+1|^{-\gamma} \\ &-\frac{\gamma}{2}\left(B\overline{db}\cdot(Bb+1)+(B\overline{b}+1)\cdot Bdb\right)\cdot(Aa_i - X_l\cdot Bb)\cdot|Bb+1|^{-\gamma-2}. \end{aligned} \tag{1.8.46}$$

where $M \cdot N$ refers to the element-wise multiplication of M and N, and $\overline{x}$ is the element-wise complex conjugate of x.

1.8.6.2 Backward mode

Let py be a vector of the same size than y

$$pa_i = \sum_{i=1}^{n_{red}} A^T \left(py_{i,\cdot}^T \cdot |Bb+1|^{-\gamma} \right). \tag{1.8.47}$$

$$\begin{aligned} pb = &- \sum_{i=1}^{n_{\text{red}}} B^T \left(py_{i,\cdot}^T \cdot \overline{X_l} \cdot |Bb+1|^{-\gamma} \right) \\ &- \sum_{i=1}^{n_{\text{red}}} B^T \frac{\gamma}{2} B^T \left(\left(py_{i,\cdot}^T + \overline{py_{i,\cdot}^T} \right) \cdot (B\overline{b}+1) \cdot \left(A\overline{a}_i - \overline{X_l} \cdot B\overline{b} \right) \cdot |Bb+1|^{-\gamma-2} \right). \end{aligned} \tag{1.8.48}$$

1.8.7 Postprocessing: Identification of new coefficients a

At this stage of the process, we have identified vector b that defines the denominator β. The model is almost completely built. For now, we only have a model $\hat{y}$ of the reduced vector $\hat{x}$. We have to build a model y of x.
We have two main alternatives:

Method 1 If we took a large enough number n_{red} in a first place, we can perform an almost inverse transform to get back to the natural space:

$$y(\omega, p) = U_{\text{red}} \hat{y}(\omega, p) \tag{1.8.49}$$

which is equivalent to get a new and larger a for the model y in the form

$$a \leftarrow U_{\text{red}} a \tag{1.8.50}$$

Method 2 A good alternative is to only keep β and look for new α_i suited for the x_i in the natural space. As β is fixed, this gets quite easy to achieve. The problem gets separated and linear. We look for

$$\min_a \sum_{k=1}^{n_{bf}} \sum_{i=1}^{n_x} \| G_{i,k}(\omega_k, p_k, a_{i,.}, b) \|_2^2 \tag{1.8.51}$$

where

$$G_{i,k}(\omega_k, p_k, a) = \frac{\alpha_i(\omega_k, p_k, a)}{\beta(\omega_k, p_k, b) + 1} - x_i(\omega_k, p_k) \tag{1.8.52}$$

The only unknown here is vector a and we can solve independently every small linear least squares problem of the type

$$\min_a \sum_{k=1}^{n_{bf}} \|G_{i,k}(\omega_k, p_k, a_{i,.})\|_2^2, \quad \forall i \in [1, \ldots, n_x] \tag{1.8.53}$$

Note: We can also mix those two methods and use a small n_{red} while looking for a good denominator, which is the complicated and non-linear part, and then find good numerators for a large n_{red} when the reduced basis from the SVD catches most the energy. That is particularly relevant if the meshing of the system is too refined compared to the highest frequency we work with.

Algorithm 3 zROM postprocessing

procedure POSTPROCESSING($\omega_{\text{set}}, p_{\text{set}}, X_l, n_{bf}, a, b, U_{\text{red}}$,method)
  if method=1 **then**
    $a = U_{\text{red}} a$;
  else
    for $i = 1; i \leq n_x; i = i + 1$ **do**
      $a_{i,.} \leftarrow \text{argmin} \|G_{i,k}(\omega_k, p_k, a_{i,.})\|_2^2$;
    end for
  end if
  return a and b;
end procedure

1.8.8 Evaluate the model for a new configuration

Now we can define the algorithm that evaluates the model for a new $(\omega_{\text{new}}, p_{\text{new}})$ configuration.

Algorithm 4 zROM evaluation

procedure EVALUATE($\omega_{\text{new}}, p_{\text{new}}, a, b$)
  $\alpha_{\text{new}} \leftarrow \sum_{k=1}^{n_{bf}} a(:,k)^T g_k(\omega_{\text{new}}, p_{\text{new}})$
  $\beta_{\text{new}} \leftarrow \sum_{k=1}^{n_{bf}-1} b_k g_{k+1}(\omega_{\text{new}}, p_{\text{new}})$
  $y_{\text{new}} \leftarrow \alpha_{\text{new}}/(\beta_{\text{new}}+1)$
  return y_{new}
end procedure

1.8.9 Recovering the eigenelements

It can still be very useful to know how the singularities of $\mathscr{A}$ and the corresponding modes behave with respect to p.

Depending on the learning data we use, we expect to build a reliable model for $\omega \in [\omega_{min}, \omega_{max}]$ and $p \in \mathscr{D}_p$. Assuming that $L(p)$ is not orthogonal to one or several eigenvectors for every p (which is unlikely given the fact that their behavior with respect to p is as stiff as the eigenvalues'), and that the model is well built (i.e. $y(\omega,p)$ is really close to $x(\omega,p)$), what can we expect from α and β?
In the general case, α_i and β will be good approximations (up to the multiplication by a constant to $\det(\mathscr{A}_i(\omega,p))$ and $\det(\mathscr{A}(\omega,p))$ on $[\omega_{min},\omega_{max}] \times \mathscr{D}_p$. β will even share the roots of $\det(\mathscr{A}(\lambda,p))$ that are close enough to $[\omega_{min},\omega_{max}]$ (for any $p \in \mathscr{D}_p$).
However, there are particular cases to consider. They may occur when the mechanical system we consider has symmetry properties that are preserved when p changes. In that case some eigenvalues of $\mathscr{A}$ have a multiplicity greater than 1 in a consistent manner when p moves all over $\mathscr{D}_p$. In this instance we can extend the notion of *minimal polynomial* to multivariate eigenvalue problems. We call it $\Pi(\omega,p)$. We can show that the induced simplification can be extended to $detA_i(\omega,p)$, so we define $\Pi_i(\omega,p)$.
In that case,

$$x_i(\omega,p) = \frac{\Pi_i(\omega,p)}{\Pi(\omega,p)}, \tag{1.8.54}$$

and consequently, α_i and β are more likely to behave respectively like Π_i and Π.

In both cases, if α and β are regular enough (and they must be by construction) we have the following properties

- The zeros of β are **eigenvalues** of $\mathscr{A}$
- The corresponding **eigenvectors** are $\alpha(\lambda_i)$ (and its derivatives with respect to λ at λ_i if the multiplicity of the eigenvalue is greater than 1 on a finite number of points).

As α and β are much easier to model than eigenvalues and eigenvectors themselves. In a way this is an implicit way to address the eigenvalue problem for a given range of frequencies.

1.9 Numerical experiments

It is now time to test the method using experimental test cases. In this section we are going to consider two different geometries:

- The pipe full of fluid, already introduced in 1.2 is the first case. The only parameter is the length. There will be two different experiments:
  1 The fluid is viscous (damping),
  2 The fluid is not viscous (no damping).
- The second case is a rectangular clamped vibrating plate. This time there are two parameters: the two lengths of the rectangle. The damping is really low. What make this case particularly interesting is that when the plate becomes a square, the multiplicity of the poles doubles. We will see how the ROMs deal with that.

1.9.1 The industrial case

This industrial test case is provided by Ansys and General Electric India.
We consider a pipe full of fluid of length p_L. It is clamped at one end and excited with a longitudinal force at frequency f at the other end.
Figure 1.10 shows the geometry of the pipe.
Here we will more naturally refer to f instead of ω, which does not change much as $\omega = 2\pi f$.
Our goal is to predict the harmonic longitudinal displacement $x^{\text{long}}(f, p_L)$ of

the pipe for any (f, p_L) pair within a specified range.
The learning data are computations of $x^{\text{long}}(f, p_L)$ made with Mechanical, a solver from ANSYS.
Let us remind that the method is fully **non-intrusive**. We do not need further information from the solver. Thus we have no information about Lamé's coefficient, about the thickness of the tube, about its diameter...

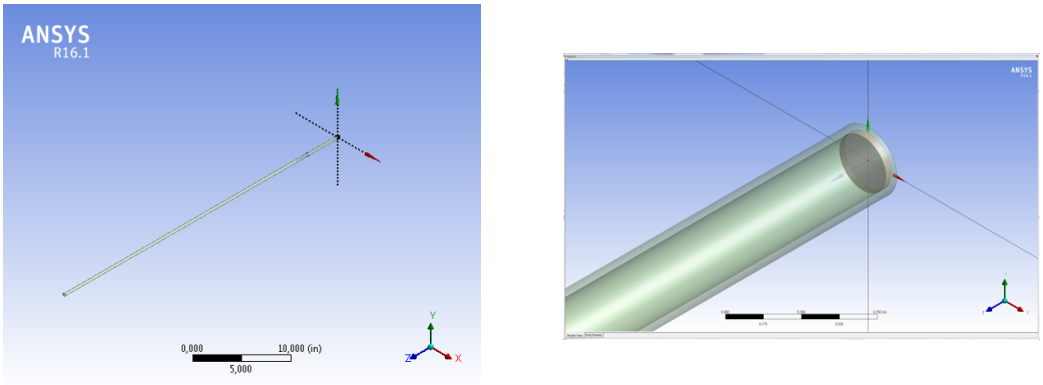


Figure 1.10 – Geometry

The pipe has been meshed using 13564 nodes. ANSYS Mechanical, the solver that is used to generate the learning data, solves systems in the form

$$\mathscr{A}(f, p_L)x(f, p_L) = L(p_L) \tag{1.9.1}$$

where (f, p_L) is set. The size of this square system is greater than 13564×3 (3 comes from the 3 components of the displacement) as it takes into account the fluid as well.
From this computation, we got only access to the elements of x that are related to the **longitudinal displacement**. As already mentioned, we call this 13564-long vector x^{long}.

Two different cases are considered depending on the damping coefficient for the material. In a first experiment it is settled to a non-zero value 1.9.2. Alternatively, it is equal to zero in a second experiment 1.9.3.

1.9.2 Case with damping

In this section the damping coefficient is set to a non-zero value.
We have plenty of computed data through a 25×37 grid in $[f_{\min}, f_{\max}] \times [p_{L\min}, p_{L\max}]$,where

- $f_{\text{min}} = 838\text{Hz}$
- $f_{\text{max}} = 1000\text{Hz}$
- $p_{L\text{min}} = 1\text{m}$
- $p_{L\text{max}} = 1.12\text{m}$

The learning data set is made of a 4×4 sub grid as shown in Figure 1.11.

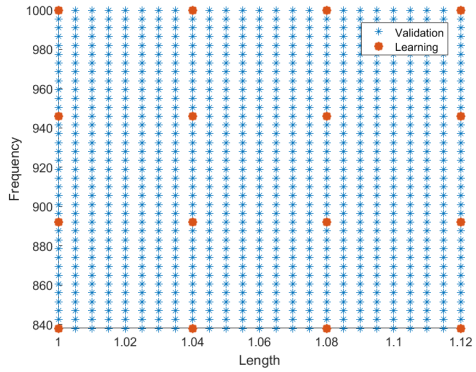


Figure 1.11 – Design of experiment

The method has been developed using the Matlab software. The learning phase lasts less than one minute on a powerful laptop.

1.9.2.1 Preprocessing

Following section 1.8.4 we first preprocess the data using the SVD. 7 modes from the SVD are used. The main goal at this stage is to identify a good denominator.

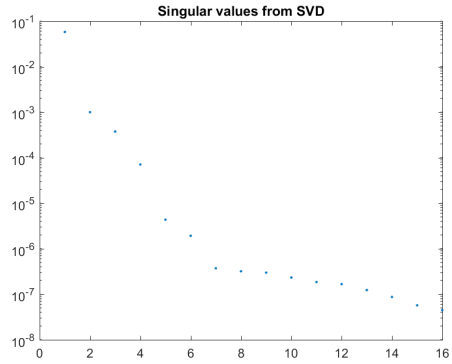


Figure 1.12 – Spectrum of the SVD

We choose to keep the $n_{\text{red}} = 7$ first modes. The new learning data are

$$\hat{X}_l = U^*_{\text{red}} X^{\text{long}}_l \tag{1.9.2}$$

1.9.2.2 Results for $\hat{x}$

The set of functions that have been selected (according to the method from 1.8.3.3) is depicted in Figure 1.13.

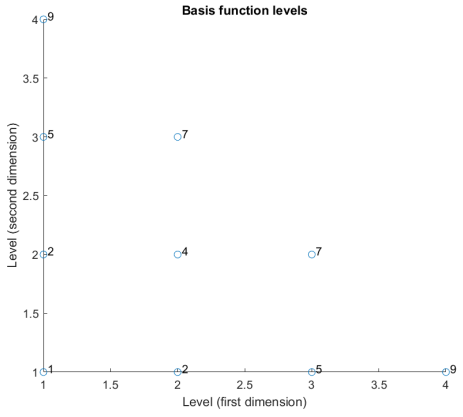


Figure 1.13 – Levels of selected functions

Now, let us have a look at the 7 elements of vector $\hat{x}(f, p_L)$.
In that case we do not really need to use as many SVD modes to identify an

efficient denominator function. The magnitude of the first modes actually decrease really quickly, and the influence of the last ones is negligible. But they bring some interesting patterns to look at.
Before going into the results, let us mention that the learning data (on the left of Figures 1.14 to 1.20) are **displayed using a linear interpolation in order to make them easier to observe**. At the center are the validation data (called *reference*), and on the right is the evaluation of the model. The upper part corresponds to the real part of the coefficients. Receptively, the lower part corresponds to the imaginary part.

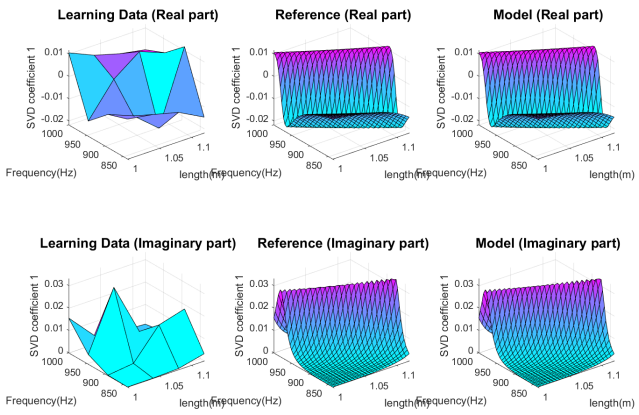


Figure 1.14 – $\hat{x}_1$ - SVD mode coefficient 1

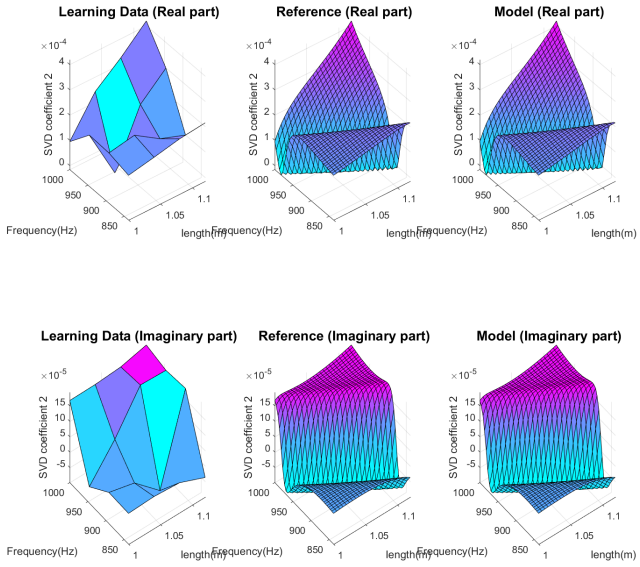


Figure 1.15 – $\hat{x}_2$ - SVD mode coefficient 2

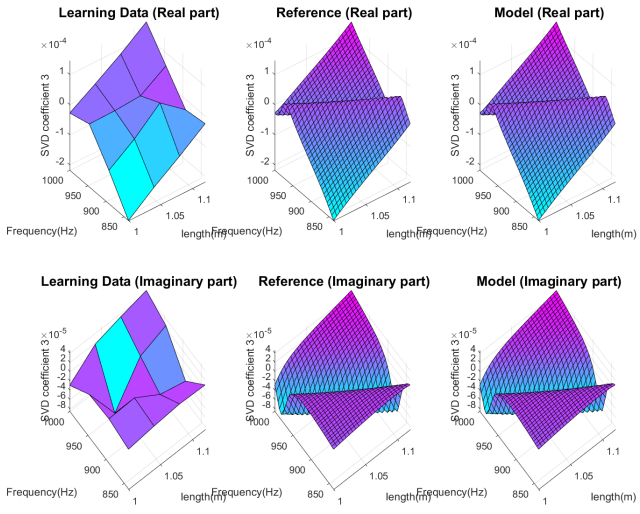


Figure 1.16 – $\hat{x}_3$ - SVD mode coefficient 3

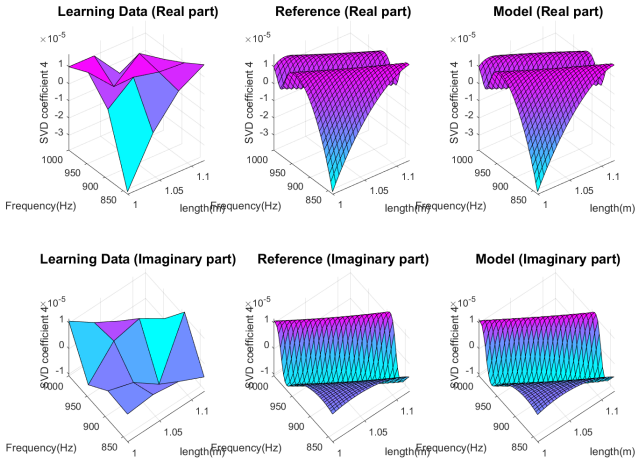


Figure 1.17 – $\hat{x}_4$ - SVD mode coefficient 4

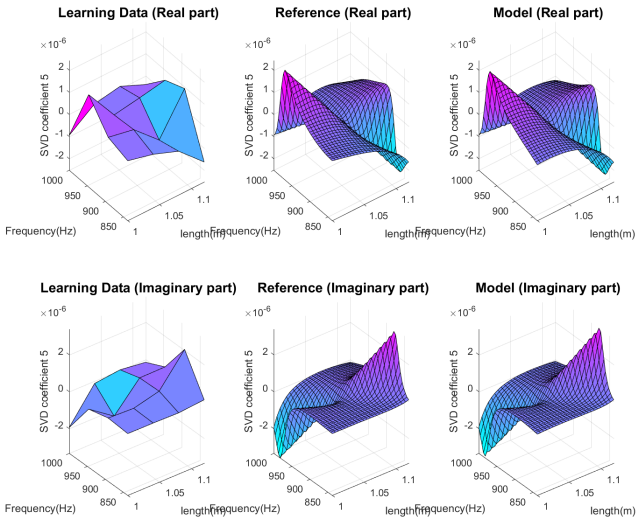


Figure 1.18 – $\hat{x}_5$ - SVD mode coefficient 5

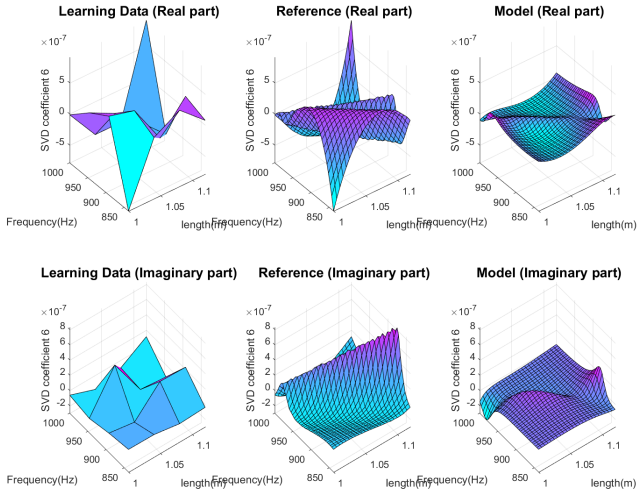


Figure 1.19 – $\hat{x}_6$ - SVD mode coefficient 6

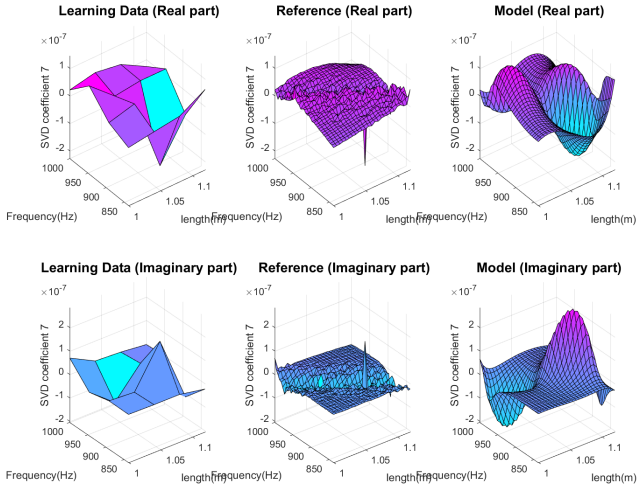


Figure 1.20 – $\hat{x}_7$ - SVD mode coefficient 7

1.9.2.3 Back to the natural space: results for x^{long}

Now that we have identified the denominator function, we can build a model $\hat{x}$ for every element of x^{long}. Figure 1.21 depicts the following relative error measure

$$\varepsilon_i = \frac{\sqrt{\sum_{j=1}^{n_v} |\hat{x}_i(f_j, p_{Lj}) - x_i^{\text{long}}(f_j, p_{Lj})|^2}}{\sqrt{\sum_{j=1}^{n_v} |x_i^{\text{long}}(f_j, p_{Lj})|^2}} \tag{1.9.3}$$

where the set of (f_j, p_{Lj}) is the validation set.

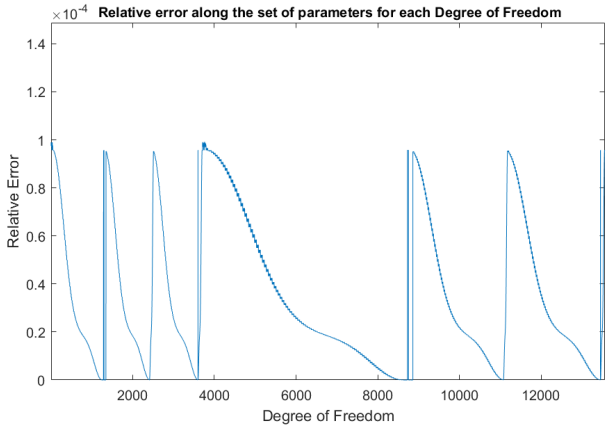


Figure 1.21 – Plot of ε, the vector of the relative errors for the elements in x^{long}

From Figure 1.21 we can locate the element x_i^{long} of x^{long} for which the error is the largest. Here it occurs at $i = 3760$. Figure 1.22 shows how the model behaves for this element.

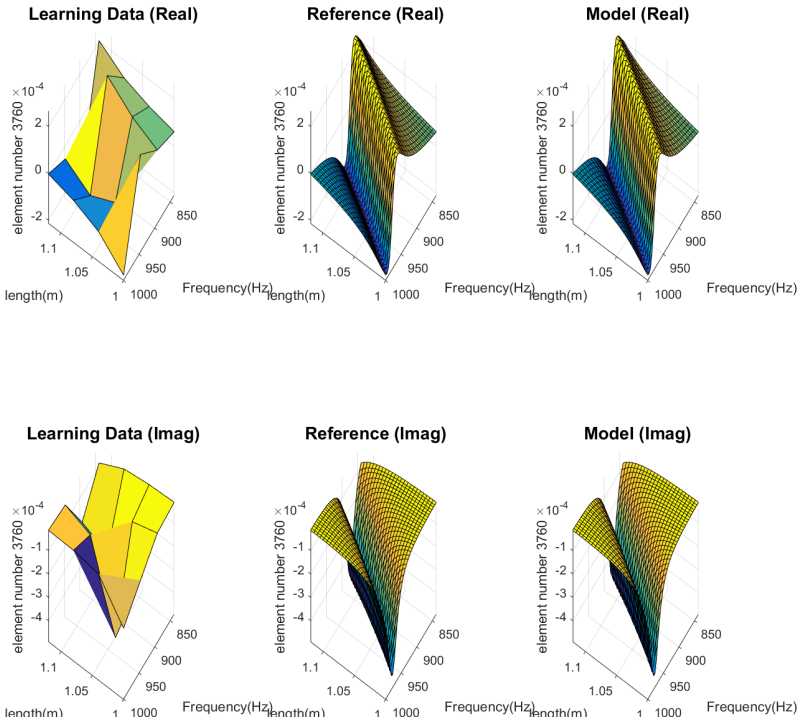


Figure 1.22 – Model of the element of x^{long} for which the validation error is the largest.

These results are satisfactory. The method makes it possible to fit well x^{long} using a few learning data.
It might be interesting to compare the model to a state-of-the-art function approximation technique. As the learning points form a regular grid we choose multivariate cubic spline interpolation rather than Kriging (that is very popular otherwise). The results may be observed in Figure 1.23 for the same element of x^{long} as before.

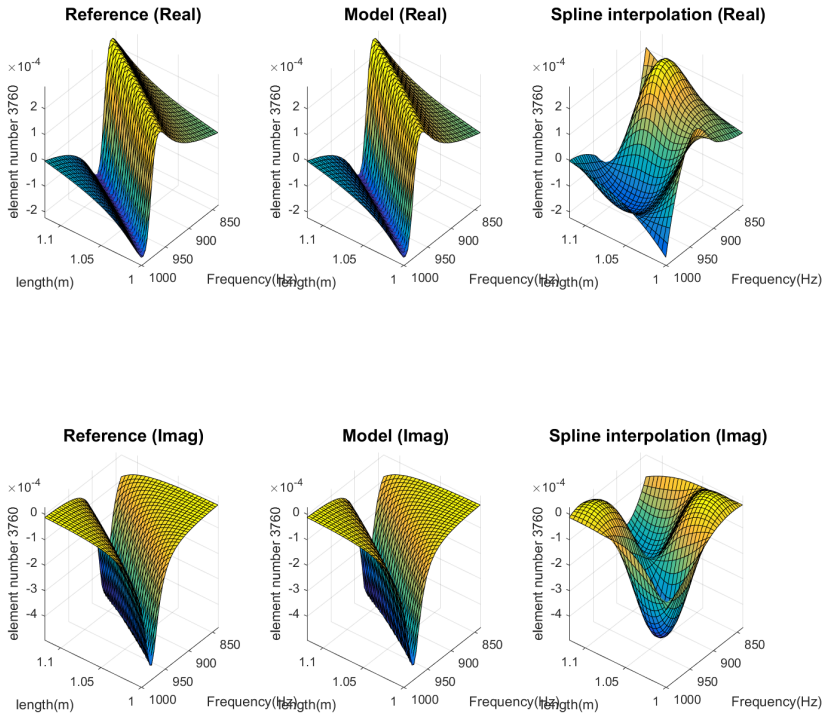


Figure 1.23 – Comparison between the model and multivariate spline interpolation

The multivariate spline interpolation does not perform as well as the model even though it uses the same learning data. The denominator plays a major role in the behavior of the function. And this will be even more noticeable with the next case, where there is no more damping.

1.9.3 Case with no damping

Here the damping coefficient is set to zero. Thus the imaginary part of the field is zero as there is no more phase change coming from the damping effect.
The domain is now $[f_{\min}, f_{\max}] \times [p_{L\min}, p_{L\max}]$,where

- $f_{\min} = 103$Hz
- $f_{\max} = 967$Hz
- $p_{L\min} = 1$m

- $p_{L_{\max}} = 1.12\text{m}$

Learning and validation points that are used are displayed in figure 1.24.
Overall we have

- available data: a 289×8 points grid,
- learning data: only a 13×8 sub grid.

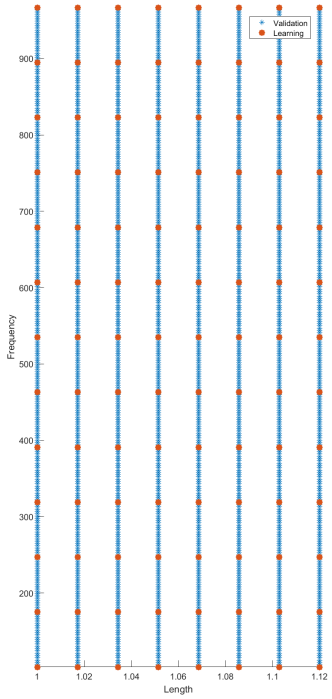


Figure 1.24 – Design of experiment

1.9.3.1 Preprocessing

Following section 1.8.4 we first use a preprocessing of the data using the SVD.
The singular values from the SVD are shown in Figure 1.25. They decrease quite quickly.

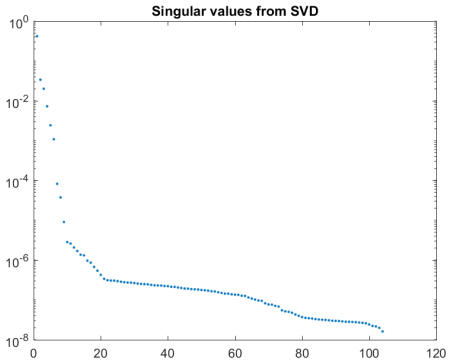


Figure 1.25 – Spectrum of the SVD

We choose to keep the $n_{\text{red}} = 8$ first modes. The ROM building would still have been efficient keeping a smaller number of modes but those additional modes provide interesting graphics.
The learning data are now

$$\hat{X}_l = U_{\text{red}}^* X_l^{\text{long}} \tag{1.9.4}$$

1.9.3.2 Results for $\hat{x}$

The set of functions that have been selected (according to the method from 1.8.3.3) is depicted in Figure 1.26.

