Adaptive tensor methods for high dimensional problems

María Fuente Ruiz

Adaptive tensor methods for high dimensional systems

THÈSE DE DOCTORAT

Présentée par

María Fuente Ruiz

pour obtenir le grade de

DOCTEUR
de Sorbonne Université

Spécialité : Mathématiques Appliquées

Examinatrice	Marie BILLAUD-FRIESS	Maître de conférences de Ecole Centrale Nantes
Examinateur	Mi-Song DUPUY	Maître de conférences de Sorbonne Université
Co-directrice de Thèse	Virginie EHRLACHER	Chargé de recherche de CERMICS–Ponts ParisTech
Rapporteur	Antonio FALCÓ MONTESINOS	Professeur de Universidad CEU Cardenal Herrera
Directeur de Thèse	Damiano LOMBARDI	Chargé de recherche de l'Inria Paris
President	Yvon MADAY	Professeur de Sorbonne Université
Rapporteur	Guillaume PERRIN	Professeur de Université Gustave Eiffel
Examinateur	André USCHMAJEW	Professeur de Universität Augsburg University

Après avis favorables des rapporteurs: Antonio FALCÓ MONTESINOS et Guillaume PERRIN

Thèse préparée au sein de l'équipe-projet Commedia
Centre de Recherche Inria de Paris
2 rue Simone Iff
75589 Paris Cedex 12
et **Laboratoire Jacques-Louis Lions**
de **Sorbonne Université**

Acknowledgments

I start acknowledging my thesis directors: Virginie Ehrlacher and Damiano Lombardi. Thank you for the inspiration, the example, the professional advice, and for all the things you have thought me.

I am very grateful to Antonio Falcó and Guillaume Perrin for having taken time to review this manuscript and for the encouraging reports. I also thank Marie Billaud-Friess, Mi-Song Dupuy, Yvon Maday and André Uschmajew for being members of the jury for which I am honored.

Thank you to Adrien Beguinet, Roberta Flenghi, Olga Mula and Agustin Somacal, members of a research team that started in CEMRACS and have continued till today, conforming the last contribution of this thesis. Also for the help inside and outside the professional environment.

Also, to all the members of the COMMEDIA team, for giving me the opportunity and the support of doing this project. A special thank you to all the current and former colleagues of the team that have become close friends. You've been the best teammates that one could ask for.

Thanks to all my friends in Paris that have been more my family than my friends here and that were with me during all this three years and I'm sure for a lot more. A special thank you to my flatmates, that have lived my thesis submission from a very close and stressful perspective, as if it was theirs. Also to the ones that have been sending me love and support even in the distance. All of you have filled these three years of unforgettable moments. To my capoeira and dance group friends, for the daily disconnection time.

Thanks to my family. It's been years since you support me unconditionally, you are my example in life. I am very lucky to have you always having my back.

To finalize, thanks to Alberto. Without you none of this would have been possible. Thank you for the patience, the support, the love and for always stand by my side. This journey has only started.

Abstract

Abstract: In this thesis, the aim is to find numerical methods in order to approximate multivariate functions (tensors). They can be derived from datasets, multiway arrays, or solutions of PDEs coming from physics, biology, economy or other disciplines. The task of approximating multivariate functions is done from three different perspectives in this thesis:

1. A tensor approximation method, Sum of Tensor Trains (SoTT), whose output is the approximated solution in a sum of TTs format.
2. A local tensor approximation method based on clustering, that retrieves an approximated solution in the introduced local HOSVD format.
3. A deep learning-based method that computes the approximation of the solution of a convection diffusion PDE when the diffusion parameter is very small.

In the first part of the work, a method is proposed in order to compute an approximation of a given tensor as a sum of Tensor Trains (TTs), where the order of the variates and the values of the ranks can vary from one term to the other in an adaptive way. The numerical scheme is based on a greedy algorithm and an adaptation of the TT-SVD method. It achieves good performances without depending on the variable ordering. The proposed approach can also be used in order to compute an approximation of a tensor in a Canonical Polyadic format (CP), as an alternative to standard algorithms. Some numerical experiments are proposed, in which the proposed method is compared to Alternating Least Squared (ALS) and Alternating Singular Value Decomposition (ASVD) methods for the construction of a CP approximation of a given tensor and performs particularly well for high-order tensors. The interest of approximating a tensor as a sum of Tensor Trains is illustrated in several numerical test cases.