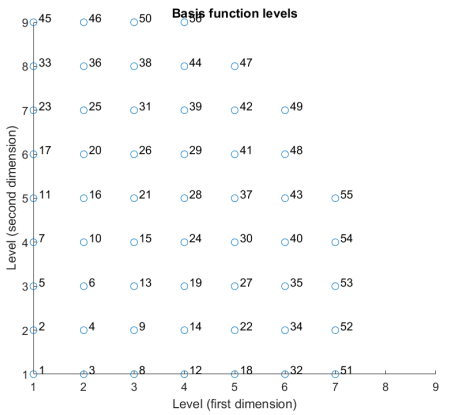


Figure 1.26 – Levels of selected functions

As for the damping case, the learning data are on the left, the validation data at the center (called *reference*), and the evaluation of the model on the right. Unlike the damping case, the imaginary part is zero here. Thus only the real parts are shown.
The main difference with previous case is that the function to be fitted is much sharper. The poles are real and the denominator actually is equal to zero for some (f, p_L) pairs in the domain. The theoretical magnitude of the function at these points may be infinite. However the ROM deals well with that as we can see in Figures 1.27 to 1.34.

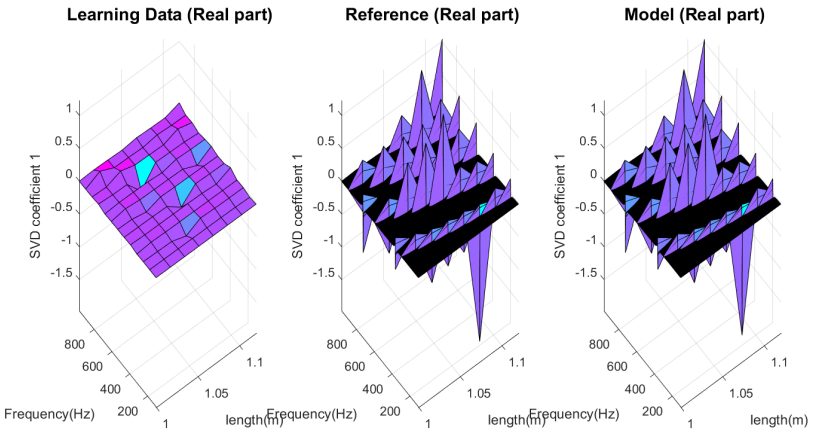


Figure 1.27 – $\hat{x}_1$ - SVD mode coefficient 1

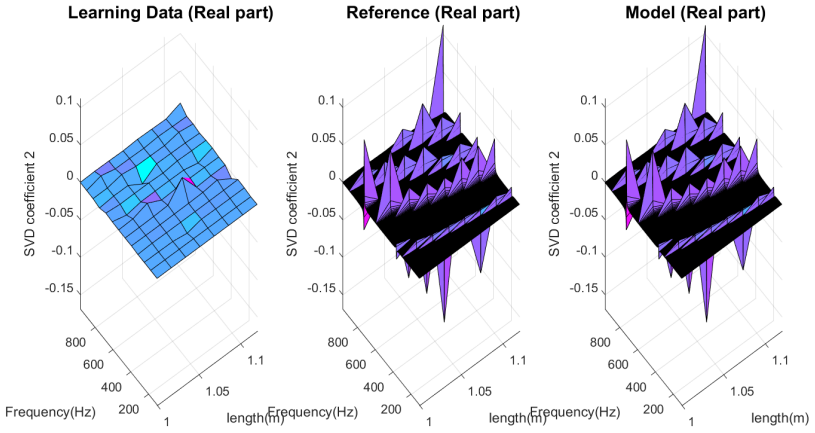


Figure 1.28 – $\hat{x}_2$ - SVD mode coefficient 2

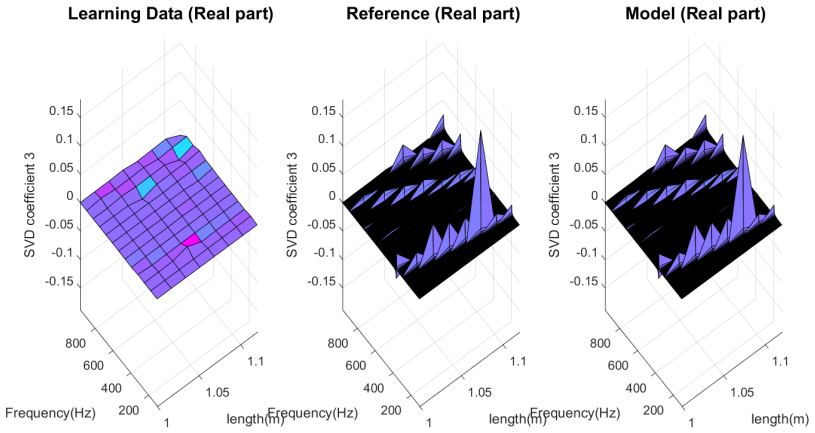


Figure 1.29 – $\hat{x}_3$ - SVD mode coefficient 3

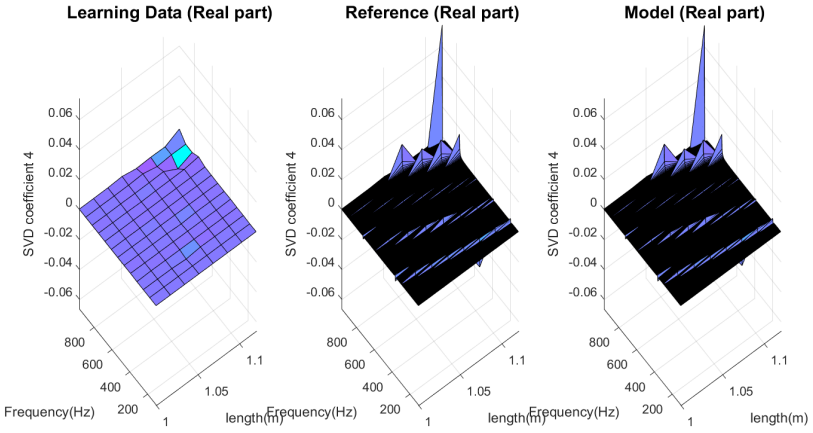


Figure 1.30 – $\hat{x}_4$ - SVD mode coefficient 4

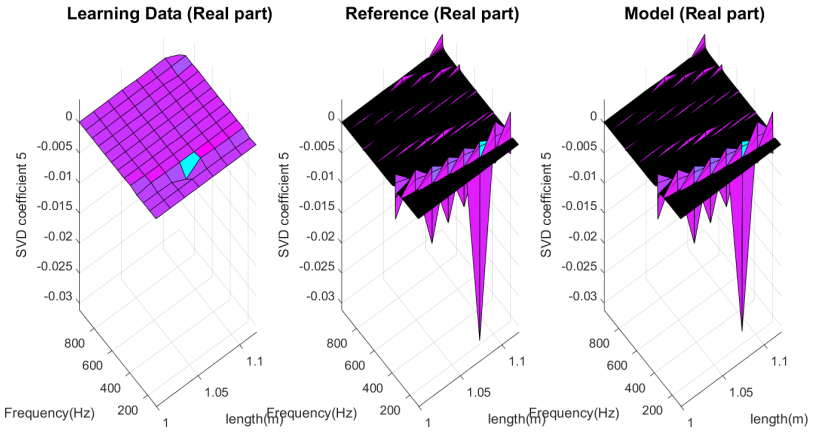


Figure 1.31 – $\hat{x}_5$ - SVD mode coefficient 5

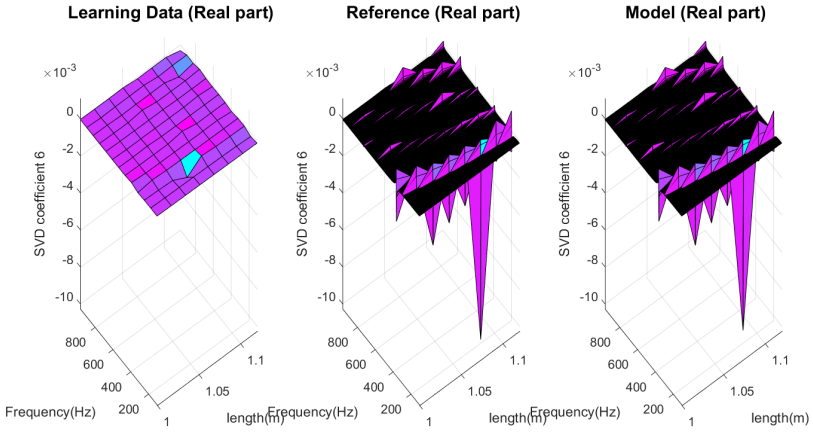


Figure 1.32 – $\hat{x}_6$ - SVD mode coefficient 6

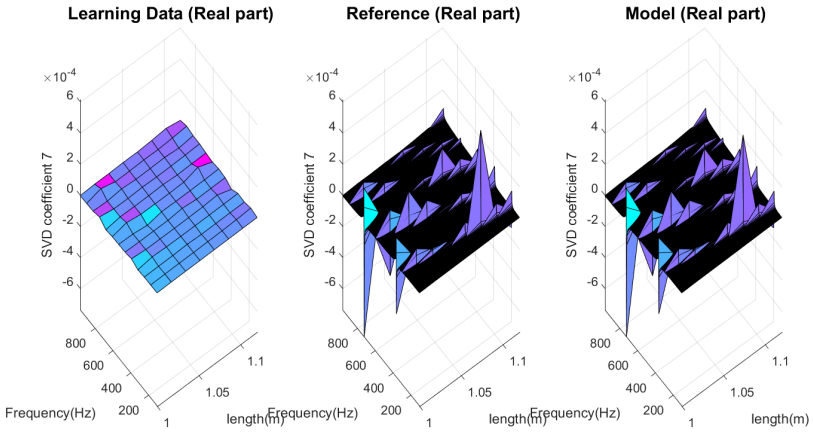


Figure 1.33 – $\hat{x}_7$ - SVD mode coefficient 7

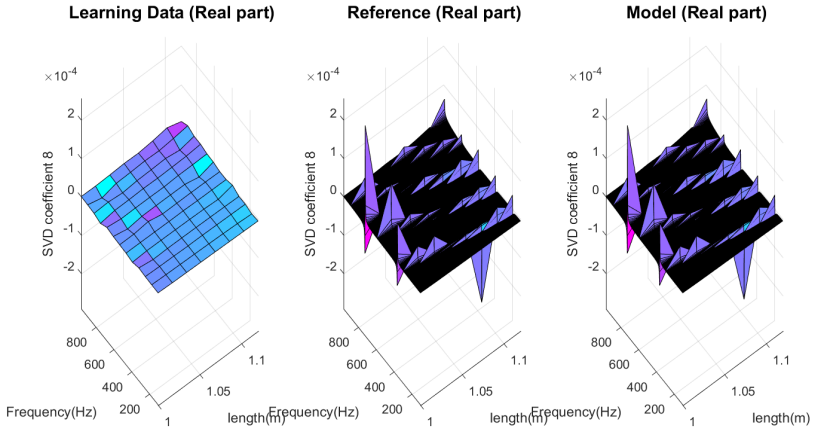


Figure 1.34 – $\hat{x}_8$ - SVD mode coefficient 8

1.9.3.3 Back to the natural space: results for x^{long}

As in the damping case, now that we have identified the denominator function, we can build a model $\hat{x}$ for every element of x^{long}. Figure 1.35 depicts the following relative error measure

$$\varepsilon_i = \frac{\sqrt{\sum_{j=1}^{n_v} |\hat{x}_i(f_j, p_{Lj}) - x_i^{\text{long}}(f_j, p_{Lj})|^2}}{\sqrt{\sum_{j=1}^{n_v} |x_i^{\text{long}}(f_j, p_{Lj})|^2}} \tag{1.9.5}$$

where (f_j, p_{Lj}) form the set of validation points.

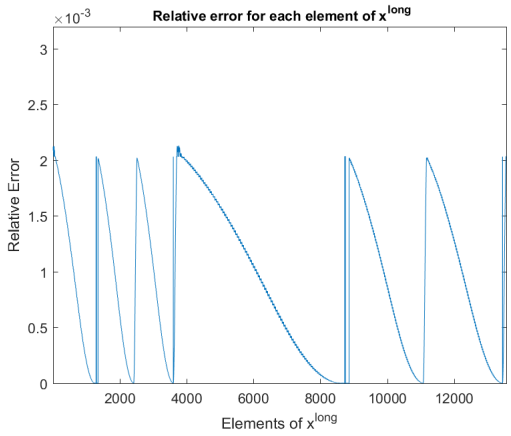


Figure 1.35 – Plot of ε, the vector of the relative errors for the elements in x^{long}

From Figure 1.35 we can locate the element x_i^{long} of x^{long} for which the error is the largest. Here it occurs at $i = 3760$. Figure 1.36 shows how the model behaves for this element.

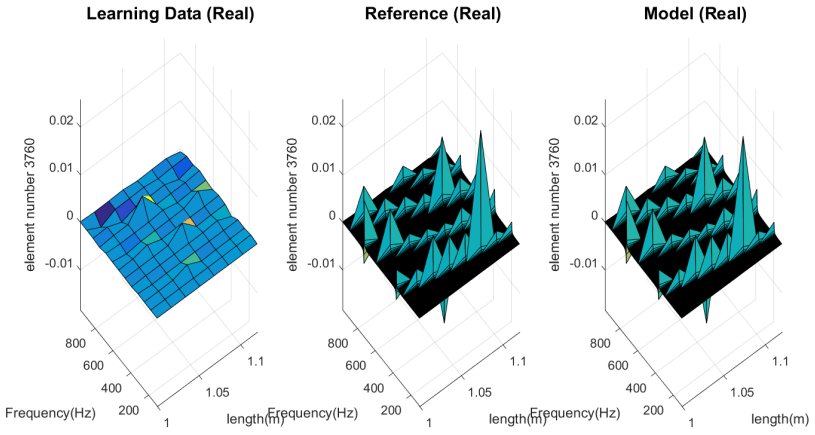


Figure 1.36 – Model of the element of x^{long} for which the validation error is the largest.

From these results we can deduce that the method has been very efficient at locating the actual poles of the system without explicitly looking for them. This is a valuable outcome that Ansys consider very promising.

1.9.3.4 The denominator

Let us have a look at the denominator on a refined grid in the domain.

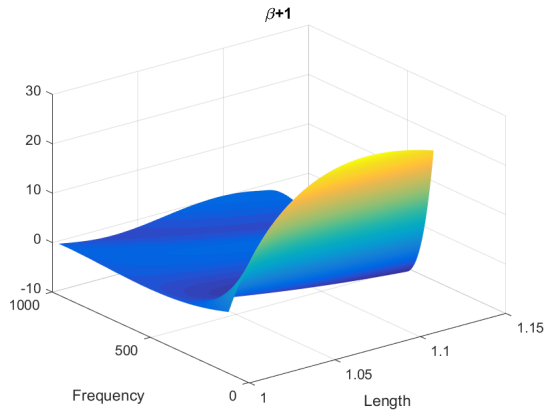


Figure 1.37 – The denominator

From figure 1.37, we observe that the denominator is quite smooth. It remains a simple function. It could be interesting to locate the poles. An efficient way to visualize them is to plot the logarithm of the modulus of the denominator $\log_{10}(|\beta+1|)$.

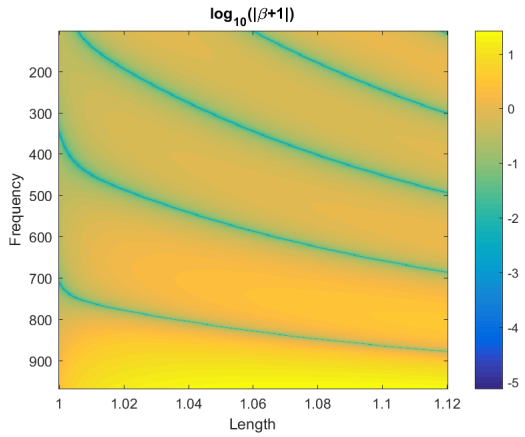


Figure 1.38 – The logarithm of the modulus of the denominator

1.9.4 The vibrating plate

The previous test case was interesting because it came from an industrial question. But there was only one parameter aside from the frequency. We would like to test the method into a problem for which there are at least two parameters. Because we did not have the chance to get this king of data from our partners we decided to study a simple but informative example. It comes from the file exchange platform of MathWorks and is called the *Vibration of rectangular clamped thin plate*. The design is a simple rectangular plate defined by both lenghts p_{L_1} and p_{L_2}.
We are going to build a ROM where (f, p_{L_1}, p_{L_2}) live in a given domain.
The specific feature here is that when $p_{L_1} = p_{L_2}$ eigenvalues meet because of the symmetry (the harmonic point force whose coordinates are (x, y) is chosen so that $x = y$). We want to find out how the ROM can deal with that.
Even if this is a simple test case, the computation of both learning and validation data was quite intensive. Producing data for a Cartesian grid of $97 \times 45 \times 45$ points for respectively f, p_{L_1} and p_{L_2} can take almost an entire day on a powerful laptop. We chose to set the damping coefficient to 0. We found out that interesting phenomena occurred for f varying between 100 and 100.97 and p_{L_i} between 1.14 and 1.18. It may seem to be a small domain but the lack of damping makes the behavior really stiff.
As learning data we took a $4 \times 5 \times 5$ sub-grid.
The basis functions we chose are depicted in Figure 1.39.

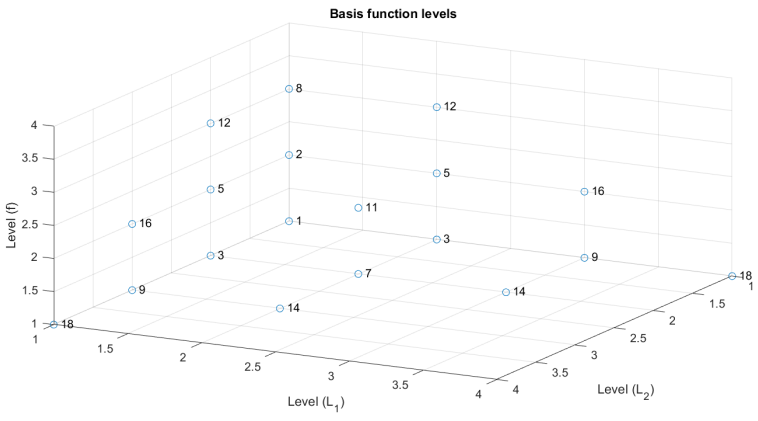


Figure 1.39 – Basis functions levels

1.9.4.1 Preprocessing

Following section 1.8.4 we first preprocess the data using a SVD. This time there are only two significant SVD modes as it can be seen in figure 1.40. Their behavior is still really interesting to observe.

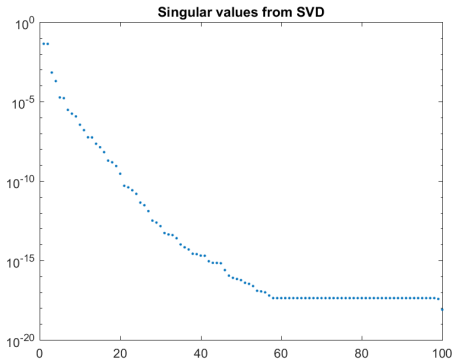


Figure 1.40 – SVD spectrum

1.9.4.2 Results over frequencies included in the learning set

Figures 1.41 to 1.44 display the performance of the model for the 4 frequencies that were included in the learning set. Although we are looking at frequencies that are among the Cartesian learning grid, we are displaying some validation data with respect to p_{L_1} and p_{L_2}. Thee learning data are on the left (and are linearly interpolated), the reference is at the center and the ROM is on the right. The upper part corresponds to first coefficient from the SVD, whereas the lower part corresponds to the second one.

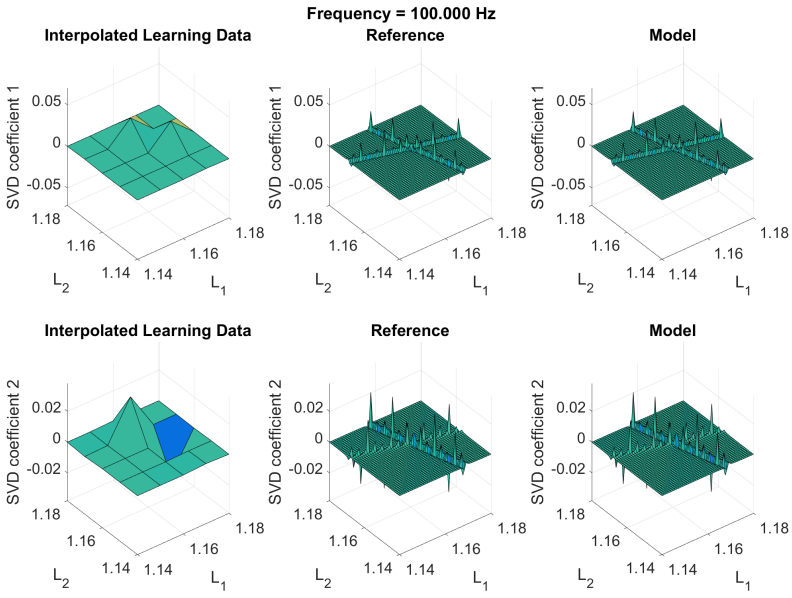


Figure 1.41 – f=100 Hz

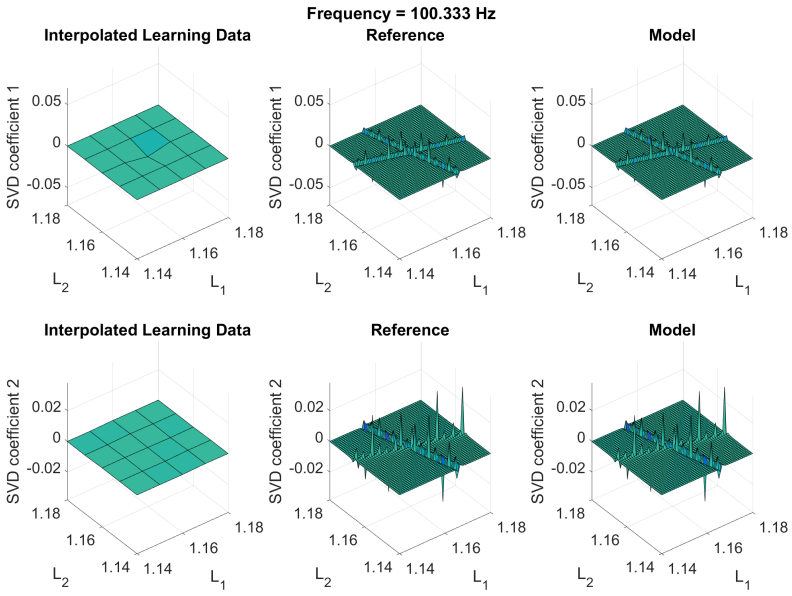


Figure 1.42 – f=100.33 Hz

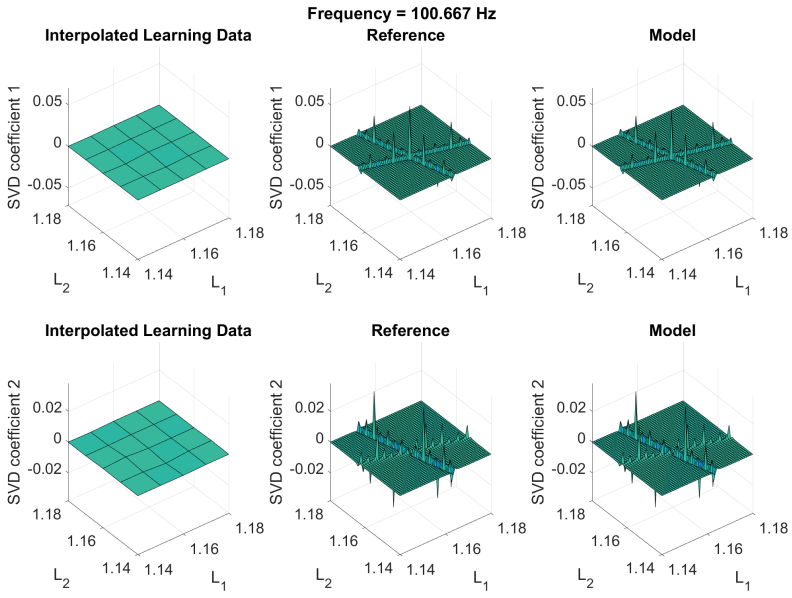


Figure 1.43 – f=100.65 Hz

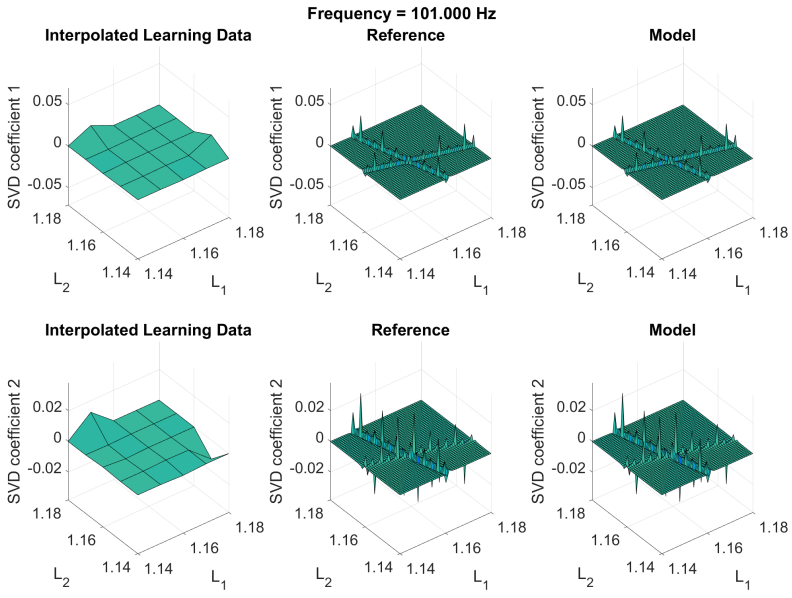


Figure 1.44 – f=100.97 Hz

1.9.4.3 Results over validation data

Let us now have a look at the results for some frequencies that are not in the learning set (Figures 1.45 to 1.47). The reference is on the left and the ROM is on the right.

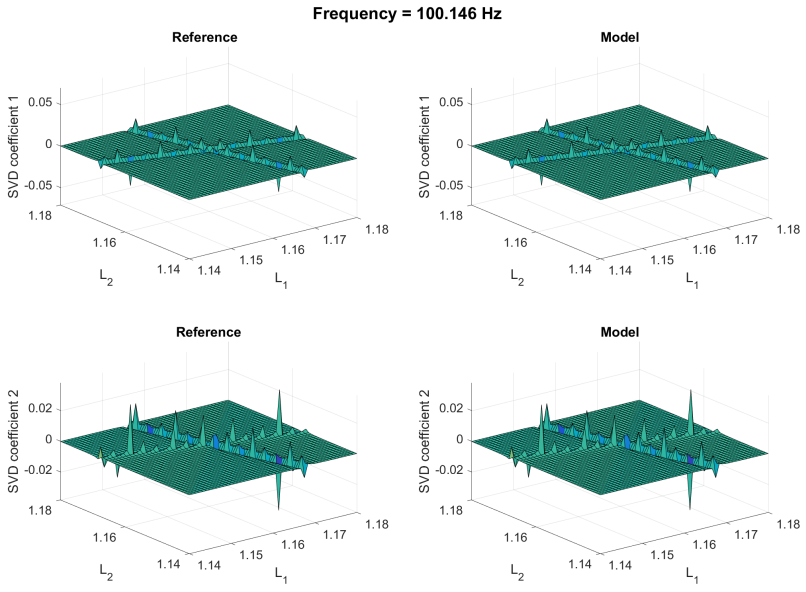


Figure 1.45 – f=100.15 Hz

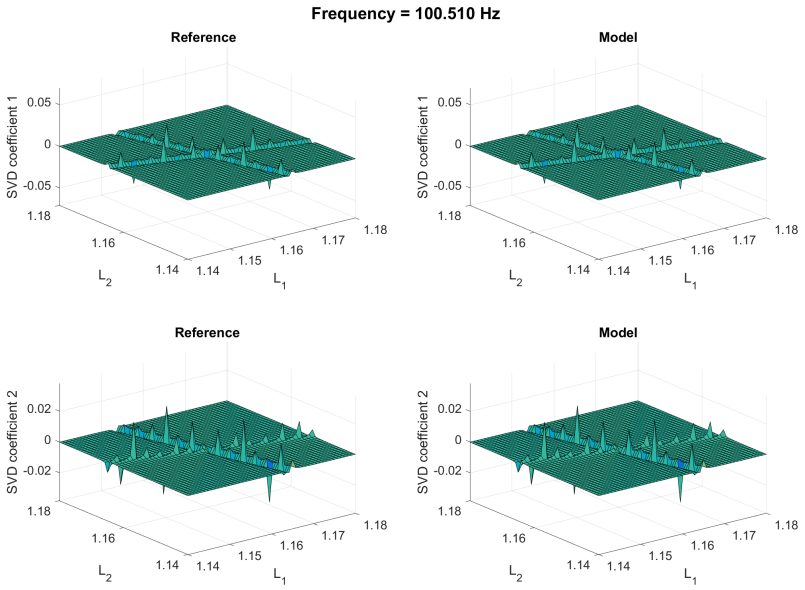


Figure 1.46 – f=100.50 Hz

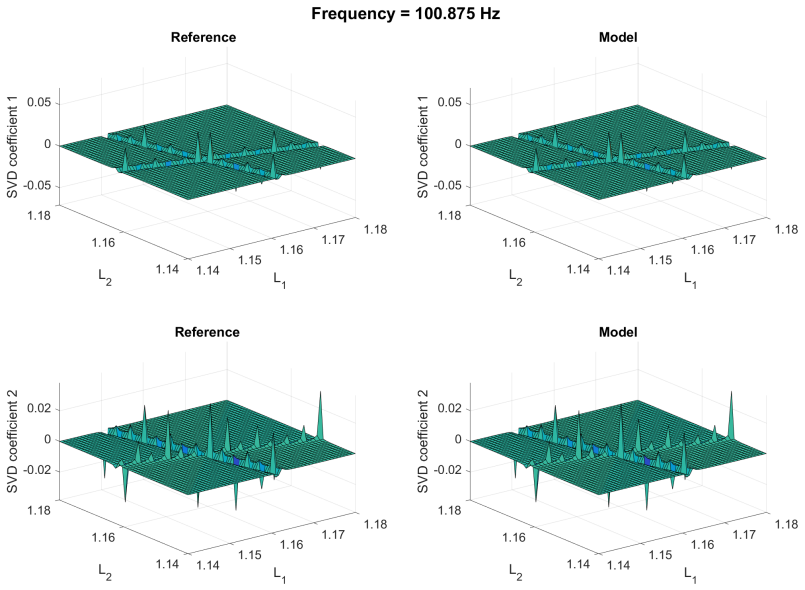


Figure 1.47 – f=100.85 Hz

The relative validation error for the SVD coefficients is **0.0217**.