In the second part of the work, we propose a local tensor approximation method based on clustering, that retrieves an approximated solution in the introduced local High Order Singular Value Decomposition (local HOSVD) format. The clustering method is performed direction per direction and it provides a dyadic partition tree that encloses the different possible partitions of the domain. This can be done due to the separability of the domain. In order to find the best partition to approximate the function, an extensive search between the possible partition combinations is done. Then, a HOSVD approximation of the tensor on each one of the partitions is computed. The criterion selected in order to choose one partition over the others is the memory needed to store the approximation. In the light of the numerical results obtained, we can say that the method achieves a good compression rate with respect to the HOSVD method.

In the third part of the work, a study on Deep learning-based numerical schemes, such as Physics-Informed Neural Networks (PINNs) as an alternative to classical numerical schemes for solving Partial Differential Equations (PDEs) is addressed. These methods are very appealing at first sight because of its simple implementation. Vanilla versions of PINNs, based on strong residual forms and neural networks, offer very high approximation capabilities. However, when the PDE solutions are low regular, optimization solvers are significantly challenged, and can potentially spoil the final quality of the approximated solution due to the convergence to bad local minima, and bad generalization capabilities. In this work, we present a numerical study of the merits and limitations of these schemes when solutions exhibit low-regularity, and compare performance with respect to more benign cases when solutions are very smooth. As a support for our study, we consider singularly perturbed convection-diffusion problems where the regularity of solutions typically degrades as certain multiscale parameters go to zero.

Keywords: Tensor methods, Canonical Polyadic, Tesor Train, Local Tensor Methods, Model Order Reduction, Neural Networks, PINNs, Singulary Perturbed Problems.

Résumé

Résumé: Dans cette thèse, nous cherchons à développer des méthodes numériques afin d'approcher des fonctions multivariées (tenseurs). Celles-ci peuvent être dérivées d'ensembles de données, de tableaux à entrée multiple , ou de solutions d'EDP qui provient de la physique, de la biologie, de l'économie ou d'autres disciplines. L'approximation de ces fonctions multivariées est abordée sous trois angles:

1. Une méthode d'approximation tensorielle, Sum of Tensor Trains (SoTT), dont la sortie est la solution approchée fournie sous la forme d'une somme de TTs.

2. Une méthode d'approximation tensorielle locale basée sur un methode de clustering, qui récupère une solution approchée dans le format local HOSVD.

3. Une méthode basée sur l'apprentissage profond qui calcule l'approximation de la solution d'une équation de convection-diffusion lorsque le paramètre de diffusion est très petit.

Dans la première partie de ce travail, la méthode proposée permet de calculer une approximation d'un tenseur donné comme une somme de Tensor Trains (TT), où pour lequel l'ordre des variables et les valeurs des rangs peuvent varier d'un terme à l'autre de manière adaptative. Le schéma numérique est basé sur un algorithme glouton et une adaptation de la méthode TT-SVD. Il permet d'obtenir de bonnes performances de bons résultats sans dépendre indépendemment de l'ordre des variables. L'approche proposée peut également être utilisée pour calculer une approximation d'un tenseur dans un format Canonical Polyadic (CP), comme alternative aux algorithmes standards. Quelques résultats numériques sont proposés, dans lesquelles la méthode développée est comparée aux méthodes Alternating Least Squared (ALS) et Alternating Singular Value Decomposition (ASVD) pour la construction d'une approximation CP, celle-ci se comporte particulièrement bien pour les tenseurs d'ordre élevé. L'intérêt d'approximer un tenseur comme une somme de Trains Tensoriels est illustré dans plusieurs cas tests numériques.

Dans la deuxième partie du travail, nous proposons une méthode d'approximation tensorielle locale basée sur le clustering, qui récupère une solution approchée dans le format introduit local High Order Singular Value Decomposition (local HOSVD). La méthode de clustering est réalisée direction par direction et fournit un arbre de partition dyadique qui englobe les partitions du domaine. Ceci peut être fait grâce à la séparabilité du domaine. Afin de trouver la meilleure partition pour approximer la fonction, une recherche extensive entre les combinaisons de partitions possibles est effectuée. Ensuite, une approche HOSVD du tenseur sur chacune des partitions est calculée. Le critère retenu pour choisir une partition plutôt que les autres qu'une autre est la mémoire nécessaire pour stocker la solution approchée. À la lumière des résultats numériques obtenus, nous pouvons dire que la méthode atteint un bon taux de compression par rapport à la méthode HOSVD.

Dans la troisième partie du travail, une étude sur les schémas numériques basés sur l'apprentissage profond, tels que les Physics-Informed Neural Networks (PINNs), comme alternative aux schémas numériques classiques pour résoudre les équations différentielles partielles (PDEs), est abordée. Ces méthodes sont très attrayantes à première vue en raison de leur mise en œuvre très simple. Les versions anciennes des PINNs, basées sur des formes résiduelles fortes et des réseaux de neurones, offrent des capacités d'approximation très élevées. Cependant, lorsque les solutions des EDP sont peu régulières, les solveurs d'optimisation peuvent potentiellement restituer une mauvaise solution approchée en raison de la convergence vers de mauvais minima locaux et de mauvaises capacités de généralisation. Dans ce travail, nous présentons une étude numérique mettant en excergue les des mérites avantages mais aussi les limites de ces schémas lorsque les pour des solutions présentent une faible régularité peu régulières, et nous comparons également les performances pour des cas plus simples avec des solutions lisses. Comme support à notre étude, nous considérons des problèmes de convection-diffusion singulièrement perturbés où la régularité des solutions se

dégrade typiquement lorsque certains paramètres multi-échelles tendent vers zéro.

Mots-clés : Méthodes Tensorielles, Canonical Polyadic, Tensor Train, Méthodes Tensorielles Locales, Modèles d'Ordre Réduit, Réseaux Neuronaux, PINNs, Problèmes Perturbés Singuliers.

Contents

1 Introduction **1**
- 1.1 State of the art of Tensor methods . . . 2
  - 1.1.1 Proper Orthogonal Decomposition (POD) . . . 3
  - 1.1.2 Canonical Polyadic (CP) decomposition . . . 4
  - 1.1.3 Tucker decomposition . . . 7
  - 1.1.4 Hierarchical Tucker (H-Tucker) format . . . 8
  - 1.1.5 Tensor Train format . . . 10
  - 1.1.6 Approximations in local subdomains . . . 10
- 1.2 State of the art of clustering methods . . . 13
  - 1.2.1 Physics-informed clustering . . . 14
  - 1.2.2 Other classification methods . . . 15
- 1.3 State of the art of Deep Learning-based Schemes . . . 15
  - 1.3.1 Deep learning scheme . . . 16
  - 1.3.2 Physics-Informed Neural Networks (PINNs) . . . 18
- 1.4 Contributions of the thesis . . . 22
  - 1.4.1 Contribution on Tensor Formats: The Sum of Tensor Trains method . . . 22
  - 1.4.2 Contribution in local tensor methods: The local HOSVD method . . . 24
  - 1.4.3 Contribution in Deep Learning-based Schemes for Singularly Perturbed Convection-Diffusion Problems . . . 25
- 1.5 Organization of the manuscript . . . 27

2 Sum of Tensor Trains: SoTT **29**
- 2.1 Introduction . . . 29
  - 2.1.1 Organization of the chapter . . . 30
- 2.2 Notation and preliminaries . . . 30
  - 2.2.1 Tensor spaces . . . 31
  - 2.2.2 Object definition and POD decomposition . . . 32
  - 2.2.3 Ranks and tensor formats: Canonical Polyadic (CP) and Tensor Train (TT) . . . 33
- 2.3 The Sum of Tensor Trains (SoTT) algorithm . . . 36
  - 2.3.1 Presentation of the SoTT algorithm . . . 36
  - 2.3.2 Exponential convergence of the SoTT algorithm in finite dimension . . . 40
  - 2.3.3 Complexity estimate of the SoTT algorithm . . . 41
- 2.4 CP-TT: fixed-rank SoTT algorithm with rank 1 . . . 42
- 2.5 Numerical Experiments . . . 43
  - 2.5.1 Comparison between CP-TT and other rank-one update methods . . . 43
  - 2.5.2 SoTT method for the compression of multivariate functions . . . 47
- 2.6 Conclusions and perspectives . . . 53

3 Local tensor methods **55**
- 3.1 Introduction . . . 55
  - 3.1.1 Organization of the chapter . . . 59
- 3.2 Notation and preliminaries . . . 59
  - 3.2.1 Partitioning of tensors . . . 60
  - 3.2.2 The High Order Singular Value (HOSVD) decomposition . . . 61
- 3.3 Local tensor spaces . . . 61

3.3.1 Construction of local subdomains . . . 61
3.3.2 Computation of the leaves . . . 62
3.3.3 Merging local subdomains . . . 65
3.4 Computing local HOSVD method . . . 67
3.5 Cost and complexity of the algorithm . . . 71
3.6 Summary . . . 73
3.7 Numerical results . . . 74
3.7.1 Compression of a Gaussian function . . . 74
3.7.2 Compression of the solutions of the Fitz-Hugh-Nagumo equation . . . 75
3.8 Conclusions and perspectives . . . 80