When we go back to original space, the relative validation error for the entire field is **0.0268**.

From these graphics we see that despite the limited amount of information provided by the learning data the model is able to position really accurately the moving poles of the system as indicated by the peaks that perfectly fit.
Actually as there is no damping the magnitude of the modes could be infinite at the (f, p_{L_1}, p_{L_2}) triplets defining a pole. The points of the grid we used to produce the validation data can be more or less close to those poles and these distances define the magnitude of the peaks we see. For that reason we can claim that the poles are well located. Plus we can observe that the point where two different pole trajectories meet (leading to an higher order pole) does not cause any difficulty.

1.10 From mechanics to electromagnetism

We are now switching to another kind of physics, that is electromagnetism. In fact, equations from electromagnetism are really close to the ones of mechanics that we considered in the previous section. The common principles ruling those two physics are well exhibited in the book of Gilbert Strang [17].

1.10.1 Maxwell's equations

Electromagnetism phenomena are modeled by Maxwell's equations. They can be written as follows:

$$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t} \tag{1.10.1}$$

$$\nabla \times \mathbf{B} = \mu_0(\mathbf{J} + \varepsilon_0 \frac{\partial \mathbf{E}}{\partial t}) \tag{1.10.2}$$

$$\nabla \cdot \mathbf{E} = \frac{\rho}{\varepsilon_0} \tag{1.10.3}$$

$$\nabla \cdot \mathbf{B} = 0. \tag{1.10.4}$$

Where 1.10.1 is the Maxwell-Faraday equation, 1.10.2 is Ampère's circuital law, 1.10.3 and 1.10.4 are Gauss's laws. ε_0 and μ_0 are respectively the permittivity

and the permeability of free space.
We can add to these equations a continuity equation

$$\nabla \cdot \mathbf{J} = -\frac{\partial \rho}{\partial t}. \tag{1.10.5}$$

Those equations are linear and hyperbolic. That reminds us of the mechanics equations.

1.10.2 Test case: The microstrip line

Ansys provided us with simulated data of a Microstrip line. Microstrip is a type of electrical transmission line. It consists of a conducting strip separated from a ground plane by a dielectric layer known as the substra. It is usually made using printed circuit board technology. Its role is to convey microwave-frequency signals. It behaves to a certain extent like a waveguide.
The microstrip geometry can be seen in figure 1.48.

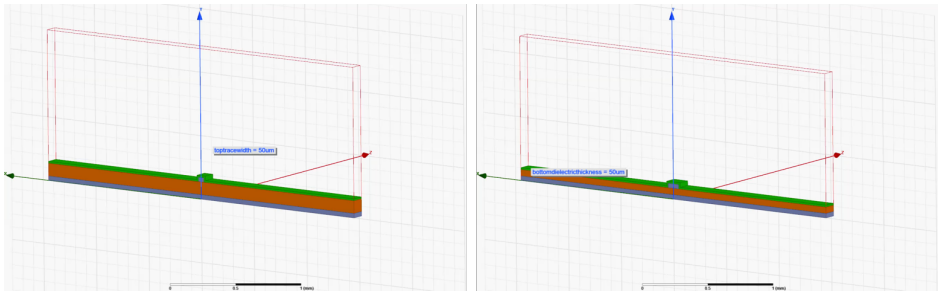


Figure 1.48 – Microstrip geometry

The design parameters that will play the role of p are the so called *trace width* (TW in short) and *dieletric thickness* (DT in short).
In that case we are particularly interested in the behavior of the scattering matrix S that depicts the linear relation between incoming waves and outgoing waves in the Microstrip line. It is the same notion as the scattering matrix for waveguides.
This matrix S is 2×2 here and depends on the frequency f of the incoming waves, on TW and on DT

$$(f, TW, DT) \rightarrow S(f, TW, DT). \tag{1.10.6}$$

It has to be said that $f \to S(f)$ is not meromorphic in a general manner. Depending on the domain there can be cutoff frequencies that break that property as it is reminded in [18]. That feature could imply a change in the basis functions we use. However, in this particular case there is no such a problem.

The design of experiment is shown in figure 1.49. 10 learning points for the design configuration (TW, DT) are considered. Additionally, there are 30 extra design configurations to validate. Regarding the frequency, 10 learning points are used (equally spaced from 0 to 5GHz) and about 500 validation frequencies are in stock for every design configuration.

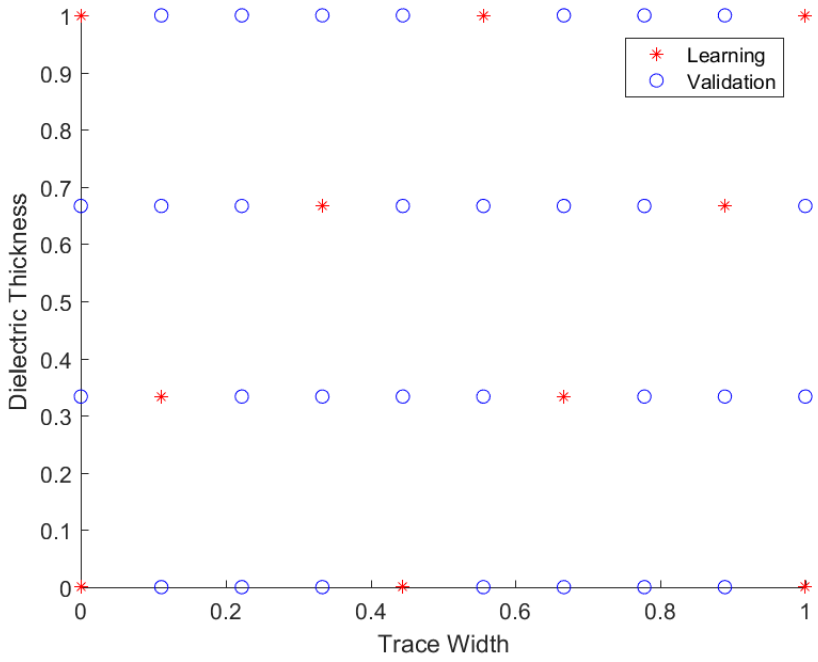
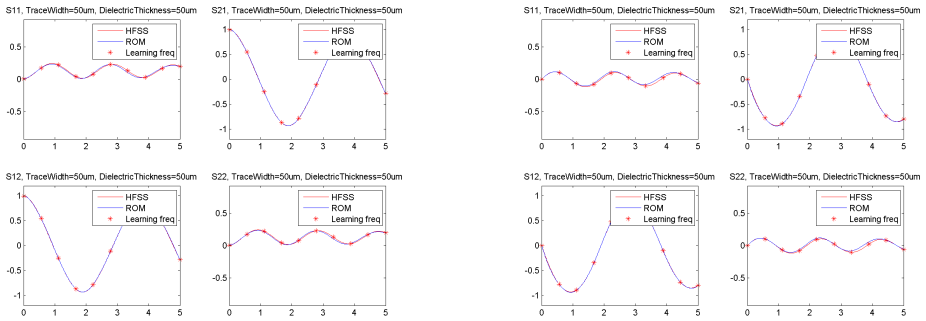


Figure 1.49 – Design of experiment

As already mentioned, we are trying to model matrix S. Thus it is no longer a field, and we will not use the singular value decomposition as a preprocessing.

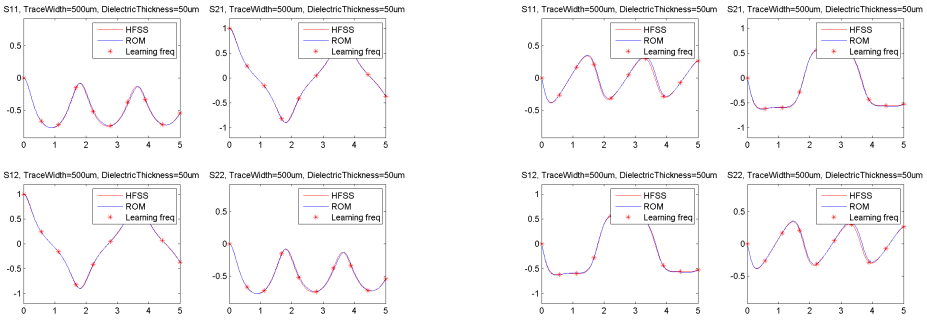
1.10.2.1 Results for the geometries in the learning set

Figures 1.50 to 1.51 display the results for designs of the Microstrip Line that are included in the learning set. One should keep in mind that only 10 frequencies are available for each design configuration. They are labeled with $*$. Thus the remaining displayed frequency points are already validation data. For a given design configuration a first graphic shows the real part of the frequency response of each of the four elements of S and a second one displays the corresponding imaginary part.



(a) Real part (b) Imaginary part

Figure 1.50 – Learning set 1/2

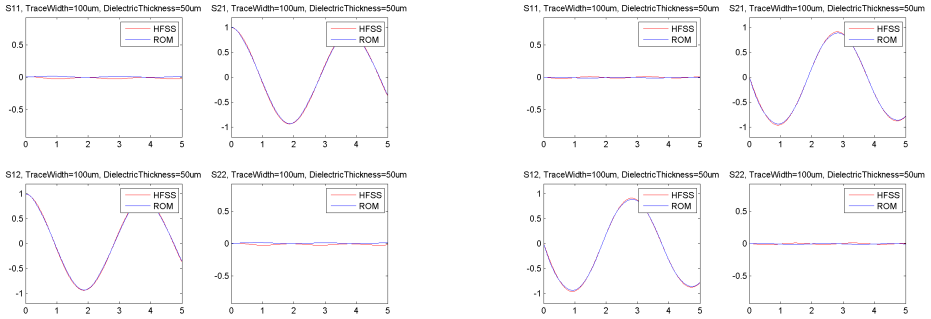


(a) Real part (b) Imaginary part

Figure 1.51 – Learning set 2/2

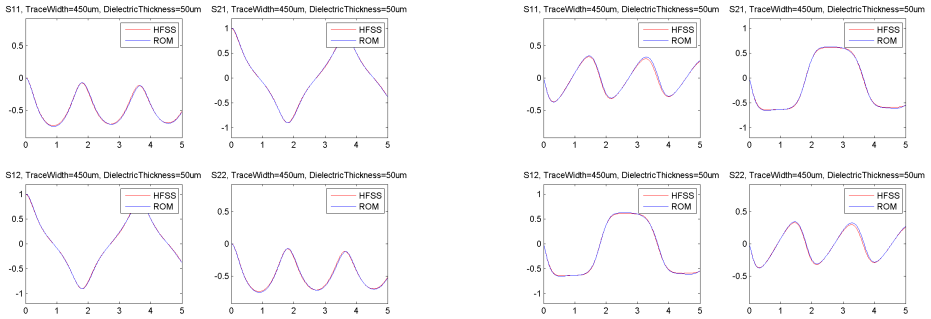
1.10.2.2 Results for the geometries in the validation set

Figure 1.52 to 1.54 display the results for some designs of the Microstrip Line that where not seen during the learning phase.



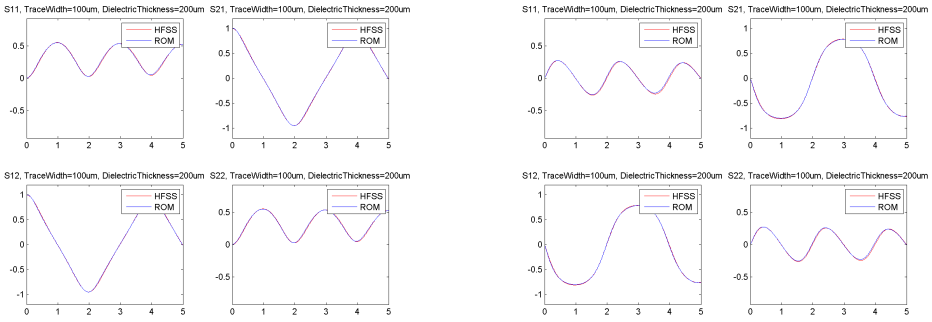
(a) Real part (b) Imaginary part

Figure 1.52 – Validation set 1/3



(a) Real part (b) Imaginary part

Figure 1.53 – Validation set 2/3



(a) Real part (b) Imaginary part

Figure 1.54 – Validation set 3/3

We can see that the ROM performs efficiently. The computed relative error in Frobenius norm over the validation set is equal to 2%.

Note: We have noticed that we could not get better results by taking additional learning data. The reason to this is presumably that every change in the geometry of the microstrip line implies a new mesh. That does not prevent from building the ROM as we are interested in the scattering matrix S. But the computed S is not continuous with respect to TW and DT anymore. However one can notice that the method is nevertheless stable as this numerical noise does not lead to completely irrelevant model. We could even assume that our model is a way to filter and clean the computed learning data.

1.11 Conclusion and perspectives

The proposed method to build reduced order models has proven to be efficient for the considered test cases. This method is not restricted to linear mechanics and electromagnetism. More generally, it can be applied to parameterized linear systems that are close to singularities at some point.
However, there is still work to improve the method. Especially because it is not easy to know if the learning set we consider is rich enough to build a reliable model. We need to develop a new option that allows the model to pilot the learning set creation. This could be particularly useful and is something that Adagos's potential customers ask for. We have seen in 1.8.3.3 that it is safer to choose the basis functions according to the learning set. Actually,

those functions should, above all, be adapted to what we want to model. We are not changing the fact that the selection of the basis functions is made from the learning set though. This proved to be healthy and efficient. Hence, to get functions that are adapted to what we model we need to monitor selection of the learning points iteratively. In this manner, the basis functions will adapt to what they model through the appropriate selection of the learning points.
This work is in progress and relies mainly on sparse grids method 3. But it brings a lot of additional difficulties. The main reason is that the method does not interpolate data contrary to usual sparse grids heuristic.

Chapter 2

Dynamical Reduced Order Models

Adagos has been developing tools to build reduced order models of nonlinear dynamical systems named dynaROM. They initially focused on the field of fluid dynamics and thermal transfer. The goal was at first to be able to learn the response of a fluid flow under time dependent excitations (fluid velocity at the entries, external pressure, heating, etc...). Contrary to chapter 1, the problem here is in a general manner non-linear and it cannot be addressed in the frequency space using the principle of superposition. This is a new challenge to address in the field of reduced order models. We consider here systems described by the Navier-Stokes equations.
State-of-the-art methods of model reduction in the field of fluid dynamics rely on proper orthogonal decomposition (POD) [14] and on the Galerkin method [15]. Most of the energy of the field is captured using a small number of POD modes, then a Galerkin projection of the Navier-Stokes equations onto this smaller space allows to speed the computations up. These methods are very popular and can lead to very good results. They can even allow for real-time computations. But they are usually intrusive and rely on a CFD solver.
Similarly to zROM, we want dynaROM to perform fully non-intrusive creations of ROMs. The identification of the coefficients of the model is made by learning from data that have usually been produced by a CFD software like Fluent from Ansys. Learning data correspond to the fluid flow for a given time dependent input (for example the speed of the flow at an inlet). Then dynaROM makes it possible to predict what will the fluid flow be for new inputs in an autonomous manner, without using a CFD software. We will see that this method should even allow to learn from measured data.

2.1 PDE model

The dynaROM version we are using here aims at modeling the velocity of a fluid flow depending on some dynamical inputs. We only consider the viscous and incompressible case. The structure of the model is based on the Navier-Stokes equations in a domain Ω

$$\begin{cases} \dfrac{\partial u}{\partial t} + (u\cdot\nabla)u - \nu\nabla^2 u + \nabla p = f, \\ \nabla\cdot u = 0 \end{cases} \tag{2.1.1}$$

where u is the flow velocity, ν is the kinematic viscosity, p is pressure, f is the external source and $(u\cdot\nabla)u$ is defined by

$$((u\cdot\nabla)\,u)_i = \sum_{j=1}^{d} u_j \frac{\partial u_i}{\partial x_j} \tag{2.1.2}$$

2.1.1 Boundary conditions

We consider problems where the boundary of the domain $\Gamma = \partial\Omega$ can be broken down as follows:

- Γ_0 the inner side of the object we consider,
- Γ_i with $i \in (1,\dots,n_{\text{out}})$ the inlets/outlets.

The number n_{out} of outlets is 2 in the case of a simple pipe or more for an aorta for example.
The following boundary conditions are prescribed on these portions of the boundary:

$$\begin{aligned} u &= 0, && \text{on} \quad \Gamma_0, \\ \nu\frac{\partial u}{\partial n} - pn + \frac{1}{2}(u\cdot n)\,u &= p_{\text{ext}} n, && \text{on} \quad \Gamma_i, i \geq 1. \end{aligned} \tag{2.1.3}$$

2.1.2 Objective

Our goal is to build a dynamical reduced order model of this system using learning data.
The learning data are some dynamical excitations p_{ext}^L and f^L, and the corresponding time dependent fluid flow u^L that is solution to 2.1.1 with the boundary conditions from 2.1.3.

The final objective is to use the reduced order model to predict what will be the fluid flow u for new excitations p_{ext} and f.
To do so we are going to work on these equations to progressively showcase the structure of the reduced order model we are implementing.

2.1.3 Weak form

2.1.3.1 Getting rid of the pressure

In order to build a ROM that only handles the flow velocity, let us try to get rid of the pressure p in the equation. First we change equation 2.1.1 into its weak form.
We choose the test functions to be in $V = \{\varphi \in H^1(\Omega); \quad \nabla\cdot\varphi = 0, \quad \varphi = 0 \quad \text{on} \quad \Gamma_0\}$.

$$\int_\Omega \frac{\partial u}{\partial t}\cdot\varphi + \int_\Omega (u\cdot\nabla)u\cdot\varphi + \int_\Omega \nabla p\cdot\varphi = \int_\Omega f\cdot\varphi. \tag{2.1.4}$$

Integrating by parts we get

$$\int_\Omega \frac{\partial u}{\partial t}\cdot\varphi + \int_\Omega (u\cdot\nabla)u\cdot\varphi + \nu\int_\Omega \nabla u\cdot\nabla\varphi - \nu\int_\Gamma \frac{\partial u}{\partial n}\cdot\varphi - \int_\Omega p\nabla\cdot\varphi + \int_\Gamma p\,(\varphi\cdot n) = \int_\Omega f\cdot\varphi. \tag{2.1.5}$$

As $\nabla\cdot\varphi = 0$ we get

$$\int_\Omega \frac{\partial u}{\partial t}\cdot\varphi + \underbrace{\int_\Omega (u\cdot\nabla)u\cdot\varphi}_{\text{Non-linear term}} + \nu\int_\Omega \nabla u\cdot\nabla\varphi - \nu\int_\Gamma \frac{\partial u}{\partial n}\cdot\varphi + \int_\Gamma p\,(\varphi\cdot n) = \int_\Omega f\cdot\varphi. \tag{2.1.6}$$

2.1.3.2 Rewriting the quadratic term using the boundary conditions

Now, let us integrate by part the non-linear term

$$\begin{aligned}
\underbrace{\int_\Omega (u\cdot\nabla)u\cdot\varphi}_{\text{NL1}} &= \int_\Omega u_j\partial_j u_i\varphi_i \\
&= -\int_\Omega u_j u_i\partial_j\varphi_i + \int_\Gamma (u\cdot n)u\cdot\varphi \\
&= \underbrace{-\int_\Omega (u\cdot\nabla)\varphi\cdot u + \int_\Gamma (u\cdot n)u\cdot\varphi}_{\text{NL2}}.
\end{aligned} \tag{2.1.7}$$

From that result, we can choose to write the non-linear term as $\frac{\text{NL1+NL2}}{2}$

$$\text{Non-linear term} = \frac{1}{2}\left(\int_\Omega \left((u\cdot\nabla)u\cdot\varphi - (u\cdot\nabla)\varphi\cdot u\right) + \int_\Gamma (u\cdot n)u\cdot\varphi\right). \tag{2.1.8}$$

Hence equation 2.1.6 changes into

$$\begin{aligned}\int_\Omega \left(\frac{\partial u}{\partial t}\cdot\varphi + \frac{1}{2}\left((u\cdot\nabla)u\cdot\varphi - (u\cdot\nabla)\varphi\cdot u\right) + \nu\nabla u\cdot\nabla\varphi\right)\\ + \int_\Gamma \left(\frac{1}{2}(u\cdot n)u\cdot\varphi - \nu\frac{\partial u}{\partial n}\cdot\varphi + p\left(\varphi\cdot n\right)\right) = \int_\Omega f\cdot\varphi.\end{aligned} \tag{2.1.9}$$

Let us focus on the terms related to the integral on the boundary Γ

$$\mathrm{I}_\Gamma = \int_\Gamma \left((u\cdot n)u\cdot\varphi - \nu\frac{\partial u}{\partial n}\cdot\varphi + p\left(\varphi\cdot n\right)\right). \tag{2.1.10}$$

As $u = 0$ and $\varphi = 0$ on Γ_0

$$\int_{\Gamma_0} \left((u\cdot n)u\cdot\varphi - \nu\frac{\partial u}{\partial n}\cdot\varphi + p\left(\varphi\cdot n\right)\right) = 0, \tag{2.1.11}$$

and

$$\mathrm{I}_\Gamma = \sum_{i=1}^{n_{\text{out}}} \int_{\Gamma_i} \left((u\cdot n)u\cdot\varphi - \nu\frac{\partial u}{\partial n}\cdot\varphi + p\left(\varphi\cdot n\right)\right). \tag{2.1.12}$$

As the boundary condition on those sections is $\nu\frac{\partial u}{\partial n} - pn + \frac{1}{2}(u\cdot n)\,u = p_{\text{ext}}n$

$$\mathrm{I}_\Gamma = -\sum_{i=1}^{n_{\text{out}}} \int_{\Gamma_i} p_{\text{ext}}\left(\varphi\cdot n\right). \tag{2.1.13}$$

Hence, going back to equation 2.1.9 we get

$$\int_\Omega \left(\frac{\partial u}{\partial t}\cdot\varphi + \frac{1}{2}\left((u\cdot\nabla)u\cdot\varphi - (u\cdot\nabla)\varphi\cdot u\right) + \nu\nabla u\cdot\nabla\varphi\right) = \int_\Omega f\cdot\varphi + \sum_{i=1}^{n_{\text{out}}} \int_{\Gamma_i} p_{\text{ext}}\left(\varphi\cdot n\right), \tag{2.1.14}$$

which is the form we will use.

2.1.3.3 Energy analysis of the PDE

Let us study how the energy of u evolves over time. It is a key feature for a dynamical system.
To do so we use $\varphi = u$ as the test function in the weak formulation of 2.1.14:

$$\int_\Omega \left(\frac{\partial u}{\partial t} \cdot u + \frac{1}{2}\left((u\cdot\nabla)u\cdot u - (u\cdot\nabla)u\cdot u\right) + \nu \nabla u \cdot \nabla u \right) = \int_\Omega f\cdot u + \sum_{i=1}^{n_{\text{out}}} \int_{\Gamma_i} p_{\text{ext}}\,(u\cdot n)\,. \tag{2.1.15}$$

That gives directly the time evolution of the energy

$$\frac{d}{dt}\int_\Omega |u|^2 = \int_\Omega \left(\underbrace{\left((u\cdot\nabla)u\cdot u - (u\cdot\nabla)u\cdot u\right)}_{=0} - 2\nu|\nabla u|^2 \right) + 2\int_\Omega f\cdot u + 2\sum_{i=1}^{n_{\text{out}}} \int_{\Gamma_i} p_{\text{ext}}\,(u\cdot n)\,. \tag{2.1.16}$$

Hence, the non-linear term has no contribution to energy and we get

$$\frac{d}{dt}\int_\Omega |u|^2 = -2\nu\int_\Omega |\nabla u|^2 + 2\int_\Omega f\cdot u + 2\sum_{i=1}^{n_{\text{out}}} \int_{\Gamma_i} p_{\text{ext}}\,(u\cdot n)\,. \tag{2.1.17}$$

and if $p_{\text{ext}} = 0$ and $f = 0$ that leads to

$$\frac{d}{dt}\int_\Omega |u|^2 = -2\nu\int_\Omega |\nabla u|^2. \tag{2.1.18}$$

That implies that the energy decreases with time. We will ensure to build a model that keeps this property.

2.2 The reduced order model

2.2.1 The semi-discrete ROM

To define the structure of the reduced order model, we need to project the solution on a reduced basis.
We define u_r by

$$u_r(t,\xi) = \sum_{i=1}^{r} x_i(t)\varphi_i(\xi), \quad (t,\xi) \in \mathbb{R}^+ \times \Omega \tag{2.2.1}$$

where the functions φ_i form an L^2-orthonormal family of V.
Hence

$$\nabla \cdot \varphi_i = 0. \tag{2.2.2}$$

Thus, by nature

$$\nabla \cdot u_r = 0, \tag{2.2.3}$$

Introducing u_r in the weak form [2.1.14] with φ_k as a test function we get

$$\sum_{i=1}^{r} \frac{dx_i}{dt} \int_\Omega \varphi_i \cdot \varphi_k + \frac{1}{2} \sum_{i,j=1}^{r} x_i x_j \int_\Omega \left((\varphi_i \cdot \nabla)\varphi_j \cdot \varphi_k - (\varphi_i \cdot \nabla)\varphi_k \cdot \varphi_j \right) + \nu \sum_{i=1}^{r} \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_k = \int_\Omega f \cdot \varphi_k + \int_{\Gamma_1} p_{\text{ext}} \left(\varphi_k \cdot n \right). \tag{2.2.4}$$

As $\int_\Omega \varphi_i \cdot \varphi_k = \delta_{i,k}$, that leads to

$$\begin{aligned} \frac{dx_k}{dt} = & -\nu \underbrace{\sum_{i=1}^{r} x_i \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_k}_{Linear\,part} \\ & + \underbrace{\frac{1}{2} \sum_{i,j=1}^{r} x_i x_j \int_\Omega \left((\varphi_i \cdot \nabla)\varphi_k \cdot \varphi_j - (\varphi_i \cdot \nabla)\varphi_j \cdot \varphi_k \right)}_{Non-linear\,part} + b_{f_k} + b_{p_k}. \end{aligned} \tag{2.2.5}$$

where

$$b_f = \left(b_{f_k} \right)_{1 \le k \le r} = \left(\int_\Omega f \cdot \varphi_k \right)_{1 \le k \le r}, \tag{2.2.6}$$

and

$$b_p = \left(b_{p_k} \right)_{1 \le k \le r} = \left(\int_{\Gamma_1} p_{\text{ext}} \left(\varphi_k \cdot n \right) \right)_{1 \le k \le r}. \tag{2.2.7}$$

At this point, if we consider every $k \in (1, \dots, r)$, with r relatively small, it is starting to look like a reduced order model. It may even look like what is used in [16]. The main difference is that we are not going to compute every terms of 2.2.5. We will not compute explicitly gradients, products and so on. Alternatively, we are going to identify these coefficients from a learning process. We do not need to know the geometry of the system or the properties of the fluid. We will only rely on some computations of the fluid flow.
We can rewrite equation 2.2.5 in the form

$$\dot{x}(t) = \mathbf{W}(x(t)) + b(t) \tag{2.2.8}$$

where

$$b(t) = b_f(t) + b_p(t). \tag{2.2.9}$$

and where W is

$$\mathbf{W}(x) = \mathbf{A}x + \sum_{k=1}^{r} x_k \mathbf{M}_k x. \tag{2.2.10}$$

$\mathbf{A}$ and $\mathbf{M_k}$ are $r \times r$ matrices. So $\mathbf{W}$ is the sum of a linear mapping and a quadratic mapping. We will denote by $\mathbf{M}$ the $r \times r \times r$ hypermatrix containing the $\mathbf{M}_k$

$$\mathbf{M} = \begin{bmatrix} \mathbf{m}_{1,1,k} & \cdots & \mathbf{m}_{1,n,k} \\ \vdots & \ddots & \vdots \\ \mathbf{m}_{n,1,k} & \cdots & \mathbf{m}_{n,n,k} \end{bmatrix}. \tag{2.2.11}$$

From 2.2.5, A is equal to

$$\mathbf{A} = -\nu \left[\int_\Omega \nabla \varphi_i \nabla \varphi_j \right], \tag{2.2.12}$$

and M to

$$\mathbf{M} = \left[\mathbf{m}_{i,j,k} \right] = \int_\Omega \left((\varphi_k \cdot \nabla)\varphi_i \cdot \varphi_j - (\varphi_k \cdot \nabla)\varphi_j \cdot \varphi_i \right). \tag{2.2.13}$$

2.2.2 The discrete-in-time ROM

In practice we are not using a continuous time. With a discrete time and a centered scheme, equation 2.2.8 changes into

$$\frac{x^{i+1} - x^i}{\delta t} = \mathbf{W}\left(\frac{x^{i+1} + x^i}{2} \right) + \frac{b^{i+1} + b^i}{2}. \tag{2.2.14}$$

In order to shorten the notation we define

$$b^{i+\frac{1}{2}} = \frac{b^{i+1} + b^i}{2}, \tag{2.2.15}$$

and

$$x^{i+\frac{1}{2}} = \frac{x^{i+1} + x^i}{2}. \tag{2.2.16}$$

Equation 2.2.14 gives us the structure of the ROM we want to build. Alternatively, we will not compute explicitly $\mathbf{W}$, but rather identify W using learning data, making sure that is has the same core properties.

Objective

Our goal is to build a reduced model in the form

$$\frac{x^{i+1}-x^i}{\delta t} = W\left(x^{i+\frac{1}{2}}\right) + b^{i+\frac{1}{2}}. \tag{2.2.17}$$

where

$$W(x) = Ax + \sum_{k=1}^{r} x_k M_k x. \tag{2.2.18}$$

A and M are the unknown to identify. We will do so only using learning data

$$(b_L^i, u_L^i)_{0\le i\le T_\ell} \tag{2.2.19}$$

where u_L is provided from a Computational Fluid dynamics (CFD) solver for the given dynamical input b_L.
A and M must be built from properties emerging in the weak formulation of the PDE 2.2.5, projected in a POD basis.
It will be a truly non-intrusive method in the sense that it does not require further information from the solver.
The ultimate goal being to be able to use this ROM to predict what will be the flow u for a new dynamical input b.

In addition we would like the prediction phase to be fast enough to allow real time computing. This would provide an extra feature compared to state-of-the-art methods like [13].

Note: *From now the bold letters* $\mathbf{W}$, $\mathbf{A}$ *and* $\mathbf{M}$ *will always refer to the theoretical operators defined above. Alternatively* W, A, *and* M *will refer to the operators of the reduced order model we try to identify.*

2.2.2.1 Identification problem

First let us say a few words about the learning data. For a given experiment, we call u_{b_L} the fluid velocity related to a learning excitation b_L.
To make things simpler we first consider the case where we get access to the coefficients of the projection of the learning flow u_{b_L} on a set of vectors $(\varphi_1,\dots,\varphi_r)$ satisfying the orthonormality criterion with the natural scalar product $\int_\Omega \varphi\cdot\psi$.

We call x_L^i the coefficients related to $u_{b_L}(t^i)$ (the learning flow at $t^i = t^0 + i\delta t$). The identification problem we want to address is

$$\min_{A,M} \sum_{i=0}^{N} \| x(A,M)^i - x_L^i \|_2^2 \tag{2.2.20}$$

where $x(A,M)^i$ is the i^{th} element of the sequence ruled by the following evolution equation:

$$\begin{aligned} \frac{x^{i+1} - x^i}{\delta t} &= W\left(\frac{x^{i+1} + x^i}{2}\right) + b^{i+\frac{1}{2}}, \quad \forall i \leq N, \\ x^0 &= x_L^0 \end{aligned} \tag{2.2.21}$$

where W directly involves A and M as in 2.2.10.

2.3 Properties and constraints on the ROM

A and M must have built in properties emerging from the weak formulation of the PDE projected in a POD 2.2.5. Let us find out what they should be.

2.3.1 Properties of A

As already mentioned, A should inherit the properties of $\mathbb{A}$

$$\mathbf{A} = -\left[\int_\Omega \nabla\varphi_i \nabla\varphi_j\right] \tag{2.3.1}$$

which is a symmetric matrix. We omit intentionally δt or 2ν that are only non-trivial positive constants multiplying the whole matrix.
In addition to being symmetric, this matrix is the opposite of a positive matrix. Indeed, let x be a vector of $\mathbb{R}^r$,

$$\begin{aligned} x^T \mathbf{A} x &= -\sum_{j=1}^{r} x_j \sum_{i=1}^{r} x_i \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \\ &= -\int_\Omega \nabla\left(\sum_{i=1}^{r} x_i\varphi_i\right) \cdot \nabla\left(\sum_{j=1}^{r} x_j\varphi_j\right) \\ &= -\int_\Omega \left|\nabla\left(\sum_{i=1}^{r} x_i\varphi_i\right)\right|_2^2 \end{aligned} \tag{2.3.2}$$

which is negative.
Thus A will be by nature symmetric and we will pay attention that it remains negative.

2.3.2 Properties of M

Let us denote by g^M the function such that

$$g^M(x) = \sum_{k=1}^{n} x_k M_k x. \tag{2.3.3}$$

Let us remind that the theoretical matrix $\mathbf{M}$ is equal to

$$\mathbf{M} = \left[\mathbf{m}_{i,j,k}\right] = \int_\Omega \left((\varphi_k \cdot \nabla)\varphi_i \cdot \varphi_j - (\varphi_k \cdot \nabla)\varphi_j \cdot \varphi_i\right). \tag{2.3.4}$$

We look for M so that $g^M = g^{\mathbf{M}}$.

2.3.2.1 General property of g^M

One can notice that in a general manner two different hypermatrices M^1 and M^2 can possibly lead to $g^{M^1} = g^{M^2}$. This is the case for instance with $M^2 = \left[m^1_{i,j,k}\right] = \left[m^1_{i,k,j}\right]$. $g_i^{M^1}$ being the i^{th} element of g we get

$$\begin{aligned} g_i^{M^1}(x) &= \sum_{k=1}^{n} x_k \sum_{j=1}^{n} m^1_{i,j,k} x_j \\ &= \sum_{k=1}^{n} \sum_{j=1}^{n} m^1_{i,j,k} x_k x_j \end{aligned} \tag{2.3.5}$$

So we see that we can switch the roles of j and k.
From this result we can look for M verifying $g^M = g^{\mathbf{M}}$ in a form such as, for $k \in (1,\dots,n)$, only the first k lines of M_k are non-zero without loss of generality.

$$M_k = \begin{bmatrix} m_{1,1,k} & \cdots & m_{1,1,k} \\ \vdots & \ddots & \vdots \\ m_{k,1,k} & \cdots & m_{k,n,k} \\ 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix}. \tag{2.3.6}$$

2.3.2.2 Energy

Let us now have a look at the energy contribution of $g^{\mathbf{M}}$:

$$\begin{aligned} x^T g^{\mathbf{M}}(x) &= \left(\sum_{k=1}^r x_k \mathbf{M}_k x\right) \\ &= \sum_{i=1}^r x_i \left(\sum_{k=1}^r x_k \left(\sum_{j=1}^r m_{i,j,k} x_j\right)\right) \\ &= \sum_{i,j,k=1}^r x_i x_j x_k \int_\Omega (\varphi_k \cdot \nabla)\varphi_i \cdot \varphi_j - (\varphi_k \cdot \nabla)\varphi_j \\ &= \int_\Omega \left(\left(\sum_k^r x_k \varphi_k\right) \cdot \nabla\right) \left(\sum_i^r x_i \varphi_i\right) \cdot \left(\sum_j^r x_j \varphi_j\right) \\ &\quad - \left(\left(\sum_k^r x_k \varphi_k\right) \cdot \nabla\right) \left(\sum_j^r x_j \varphi_j\right) \cdot \left(\sum_i^r x_i \varphi_i\right) \\ &= 0. \end{aligned} \tag{2.3.7}$$

So we have to make sure to look for a matrix M that lives in a subspace verifying this criterion. Let us calculate:

$$\begin{aligned} x^T g^M(x) = x^T (\sum_{k=1}^n x_k M_k x) \\ = \sum_{k=1}^n x_k x^T M_k x \\ = \sum_{k=1}^n x_k \sum_{i=1}^n \sum_{j=1}^n x_i m i, j, k x_j \\ = \sum_{k=1}^n \sum_{i=1}^n \sum_{j=1}^n m i, j, k x_i x_j x_k \end{aligned} \tag{2.3.8}$$

The stability condition is satisfied when the sums of the permutations of the i, j and k in $m_{i,j,k}$ are zero.
Thus we look for M in the form

- for every triplet (i, j, k) so that $i < j < k$

$$m_{i,j,k} = -m_{i,k,j} - m_{j,i,k}, \tag{2.3.9}$$

- for every couple (i, k) so that $k < i$

$$m_{i,k,i} = -m_{k,i,i}, \tag{2.3.10}$$

- for every couple (i, k) so that $i < k$

$$m_{i,k,i} = -m_{i,i,k}, \tag{2.3.11}$$

- for every i

$$m_{i,i,i} = 0. \tag{2.3.12}$$

The number of constraints is then

$$\begin{aligned} N_c &= n + n(n-1) + \frac{n(n-1)(n-2)}{6} \\ &= n^2 + \frac{n(n-1)(n-2)}{6}. \end{aligned} \tag{2.3.13}$$

The number of *a priori* non-zero elements being

$$N_{nz} = \frac{n^2(n+1)}{2}, \tag{2.3.14}$$

the number of degrees of freedom is

$$N_{dof} = n\left(\frac{n(n+1)}{2} - n - \frac{(n-1)(n-2)}{6}\right). \tag{2.3.15}$$

Note: It is interesting to notice that when we look at the whole model (i.e linear + nonlinear parts), even though the energy can not diverge it does not mean that its linearization has eigenvalues with real parts at least nonpositive.

2.3.3 Invariance with respect to change of basis

As we are going to identify the model, we can wonder if this general structure is preserved when the basis changes. The answer is no, in a general way. But if we choose the new basis so it is othonormal it works.
Let us consider

$$x^{\text{out}} = Ax^{\text{in}} + g(x^{\text{in}}) \tag{2.3.16}$$

and $P = [\varepsilon_1, \ldots, \varepsilon_n]$ the matrix whose columns are the vectors of the new orthonormal basis.
We note

$$y^{\text{in}} = P^T x^{\text{in}}, \tag{2.3.17}$$

$$y^{\text{out}} = P^T x^{\text{out}}. \tag{2.3.18}$$

Thus

$$\begin{aligned} y^{\text{out}} &= P^T A x^{\text{in}} + P^T g(x^{\text{in}}), \\ &= P^T A P y^{\text{in}} + P^T g(P y^{\text{in}}). \end{aligned} \tag{2.3.19}$$

Linear part The matrix $\tilde{A} = P^T AP$ will be the linear operator in the new basis. First, let us notice that $\tilde{A}$ remains symmetric. Then A and $\tilde{A}$ are similar so they share the same eigenvalues. Hence the stability constraint is preserved.

Non-linear part Let us note g_P the new non-linear term

$$\begin{aligned} g^P(y^{\text{in}}) &= P^T g(x^{\text{in}}), \\ &= P^T \sum_{k=1}^{n} \left(Py^{\text{in}}\right)_k M_k P y^{\text{in}}, \\ &= \sum_{k=1}^{n} \left(Py^{\text{in}}\right) P^T M_k P y^{\text{in}}. \end{aligned} \tag{2.3.20}$$

We define $C = \left[c_{i,j,k}\right]$ so that $C_k = P^T M_k P$,

$$\begin{aligned} g^P(y^{\text{in}}) &= \sum_{k=1}^{n} \left(Py^{\text{in}}\right)_k C_k y^{\text{in}} \\ &= \sum_{k=1}^{n} \left(\sum_{j=1}^{n} p_{k,j} y_i^{\text{in}} \right) C_k y^{\text{in}}, \end{aligned} \tag{2.3.21}$$

Let us focus on $g_i^p(y^{\text{in}})$ the i^{th} element of $g^p(y^{\text{in}})$

$$\begin{aligned} g_i^P(y^{\text{in}}) &= \sum_{k=1}^{n} \left(\sum_{j=1}^{n} p_{k,j} y_j^{\text{in}} \right) \left(\sum_{l=1}^{n} c_{i,l,k} y_l^{\text{in}} \right) \\ &= \sum_{k=1}^{n} \sum_{j=1}^{n} \sum_{l=1}^{n} p_{k,j} c_{i,l,k} y_j^{\text{in}} y_l^{\text{in}} \\ &= \sum_{k=1}^{n} \sum_{j=1}^{n} \left(\sum_{l=1}^{n} p_{k,j} c_{i,l,k} \right) y_j^{\text{in}} y_l^{\text{in}} \end{aligned} \tag{2.3.22}$$

By defining M^p so that

$$m_{i,j,k}^p = \left(\sum_{m=1}^{n} p_{m,j} c_{i,k,m} \right), \tag{2.3.23}$$

we get

$$g^p(y^{\text{in}}) = \sum_{k=1}^{n} y_k^{\text{in}} M_k^p y^{\text{in}} \tag{2.3.24}$$

which is the result we want.
What about the stability constraint ? Let $\left(y^1, y^2\right)$ and $\left(x^1, x^2\right)$ be so that $y^1 = P^T x^1$ and $y^2 = P^T x^2$,

$$\begin{aligned} {y^2}^T g^p(y^1) &= {y^2}^T P^T g(P y^1), \\ &= P^T {x^2}^T P^T g(x^1), \\ &= {x^2}^T P P^T g(x^1), \\ &= {x^2}^T g(x^1). \end{aligned} \tag{2.3.25}$$

Hence this is invariant in the new basis. In particular, for $y = P^T x$

$$y^T g^p(y) = x^T g(x) = 0. \tag{2.3.26}$$

The stability criterion is preserved for a new orthonormal basis.

2.4 Computational aspects

2.4.1 Evaluate the model: solving the nonlinear equation

In order to identify A and M, we have to be able to evaluate the model for a given couple A and M. As the scheme is implicit and the equation is nonlinear it is not trivial.
Assuming that we know x^i, let us explain how we get to x^{i+1}. We need to solve the following problem

$$\frac{s - x^i}{\delta t} = W\left(\frac{s + x^i}{2}\right) + b^{i+\frac{1}{2}} \tag{2.4.1}$$

where s is the unknown. As there are r unknowns and r equations we choose to use the Newton method to perform this resolution.
On this basis, we can compute the sequence of x^i from 1 to N by solving this system successively starting from x^0.

2.4.2 Preprocessing of the learning data

At this point we have made the unlikely hypothesis that we have access to the projection of a theoretical solution of the PDE in an orthonormal reduced basis.

In practice we deal with a discrete solution coming from a solver. The solver usually uses the finite volume method.
The reference learning solution related to the excitation b_L is denoted by

$$U_{b_L} = \left[U^0_{b_L}, \dots, U^N_{b_L} \right] \tag{2.4.2}$$

where

$$U^i_{b_L} = \begin{pmatrix} {u_{b_L}}^i_1 \\ \vdots \\ {u_{b_L}}^i_n \end{pmatrix}. \tag{2.4.3}$$

Here ${U_{b_L}}^i_k$, is a vector of $\mathbb{R}^d$ ($d = 2$ or 3) representing the flow velocity at point k.
However U_{b_L} is not a sufficient piece of information. Each of the n points is linked to a volume. As the fluid is incompressible we can directly link the volume to the mass.
We define $\mathbf{m} = \text{diag}(m_k)$ the diagonal mass matrix where an element of the diagonal is the mass of the corresponding point. This matrix is necessary to compute the energy. We call E^i the energy at time step i:

$$E^i = \sum_{k=1}^{n} m_k {u_{b_L}}^i_k{}^T {u_{b_L}}^i_k \tag{2.4.4}$$

Assuming that the CFD computation is reliable enough we expect that E^i has the same properties as the continuous equivalent from the original PDE: it should decrease when the excitations are zero.

$$E^{i+1} \leq E^i \quad \text{for} \quad b^i_L = 0 \tag{2.4.5}$$

Furthermore, if the computation is accurate enough, once the solution is projected on an orthonormal reduced basis of $\mathbb{R}^{(d\times n)}$, the properties of the reduced order model introduced in 2.3.3 are compatible to model the solutions of this system.
To build an orthonormal reduced basis we use the Singular Value Decomposition (SVD) method.
First, we introduce the $\tilde{\cdot}$ notation

$$\tilde{u} = \begin{pmatrix} \sqrt{m_1} u_1 \\ \vdots \\ \sqrt{m_n} u_n \end{pmatrix}. \tag{2.4.6}$$

where each u_k is a vector of $\mathbb{R}^d$. This is convenient to write simply the natural scalar product

$$(u|v) = \sum_{k=1}^{n} m_k u_k^T v_k = \tilde{u}^T \tilde{v} \tag{2.4.7}$$

Then we apply the SVD to

$$\tilde{U}_{b_L} = \begin{bmatrix} \sqrt{m_1} u_{b_L}{}_1^1 & \cdots & \sqrt{m_1} u_{b_L}{}_1^N \\ \vdots & \ddots & \vdots \\ \sqrt{m_n} u_{b_L}{}_n^1 & \cdots & \sqrt{m_n} u_{b_L}{}_n^N \end{bmatrix}. \tag{2.4.8}$$

which gives

$$\tilde{U}_{b_L} = \Phi \Sigma \Psi^T \tag{2.4.9}$$

where

- Φ is a $dn \times dn$ unitary matrix,
- Σ is a diagonal $n \times N$ matrix with the singular values σ_k on the diagonal which are non-negative real numbers,
- Ψ_{b_L} is an $N \times N$ unitary.

Depending on how the singular values (contained in Σ) decrease we chose a number r of reduced basis vectors. We define then

$$\Phi_r = \begin{bmatrix} \varphi_1 & \cdots & \varphi_r \end{bmatrix} \tag{2.4.10}$$

$$\Sigma_r = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \tag{2.4.11}$$

$$\Psi_r = \begin{bmatrix} \psi_1 & \cdots & \psi_r \end{bmatrix} \tag{2.4.12}$$

where only the r first columns of each matrix are kept.
Hence we get the corresponding discrete-time coefficients $x_{b_l} \in \mathbb{R}^{r \times N}$:

$$x_{b_l} = \Phi_r^T \tilde{U}_{b_L} = \Sigma_r \Psi_r^T. \tag{2.4.13}$$

Hence, x_{b_l} is then the reference learning data used to identify the reduced order model.

2.4.3 The identifcation problem in the learning process

We can now try to solve the identification problem

$$\min_{A,M} \sum_{i=0}^{N} \| x(A,M)^i - x_{b_l}^i \|_2^2. \tag{2.4.14}$$

This is a nonlinear least-squares problem. We are going to use a Gauss-Newton method. At each time step i we need to get the derivative of x^{i+1} with respect to A, M and x^i. As we do not want to compute explicitely the Jacobian matrix related to A, M and x^i we need to compute the direct and inverse derivatives in order to proceed as defined in appendix A.

2.4.3.1 Forward mode

First, let us study the forward mode for $x \mapsto y = W(x, A, M)$

$$\begin{aligned} \delta y &= dW(x, A, M, \delta x, \delta A, \delta M) \\ &= \delta A x + A\delta x + \sum_{k=1}^{r} \left(\delta x_k M_k x + x_k \delta M_k x + x_k M_k \delta x \right). \end{aligned} \tag{2.4.15}$$

We note J_x the $r \times r$ matrix such that

$$J_x = A + \left[M_1 x, \quad \ldots, \quad M_r x \right] + \sum_{k=1}^{r} x_k M_k. \tag{2.4.16}$$

Thus

$$\begin{aligned} \delta y &= dW(x, A, M, \delta x, \delta A, \delta M) \\ &= \delta A x + J_x \delta x \\ &+ \sum_{k=1}^{r} x_k \delta M_k x \end{aligned} \tag{2.4.17}$$

Hence

$$\frac{\delta x^{i+1} - \delta x^i}{\delta t} = \delta A x^{i+\frac{1}{2}} + J_{x^{i+\frac{1}{2}}} \delta x^{i+\frac{1}{2}} + \sum_{k=1}^{r} x_k^{i+\frac{1}{2}} \delta M_k x^{i+\frac{1}{2}}. \tag{2.4.18}$$

We want to identify δx^{i+1}. As $\delta x^{i+\frac{1}{2}} = \frac{\delta x^{i+1} + \delta x^i}{2}$, we have

$$\frac{\delta x^{i+1} - \delta x^i}{\delta t} = \delta A x^{i+\frac{1}{2}} + J_{x^{i+\frac{1}{2}}} \frac{\delta x^{i+1} + \delta x^i}{2} + \sum_{k=1}^{r} x_k^{i+\frac{1}{2}} \delta M_k x^{i+\frac{1}{2}}, \tag{2.4.19}$$

which leads to

$$\left(\frac{1}{\delta t}I_r - \frac{1}{2}J_{x^{i+\frac{1}{2}}}\right)\delta x^{i+1} = \delta A x^{i+\frac{1}{2}} + \left(\frac{1}{\delta t}I_r + \frac{1}{2}J_{x^{i+\frac{1}{2}}}\right)\delta x^i + \frac{\delta x^{i+1}+}{2} + \sum_{k=1}^{r} x_k^{i+\frac{1}{2}}\delta M_k x^{i+\frac{1}{2}}. \tag{2.4.20}$$

This system being small it is solved using a direct method.

2.4.3.2 Reverse mode

In the same manner as earlier, we first look at the reverse mode derivative of $y = W(x, A, M)$:

$$px = A^T + \sum_{k=1}^{r}\left(e_k x^T M_k^T + x_k M_k^T\right) py, \tag{2.4.21}$$

$$pA = pyx^T, \tag{2.4.22}$$

$$pM_k = x_k pyx^T, \quad \forall k \in [1,\dots,r]. \tag{2.4.23}$$

Introducing this into the main equation, it leads to

$$px^{i+\frac{1}{2}} = \left(A^T + \sum_{k=1}^{r}\left(e_k {x^{i+\frac{1}{2}}}^T M_k^T + x_k^{i+\frac{1}{2}} M_k^T\right)\right)\left(px^{i+1} - px^i\right), \tag{2.4.24}$$

hence we get

$$\begin{aligned}&\left(\frac{1}{\delta t}\left(A^T + \sum_{k=1}^{r}\left(e_k {x^{i+\frac{1}{2}}}^T M_k^T + x_k^{i+\frac{1}{2}} M_k^T\right)\right) + \frac{1}{2}I\right)px^i \\ =&\left(\frac{1}{\delta t}\left(A^T + \sum_{k=1}^{r}\left(e_k {x^{i+\frac{1}{2}}}^T M_k^T + x_k^{i+\frac{1}{2}} M_k^T\right)\right) - \frac{1}{2}I\right)px^{i+1}.\end{aligned} \tag{2.4.25}$$

Once again we solve this relatively small system using a direct method.
Then we get

$$pA = \frac{px^{i+1} - px^i}{\delta t}{x^{i+\frac{1}{2}}}^T, \tag{2.4.26}$$

and

$$pM_k = x_k^{i+\frac{1}{2}}\frac{px^{i+1} - px^i}{\delta t}{x^{i+\frac{1}{2}}}^T, \quad \forall k \in [1,\dots,r]. \tag{2.4.27}$$

2.4.4 Dealing with A nonpositive

Up to there we omitted the constraint that the symmetric matrix A should be nonpositive. Yet it is obviously an important matter. The semidefinite programming (SDP) [21] is sometimes the way to go to deal with this kind of constraints. But this method is suitable to the cases where there is a linear form to minimize. We did not find it to be most convenient solution here. So we decided to use an alternative method.
Alternatively, we use the fact that any nonnegative symmetric S matrix can be decomposed as

$$S = LL^T, \tag{2.4.28}$$

where L is a lower-triangular matrix with real and nonnegative diagonal entries. This is the Cholesky decomposition and it is unique.
Hence we will look for the nonpositive symmetric matrix A in the form

$$A = -LL^T. \tag{2.4.29}$$

The constraint that the diagonal entries must be nonnegative is then easy to handle. This L matrix will be the new unknown of the optimization problem instead of A. Of course it brings nonlinearity and complexity to the problem, but it appeared to be the most convenient way to insure the constraint is met. We just need to connect the derivatives of $F : L \mapsto A = -LL^T$ to the ones defined in (2.4.3).
The forward mode is

$$dF(L, \delta L) = -L\delta L^T - \delta L L^T, \tag{2.4.30}$$

and reverse mode is

$$pF(L, pA) = -L^T pA - pAL^T. \tag{2.4.31}$$

2.4.4.1 The mixed approach

In practice, during the optimization process, the constraint can be far from being active. In other words, A can be very negative. In that case, we may just ensure that A is symmetric (which is straight forward) as long as the constraint is far enough from being active, without introducing the decomposition with L. As soon as a step of the optimization process breaks the constraint, we go back to the previous step of optimization and apply the Cholesky factorization to the current $-A$ (which is indeed nonnegative) and keep going dealing with L as long as necessary. This works well as the optimization algorithm has no memory.

2.4.5 Alternative case: no access to the mass matrix m

We have seen in paragraph 2.4.2 how we proceed to work with a reduced basis using the mass matrix m.
This m matrix allowed us to identify a convenient orthonormal basis to work with using SVD.
To remain as non-intrusive as possible we should be able to build a ROM only using the field U_{b_L} (the learning flow), even if we do not have access to the diagonal mass matrix **m**. It is the case we will consider in our first experiment 2.6.
When we cannot get access to m, we are not able to work with the right scalar product. Thus we cannot make any reliable assumption about energy and orthonormality. We have to overcome this difficulty.

First, we proceed by directly performing an SVD on U_{b_L}.

$$U_{b_L} = \Phi \Sigma \Psi^T \tag{2.4.32}$$

That provides us with discrete-time coefficients that we note y_{b_l}.

$$y_{b_L} = \Phi_r^T U_{b_L} = \Sigma_r \Psi_r^T \tag{2.4.33}$$

The columns of U_{b_L} are orthonormal with respect to the usual scalar product in $\mathbb{R}^{(\times)}$, but they are not if we consider the natural scalar product of the system defined with the mass matrix m.
As there is a default of orthonormality in the appropriate scalar product, we cannot attempt to build a ROM with the correct properties working directly with y_{b_l} if we do not make adjustments in the method.
We are going to describe 3 different ways to address this situation in paragraphs 2.4.5.1 to 2.4.5.3. And we will eventually decide which solution we choose.

2.4.5.1 Option 1: identifying the matrix of the scalar products

Going back to equation 2.2.4, the default of orthogonality of the basis functions leads to the introduction of C such that

$$C\left(y^{i+1} - y^i\right) = W\left(y^{i+\frac{1}{2}}\right) + b^{i+\frac{1}{2}}. \tag{2.4.34}$$

C is to be identified through learning and c_{ij} should ideally correspond to $(\varphi_i|\varphi_j)$ (where the φ_i come from the SVD and are not orthonormal in the appropriate

scalar product $(\cdot|\cdot)$). Thus C should be symmetric.
Let us demonstrate that C is a positive-definite matrix in the ideal case where it is equal to $C = \left[\left(\varphi_i|\varphi_j\right)\right]$. For y nontrivial

$$\begin{aligned} y^T C y &= \sum_{i,j}^{n} y_i y_j \left(\varphi_i|\varphi_j\right) \\ &= \left(\sum_{i}^{n} y_i\varphi_i|\sum_{j}^{n} y_j\varphi_j\right) \quad = \|\sum_{i}^{n} y_i\varphi_i\|^2 > 0. \end{aligned} \tag{2.4.35}$$

Hence, if we try to identify C, we have to ensure that it remains positive-definite in the model. This new constraint may be inconvenient to deal with.

2.4.5.2 Option 2: the orthonoramalization matrix

Alternatively we may try to identify a lower-triangular matrix R that would represent the change of basis of the vectors we got from the SVD (without the mass matrix) into an orthonormal set.
That means that we rewrite the equation with $x^i = Ry^i$ in the form

$$\left(x^{i+1} - x^i\right) = W\left(x^{i+\frac{1}{2}}\right) + Rb^{i+\frac{1}{2}}. \tag{2.4.36}$$

where R is an unknown of the optimization problem. This optimization problem requires the use of R^{-1} as we want to minimize

$$\min_{A,M,R} \sum_{i=0}^{N} \|R^{-1}x(A,M)^i - y_L^i\|_2^2 \tag{2.4.37}$$

The inversion of R (that is simple as the matrix is triangular) is not convenient for the optimization process.

2.4.5.3 Option 3: the linear observation operator

The third option we consider involves a so-called *observation operator*. This is the method we are going to implement.
It consists in decomposing the model into a dedicated dynamical part that has to fit the properties from the physics involved, and an observation operator that makes the link between the dynamical part and the data we want to learn from

$$\begin{aligned} \left(x^{i+1} - x^i\right) &= W\left(x^{i+\frac{1}{2}}\right) + Bb^{i+\frac{1}{2}} \\ y^{i+1} &= Cx^{i+1}. \end{aligned} \tag{2.4.38}$$

Here B is a linear operator that multiplies the dynamical inputs and C is the so-called linear observation operator.
Proceeding this way, we also need to identify the initial condition for the dynamical part x^0. The optimization problem becomes then

$$\min_{A,M,B,C,x^0} \sum_{i=0}^{N} \| y(A,M,B,C,y^0)^i - y_L^i \|_2^2 \tag{2.4.39}$$

The key idea is to work with a dynamical part of the model that preserves the basic structure and properties issued from the Navier-Stokes equations as depicted above.
The x^i vectors correspond to time dependent coefficients in some orthonormal reduced basis that we do not need to identify and that is efficient to represent the fluid dynamics system.
Then the observation matrix C makes the link between those coefficients and the quantities we want to model eventually.
This allows to use straightforwardly the dynamical model introduced before connecting it with new operators. The operators A and M will have to follow the exact rules we set above.
This method may seem more complicated at first glance, but in practice it proves to be pretty efficient and offer new opportunities.