4 Deep Learning-based schemes **83**
4.1 Introduction . . . 83
4.1.1 Organization of the chapter . . . 85
4.2 A singularly perturbed convection-diffusion equation . . . 85
4.2.1 Problem definition . . . 85
4.2.2 General formulation . . . 86
4.2.3 Vanilla (V) formulation . . . 86
4.2.4 Weak variational (W) formulation . . . 87
4.2.5 Rescaled formulation . . . 88
4.2.6 Summary of the methods . . . 89
4.3 Neural networks based numerical schemes . . . 89
4.3.1 General principle . . . 90
4.3.2 Neural Network classes of functions . . . 90
4.3.3 Sampling schemes . . . 91
4.3.4 Comparison with finite element schemes . . . 92
4.4 Numerical Results . . . 93
4.4.1 Test case and comparison criteria . . . 93
4.4.2 Our code and practical implementation details . . . 94
4.4.3 Discussion . . . 94
4.4.4 Conclusions from the numerical experiments . . . 95
4.5 Future research directions and extensions . . . 97

5 Conclusions and perspectives **99**
5.1 Conclusions of SoTT . . . 99
5.2 Conclusions of local HOSVD method . . . 99
5.3 Conclusions of Deep Learning-based schemes . . . 100

A Appendix of Sum Of Tensor Trains: SoTT **103**
A.1 Optimization of the computation . . . 103
A.1.1 Optimization of the coefficients . . . 103
A.1.2 Computing the SVD of the unfolding without explicitly computing and assembling the unfolding . . . 103
A.2 Results for functions in H1 . . . 104

B Appendix of Deep Learning-based schemes **107**
B.1 l2 error plots . . . 107
B.2 Architecture of the NN plots . . . 108
B.3 Training of the PINN . . . 108

Chapter 1

Introduction

Contents

1.1 State of the art of Tensor methods 2
1.1.1 Proper Orthogonal Decomposition (POD) 3
1.1.2 Canonical Polyadic (CP) decomposition 4
1.1.3 Tucker decomposition 7
1.1.4 Hierarchical Tucker (H-Tucker) format 8
1.1.5 Tensor Train format 10
1.1.6 Approximations in local subdomains 10
1.2 State of the art of clustering methods 13
1.2.1 Physics-informed clustering 14
1.2.2 Other classification methods 15
1.3 State of the art of Deep Learning-based Schemes 15
1.3.1 Deep learning scheme 16
1.3.2 Physics-Informed Neural Networks (PINNs) 18
1.4 Contributions of the thesis 22
1.4.1 Contribution on Tensor Formats: The Sum of Tensor Trains method 22
1.4.2 Contribution in local tensor methods: The local HOSVD method 24
1.4.3 Contribution in Deep Learning-based Schemes for Singularly Perturbed Convection-Diffusion Problems 25
1.5 Organization of the manuscript 27

Machine learning and data mining algorithms are becoming increasingly important in analyzing large volume, multi-relational and multi-modal datasets, which are often conveniently represented as multiway arrays or tensors [1, 2, 3]. Many problems with practical interest in physics, chemistry or mathematical applications naturally lead to high-dimensional or multivariate approximation problems. As a consequence, these can't be treated naively.

The main challenge in dealing with such amount of data is the so called *curse of dimensionality*, introduced by Bellman in [4], that refers to the need of using a number of degrees of freedom exponentially increasing with the dimension [5]. In many applications, functions may depend on a potentially high number of variables d. When the dimension d increases, standard methods lead to a complexity of the numerical approximation which grows exponentially with d. As a consequence, the number of evaluations necessary to deal with such large datasets with naive tools may become prohivitive.

The numerical simulation of physical models takes today an important place in numerous branches of science and engineering. Due to the increasing complexity of models, more and more refined discretizations and robust numerical solution techniques are needed in order to obtain reliable predictions of their responses. Furthermore, in the context of optimization or model identification, the aim is not to predict the response of a unique model but of a family of models. In order to achieve these goals, traditional solution techniques require the optimal use of constantly evolving computational resources. This is closely linked with the

curse of dimensionality introduced before. However, for many applications, innovative methodologies as an alternative to the brute force approach are obviously necessary to access numerical prediction. In other contexts, standard models may become multidimensional if some of the parameters that they involve are considered as new coordinates. This possibility is specially attractive when these coefficients are not well known, they have a stochastic nature, or when one is interested in optimization or inverse identification.