2.5 Giving flexibility to the model

We just introduced the linear operator method to overcome the lack of knowledge about the mass matrix m. And we are going to implement it as the default method as it has further advantages. We will even go further and add extra elements to the ROM.
The reason we want to bring flexibility to the model is because we need to address problems where the boundary conditions we deal with are not strictly the same as the ones defined in 2.1.3. We chose these boundary conditions because they allowed to exhibit a convenient structure for the dynamical part of the ROM. Especially for the purpose of creating a nonlinear dynamical model that cannot diverge. But there are not commonly encountered and can be restrictive.
In addition to the observation operator we are going to introduce new elements that will allow to bring flexibility to the ROM without modifying the core properties and structure of the dynamical part of the model we have developed

throughout this chapter. We do not want to modify the properties of W in the dynamical part, and only add elements around it that guarantee the ROM cannot diverge.

2.5.1 Versatility provided by the observation operator

The observation operator brings some safe flexibility to the system. For example, it allows to learn quantities that linearly derive from the fluid flow field, like local observations or averaged quantities.
One extra feature is that the dynamical part of the model no longer has to be the same size as the learning data coefficients vector (i.e. C is not necessarily square).
The dynamical part of the ROM only allows the L^2 norm of x to decrease when the dynamical input b is set to zero. That is what we wanted as it was the simplest way to go for a nonlinear dynamical model that could not diverge. But this property can be really restricting.
However, the use of the observation operator allows the ROM to increase for a finite amount of time with no risk of exploding when $b = 0$.

Example To illustrate this property let us consider a simple example. In order to simplify we can only consider a linear ROM.

$$\begin{aligned} x^{i+1} - x^i &= Ax^{i+\frac{1}{2}} \\ y^{i+1} &= Ox^{i+1} \end{aligned} \tag{2.5.1}$$

where

$$A = 10^{-3} \begin{bmatrix} -2 & 1 \\ 1 & -2 \end{bmatrix} \tag{2.5.2}$$

This matrix is symmetric and non-positive. Now, let us consider

$$x^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{2.5.3}$$

as the initial condition.
Figure [2.1] shows how the magnitude of each coefficient in x evolves with time.

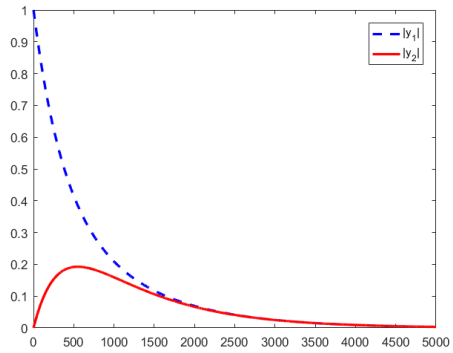


Figure 2.1 – Time evolution of the magnitude of x_1 and x_2

We observe that x_2 first increases in magnitude before decreasing. In the same time the norm of x only decreases as shown in Figure [2.2].

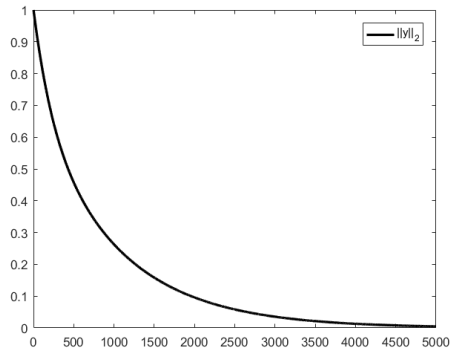


Figure 2.2 – Time evolution of the norm of y

That means that if use the following observation operator

$$O = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.5.4}$$

the energy of the output y can increase at first despite the dynamical input being 0. Which proves our point.
Combining the dynamical model and the observation operator is close to the spirit of reservoir computing [20]. The reservoir computing consists in picking

in a reservoir of dynamical functions of the input what is needed to fit the targeted output. This is done using an observation operator to be identified (this is called the trained readout). In the usual reservoir computing methods the reservoir layer may be a random neural network. In our case we train it as well and make it live in a small space where it has to match basic properties issued from the specific physics involved.
Now we want to build around this model and even add some new features. Doing so we are entering a bit more the world of machine learning.

2.5.2 The observation offset

To give even more flexibility we add an offset term called d

$$\begin{aligned} \left(x^{i+1} - x^i\right) &= W\left(x^{i+\frac{1}{2}}\right) + Bb^{i+\frac{1}{2}} \\ y^{i+1} &= Cx^{i+1} + d \end{aligned} \tag{2.5.5}$$

This term adds an offset to the model but does not affect the property that the energy cannot explode if b is set to zero. From the performed experiments we observed that this new term is really beneficial as it prevent the eigenvalues of the linear term A from being to close to zero.

2.5.3 The excitation offset

We want to be able to modify the input entering the model. The new model is

$$\begin{aligned} \left(x^{i+1} - x^i\right) &= W\left(x^{i+\frac{1}{2}}\right) + Bb^{i+\frac{1}{2}} + e \\ y^{i+1} &= Cx^{i+1} + d \end{aligned} \tag{2.5.6}$$

where B and e define an affine transformation of the input. In the case of the coronary the dynamical input we get is just a scalar. Hence B is simply a vector distributing the input to all the dynamical variables.
This transformation is a bit more tricky than the previous one as it implies a significant change of the energy. Indeed if we set b to zero there may still be some energy entering the system because of e.
This new feature is adapted to the fact that in some cases the boundary condition is such that a constant pressure is set at some outlets of the system for example.

We are going to prove that the energy divergence is at most linear. Let us do it with the time continuous case as it is more general.

$$\frac{dx}{dt} = W(x) + e, \tag{2.5.7}$$

Applying a left multiplication with x^T we get

$$\frac{1}{2}\frac{d\|x\|_2^2}{dt} = x^T W(x) + x^T e, \tag{2.5.8}$$

As $x^T W(x) \leq 0$ by nature that gives

$$\begin{aligned}
\frac{1}{2}\frac{d\|x\|_2^2}{dt} &\leq x^T e, \\
&\leq \sum_i^r x_i e_i, \\
&\leq \sum_i^r |x_i||e_i|, \\
&\leq \|e\|_\infty \sum_i^r |x_i|, \\
&\leq r\|e\|_\infty \|x\|_\infty, \\
&\leq r\|e\|_\infty \|x\|_2.
\end{aligned} \tag{2.5.9}$$

Hence, to study the worst case scenario we may focus on the following differential equation

$$\begin{aligned}
&\frac{d\alpha^2}{dt} = c\alpha, \\
\iff &2\alpha\frac{d\alpha}{dt} = c\alpha.
\end{aligned} \tag{2.5.10}$$

In this worst case scenario, we obviously look for the nontrivial solutions for α, which leads to

$$\begin{aligned}
&\frac{d\alpha}{dt} = \frac{1}{2}c, \\
\iff &\alpha(t) = \frac{1}{2}ct + \alpha(0).
\end{aligned} \tag{2.5.11}$$

From this we deduct that

$$\|x(t)\|_2 - \|x(0)\|_2 = \mathcal{O}(t). \tag{2.5.12}$$

In conclusion the excitation offset allows the system to diverge but only linearly at most. This choice is worth the risk as it allow to accommodate various boundary conditions.

All the elements we introduced to this point allow the ROM to be more versatile. And they only involve a small number of new degree of freedom to identify. Especially if we compare to the size of the hypermatrix of quadratic term.
Moreover we can observe how the method now differs from usual POD-Galerkin reduced order models. With the latter, the space defined by the POD modes on which the equations are projected is extremely important as the dynamical computation is directly performed on the corresponding time coefficients. Alternatively, the method proposed here separates somehow the space on which we project the learning data and the dynamical part of the model. Then we rely on learning to identify a correct dynamical model that performs well over time. We could potentially project the learning field on a uniform function equal to 1 (calculating its spatial average) and still be able to catch in the dynamical part what is necessary to get a reduced order model for this quantity. We can go further and use this method to learn from measured data (coming from wind tunnel experiments for example or from real life systems).

2.5.4 The updated learning process

The use of the observation operator changes slightly the learning process compared to the case we directly use the dynamical part of the ROM to fit the learning data2.4. In particular, given the fact that the size of the dynamical part and the size of the learning data have no longer to be the same.

Step 1 We first perform a SVD over the learning field U_{b_L} and select r_o modes in order to get a good approximation of the learning field.

Step 2 We start the learning process with only one variable in the dynamical part of the model (i.e. C is a matrix $r_o \times 1$).

Step 3 Once the optimization is done, if we are not satisfied, we increased the size of the dynamical part, adding a new variable, and start again the process again. We iterate this way to add as many dynamical variables as needed. We call r_d the number of dynamical variables. Depending on the case r_d can even be greater than r_o in the end.

2.5.5 The updated prediction process

We need to be able to predict what the flow will be for a new dynamical input. There are two different cases to address:

Case 1 The flow at $t = 0$ is in the exact same configuration as during the learning phase.

Case 2 The flow is not in the same configuration at $t = 0$.

In the first case there is no difficulty, we can perform a prediction straightforwardly as the initial values of x^0 identified throughout the learning phase can be reused to predict.
In the second case, however, we need to identify x^0. In that case we can use the CFD solver to get the first time steps of the flow velocity. We use them to identify x^0 using Gauss-Newton, freezing any other element of the ROM. Then we can predict the flow for the following time steps using the ROM.

2.6 The coronary test case

ANSYS provided us with experimental data produced with their CFD solver Fluent. In particular, we got to study the blood flow in an coronary depending on the massflow rate at one inlet. The goal being to build a model that predicts what will the flow be for new time dependent massflow rates. To identify this model we use a learning massflow rate and the corresponding blood flow. The CFD solver used the finite volume method [22]. **The learning data were generated by Ansys** which means that we did not have the opportunity to tune the experiments in order to help the building of the ROM.
In that case the data come as follows:

- The time discretization is regular,
- The only dynamical output is the mass flow rate that comes as a scalar. There is no external source f. Hence b^i, at time step i, is just a scalar representing only the mass flow rate.
- The blood flow velocity is represented via Ux, Uy and Uz that are vectors of $\mathbb{R}^n$ where $n \approx 6 \times 10^5$. Every element of the vectors correspond to a given volume.

The learning mass flow rate and the learning blood flow field are then respectively denoted b_L and U_{b_L}.
As the method aims to be as little intrusive as possible we have little information about the numerical simulation

- There are 0.26M polyhedral cells,
- Outlets massflow repartition is made based on outlet surface ratio to sum of all the outlets surfaces,
- The duration of one computation is 4h on 12 CPUs for about 250 time steps.

This particular case does not match exactly the properties introduced in 2.1. In particular, the boundary conditions do not fit the ones described in 2.1.3. Furthermore, we do not know the mass matrix m. For all theses reasons, we will be using the flexibility given by the observation operator 2.5.

2.6.1 Experimental results

As mentioned earlier, we are first going to train the model using a learning mass flow rate at the inlet (the learning dynamical input) and the corresponding learning flow field issued from a CFD computation (Fluent) 2.6.1.1. Once this is completed, we will be using the model to predict what happens for a new mass flow rate. We will consider two profiles for the mass flow rate and will compare the resulting flow with reference data computed with Fluent 2.6.1.2.

2.6.1.1 Learning phase

We denote by $b_L = \left[b_L^0, \ldots, b_L^{N_L}\right]$ the learning dynamical input, i.e. the mass flow rate. Its profile can be observed in Figure 2.3. The corresponding sequence for the flow field issued from the CFD computation is denoted by $U_{b_L} = \left[U_{b_L}^0, \ldots, U_{b_L}^{N_L}\right]$. Here the number of time steps is $N_L = 471$. The data were generated by Ansys, so we could not monitor the learning data. We have to deal with a learning dynamical input (i.e. the massflow rate) that is not optimal in term of quantity of provided information. We would have preferred a signal that carries more information. Anyway we are going to show that it still allows to obtain good results.

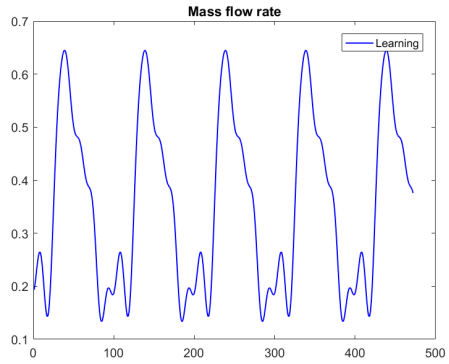


Figure 2.3 – Learning massflow rate (dynamical input)

First of all we perform a SVD (Singular Value Decomposition) on the learning flow field U_{b_L}.

$$U_{b_L} = \Phi \Sigma \Psi^T \tag{2.6.1}$$

We can see in Figure [2.4] how quickly the singular values in Σ decrease.

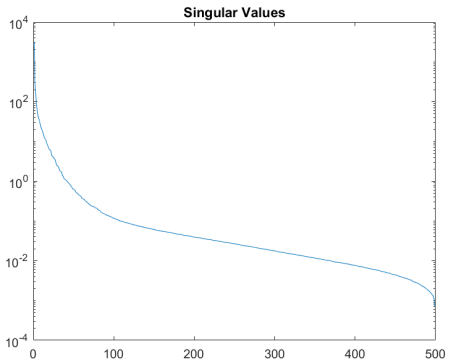


Figure 2.4 – Singular values for the learning flow field (logarithmic scale)

Keeping only the first r_o columns of Φ the field is projected in a smaller space

$$y_{b_l} = \Phi_{r_o}^T U_{b_L}, \tag{2.6.2}$$

where $y_{b_l} \in \mathbb{R}_{r_o}^{N_L}$. We took $r_o = 6$. Doing so the projection error is 0.02199. That r_o sets the size of the observation part.
As mentioned earlier, during the learning process the size r_d of the dynamical

part grows iteratively starting from 1. It reaches size $r_d = 6$ at the end of the learning process.
Eventually the quality of the learning can be measured in Figure 2.5.

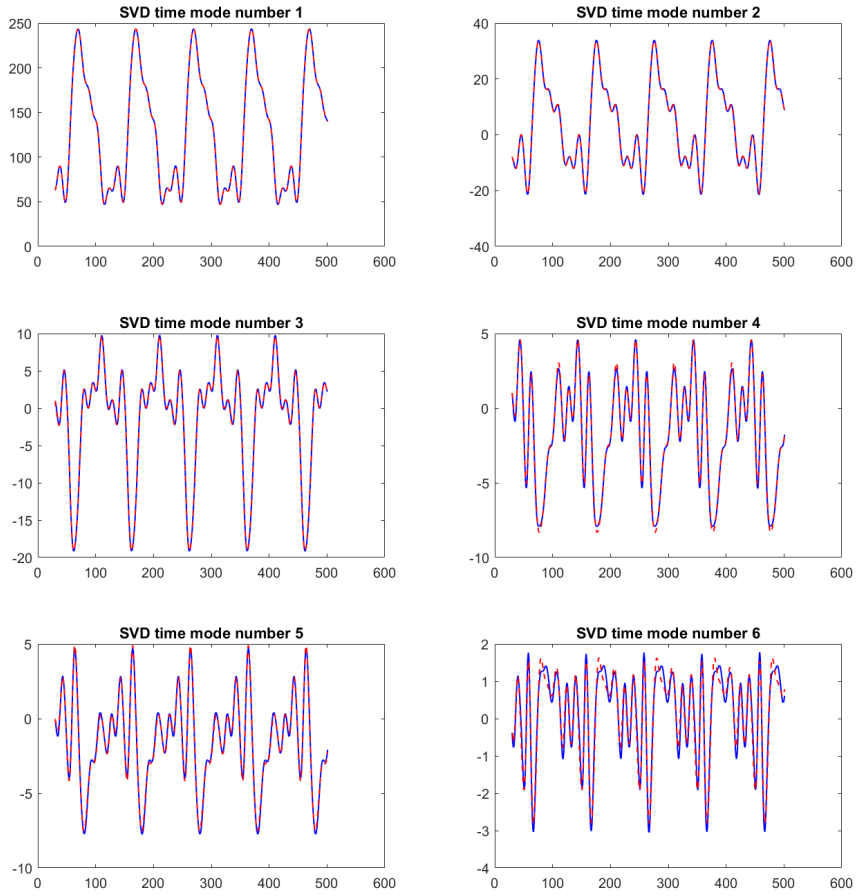


Figure 2.5 – Learning - Blue line is the reference and dashed red line is the ROM

The relative error (compared to the global norm of y_{b_l}) for each time mode is depicted in the table below

Table 2.1 – Learning - Relative error for the SVD time coefficients

	1	2	3	4	5	6
L^2 relative error	0.0030	0.0021	0.0014	0.0018	0.0014	0.0020
L^∞ relative error	0.0050	0.0033	0.0025	0.0033	0.0020	0.0048

The global relative L^2 error is 0.0048.

2.6.1.2 Validation

Now that identification of the model is done it is time to test the prediction on validation data. We have two sets of validation data (namely 1 and 2). In particular there are two new dynamical inputs that are shown alongside the learning one in Figure [2.6].

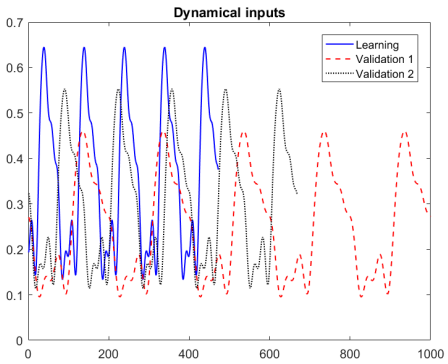


Figure 2.6 – Learning and validation 1 and 2 massflow rates

We do not know if the flow starts from the same situation as in the learning case. So we need as a first step to identify the initial condition x_0. In both validation case, this is done using the 50 first time steps of the validation flow. Then we can really predict what is happening without any help from the validation data.
Each validation flow is projected in the reduced space from the learning phase the uses Φ_{r_o}

$$y_v = \Phi_{r_o}^T U_v, \tag{2.6.3}$$

In the first validation case the projection error is 0.0250. In the second one it is 0.0212. As the vector in Φ_{r_o} are othonormal we know that if the results are good in the small space, it will be so in the large one.

Validation 1 We can observe the results for the first validation case in Figure 2.7.

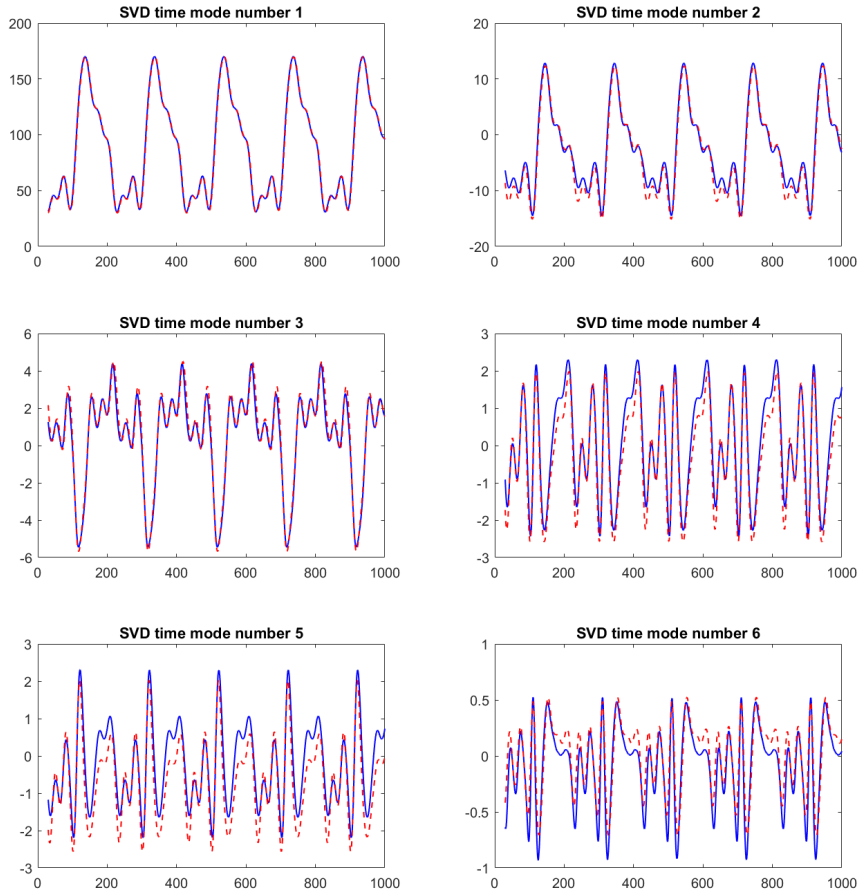


Figure 2.7 – Validation 1 - Blue line is the reference and dashed red line is the ROM

The relative error for each time coefficient is depicted in the table below

Table 2.2 – Validation 1 - Relative error for the SVD time coefficients

	1	2	3	4	5	6
L^2 relative error	0.0236	0.0120	0.0036	0.0063	0.0056	0.0108
L^∞ relative error	0.0599	0.0832	0.0373	0.0366	0.0349	0.0747

The global relative L^2 error is 0.0312.

Validation 2 We can observe the quality of the results for the second validation case in Figure 2.7.

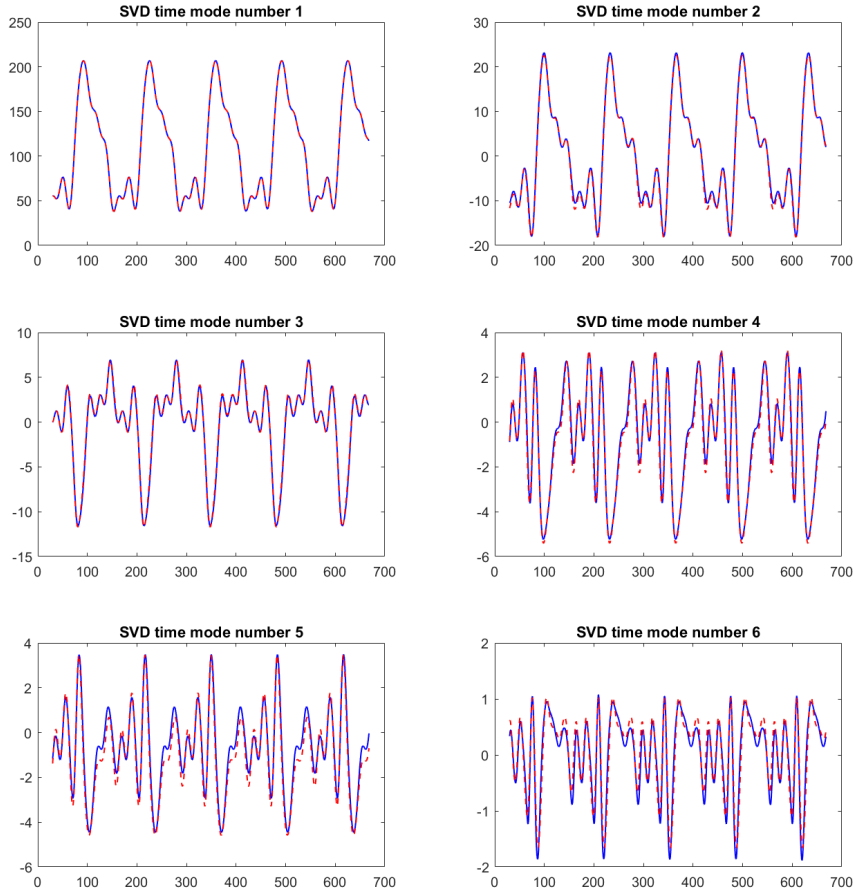


Figure 2.8 – Validation 2 - Blue line is the reference and dashed red line is the ROM

The relative error for each time coefficient is depicted in the table below

Table 2.3 – Validation 1 - Relative error for the SVD time coefficients

	1	2	3	4	5	6
L^2 relative error	0.0264	0.0094	0.0047	0.0050	0.0058	0.0078
L^∞ relative error	0.3907	0.0835	0.0489	0.0276	0.0548	0.0582

The global relative L^2 error is 0.0364.
Overall the model proves to be reliable for this test case. But it would be interesting to get a more complicated dynamical input in the learning phase in order to test the ROM for validation data involving frequencies and magnitudes in a wider span.
Now we would like to highlight the importance of the quadratic term in the model. To do so we can test how the ROM performs without the nonlinear term as a comparison.

2.6.1.3 Learning without the quadratic term

We reproduce the exact same experiment. Unless this time we remove the quadratic term in the model

$$W(x) = Ax. \tag{2.6.4}$$

To consider the best linear model we performed several model creations for different size r_d of the dynamical part. And we kept the one giving the best validation results. That happens for $r_d = 6$.

Learning The results of the learning phase is shown in Figure 2.9.

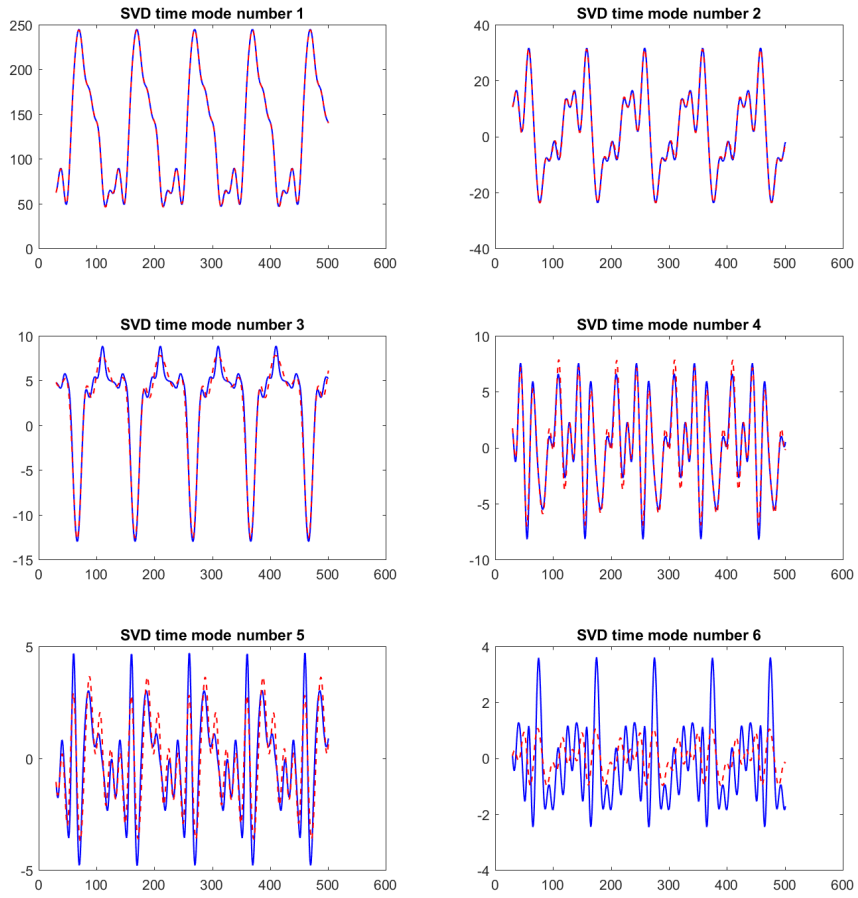


Figure 2.9 – Learning - Blue line is the reference and dashed red line is the ROM

The global relative L^2 learning error is 0.0136.

Validation 1 The results for the first validation set are shown in Figure 2.10.

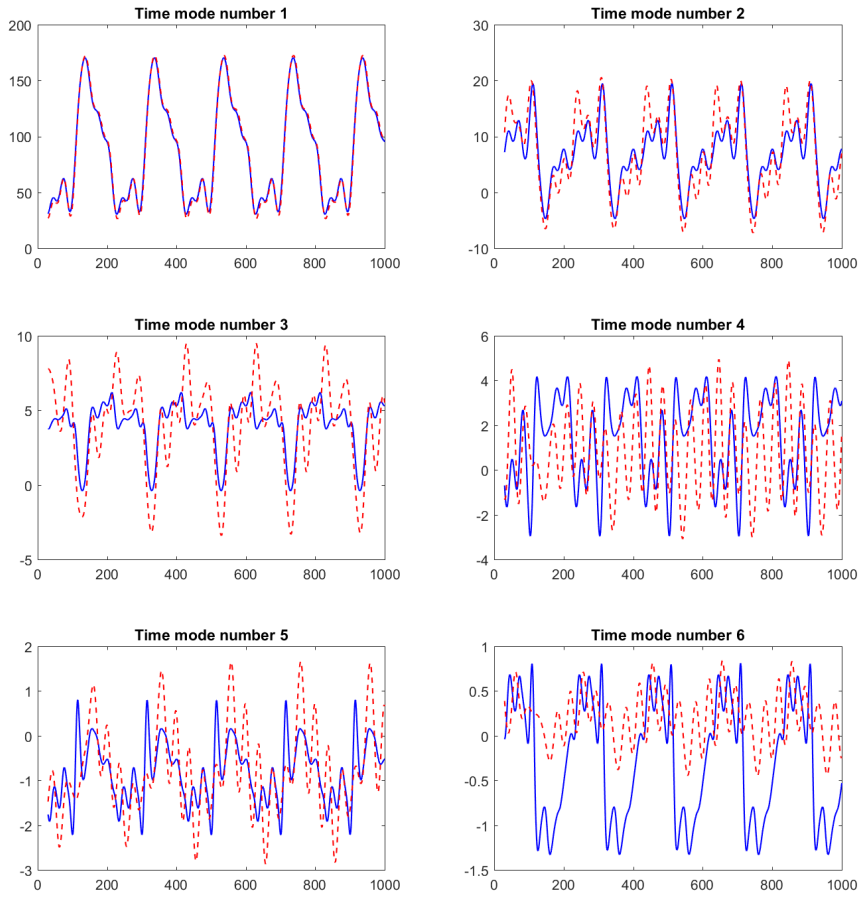


Figure 2.10 – Validation 1 - Blue line is the reference and dashed red line is the ROM

The global relative L^2 error is 0.0579.