The concept of model reduction (MOR, [6, 7]) seems to be a path for solving these computational issues. Model order reduction aims to lower the computational complexity of certain problems as simulations of large-scale dynamical systems and control systems. Model reduction methods exploit the fact that the response of complex models can often be approximated with a reasonable precision by the response of a surrogate model.

For instance, in Reduced Basis (RB, [8, 9, 10, 11, 12, 13]) methods this is usually achieved by projection of full-order models. In many studies, the reduced basis is obtained by Proper Orthogonal Decomposition-based methods [14, 15, 16]. The dimension of reduced basis may be of several orders of magnitude lower than the dimension of the classically used numerical models. We refer to [7] for a complete overview of the Model Order Reduction picture.

Consequently, the goal is to provide methods for generating reduced models that are simultaneously accurate (concerning the approximation error) and performant (concerning the simulation time) independently of the complexity in parameters and the complexity in the time evolution of the original problem.

Currently, there are several classes of methods under investigation in order to propose parsimonious representations in high dimensions. They try to address an approximation in classes of functions that make it possible to circumvent the curse of dimensionality, [17, 7, 18, 19]. Standard structured approximations include n-terms approximations [20, 21, 22], sparse approximations [23], low-rank approximations [24, 25], or deep learning approximation methods [26].

This work is focused on low-rank representations of multivariate functions from three different points of view. From the so-called low-rank tensor approximations or low-rank tensor decompositions, [27, 28], its local variant local low-rank tensor approximation models and from a deep learning-based approach.

Find in the present chapter the state of the art of Tensor methods in Section 1.1, an overview on clustering methods in Section 1.2 and the state of the art of Physics-Informed Neural Networks in Section 1.3. At the end of the chapter, in Section 1.4 the reader can find also the contributions of this thesis to the scientific framework.

1.1 State of the art of Tensor methods

In the literature, these high-order equivalents of vectors or matrices are called high-order tensors or multi-way arrays. For a lot of applications involving high-order tensors, the existing algebraic methods face the cited curse of dimensionality, what makes them inappropriate for some problems. This problem can be alleviated by using various tensor formats, achieved by low-rank tensor approximations, for the compression of the full tensor as described for instance in [29, 30, 31, 32]. Tensors are well-studied objects in the standard mathematics literature [33, 34, 35] and more specifically in multilinear algebra, [36, 37].

Let p be the dimension of the system and let D be a subset of $\mathbb{R}^p$ with a product structure $D := D_1 \times \ldots \times D_d$. Let us consider $d \in \mathbb{N}^*$. Then, for all $1 \leq i \leq d$, D_i is an open bounded subset of $\mathbb{R}^{p_i}$ for some $p_i \in \mathbb{N}^*$. We call a *tensor* any real-valued multivariate function $F \in L^2(D_1 \times \cdots \times D_d)$. The following definitions are set for continuous tensors. Let a discrete tensor be the discretizated form of a multivariate function.
For each $1 \leq i \leq d$, let us have a certain $\mathcal{H}_i$ Hilbert space of univariate functions defined of D_i, equipped with the corresponding inner product $(\cdot,\cdot)_{\mathcal{H}_i}$ and the associated norm $\|\cdot\|_{\mathcal{H}_i}$.
The elementary tensor product $u^{(1)} \otimes \ldots \otimes u^{(d)}$ of the d univariate functions $u^{(i)} \in \mathcal{H}_i$ has associated a multilinear mapping $D_1 \times \ldots \times D_d \to D_1 \otimes \ldots \otimes D_d$ and is defined such that for a multivariate function F and $x = (x_1, \ldots, x_d) \in D$,

$$F(x) = u^{(1)}(x_1), \otimes \ldots \otimes u^{(d)}(x_d).$$

The space $L^2(D)$ of square-integrable, real-valued functions defined on D equipped with a scalar product

and the corresponding canonical norm $\|\cdot\|_{L^2(D)}$ that is defined for $F, G \in L^2(D)$ by

$$\langle F, G\rangle_D := \int_D F(x)G(x)dx \quad \text{and} \quad \|F\|_{L^2(D)} := \left(\int_D F^2(x)dx\right)^{1/2}.$$

From now on, in order to simplify the notation of the manuscript (every time that there's no ambiguity) $L^2(D)$ will be simplified as D, equipped with the norm $\|\cdot\|_D$ and its associated inner product $(\cdot,\cdot)_D$.