Validation 2 The results for the second validation set are shown in Figure 2.11.

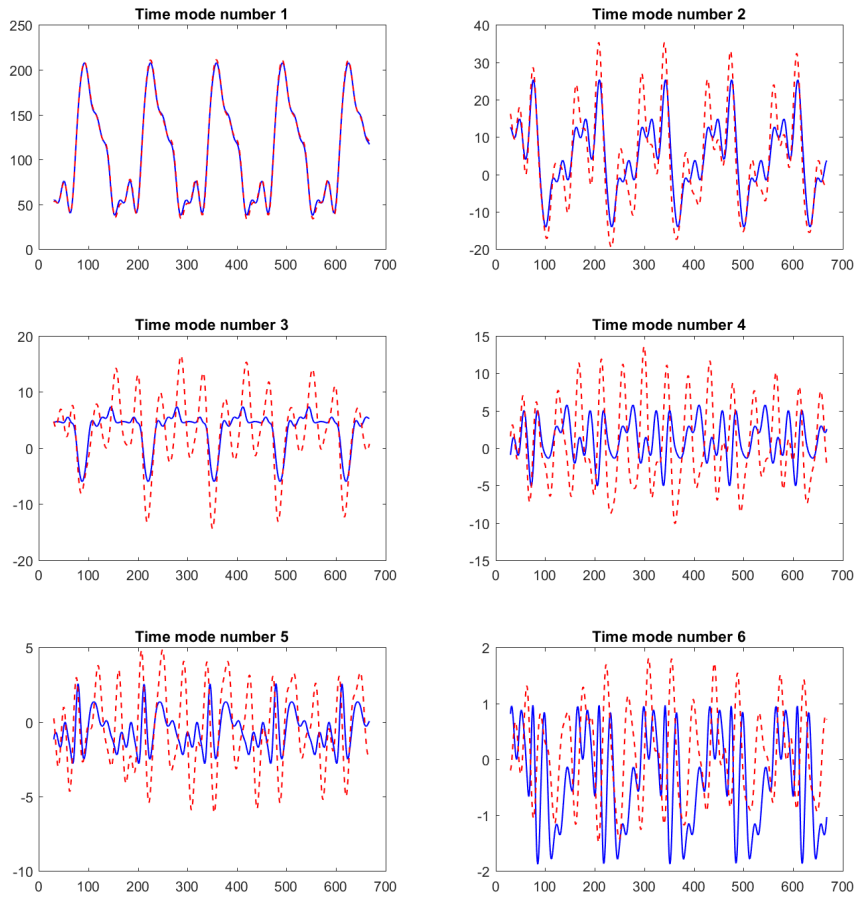


Figure 2.11 – Validation 2 - Blue line is the reference and dashed red line is the ROM

The global relative L^2 error is 0.0906.
We observe that the validation error is not so bad. This is mainly due to the fact that the first time coefficient, that is the most significant, seems to be well learned and predicted. However we can see that the smaller coefficients are way better learned and predicted with the quadratic version. Which proves the importance of taking into account this nonlinearity. It will certainly be even more significant with validation data that differ more from the learning case.

2.7 Modeling the pressure

In the coronary test case, the data we got from the CFD solver include, besides the flow, the pressure field. Let us look for a way to learn and predict this pressure field. We are going to take advantage of the previous reduced order model of the fluid flow. As for the ROM of the flow, we are starting from the PDE of the pressure to make a ROM emerge.

2.7.1 Mixed boundary conditions

Applying the divergence operator to equation 2.1.1 and using the boundary conditions from 2.1.1 for u on Γ_i, $i > 0$, we get the following system of equations:

$$\begin{cases} \Delta p = \nabla \cdot \left(f - \dfrac{\partial u}{\partial t} - (u \cdot \nabla) u + \nu \nabla^2 u \right), & \text{on} \quad \Omega, \\ p = \nu \dfrac{\partial u}{\partial n} + \dfrac{1}{2} (u \cdot n)\, u - p_{\text{ext}}, & \text{on} \quad \Gamma_i, \quad 1 \le i, \\ \dfrac{\partial p}{\partial n} = \left(f - \dfrac{\partial u}{\partial t} - (u \cdot \nabla) u + \nu \nabla^2 u \right) \cdot n, & \text{on} \quad \Gamma_0, \end{cases} \tag{2.7.1}$$

From $\nabla \cdot u = 0$ on Ω and $u = 0$ on Γ_0, we can get rid of the time derivatives and the system simplifies into

$$\begin{cases} \Delta p = \nabla \cdot \left(f - (u \cdot \nabla) u + \nu \nabla^2 u \right), & \text{on} \quad \Omega, \\ p = \nu \dfrac{\partial u}{\partial n} + \dfrac{1}{2} (u \cdot n)\, u - p_{\text{ext}}, & \text{on} \quad \Gamma_i, \quad 1 \le i, \\ \dfrac{\partial p}{\partial n} = \left(f - (u \cdot \nabla) u + \nu \nabla^2 u \right) \cdot n, & \text{on} \quad \Gamma_0, \end{cases} \tag{2.7.2}$$

Let p_p be a particular solution of 2.7.2.
In order to write the weak form of 2.7.2, let us define

$$V = \left\{ q \in H^1(\Omega) \quad | \quad q = 0 \quad \text{on} \quad \Gamma_i, \quad 1 \le i \right\}. \tag{2.7.3}$$

and

$$U = V + p_p, \tag{2.7.4}$$

We look for $p \in U$ so that

$$\int_\Omega \Delta p q = \int_\Omega \nabla \cdot \left(f - (u \cdot \nabla) u + \nu \nabla^2 u \right) q, \quad \text{on} \quad \Omega, \quad \forall q \in V. \tag{2.7.5}$$

Integrating by parts we have

$$\begin{aligned}\int_\Omega \Delta p q &= -\int_\Omega \nabla p \nabla q + \int_{\Gamma_0} \frac{\partial p}{\partial n} q \\ &= -\int_\Omega \nabla p \nabla q + \int_{\Gamma_0} \left(\left(f - (u\cdot\nabla)u + \nu\nabla^2 u \right) \cdot n \right) q\end{aligned} \tag{2.7.6}$$

and

$$\begin{aligned}\int_\Omega \nabla \cdot \left(f - (u\cdot\nabla)u + \nu\nabla^2 u \right) q = &-\int_\Omega \left(f - (u\cdot\nabla)u + \nu\nabla^2 u \right) \nabla q \\ &+ \int_{\Gamma_0} \left(\left(f - (u\cdot\nabla)u + \nu\nabla^2 u \right) \cdot n \right) q\end{aligned} \tag{2.7.7}$$

Hence, equation 2.7.5 is equivalent to

$$\int_\Omega \nabla p \nabla q = \int_\Omega \left(f - (u\cdot\nabla)u + \nu\nabla^2 u \right) \nabla q. \tag{2.7.8}$$

This last equation seems convenient. However, some difficulties emerge when handling the particular solution p_p. Indeed, we would need to define a particular solution that depends gently on u. At this stage it seems complicated. Furthermore, we cannot easily find a suitable basis for V, guaranteeing the homogeneous Dirichlet condition, from the learning data. We would need to get further information for the field (the location of each element) which would jeopardize the simple use of the SVD on the learning data.

2.7.2 Full Neumann boundary conditions

We can set the problem in a slightly different manner to make appear a ROM structure more naturally. We can use the Neumann boundary conditions on every Γ_i, $i \geq 0$ (which is possible from 2.1.1):

$$\begin{cases} \Delta p = \nabla \cdot \left(f - \dfrac{\partial u}{\partial t} - (u\cdot\nabla)u + \nu\nabla^2 u \right), & \text{on} \quad \Omega, \\ \dfrac{\partial p}{\partial n} = \left(f - \dfrac{\partial u}{\partial t} - (u\cdot\nabla)u + \nu\nabla^2 u \right) \cdot n, & \text{on} \quad \Gamma_i, \quad i \geq 0, \end{cases} \tag{2.7.9}$$

This time we still can get rid of $\frac{\partial u}{\partial t}$ in the first line of the system thanks to the $\nabla \cdot u = 0$, but we cannot do the same in the second line as u is not equal to zero

on Γ_i when $i > 0$:

$$\begin{cases} \Delta p = \nabla \cdot \left(f - (u \cdot \nabla) u + \nu \nabla^2 u \right), & \text{on} \quad \Omega, \\ \dfrac{\partial p}{\partial n} = \left(f - \dfrac{\partial u}{\partial t} - (u \cdot \nabla) u + \nu \nabla^2 u \right) \cdot n, & \text{on} \quad \Gamma_i, \quad i \geq 0, \end{cases} \tag{2.7.10}$$

Now let us define

$$V = \left\{ p \in H^1(\Omega) \quad | \quad \int_\Omega p = 0 \right\} \tag{2.7.11}$$

We can decompose $H^1(\Omega)$ as the direct sum of two subspaces:

$$H^1(\Omega) = V \bigoplus \text{span}(1) \tag{2.7.12}$$

The pressure p can be decomposed into the sum of a function of V and a uniform function e

$$p = p_0 + e, \quad p_0 \in V. \tag{2.7.13}$$

Equation 2.7.10 is well posed in V.
After having integrated by parts the weak formulation of 2.7.10, we look for $p_0 \in V$ that satisfies

$$\int_\Omega \nabla p_0 \nabla q = \int_\Omega \left(f - (u \cdot \nabla) u + \nu \nabla^2 u \right) \nabla q + \sum_{i>0} \int_{\Gamma_i} \left(\frac{\partial u}{\partial t} \cdot n \right) q, \quad \text{on} \quad \Omega. \tag{2.7.14}$$

With this equation, p_0 is well defined. Its dependency with respect to u is linear and quadratic, and its dependency with respect to $\frac{\partial u}{\partial t}$ is linear.
We can now determine e thanks to the Dirichlet condition on Γ_i, $i > 0$. For example if

$$p = \nu \frac{\partial u}{\partial n} + \frac{1}{2} (u \cdot n) u - p_{\text{ext}}, \quad \text{on} \quad \Gamma_i, \quad i > 0, \tag{2.7.15}$$

we have

$$e = \frac{\sum_{i>0} \int_{\Gamma_i} \left(\nu \frac{\partial u}{\partial n} + \frac{1}{2} (u \cdot n) u - p_{\text{ext}} \right) - p_0}{\sum_{i>0} \int_{\Gamma_i} 1}. \tag{2.7.16}$$

So e depends in a linear and quadratic way on u.
From this, we cans state that the pressure p only depends on the fluid flow u (in a linear and quadratic way), its time derivative (linearly), and on the dynamical inputs f and p_{ext}. We are going build a static reduced order model of p based

on this result.
Projecting u and p on reduced bases we get

$$u \approx \sum_{i=1}^{r_u} x_{ui}\varphi_i, \tag{2.7.17}$$

$$f \approx \sum_{i=1}^{r_u} b_{f_i}\varphi_i, \tag{2.7.18}$$

$$p_{\text{ext}} \approx \sum_{i=1}^{r_u} b_{p_i}(\varphi_k \cdot n). \tag{2.7.19}$$

and

$$p \approx \sum_{i=1}^{r_p} x_{p_i}\psi_i, \tag{2.7.20}$$

The projection on this reduced basis leads therefore to the following model of the pressure

$$x_p = A_u x_u + q^N(x_u) + A_{\dot{u}}\dot{x}_u + B_f b_f + B_p b_p + \epsilon, \tag{2.7.21}$$

where A_u, $A_{\dot{u}}$, B_f and B_p are in $\mathbb{R}^{\times}$, and ϵ is just a scalar.
The quadratic term q^N makes the link between the flow and the pressure as follows

$$q_k^N(x) = \sum_{k=1}^{n} x^t N_k x \tag{2.7.22}$$

where N_k is $r_u \times r_u \times r_p$.
Here the hypermatrix N is not cubic and there is no stability constraint for its coefficients contrary to what is done for the dynamical part in section 2.3. Indeed, there is no stability constraint to watch for, the nonlinear term in 2.7.21 does not imply clear energy properties and that is not a problem as it only plays a static role. Nevertheless we keep reducing the number of non-trivial elements of N in order to avoid that two different N^1 and N^2 lead to the same application (as done in section 2.3).

2.7.3 Experimental results: back to the coronary test case

To test this ROM, let us go back to the coronary test case.
The whole model is then

$$\begin{aligned} \left(x_u^{i+1} - x_u^i\right) &= W\left(x_u^{i+\frac{1}{2}}\right) + Bb^{i+\frac{1}{2}} + e \\ x_p^{i+1} &= A_1 x_u^{i+1} + A_0 x_u^i + q^N(x_u^{i+1}) + Bb^{i+1} + \epsilon. \end{aligned} \tag{2.7.23}$$

The upper part of the model is already identified by the previous experiment. The lower part is the one that is going to be identified from learning here. It has been marginally modified compared to 2.7.21 in order to adapt to the discrete time decomposition. The time derivative does not appear explicitly. Instead, two time steps are used in the linear part. Hence A_0 and A_1 play together the role of A_u and $A_{.u}$. The scheme is not centered, but this has no major impact as we are not building a dynamical model here but rather post-processing the flow x_u to model the pressure x_p.
The only unknowns to be identified are N, A_1, A_0, B and ϵ.
The experiment is performed following the same steps as in section 2.6.1. The learning and validation dynamical inputs are the same as earlier and can again be observed in Figure 2.6.

2.7.3.1 Preprocessing the learning data

As a first step we perform a SVD of the learning pressure field. The singular values behavior is shown in Figure 2.12.

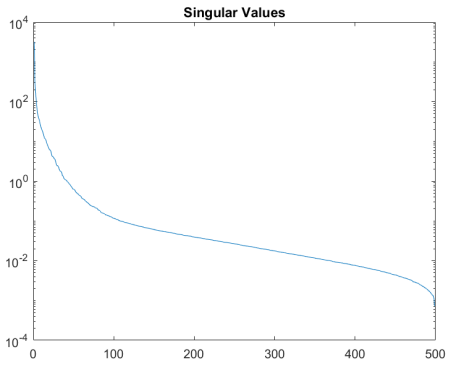


Figure 2.12 – Singular values for the learning pressure field (logarithmic scale)

We choose to keep only the first $r_o = 4$ modes from the SVD. The projection error is 0.0305. The SVD time coefficients are used as the x_p from 2.7.23.

2.7.3.2 Learning phase

Let us remind that the number of fluid coefficients r_d from previous section is 6.
The quality of the learning is shown in Figure 2.13.

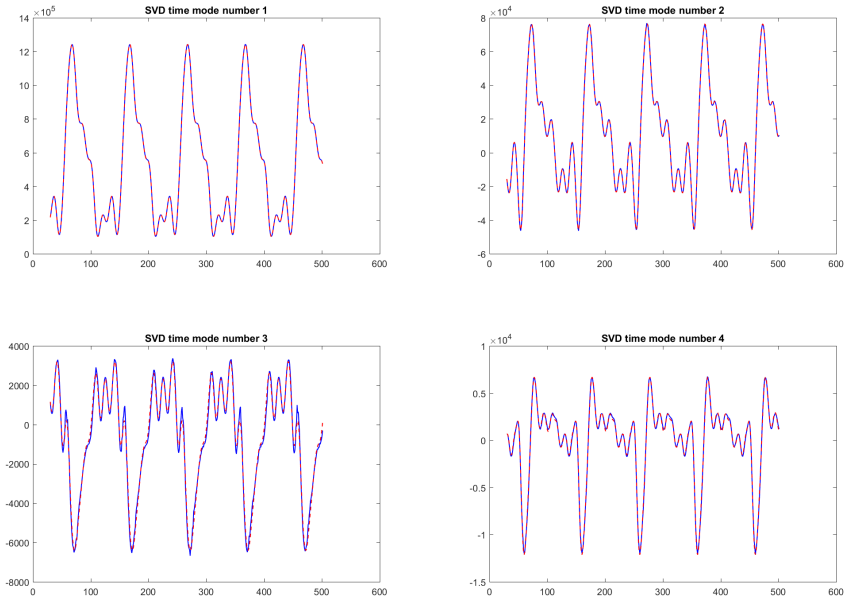


Figure 2.13 – Learning - Blue line is the reference and dashed red line is the ROM.

The relative error for each time coefficient is depicted in the table below

Table 2.4 – Learning - Relative error for the SVD time coefficients

	1	2	3	4
L^2 relative error	0.0019	0.0082	0.0038	0.0026
L^∞ relative error	0.0065	0.0129	0.0054	0.0035

In the first validation case the projection error is 0.0312. In the second one it is 0.0289.

2.7.3.3 Validation

Validation 1 We can observe the quality of the first validation case in Figure 2.14.

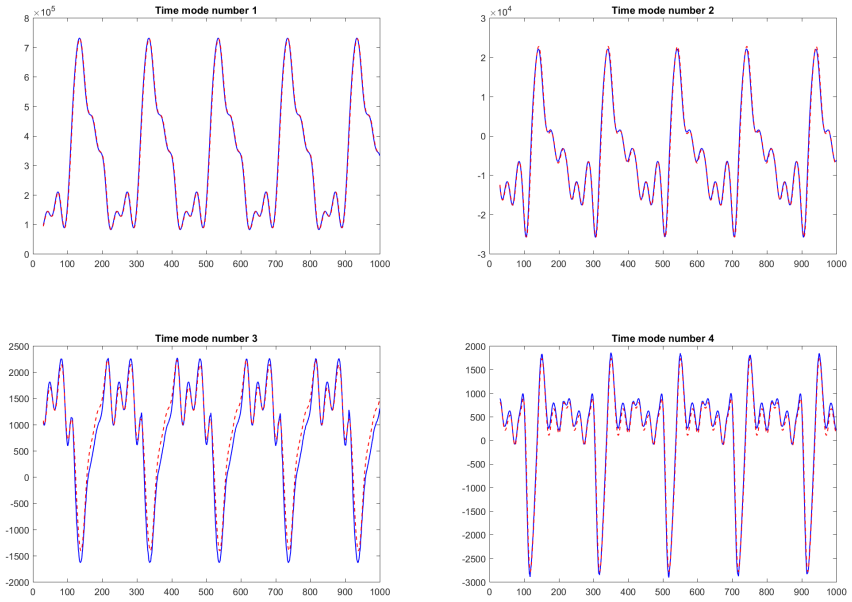


Figure 2.14 – Validation 1 - Blue line is the reference and dashed red line is the ROM.

The relative error for each time coefficient is depicted in the table below

Table 2.5 – Validation 1 - Relative error for the SVD time coefficients

	1	2	3	4
L^2 relative error	0.0261	0.0504	0.0082	0.0024
L^∞ relative error	0.0357	0.0854	0.0391	0.0048

Validation 2 We can observe the quality of the first validation case in Figure 2.15.

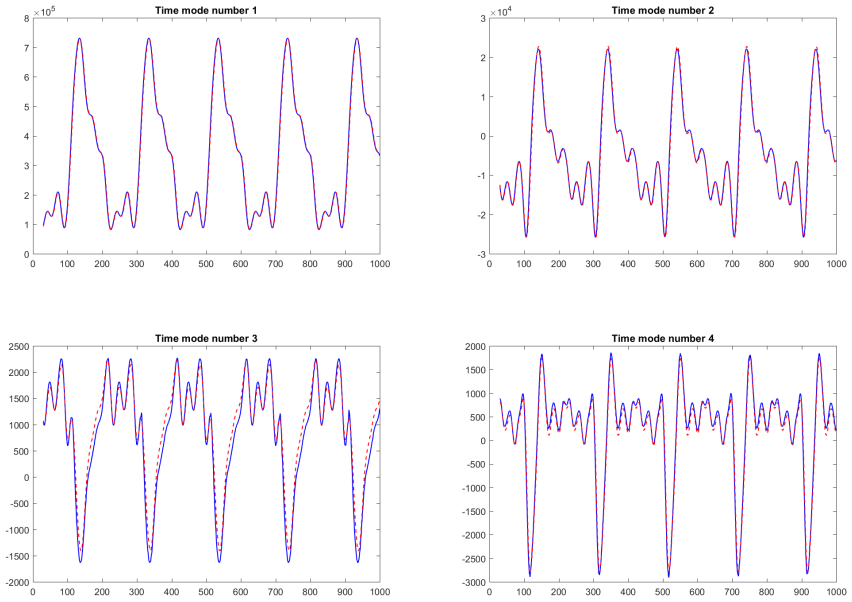


Figure 2.15 – Validation 2 - Blue line is the reference and dashed red line is the ROM.

The relative error for each time coefficient is depicted in the table below

Table 2.6 – Validation 2 - Relative error for the SVD time coefficients

	1	2	3	4
L^2 relative error	0.0261	0.0504	0.0082	0.0024
L^∞ relative error	0.0357	0.0854	0.0391	0.0048

As a result we observe that this ROM is quite efficient to recover a global behavior for the pressure. In the future we would like to be able to build the model relying only on a learning pressure field. That means being able to build at the same time a compatible dynamical model for the fluid and a fluid-to-pressure model that make it possible to fit the targeted pressure. Unfortunately, the data we have been provided here are not rich enough to attempt that. Especially because the learning dynamical input does not provide enough information.

2.8 Conclusion and perspectives

We have seen thanks to the coronary test case that dynaROM can give satisfactory results on experiments coming from CFD computations.
However the most important feature with this ROM is that it only requires learning data to be created. Therefore it can be considered as part of the world of machine learning. Measured data can be used to build a stable ROM.
Unfortunately, we did not get access to appropriate data of this kind. The only opportunity we got to experiment building a ROM with measures was in the field of weather forecasting. Adagos has been working with a start-up named Wezr on short-term weather forecasting using artificial neural networks. Besides that, we decided to try to use dynaROM to make a self-consistent model of the wind measured by weather stations in the region of the San Francisco Bay. A document presenting the general work we have done in the field of weather forecasting and more specifically the experiment using dynaROM can be found at www.adagos.com/romweather201805111708. However the use of dynaROM in this particular case is a bit frustrating as there is no dynamical excitation. DynaROM is only used as a data assimilation tool where the initial condition x^0 is tuned using the latest measures to get a new prediction.
We look forward to getting a more suitable test case involving measured data and dynamical excitations.

Chapter 3

A contribution to sparse grids

Sparse grids (SPG) are used as part of a hierchical numerical method of function approximation. They were originally developed by Sergey A. Smolyak in [24].
They are made to approximate high dimensional functions. They naturally offer the possibility to build step by step a design of experiment (i.e. the learning set of data).
In this chapter we will first introduce the key principles of state of the art SPG techniques in section 3.1. We will also present the alternative choice of basis functions we have made.
Then we will study how SPG methods perform compared to a reference method that is Kriging 3.2.
Finally, we will introduce a way to enhance the dimensional adaptavity strategy that features in the SPG method 3.3. The efficiency of this method will be illustrated with an industrial experiment in the field of nuclear energy.
We will also say one word about the fact that SPG method can be adapated to zROM 1, the reduced order model for structural analysis, in order to release the latter as a product.

3.1 How do they work?

The goal is to build an estimate g of computationally intensive function G. As it stands there is no prior information on the model.
That method is particularly interesting in high dimensional problems.

3.1.1 The one dimensional case

Before getting into more complex material it is instructive to have an insight into the 1D case in order to get familiar with the mathematical notations.
The reference function is $G : [0,1] \mapsto \mathbb{R}$ and the approximate is $g : [0,1] \mapsto \mathbb{R}$.
Let $X^i = \left\{ x_j^i \in [0,1], j = 1,2,\ldots,m_i \right\}$ be the set of nodes at level i, then $X^i \subset X^{i+1}$.
The level i interpolation of G is

$$g_i(G) = \sum_{x_j^i \in X^i} a_j^i G(x_j^i),$$

where $\left\{ a_j^i \in C([0,1]), j = 1,2,\ldots,m_i \right\}$ are the basis functions subject to:

- $\forall\, j \in 1,\ldots,m_i, a_j^i(x_j^i) = 1$,
- $\forall (j,k) \in 1,\ldots,{m_i}^2, j \neq k, a_j^i(x_k^i) = 0.$

Let Δ_i be the difference level i and level $i-1$ interpolations:

$$\Delta_i = \sum_{x_j^i \in X_\Delta^i} a_j^i (G(x_j^i) - g_{i-1}(x_j^i)) = \sum_{x_j^i \in X_\Delta^i} a_j^i w_j^i,$$

where,

- $X_\Delta^i = X^i \; X^{i-1}$ is the set added nodes to level $i-1$,
- $w_j^i = G(x_j^i) - g_{i-1}(x_j^i)$ is called the j^{th} surplus and is computed by evaluating the difference between G and g_{i-1} at x_j^i.

Therefore, to compute the level i interpolation we just need to compute w_j^i.

3.1.1.1 Usual basis functions

There are two main versions of SPG techniques. The first one uses a Clenshaw-Curtis grid and piece-wise linear basis functions. The second one relies on a Chebyshev grid and polynomial basis functions.
The Clenshaw-Curtis grid is the simplest:

$$x_j^i = \begin{cases} 0.5 & \text{for} \quad j = 1, m_i = 1 \\ \frac{j-1}{m_i - 1} & \text{for} \quad j = 1,\ldots,m_i, m_i > 1 \end{cases}$$

and

$$m_i = \begin{cases} 1 & \text{for} \quad i = 1 \\ 2^{i-1}+1 & \text{for} \quad j > 1 \end{cases}$$

Generally this type of grid is paired with piece-wise linear basis functions

$$a_1^1 = 1,$$

$$a_j^i(x) = \begin{cases} 1-(m_i-1)|x-x_j^i| & \text{if} \quad |x-x_j| < \frac{1}{m_i-1}, \\ 0. \end{cases}$$

The basis functions from level 0 to 2 are pictured in Figure 3.1.

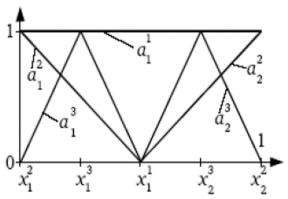


Figure 3.1 – Piece-wise linear basis functions

The second major option is to use the Chebyshev grid. It is the way to go when we want to deal with polynomial basis functions. The reason is that it allows to avoid the Runge's phenomenon [26]. The grid is then

$$x_j^i = \begin{cases} 0.5 & \text{for} \quad j = 1, m_i = 1 \\ \frac{1}{2}\left(1 - cos\left(\pi \frac{j-1}{m_i-1}\right)\right) & \text{for} \quad j = 1, \ldots, m_i, m_i > 1 \end{cases}$$

$$m_i = \begin{cases} 1 & \text{for} \quad i = 1 \\ 2^{i-1}+1 & \text{for} \quad j > 1 \end{cases}$$

And the basis functions come from Lagrangian polynomials

$$a_j^i(x) = \left\{ \prod_{k=1,k\neq j}^{m} \frac{x-x_k^i}{x_j^i - x_k^i} \text{ for } i > 1, j = 1, \ldots, m_i. \right.$$

Those functions are pictured in 3.2.

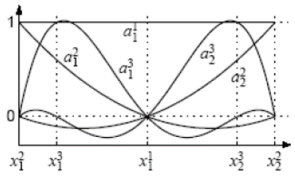


Figure 3.2 – Lagrangian polynomials

3.1.2 Multiple Dimensions

At this point we did not get the chance yet to observe the benefits of a sparse grid. To do so we need to examine the multidimensional case. We now consider the reference function $G : [0,1]^d \mapsto (R)$ and the sparse grid approximate g.
For $k = 1, ..., d$, let X^{i_k} be a set of nodes at level i_k.
Using tensor product, the interpolation formula over $X = \prod_{k=1}^{d} X^{i_k}$ is

$$g(i_1, \ldots, i_d)(G) = \sum_{x_{j_1}^{i_1} \in X^{l^1}} \cdots \sum_{x_{j_d}^{i_d} \in X^{l^d}} G\left(x_{j_1}^{i_1}, \ldots, x_{j_d}^{i_d}\right)\left(a_{j_1}^{i_1} \otimes a_{j_2}^{i_2} \otimes \ldots a_{j_d}^{i_d}\right)$$

The sparse grid relies on the fact that we do not compute every possible tensor product, but only the ones such that

$$|k|_1 = \sum_{i=1}^{d} k_i \leq d + N - 1,$$

with $k = (k_1, \ldots, k_d)$.

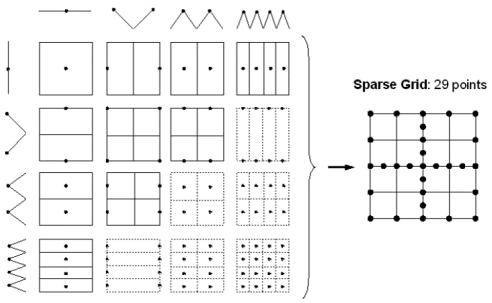


Figure 3.3 – A sparse grid

Table 3.1 – Number of grid points depending on the method

Method	dim=2	dim=10	dim = 100
Plain Grid	$5^2 = 25$	$5^10 \approx 10^7$	$5^100 \approx 7 \times 10^9$
Sparse Grid	13	221	20201

The sparse grid method is a way to overcome the so called curse of dimensionality. The accuracy obtained with piecewise linear basis functions, with respect to the L_2 and L_∞ norms, in the case the solution has bounded second order mixed derivatives is mentioned in [25]. These results are presented in Table 3.2. We can observe from the accuracy behavior with respect to d the benefits of sparse grids compared to plain ones.