1.1.1 Proper Orthogonal Decomposition (POD)

The Proper Orthogonal Decomposition (POD), [38], is a popular dimensionality reduction method used in model reduction (see [39, 40, 41]) to define the trial subspace.

Definition. For any domain $D = D_x \times D_y$, where D_x and D_y are open subdomains of $\mathbb{R}^{d_x}$ and $\mathbb{R}^{d_y}$ for some $d_x, d_y \in \mathbb{N}^*$ respectively, and any $W \in L^2(D)$, it holds that there exists an orthonormal basis $(U_k)_{k\in\mathbb{N}^*}$ of $L^2(D_x)$, an orthonormal basis $(V_k)_{k\in\mathbb{N}^*}$ of $L^2(D_y)$ and a non-increasing sequence $(\sigma_k)_{k\in\mathbb{N}^*}$ of non-negative real numbers which converges to 0 as k goes to ∞, such that

$$W = \sum_{k\in\mathbb{N}^*} \sigma_k U_k \otimes V_k. \tag{1.1}$$

A decomposition of W under the form (1.1) is called a Proper Orthogonal Decomposition (POD) of W according to the separation of variables (D_x, D_y). It is essentially the same as the Singular Value Decomposition (SVD) in a finite-dimensional space or in Euclidean space. The sequence $(\sigma_k)_{k\in\mathbb{N}^*}$ is known to be unique and is called the sequence of singular values of W associated to the separation of variables (D_x, D_y) of the set D. The orthonormal basis $(U_k)_{k\in\mathbb{N}^*}$ (respectively $(V_k)_{k\in\mathbb{N}^*}$) may not be unique but is called a sequence of left (respectively right) singular vectors of W associated to this partitioning.

For any set E, we denote in the sequel by $\#E$ the cardinality of E. Assuming that $\mathcal{N}_x := \#D_x < +\infty$ and $\mathcal{N}_y := \#D_y < +\infty$, the complexity of the computation of an SVD decomposition of the form (1.1) scales like

$$\mathcal{O}\left(\max(\mathcal{N}_x, \mathcal{N}_y)\min(\mathcal{N}_x, \mathcal{N}_y)^2\right). \tag{1.2}$$

The definition of these different tensor formats relies on the well-known separation of variables principle. We refer the reader to [24] and [42] for extensive reviews on tensor theory and extended analysis of tensor decompositions and their numerous applications. Tensor formats are also used for solving time-dependent and stochastic/parametric PDEs ([43, 44]).

Principal component analysis (PCA), [45, 46], is a multivariate technique that analyzes a data set in which observations are described by several intercorrelated quantitative dependent variables. PCA is probably the most popular multivariate statistical technique and it is used by almost all scientific disciplines. Its goal is to extract the important information from these data, to compress the size of the data set by keeping only this important information, to represent them as a set of new orthogonal variables called principal components (obtained as linear combinations of the original variables), and to display the pattern of similarity of the observations and of the variables as points in certain maps. Mathematically, its components are obtained from the singular value decomposition (SVD) introduced in Equation 1.1 of the data. Thus, PCA depends on the singular value decomposition of positive semidefinite matrices, i.e. on the SVD of rectangular matrices. Its generalized form, Generalized principal component analysis (GPCA) is introduced in [47].

In the context of approximating a multivariate function and in model order reduction, the POD is also known as PCA. It belongs to the family of linear dimensionality reduction techniques. POD-based reduced models consist in finding a low-dimensional afine subspace minimizing the projection error between the projected data and the original data. We call a solution $\{\phi_1, \ldots, \phi_M\}$ a POD basis of order M. Computing a POD basis of order M as given by Problem 1.1 is equivalent to solve a minimization problem of the form:

For $X = \{x_i\}_{i=1}^P$ given, find orthonormal functions $\{\phi_j\}_{j=1}^M$ solving:

$$\min_{\phi_j} \sum_{i=1}^{P} \|x_i - \sum_{j=1}^{M} (x_i, \phi_j)\phi_j\|^2$$

Eckart-Young theorem, [48], shows the optimality of the POD for two variables. Let us define then the maximal POD basis $\{\psi_j\}_{j=1}^{M}$ and then:

$$\sum_{i=1}^{P} \|x_i - \sum_{j=1}^{M} (x_i, \psi_j)\psi_j\|^2 \leq \min_{\phi_j} \sum_{i=1}^{P} \|x_i - \sum_{j=1}^{M} (x_i, \phi_j)\phi_j\|^2 \tag{1.3}$$