Table 3.2 – Number of grid points depending on the method and the dimension and the corresponding accuracy d

Level n (let $N = 2^n + 1$)	Number of points	Error $\|G - g\|_p$, $p = 2$ or ∞.
Plain Grid	N^d	$\mathcal{O}(N^{-2})$
Sparse Grid (Clenshaw-Curtis)	$\mathcal{O}(Nlog(N)^{d-1})$	$\mathcal{O}(N^{-2}\|log_2(N)\|^{3(d-1)})$

3.1.3 Error estimator

Sparse grid methods rely on their hierarchical structure. The procedure is to add gradually the grids by analyzing error estimators.
Those estimators are

- $e^l_{abs} = \max_{|i|_1=d+l} ||w^i_j||_\infty$,
- $e^l_{rel} = \frac{e^l_{abs}}{M-m}$,

where $M = \max_{|i|_1=d+l,j} G(X^i_j)$ and $m = \min_{|i|_1=d+l,j} G(X^i_j)$.
Thanks to these two quantities we can estimate what the error may be at every time a level is added. Thus we can target a given accuracy.

3.1.4 Dimensional adaptativity

From the error estimators we can select which level (i.e. sub-grid) is the most interesting to add. This allows the model to favor the most significant inputs.

It is very convenient to fight the *curse of dimensionality*. When dealing with a problem involving real physics and a large set of input parameters, we can often observe that a large number of the latest are not playing a major role. The Sparse Grid techniques using the dimensional adaptativity allow the model to seek for the most efficient enrichment of the design of experiment and the corresponding basis functions.
In practice, in this work, we used the Matlab Sparse Grid Interpolation Toolbox developed by Andreas Klimke at the Institute of Applied Analysis and Numerical Simulation, at the High Performance Scientific Computing lab ("Lehrstuhl für Numerische Mathematik für Höchstleistungsrechner"), Universitat Stuttgart during his Ph.D. studies. The dimensional adaptivity strategy is precisely described in the corresponding reference [27].
Basically it relies on the degree of dimensional adaptativity degree r set a priori

$$r = \frac{n_{AdaptPoints}}{n_{TotalPoints}}, \tag{3.1.1}$$

where $n_{AdaptPoints}$ is the number of points added according to a purely adaptive refinement rule based on the local error estimations, and $n_{TotalPoints}$ denotes the total number of sparse grid points.

3.1.5 Alternative basis functions

Basis functions used in sparse grid method are usually piecewise linear functions or polynomials. They both have disadvantages. The first ones are not smooth and get very local as the level number increases in the enrichment strategy. The last ones are exposed to Runge's phenomenon, a problem of oscillation at the edge of an interval when interpolating a function with high degree polynomials.
To remedy that, we decided to use cubic splines. They are more regular and global than piecewise linear functions but allow to avoid the drawbacks of polynomials. Furthermore, they perform perfectly well on the classical and "healthy" Clenshaw-Curtis grid. The first levels of these new basis functions are displayed in Figure 3.4. They are very close to polynomials at this stage.

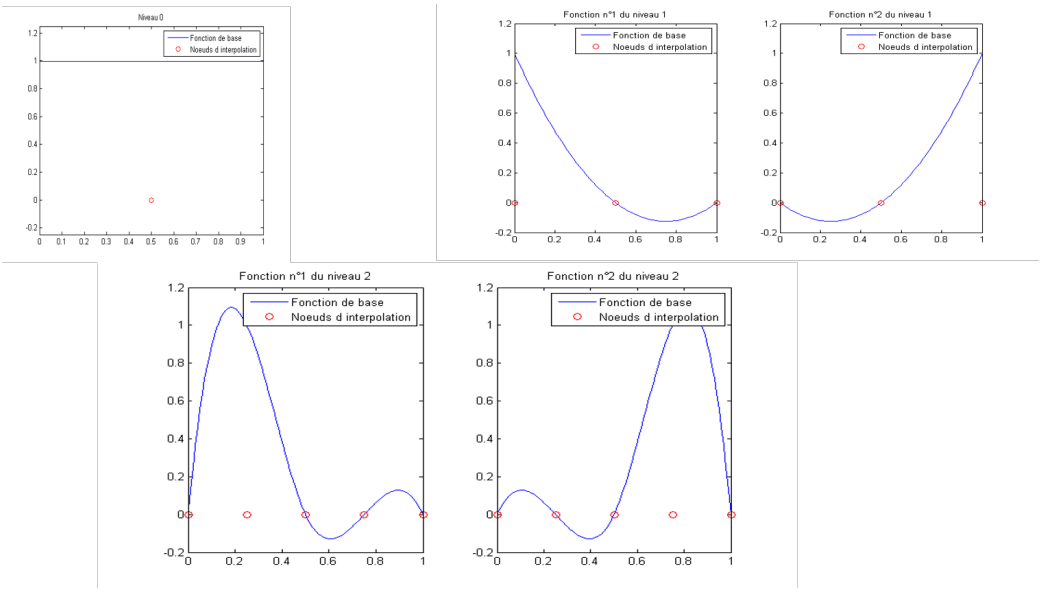


Figure 3.4 – Adagos basis functions

3.2 Comparison with Kriging

We have performed several tests to compare SPG methods to Kriging, an interpolation method that is very popular within the industrial world. For this purpose we are using two kinds of multivariate functions: the academic ones and the physical ones. The kriging toolbox we use is DACE [28].

3.2.1 Academic functions

Here are some academic functions that allow to test approximation methods. The first 3 ones have been suggested by our partner Areva.

- **G1** (d=4):

$$f(x_1,x_2,x_3,x_4) = 3x_1 + 10^{-6}x_2^3 + 2x_3 + \tfrac{2}{3}10^{-6}x_4^3, \quad \forall (x_1,x_2,x_3,x_4) \in \Omega,$$
$$\Omega = [0,0,-0.55,-0.55] \times [1200,1200,0.55,0.55] \tag{3.2.1}$$

- **G2** (d=7):

$$f(x_1\ldots,x_7) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 10x_6 - 8x_7, \\ \forall (x_1,\ldots,x_7) \in \Omega, \Omega = [-10,10]^7 \tag{3.2.2}$$

- **G3** (d=2):

$$f(x_1,x_2) = x_1^2 x_2^3, \quad \forall (x_1,x_2) \in \Omega, \\ \Omega = [-5.5,5.5]^2 \tag{3.2.3}$$

- **CarreDec** (d=N):

$$f(x_1,\ldots,x_N) = \sum_{i=1}^N (x_i-1)^4, \quad \forall (x_1,\ldots,x_N) \in \Omega, \\ \Omega = [-10,10]^N \tag{3.2.4}$$

- **DFunc** (d=2):

$$f(x_1,\ldots,x_N) = \sum_{i=1}^N |x_i - 0.5|, \quad \forall (x_1,\ldots,x_N) \in \Omega, \\ \Omega = [-10,10]^N \tag{3.2.5}$$

- **Rosenbrock** (d=2):

$$f(x_1,\ldots,x_N) = \sum_{i=1}^N \left(1 - x_i^2 + 1000\left(x_{i+1} - x_i^2\right)^2\right), \\ \forall (x_1,\ldots,x_N) \in \Omega, \Omega = [-5,10]^N \tag{3.2.6}$$

- **Rosenbrock anisotropic** (d=2):

$$f(x_1,\ldots,x_N) = \sum_{i=1}^N \left(1 - \left(\frac{2i}{N}x_i\right)^2 + 1000\left(\frac{2(i+1)}{N}x_{i+1} - \left(\frac{2i}{N}x_i\right)^2\right)^2\right), \\ \forall (x_1,\ldots,x_N) \in \Omega, \Omega = [-5,10]^N \tag{3.2.7}$$

- **Welch et al. function** (d=2):

$$f(x_1,\ldots,x_{20}) = \frac{5x_{12}}{1+x_1} + 5(x_4 - x_{20})^2 + x_5 + 40x_{19}^3 - 5x_{19} \\ +0.05x_2 + 0.008x_3 - 0.03 - 0.03x_7 - 0.09x_9 - 0.01x_{10} - \\ 0.07x_{11} + 0.25x_{13}^2 - 0.04x_{14} + 0.06x_{15} - 0.01x_{17} - 0.03x_{18}, \\ \forall (x_1,\ldots,x_20) \in \Omega, \Omega = [-0.5,0.5]^20 \tag{3.2.8}$$

3.2.2 Functions from physical problems

- **Wingweight** (d=10): The Wing Weight function models a light aircraft wing. The response is the wing's weight.

- **Otlcircuit** (d=6): The OTL Circuit function models an output transformerless push-pull circuit. The response Vm is midpoint voltage.

- **Piston** (d=7): The Piston Simulation function models the circular motion of a piston within a cylinder. It involves a chain of nonlinear functions. The response C is cycle time in seconds.

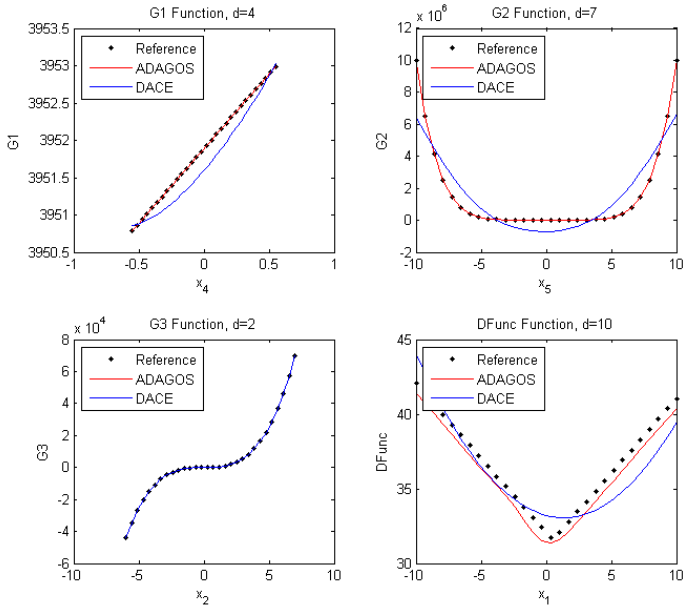
- **Borehole** (d=8): The Borehole function models water flow through a borehole. The response is water flow rate, in m^3/yr.

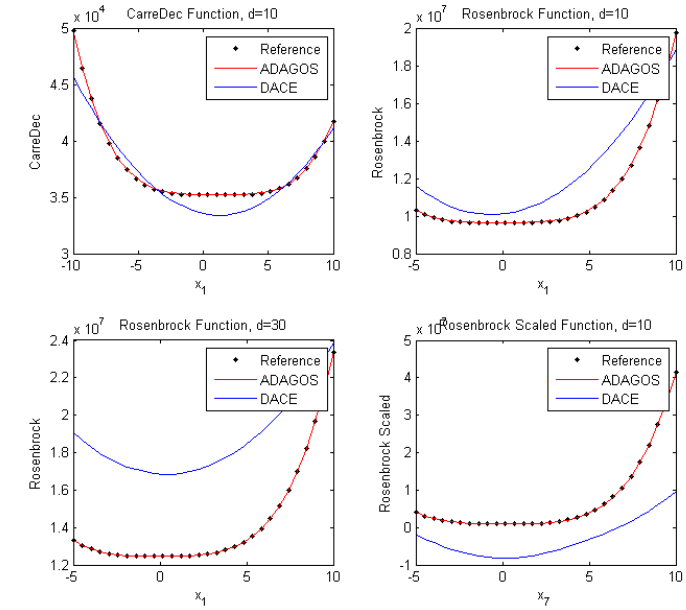
3.2.3 Results table

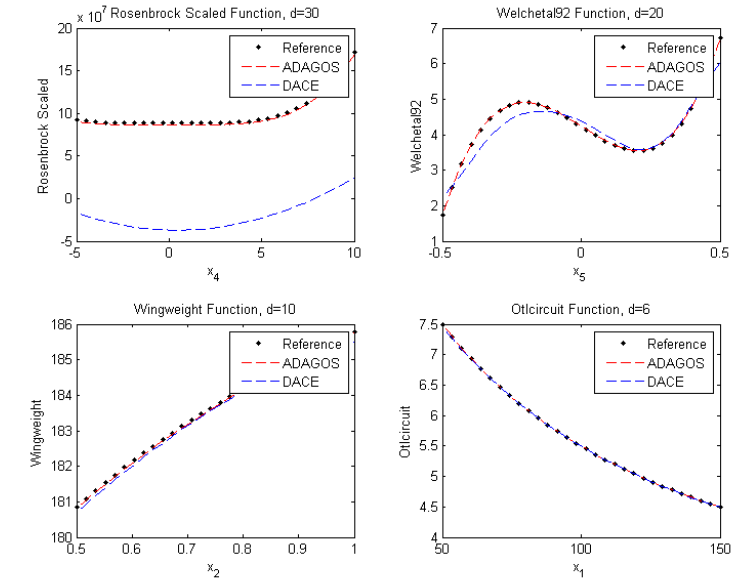
The results are displayed in Table 3.3. We can read the name of the function, the dimension number (Dim), the targeted tolerance (Tolerance), the number of points (Number of points), and the mean root mean squared error (RMSE) for both sparse grid (SPG) and kriging (DACE).
The number of point is set by the sparse grid method through its enrichment strategy to get to a given tolerance target. Then we give DACE (the Kriging toolbox) the same number of points of experiment in the form of a latin hypercube sampling, that is the design of experiment that suits Kriging best.

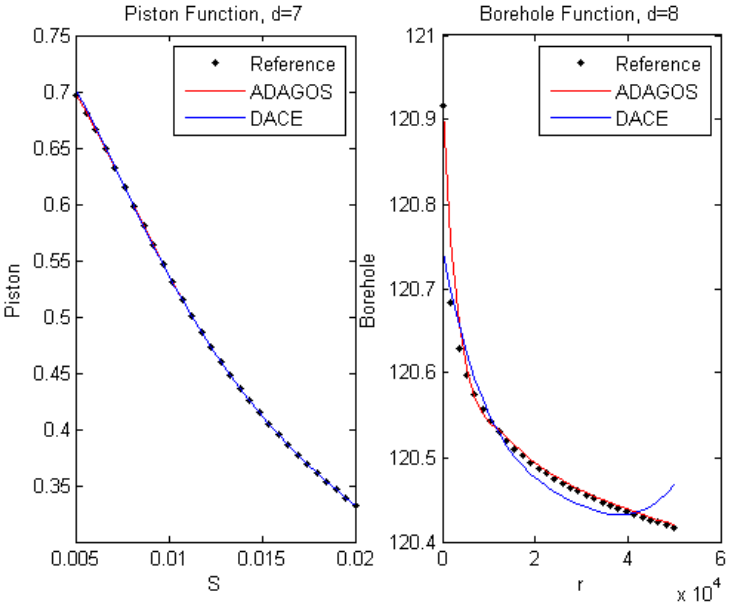
Function	Dim	Tolerance	Number of points	SPG (RMSE)	DACE (RMSE)
G1	4	0.1	17	1.24×10^{-11}	0.0286
G2	7	0.1	29	0.00212	1.05
G3	2	0.1	37	2.51×10^{-16}	0.0299
DFunc	10	0.1	261	0.00669	0.0562
CarreDec	10	0.1	261	0.0015	0.16
Rosenbrock	10	0.1	257	0.000848	0.18
Rosenbrock	30	0.1	1980	0.000676	0.116
Rosenbrock anisotropic	10	0.1	57	0.0222	0.323
Rosenbrock anisotropic	30	0.1	119	0.0467	0.568
Welchetal92	20	0.1	173	0.232	0.612
Welchetal92	20	0.01	557	0.0332	0.447
Wingweight	10	0.1	119	0.00211	0.00379
Wingweight	10	0.01	263	0.000628	0.00148
Wingweight	10	0.001	629	8.83×10^{-5}	0.000686
Otlcircuit	6	0.1	21	0.0116	0.0265
Otlcircuit	6	0.01	71	0.00125	0.00842
Otlcircuit	6	0.001	95	0.000178	0.00302
Piston	7	0.1	55	0.0897	0.0983
Piston	7	0.01	241	0.0096	0.0128
Piston	7	0.001	1460	0.00203	0.00261
Borehole	8	0.1	89	0.00899	0.0208
Borehole	8	0.01	113	0.00494	0.017
Borehole	8	0.001	251	0.000812	0.0034

Table 3.3 – Table of results


G1 Function, d=4
Reference
ADAGOS
DACE
G1
x4
x 10^6
G2 Function, d=7
G2
x5
x 10^4
G3 Function, d=2
G3
x2
DFunc Function, d=10
DFunc
x1


x 10^4
CarreDec Function, d=10
Reference
ADAGOS
DACE
CarreDec
x1
x 10^7
Rosenbrock Function, d=10
Rosenbrock
x 10^7
Rosenbrock Function, d=30
x 10^8
Rosenbrock Scaled Function, d=10
Rosenbrock Scaled
x7


x 10^7 Rosenbrock Scaled Function, d=30
Rosenbrock Scaled
x4
Welchetal92 Function, d=20
Welchetal92
x5
Wingweight Function, d=10
Wingweight
x2
Otlcircuit Function, d=6
Otlcircuit
x1
Reference
ADAGOS
DACE


Piston Function, d=7
Piston
S
Borehole Function, d=8
Borehole
r
x 10^4
Reference
ADAGOS
DACE

3.3 Enhanced dimensional adaptativity

While testing the benefits of SPG on high dimensional problems we found out that the built-in dimensional adaptativity strategy may have defects.
Usual sparse grid methods cannot interpolate a dimension d linear function with $\mathcal{O}(d)$ points.
Let us detail the first steps of the heuristic of the usual sparse grid toolbox [27] in the case of multivariate linear function. d denotes the dimension.

$$G(x) = b + a^T x, \quad x \in [-1,1]^d .$$

What happens during the building process if we assume that every a_k is greater than the absolute tolerance that is targeted (tol)? We are going to check this out step by step.

Level 0 The coefficient multiplying the constant function linked to the level 0 grid is $G(0) = b$.

Level 1 The set of points of sub-grids at level 1 are the ones such that $x_i = \pm 1$ and $x_k = 0, \forall k \in [\![1,d]\!] \backslash \{i\}$. In dimension 3 the grid is displayed in Figure 3.5.

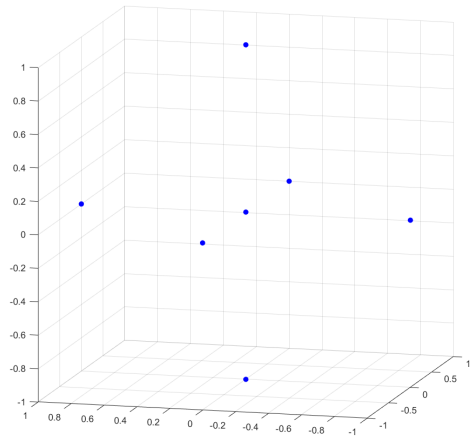


Figure 3.5 – 3d sparse grid at level 1

Level 2 At this point, all the error estimators are greater than the target tolerance, because of the assumption $tol \leq a_k$. Thus, no matter what the dimensional adaptativity degree r is, every sub-grid of level 2 is added according to the strategy implemented in the reference sparse grid toolbox [27]. In the case $d = 3$, the grid is displayed in Figure 3.6.

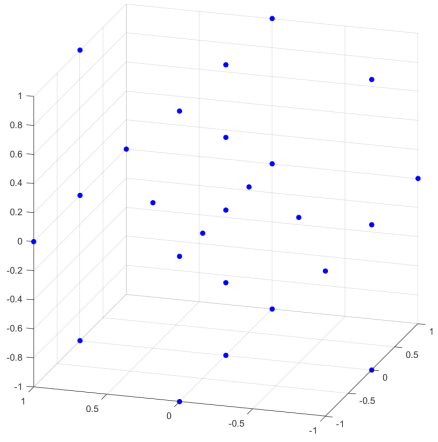


Figure 3.6 – 3d sparse grid at level 2

At this stage of the process, the number of points in the grid is

$$n_{points} = 1 + 4d + 2d(d-1).$$

This number is huge when the dimension grows. This is due to the term $2d(d-1)$. It corresponds to the introduction for the first time at this stage of *crossed levels*. These are the first sub-grids (with their corresponding basis functions) that make it possible for two separate inputs to interact. Every sub-grid contains 4 points and there are $\frac{d(d-1)}{2}$ of them (2 choose d).
If we take the example of $d = 100$, 20201 points are used to approximate G, which is huge.
The problem is that the approximation was already good at level 1. But the standard process was not able to find it out.
This problem is more general that it appears. When dealing with function coming from a physics problem involving a lot of variables (i.e. d is large), we may assume that many of them only have a small influence that can be approximate as linear, and their interactions may be negligible. We have to avoid adding unnecessary crossed levels involving those variables.
That is why we are suggesting alternative strategies.

3.3.1 Alternative strategies

At level 1 the approximation was already good. So we need to find way to know that. We imagined several ways to do that. The general principle is to assume that variables that only have a linear influence (when every other are set to zero) do not have significant interactions with remaining variables.

Method 1 The first implemented method relies on the introduction of a *linearity index* which quantifies how much the function seems to behave according to every dimension.
Let us focus on the k^{th} dimension and set every other input to 0. When getting to level 1, two surpluses, e^1 and e^2, are computed at x^1 and x^2 the points such that $x^1_k = -1$ and $x^2_k = 1$ (all others input being set to zero).

$$e^1 = G(x^1) - G(0),$$
$$e^2 = G(x^2) - G(0).$$

If $|e^1 + e^2| \leq \varepsilon$, meaning that e^1 and e^2 are close to be inverse additives (ε being chosen to be small), we may make the assumption that the behavior of the

function according to this dimension is linear. From that conclusion we decide to stop the enrichment involving this dimension.
Doing so for each and every dimension, we build a set of dimensions that we assume have at most a linear effect in the function's behavior. We call it D_{lin} . We will not add, at least initially, any crossed level involving dimensions in D_{lin}.
In the case of a linear function that means that there is no crossed level added. Therefore the approximation of G is achieved using only $2d+1$ learning points.

Method 2 In a similar manner we want to find out the "linear dimensions", i.e. the variables playing only a linear role. To do so we decide to go to level 2 for every dimension, but paying attention of not adding any crossed level. That process allows to validate or not the surrogate that was built at level 1. Every dimension along which the level 1 approximation is validated thanks to the partial level-2 points. At this stage we form D_{lin} and forbid the addition of crossed levels involving inputs among this set (in a first place at least).
In the case of a linear function the approximation of G is achieved using only $5d+1$ learning points.
This is far more robust than method 1, and the number of evaluations is of the same order.

3.3.2 A more general method

3.3.2.1 Gathering information as soon as possible

From the last example with the linear function, we observed that it is important to postpone the moment we add computationally intensive crossed levels in order to gather as much information as needed to make a decision.
We realized that, whatever the strategy about crossed levels is, some sub-grids (and therefore learning points) will be introduced anyway to satisfy the tolerance we target. Those grids can provide us with a great piece of information about the function.
Let us get into the details and consider $G : x \to G(x)$ where $x \in \mathbb{R}^d$. Whatever the adaptativity strategy is, to satisfy the tolerance criterion, we will reach a minimum level order along every single dimension (no crossed level involved at this stage). An example of this exploration is pictured in Figure 3.7.

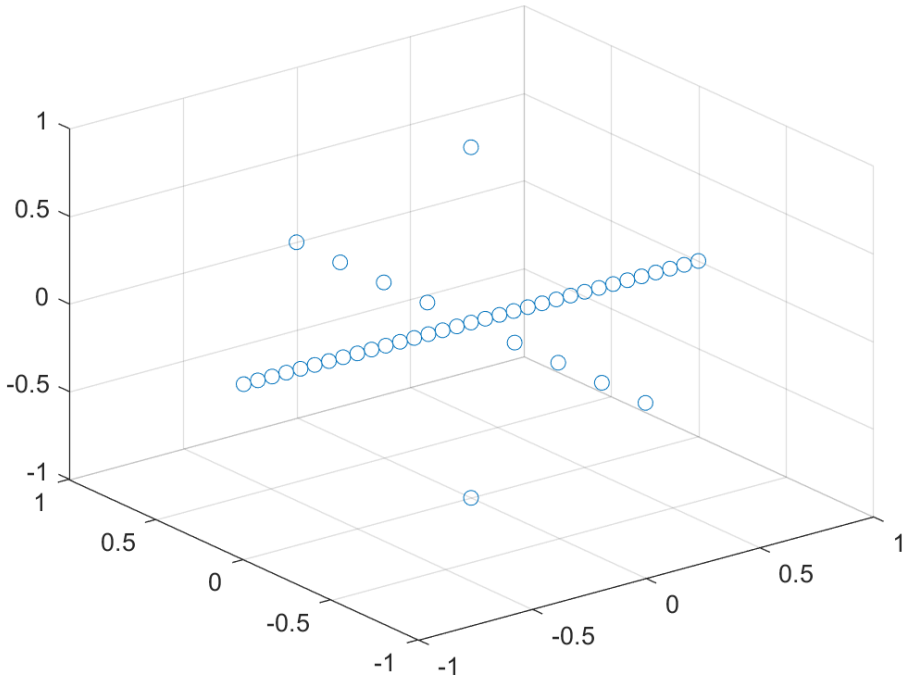


Figure 3.7 – Sparse grid without crossed level

The hypothetical crossed levels that could be candidate to be added have no impact on the 1D approximation of the function along every single dimension (every other inputs being set to 0 if the box is $[-1,1]^d$).
However, in the usual SPG method, if we have to reach the level 10 for the first dimension, we will first add all the candidate crossed levels of lower order before reaching it. But this is a shame as we should gather all the information we will have eventually anyway before making decisions about the exploration of the crossed levels.
Thus, with the strategy we propose here, we first achieve every 1D approximation. Then we can sort the variables according to their appearing contribution to the function and promote the most important ones. We will favor the addition of crossed levels between variables seeming to have a significant impact on the function.

We can adopt the same strategy at higher levels. For example as soon as we decide to introduce interactions between x_i and x_j, we can keep improving the approximation in this 2D subspace as long as the tolerance is not reached. We will use this new information to decide what crossed level involving 3 or more inputs (even more computationally intensive) should be added.
To sum up this is a way to get information, that will eventually be provided in any case, as soon as possible in order to make choices. This is a very different approach from usual sparse grid methods.

3.3.2.2 Looking for interactions

We want to find a way to quantify how strongly an input interacts with its peers for the function $G-g$, that is the difference between the reference function and the current model, and represents what remains to be learned.
Let us assume that, following the strategy described in the previous paragraph 3.3.2.1, we have already performed a satisfactory approximation along every dimension.
From then, we consider a new point of the space $c = [\pm 1, \dots, \pm 1]$, that is a corner of the $[-1,1]^d$ hypercube. Then we introduce a set of d new vectors deriving from c.

$$c^i \quad \text{s.t.} \quad \begin{cases} c^i_i &= 0 \quad , \\ c^i_k &= c_k \quad , \forall k \neq i \end{cases}$$

In practice, we set the i^{th} element of c^i to zero.
Then by computing

$$\gamma^k = \left| \left(G-g\right)(c) - \left(G-g\right)(c^k) \right|, \quad \forall k \in \{1, \dots, d\}.$$

and by sorting the γ^k we can get a good guess of the inputs that interact the most strongly with others. If γ^k is large we can bet that the interaction of the k^th input with others is strong. In fact, by setting the k^th input variable to zero, if the difference of $G-g$ at c and c^k is large, it potentially means that the k^th input is strongly involved in the value of $G-g$ at c.
Of course it is just one estimation. That is why we may want to play with different c (different corners) to be more accurate. In practice it is an operation involving only $\mathcal{O}(d)$ computations, which is very cheap compared to the cost of adding blindly new crossed levels.
In practice, this is pretty efficient at finding a subset of inputs for which we should add crossed levels (putting the other ones aside).

3.3.3 Test case: Fast Neutron Reactor

We had the chance to implement this enhanced sparse grid method into an industrial test case: a cylindrical fast reactor.
The objective is to create a parametric model of the reactivity R as a function of the combustible enrichment.
The initial configuration of the reactor is pictured in Figure 3.8.

- $FERT$: Fertile,
- ABS: Absorbing medium,
- $C11$, $C17$ and $C23$: combustible with 11, 17 and 23 enrichment percentage.

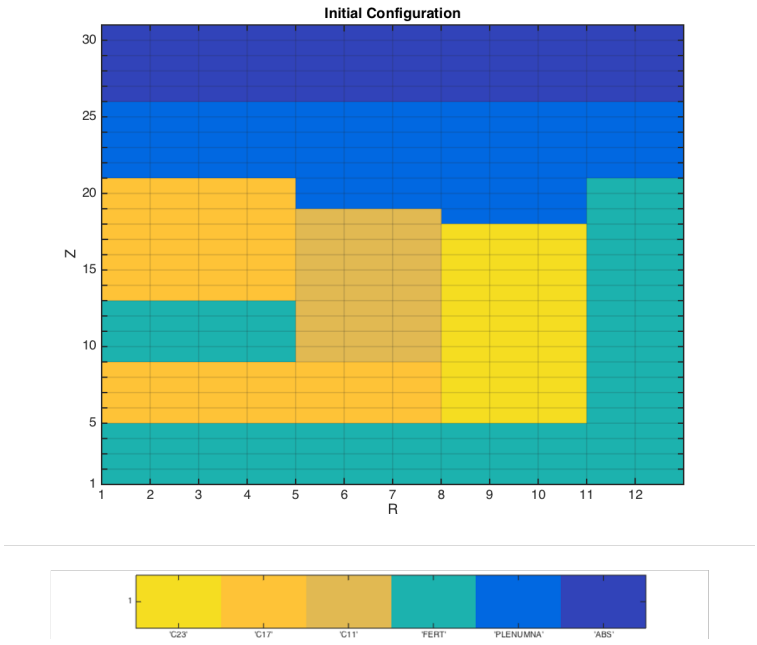


Figure 3.8 – Initial configuration

The solver we use is ERANOS [29].
The input variables are the combustible enrichment for every cell: 129 parameters p_i, $i = 1,\ldots,129$. Thus $d = 129$.
If we call c_i the initial enrichment percentage of cell i, we build a model for

$p_i \in [ci - 4\%; ci + 4\%]$.
The connection of the SPG algorithm to the ERANOS solver is a work that has been made by Mathieu Causse (Adagos), as well as the running of the following experiments.