for any other $\{\phi_j\}_{j=1}^{M}$ and $M \leq P$. Moreover the truncation error ε of using M instead of P POD basis in representing X is given by:

$$\varepsilon(M) = \sum_{i=1}^{P} \|x_i - \sum_{j=1}^{M} (x_i, \psi_j)\psi_j\|^2 = \sum_{k=M+1}^{P} \sigma_k^2$$

where $\sigma_k, \quad k = M+1, \ldots, P$ are the $P - M$ smaller singular values of the POD decomposition of 1.1. The corresponding M singular values kept indicate the variance that is captured, which enables to ordering the principal component and choosing their appropriate number to represent the dataset with a given level of accuracy ε.

The reduced order X derived by projecting onto the POD subspace, are represented as follows

$$x^M = \sum_{j=1}^{M} \alpha_j^M \psi_j$$

where α^M are the expansion coefficients. Solving the reduced order model 1.3 for the coefficients allow us to compute the reduced order solution x^M. The different choice of basis functions will lead us to different solving strategies: Galerkin, Petrov Galerkin, reduced basis methods...

Over the past years, model reduction techniques have become a very important tool for the reduction of computational requirements in the numerical simulation of complex high-dimensional models. A technique able to alleviate the already mentioned curse of dimensionality consists of using a separated representation of the unknown multivariate function. This can be done due to the separability of the domain. A family of models that rely on the construction of separated variables representation of the solution in tensor product spaces is conformed. The methods of this family can be interpreted as generalizations of Proper Orthogonal Decomposition (POD), Singular Value Decomposition (SVD) or Principal Component analysis (PCA) in other contexts. The use of these methods for tensor decomposition in high dimensional problems can be seen in [49, 50].

Tensor product spaces are introduced as the foundations of both a priori MOR methods and low-rank tensor approximations. Tensor Product Spaces are the appropriate framework to formalize the approximation in the parametric domain. Furthermore, the separated representations and tensor product spaces are closely related.
In the following, some of the most popular low-rank tensor decompositions are introduced.

1.1.2 Canonical Polyadic (CP) decomposition

As it it detailed in [51], the idea of the polyadic form of a tensor (expressing a tensor as the sum of a finite number of elementary tensors) arose initially in [52, 53]. This tensor format is also known as Candecomp Parafac. The form of Candecomp (canonical decomposition), was proposed in [54] and Parafac (parallel factors) in [55]. We refer to the Candecomp/Parafac decomposition as CP proposed independently in [56] and [57].

Let us now introduce some notation which will be used in all the sequel. From now on, we fix some $d \in \mathbb{N}^*$ and for all $1 \leq j \leq d$, let Ω_j be an open subset of $\mathbb{R}^{p_j}$ for some $p_j \in \mathbb{N}^*$. We define $\Omega := \Omega_1 \times \cdots \times \Omega_d$.

Let $F \in L^2(\Omega_1 \times \cdots \times \Omega_d)$. The function F is said to belong to the **Canonical Polyadic (CP)** format [52, 29, 58] with rank $r \in \mathbb{N}^*$ if it reads as:

$$F(x_1, x_2, ..., x_d) = \sum_{i=1}^{r} u_i^{(1)}(x_1) u_i^{(2)}(x_2) \cdots u_i^{(d)}(x_d) \tag{1.4}$$

for some functions $u_i^{(j)} \in L^2(\Omega_j)$ for $1 \leq i \leq r$ and $1 \leq j \leq d$. The CP decomposition factorizes a tensor into a sum of component rank-one (elementary) tensors.

The *rank* of a tensor is the minimum number of terms in an exact CP. A CP decomposition of a tensor F with $r = rank(F)$ number of components is called the *rank decomposition*. More details and its relation with the uniqueness can be found in [59, 51, 60] and [61].

The main advantage of this decomposition is the low memory cost needed to store it. Indeed, if $\mathcal{N}$ degrees of freedom are used per variable, the storage cost of a general function $F \in L^2(\Omega_1 \times \cdots \times \Omega_d)$ is $\mathcal{O}\left(\mathcal{N}^d\right)$. On the other hand, the storage cost of a CP tensor with rank r reduces to $\mathcal{O}(d\mathcal{N}r)$, which scales linearly in the tensor order d and size $\mathcal{N}$. However, it suffers from several drawbacks. The computation of the canonical rank is an NP-hard problem [62] and the problem of finding a best approximation of a tensor in CP format may be ill-posed [63], i.e. when the problem is written as a minimization problem, a minimizer may not exist; thus the numerical algorithms for computing an approximate representation in such cases might fail. The most classical algorithm in order to compute an approximation of a tensor in the CP format is the so-called Alternating Least Square (ALS) method.