3.3.3.1 First experiment

In this first experiment the quantity we want to model is the reactivity R.

- The adapted SPG process reaches its goal after 387 calls to ERANOS.
- The estimated relative error is 3.0×10^{-3}.
- The actual error on validation data set (latin hypercube sampling) is 4.2×10^{-4}.

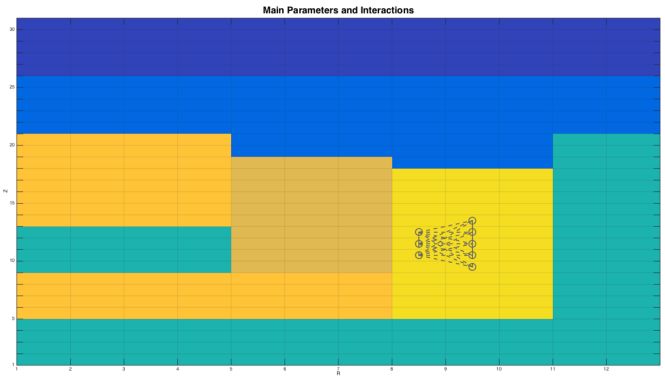


Figure 3.9 – Significant parameters and interactions

In Figure 3.9,

- the circles show the most significant parameters according to the SPG method.
- the lines connecting circles show parameters that interact in the SPG model.

With the usual SPG method, the number of calls to Eranos was exceeding 1000 (due to the interaction between input variables having only a linear influence) without getting to target tolerance yet. We stopped the model creation at this point.

3.3.3.2 Second experiment

In this second experiment the output we want to model is the void effect on reactivity $R - R_{void}$. It is the difference in reactivity before and after the reactor is being emptied wit respect to the initial configuration.
Figure 3.10 shows that our method is able to catch the most significant parameters and their interactions.

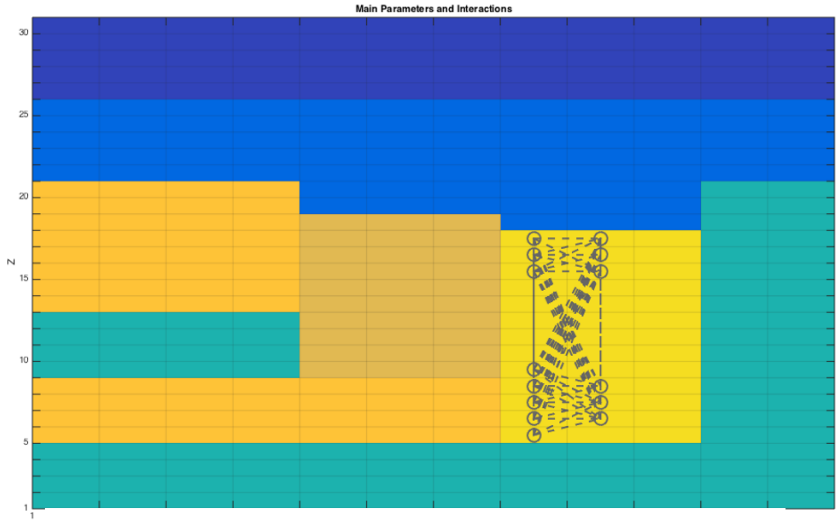


Figure 3.10 – Significant parameters and interactions

Once again,

- circles show the most significant parameters according to the SPG model,
- lines connecting circles show parameters for which interactions have been introduced.

One can observe on Figure 3.10 that two separate zones interact. This phenomenon is physically consistent and is due to the TOPAZ design as explained in [30].
That shows that the proposed method managed to exhibit meaningful interactions without adding a large number of unnecessary crossed levels.

3.4 Prospects: application to zROM

We are adapting Sparse Grids to zROM 1, the reduced order model for structural analysis, in order to release the latter as a product.
The sparse grids are used to build the design of experiment iteratively. At each stage, a zROM model is built based on the algorithm of selection of the basis functions 1.8.3.3.
The main difficulty is that the model is not interpolating contrary to usual SPG. So we have to rethink completely the error estimations and consequently the way the sub-grids are introduced.

3.4.1 Preliminary results: the vibrating plate

We have run tests with the vibrating plate test case 1.9.4. The specifications of the experiment are the same as earlier.
We chose this test case because we have access to the solver and consequently we can monitor the design of experiment. So we connected the SPG method to zROM and started the experiment.
We disabled the dimensional adaptativity as we do not know how to deal with it at the moment.
We are going to see how the ROM evolves when the sparse grid grows from level 1 to level 4.
At every level we update the preprocessing of the learning data and we keep only 2 coefficients from the SVD. We decided to set a rule of thumb to define the prescribed number of basis functions n_{bf} we want to introduce at each level:

$$n_{\text{bf}} = \frac{n_l}{5} + 1, \tag{3.4.1}$$

where n_l is the number of learning points for the current sparse grid. In practice, the actual number of basis functions in the model can exceeded n_{bf} if the last step in the selection 1.8.3.3 introduces several basis functions at once.
For every level we are going to display:

- The sparse grid and the basis function levels.
- The behavior of the model for the coefficients of the SVD (the SVD being updated at each level). This behavior will be depicted for the same validation data (i.e. the same lengths of the plate L_1 and L_2 and frequencies) as in 1.9.4. To reduce the number of graphics we only display the frequency where the error is the largest.

3.4.1.1 Level 1

A this stage the number of learning points is 7.

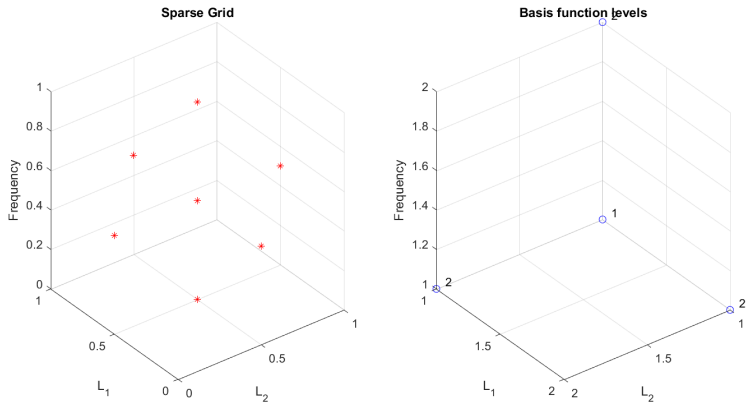


Figure 3.11 – Sparse grid and basis functions at level 1

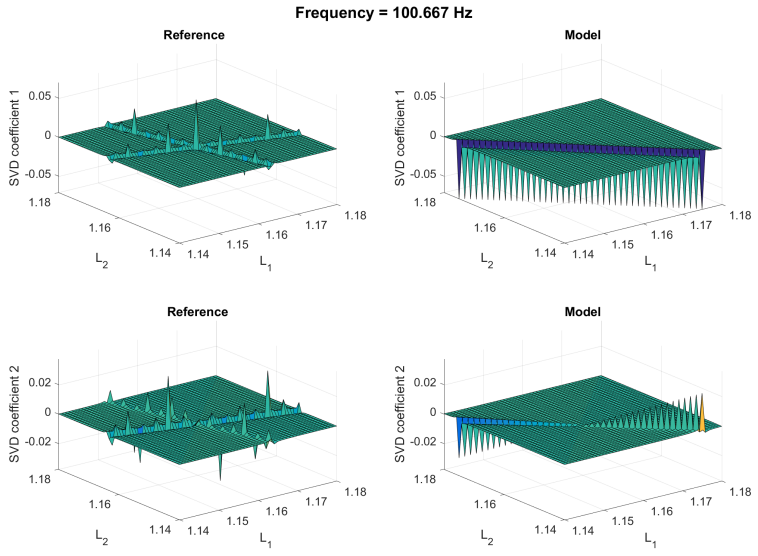


Figure 3.12 – Comparison between the reference (left) and the ROM (right) for the SVD coefficients at level 1

The validation relative error for the entire field (not the SVD coefficients) is 1.7239.

3.4.1.2 Level 2

A this stage the number of learning points is 25.

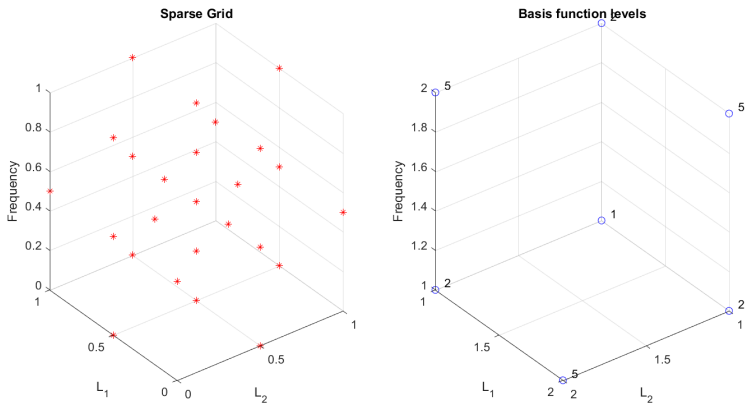


Figure 3.13 – Sparse grid and basis functions at level 2

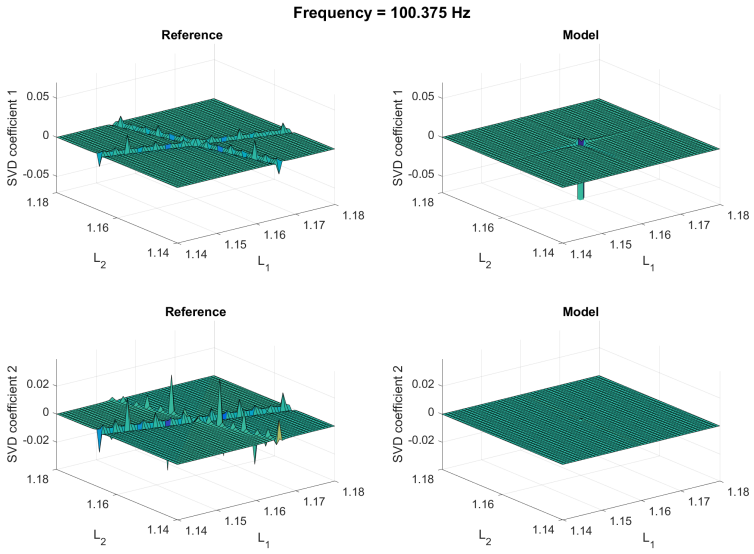


Figure 3.14 – Comparison between the reference (left) and the ROM (right) for the SVD coefficients at level 2

The validation relative error for the entire field is 2.5398.

3.4.1.3 Level 3

A this stage the number of learning points is 69.

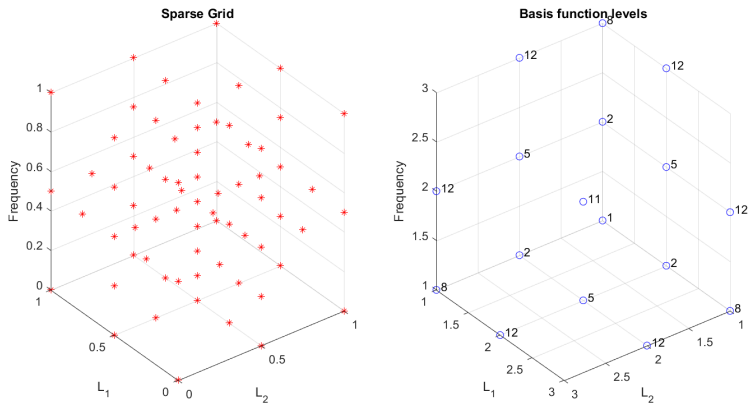


Figure 3.15 – Sparse grid and basis functions at level 3

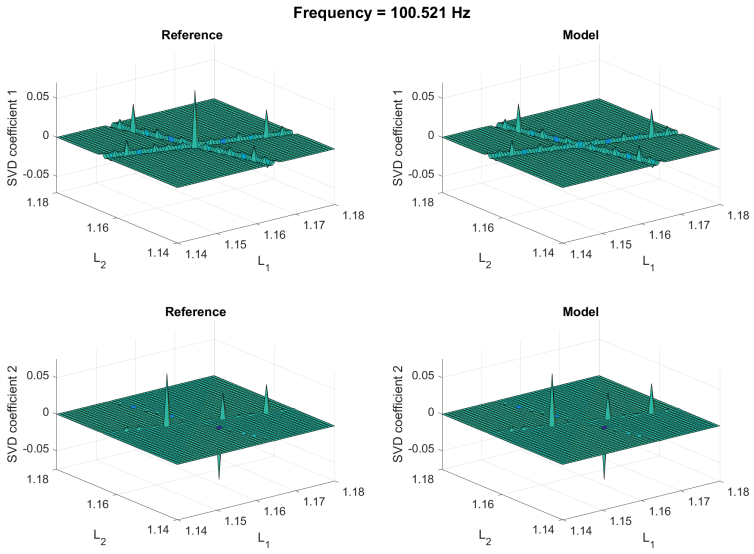


Figure 3.16 – Comparison between the reference (left) and the ROM (right) for the SVD coefficients at level 3

The validation relative error for the entire field is 0.1139.

3.4.1.4 Level 4

A this stage the number of learning points is 177.

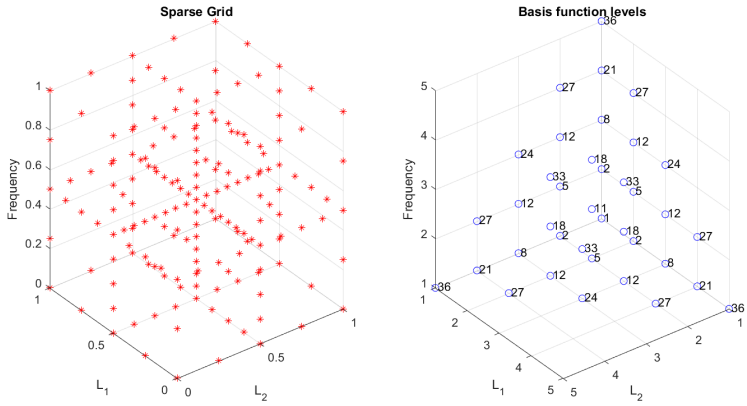


Figure 3.17 – Sparse grid and basis functions at level 4

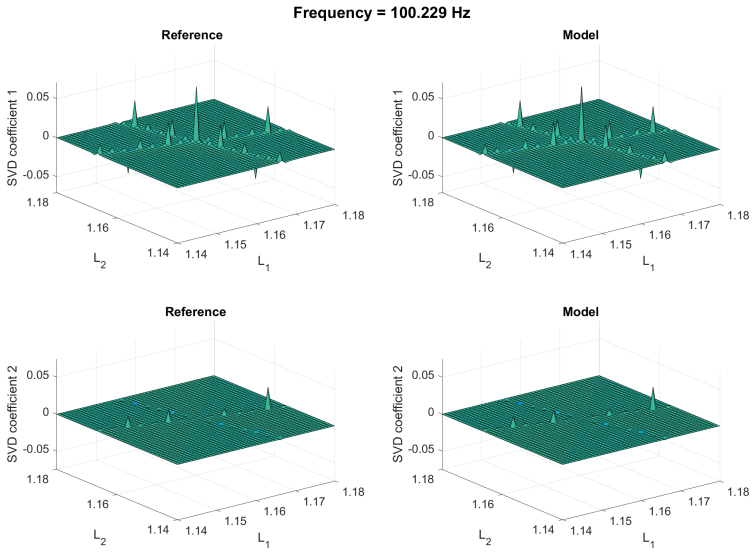


Figure 3.18 – Comparison between the reference (left) and the ROM (right) for the SVD coefficients at level 4

The validation relative error for the entire field is **0.0161**.

These first results are encouraging in the sense that the shape of the sparse grid does not seem to cause any trouble to zROM.
This sparse grid has the advantage to be easily refined iteratively (going one level further) without having to multiply the number of learning points by 2^d as it would be the case for a cartesian grid.
However we still need to find a way to implement the dimensional adaptativity strategy in this more complicated scenario.

Appendix A

Algorithmic differentiation

Calculating the gradient of a function is crucial for many applications in the field of inverse problems, optimization, machine learning and many others. Within this framework, we are going to introduce the useful concept of forward and backward mode differentiation.

A.1 Forward and backward mode differentiation

Let us define the function f

$$\begin{aligned} \mathbb{R}^n &\mapsto \mathbb{R}^m \\ y &= f(x) \end{aligned} \tag{A.1.1}$$

The forward mode differential is the usual one. For $dx \in \mathbb{R}^n$ we write

$$\left[y, dy\right] = df(x, dx) \tag{A.1.2}$$

where dy is in $\mathbb{R}^m$ and equals

$$\begin{aligned} dy &= \partial_x f\, dx \\ &= \lim_{\varepsilon \mapsto 0} \frac{f(x+\varepsilon dx) - f(x)}{\varepsilon}. \end{aligned} \tag{A.1.3}$$

Alternatively, the backward mode is called pf. For $py \in \mathbb{R}^m$ we define

$$\left[y, px\right] = pf(x, py) \tag{A.1.4}$$

where px is in $\mathbb{R}^n$ and is defined by

$$px_i = py^T \frac{\partial f}{\partial x_i} = \frac{\partial \left(py^T f\right)}{\partial x_i} \tag{A.1.5}$$

To sum up, the forward mode allows to compute the derivative of every output with respect to a scalar parameter (a linear combination of the inputs defined by dx). Symmetrically, the backward mode allows to compute the derivative of a linear combination of the outputs (defined by py) with respect to every input. The use of forward and backward modes is popular in algorithmic differentiation, and is especially used in the fied of optimization [23]. Those differentiation modes are used at every step of a computing program.
If we call Df the Jacobian matrix of f, we have in more simple manner

$$dy = Df\,dx \tag{A.1.6}$$

and

$$px = Df^T py \tag{A.1.7}$$

The role of df and pf is to avoid computing explicitly Df. In practice, for the application described in this document we compute df and pf by hand without using an automatic differentiation software.
For this reason we need a test to check that we are not mistaking. To do so we use the following property

$$\begin{aligned}(dx|px) &= (dx|Df^T py)\\ &= (Df\,dx|py)\\ &= (dy, py).\end{aligned} \tag{A.1.8}$$

This allows us to launch a test. If the scalar products mismatch, the calculation of the forward and backward mode differentials is indeed wrong.

A.2 Gauss-Newton method

Let us consider

$$\begin{array}{rccc} f: & \mathbb{R}^n & \rightarrow & \mathbb{R}^m \\ & x & \rightarrow & y = f(x). \end{array} \tag{A.2.1}$$

Our goal is to minimize $\|f(x)\|_2^2 = \sum_{i=1}^m f_i(x)^2$. To do so we use a Gauss-Newton algorithm, the well-known iterative method to solve non-linear least squares problems.
Starting from x_0 the method proceeds by the iterations

$$x^{s+1} = x^s - \left(Df^T Df\right)^{-1} Df^T f(x^s), \tag{A.2.2}$$

where Df is the Jacobian matrix of f.
Let d be such that

$$d = \left(Df^T Df\right)^{-1} Df^T f(x^s). \tag{A.2.3}$$

In practice we never invert $Df^T Df$, but rather solve

$$\left(Df^T Df\right) d = Df^T f(x^s). \tag{A.2.4}$$

If the problem is small we can compute explicitly J_f and solve the linear system via a direct method.
Alternativelyn, if it gets to large, we will not build the Jacobian matrix Df but use the forward and backward mode differentials. The linear system is then solved by a conjugate gradient method. This method is generally efficient as the eigenvalues of Df will often tend quickly towards zero in the problems we address.
Let us define the forward and backward modes as in section A.1 (taking only the second output for writing convenience)

$$[dy] = df(x, dx), \tag{A.2.5}$$

$$[px] = pf(x, py), \tag{A.2.6}$$

Thus the internal linear system to be solved in Gauss-Newton is equivalent to looking for d so that

$$pf(x^s, df(x^s, d)) = -pf(x^s, F(x^s)). \tag{A.2.7}$$

This system can be solved iteratively via a conjugate gradient method. Once d is identified, we get

$$x^{s+1} = x^s + d. \tag{A.2.8}$$

A.3 Sequence of instructions

Let f be

$$\begin{aligned} f: \quad & \mathbb{R}^n \mapsto \mathbb{R}^m \\ & x \mapsto f(x) \end{aligned} \tag{A.3.1}$$

where $x = (x_1, \ldots, x_n)$. The x_i for $i = 1, \ldots, n$ are called *independent* variables.
Usually, when dealing with complex algorithms, the functions we might want to handle can be the result of a long and complex sequence of instructions. For

example, the f function can be decomposed in a series of simpler instructions. Assuming that there are K instructions f_i, we define $(x_{n+1}, \ldots, x_N)$ the so called *intermediate* variables (where $N = n + K$). f is then defined as

$$\begin{cases} \text{for} \quad i = n+1 \quad to \quad N \quad \{ \\ \quad x_i = f_i(x_1, ..., x_{i-1}) \quad \} \\ f = x_N \end{cases} \tag{A.3.2}$$

Generally the *intermediate* functions f_i only depends on a few variables x_j, for $j < i$. Hence we define $S_i \subset (1, \ldots, i-1)$ the actual subset of useful variables. That leads to

$$\begin{cases} \text{for} \quad i = n+1 \quad \text{to} \quad N \quad \{ \\ \quad x_i = f_i(x_{S_i}) \quad \} \\ f = x_N \end{cases} \tag{A.3.3}$$

These *intermediate* functions can be simple analytic functions, as well as parts of computer codes. Let us take an example,

$$f(x) = \frac{1 + \sin(x_1 + x_2)}{x_1^2 + \exp(x_2)}. \tag{A.3.4}$$

This can be decomposed as

$$\begin{aligned} x_3 &= x_1 + x_2 \\ x_4 &= \sin(x_4) \\ x_5 &= 1 + x_4 \\ x_6 &= \exp(x_2) \\ x_7 &= x_1^2 \\ x_8 &= x_6 + x_7 \\ f = x_9 &= \tfrac{x_5}{x_8}. \end{aligned} \tag{A.3.5}$$

A.3.1 Forward mode differentiation

The forward mode is not difficult. It just consists in calculating the gradients of every *intermediate* function f_i and applying the usual rules of function composition. We can initialize the first n dx_i to the suitable value and then

$$\begin{cases} \text{for} \quad k = n+1 \quad \text{to} \quad N \quad \{ \\ \quad [x_k, dx_k] = df_k(x_{S_k}, dx_{S_k}) \quad \} \end{cases} \tag{A.3.6}$$

A.3.2 Backward mode differentiation

Assuming that we have already computed the values of the *intermediate* variables we compute the reverse mode as follows, starting with initializing every px_i, $i \leq n$ to zero, and for example px_i to py for $n+1 \leq i \leq N$.

$$\begin{cases} \text{for} \quad k = N \quad \text{down to} \quad n+1 \quad \{ \\ \quad \left[\cdot, px_{temp}\right] = df_k(x_{S_k}, px_k) \\ \quad \forall i \in S_k, \quad px_i = px_i + px_{temp} \quad \} \end{cases} \tag{A.3.7}$$

A.4 Example

We should highlight the fact that the *intermediate* functions can take various forms. For example one of the instruction can consist in solving a system. Let us take the following example

$$\begin{aligned} f: \quad \mathbb{R}^n &\mapsto \mathbb{R}^m \\ x &\mapsto f(x) \end{aligned} \tag{A.4.1}$$

where $f(x)$ is the solution to

$$A(x)u = b \tag{A.4.2}$$

Let us define

$$x_1 = f_1(x) = A(x) \tag{A.4.3}$$

and f_2 is the part of the algorithm that x_1 as an input, solves

$$x_1 u = b \tag{A.4.4}$$

and gives u as an output.
Let us take dx a small perturbation of x

$$dx_1 = df_1(x, dx) \tag{A.4.5}$$

Then we get

$$dx_1 u + x_1 du = 0 \tag{A.4.6}$$

Hence, by solving

$$x_1 du = -dx_1 u \tag{A.4.7}$$

we get du.
At this point we have determined the forward mode. In the same manner we

can look for the backward mode. Let us pick a value for pu. From f_2 instruction we get

$$px_1 = pux^T \tag{A.4.8}$$

and

$$px_{1\,temp} = x_1^T. \tag{A.4.9}$$

Then going down to instruction f_1 we get

$$px_1 = px_{1\,temp} + pf_1(x, px_1). \tag{A.4.10}$$

Bibliography

[1] A. N. Kolmogorov, "On the representation of continuous functions of several variables by superpositions of continuous functions of a smaller number of variables", Proceedings of the USSR Academy of Sciences, 108 (1956), pp. 179–182; English translation: Amer. Math. Soc. Transl., 17 (1961), pp. 369–373.

[2] Zienkiewicz, Olgierd Cecil, et al. The finite element method. Vol. 3. London: McGraw-hill, 1977.

[3] Tosio Kato, Perturbation theory for linear operators, Chapter Two, Perturbation theory in a finite-dimensional space, Springer, 1980.

[4] Osgood, William F. (1899), "Note über analytische Functionen mehrerer Veranderlichen", Mathematische Annalen, Springer Berlin / Heidelberg, 52: 462-464

[5] Hazewinkel, Michiel, ed. (2001) [1994], "Hartogs theorem", Encyclopedia of Mathematics, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4

[6] M. B. Giles, Collected matrix derivative results for forward and reverse mode algorithmic differentiation, in Advances in Automatic Differentiation, Springer, 2008, p. 35-44.

[7] Bjorn Gustaven and Adam Semlyen, Rational Approximation of frequency domain responses by vector fitting, IEEE Transactions on Power Delivery, Vol. 14, No. 3, July 1999.

[8] Philippe Guillaume, Alain Huard, Multivariate Padé approximation, Journal of Computational and Applied Mathematics 121 (2000) 197-219.

[9] Daniel Boichu and Vincent Robin, Multivariate Steinberg Theorem (http://www.eudoxuspress.com/images/jcaamv3-05.pdf)

[10] Runge, Carl (1901), "Uber empirische Funktionen und die Interpolation zwischen aquidistanten Ordinaten", Zeitschrift für Mathematik und Physik, 46: 224-243.

[11] De Boor, Carl, et al. A practical guide to splines. Vol. 27. New York: Springer-Verlag, 1978.

[12] Das, I., and Dennis, J. E. (1998). Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. SIAM Journal on Optimization, 8(3), 631-657.

[13] Ladeveze, Pierre, J-C. Passieux, and David Neron. "The latin multiscale computational method and the proper generalized decomposition." Computer Methods in Applied Mechanics and Engineering 199.21-22 (2010): 1287-1296.

[14] Lumley JL. The structure of inhomegeneous turbulent flows. In: Monin AM, Tararsky VI, editors. Atmospheric Turbulence and Wave Propagation. Moscow: Nauka; 1967. p. 166–78

[15] Aubry, N., Holmes, P., Lumley, J., & Stone, E. (1988). The dynamics of coherent structures in the wall region of a turbulent boundary layer. Journal of Fluid Mechanics, 192, 115-173. doi:10.1017/S0022112088001818

[16] Xiao, D., et al. "Non-intrusive reduced order modelling of the Navier-Stokes equations." Computer Methods in Applied Mechanics and Engineering 293 (2015): 522-541.

[17] G. Strang, Introduction to Applied Mathematics, Wellesley-Cambridge Press, Jan 1, 1986

[18] P. Guillaume, M. Masmoudi, Solution to the Time-Harmonic Maxwell's Equations in a Waveguide; Use of Higher-Order Derivatives for Solving the Discrete Problem, SIAM Journal on Numerical Analysis, 1997, Vol. 34, No. 4 : pp. 1306-1330

[19] Griewank, Andreas. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. Number 19 in Frontiers in Appl. Math. SIAM, Philadelphia, PA, 2000. ISBN 0-89871-451-6.

[20] Schrauwen, Benjamin, David Verstraeten, and Jan Van Campenhout. "An overview of reservoir computing: theory, applications and implementations." Proceedings of the 15th European Symposium on Artificial Neural Networks. p. 471-482 2007. 2007.

[21] Vandenberghe, Lieven, and Stephen Boyd. "Semidefinite programming." SIAM review 38.1 (1996): 49-95.

[22] Versteeg, Henk Kaarle, and Weeratunge Malalasekera. An introduction to computational fluid dynamics: the finite volume method. Pearson Education, 2007.

[23] Griewank, Andreas, and Andrea Walther. Evaluating derivatives: principles and techniques of algorithmic differentiation. Society for Industrial and Applied Mathematics, 2008.

[24] Sergey Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Doklady Akademii Nauk SSSR, Volume 4, 1963, pages 240-243.

[25] Sparse grids, HJ Bungartz, M Griebel - Acta numerica, 2004 - Cambridge Univ Press

[26] Runge, Carl (1901), "Uber empirische Funktionen und die Interpolation zwischen aquidistanten Ordinaten", Zeitschrift für Mathematik und Physik, 46: 224-243. (http://www.ians.uni-stuttgart.de/spinterp/doc/spinterpdoc.pdf)

[27] A. Klimke, Sparse Grid Interpolation Toolbox User's Guide V5.1, February 24, 2008

[28] Lophaven, S. N., H. B. Nielsen, and J. Sondergaard. "DACE-a Matlab Kriging toolbox; version 2; informatics and mathematical modelling." Technical University of Denmark, Technical Report No. IMM-TR-2002-12 (2002). (http://www2.imm.dtu.dk/projects/dace/)

[29] Eranos solver, OECD Nuclear Energy Agency, (http://www.oecd-nea.org/tools/abstract/detail/nea-1683/)

[30] Verrier, D., A. C. Scholer, and M. Chhor. "A New Design Option for Achieving Zero Void Effect in Large SFR Cores." Proceedings of FR13, Paris, France, 5-7 March (2013).