See the details in Algorithm 1. Several researchers have proposed improving ALS with line searches [64, 65], including the ELS [66], which adds a line search after each major iteration that updates all component matrices simultaneously based on the standard ALS search directions. Let us also cite the Modified Alternating Least Square (MALS), [67], where the idea behind is to modify the least squares format used in the ordinary ALS algorithm. In addition, weighted least squares method can be found in the literature, [68, 69, 70, 71].

Algorithm 1 Rank-one ALS greedy algorithm

Require: Prescribed tolerance $\epsilon > 0$, and tensor $W \in L^2(\Omega)$

Output: $N \in \mathbb{N}^*$, for all $1 \leq n \leq N$ and all $1 \leq i \leq d$, $R_i^n \in L^2(\Omega_i)$ so that the CP tensor $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := \sum_{n=1}^{N} R_1^n(x_1) R_2^n(x_2) \cdots R_d^n(x_d) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|_{L^2(\Omega)}^2 \leq \epsilon$.

Set $W^0 = W$, $n = 1$.

while $\|W^{n-1}\|_{L^2(\Omega)}^2 > \epsilon$ **do**

  For all $1 \leq i \leq d$, select randomly $R_i^{n,0} \in L^2(\Omega_i)$ and set $\eta := \epsilon$ and $m = 1$.

  while $\eta > \frac{1}{10}\epsilon$ **do**

    for $j = 1, \cdots, d$ **do**

      Compute $R_j^{n,m} \in L^2(\Omega_j)$ solution to

$$R_j^{n,m} \in \underset{R_j \in L^2(\Omega_j)}{\operatorname{argmin}} \left\| W_{n-1} - R_1^{n,m} \otimes \cdots \otimes R_{j-1}^{n,m} \otimes R_j \otimes R_{j+1}^{n,m-1} \otimes \cdots \otimes R_d^{n,m-1} \right\|_{L^2(\Omega)}^2$$

    end for

    Compute $\eta := \left\| R_1^{n,m} \otimes \cdots \otimes R_d^{n,m} - R_1^{n,m-1} \otimes \cdots \otimes R_d^{n,m-1} \right\|_{L^2}^2$. Set $m := m + 1$.

  end while

  Define $R_i^n = R_i^{n,m-1}$ for all $1 \leq i \leq d$.

  Compute $W_n(x_1, \cdots, x_d) = W_{n-1}(x_1, \cdots, x_d) - R_1^n(x_1) R_2^n(x_2) \cdots R_d^n(x_d)$ for all $(x_1, \cdots, x_d) \in \Omega$.

  $n = n + 1$

end while

$N = n - 1$

The ALS algorithm applied to CP is also used for tensor completion [72], i.e. the case in which the problem of tensor factorization into CP format is applied to incomplete data. For this task, some works are

proposed such as [59, 73, 74].

CP approximations have proven useful in approximating certain multidimensional operators such as the Newton potential (see [75]). CP is also used in data mining and in text analysis, [76].

Nevertheless, this separated representation is not a new proposal. It was first proposed in [77] for performing efficient solutions of complex non-linear thermo-mechanical models and more detailed later in [78] and [79]. Proper Generalized Decomposition is the common name for techniques using such separated representations. The general form of the separated representation of a generic multivariate function $F(x_1, \ldots, x_d)$ in the PGD form reads as in Equation 1.4:

$$F(x_1, \ldots, x_d) = \sum_{i=1}^{N} f_i^1(x_1) \ldots f_i^d(x_d) \tag{1.5}$$

where x_i denotes a scalar or vector coordinate defined in a domain Ω_i of moderate dimension $\Omega_i \in \mathbb{R}^d$. The PGD decomposition is thus a sum of N functional products involving each a number of d functions that are unknown a priori. The case when the function F is known, has been extensively studied over the past years in multilinear algebra as the POD decomposition [80, 81, 82].

The PGD method aims to construct a decomposition of a tensor F. The solution is sought by applying a greedy algorithm (in which every iteration is solved through a fixed point algorithm, see [83]), to the weak formulation of the problem. For each iteration of the algorithm, a mode consisting in a set of numerical values of the functional products $f_i^1(x_1) \ldots f_i^d(x_d)$ of the solution is computed. Each mode enriches the approximation of the solution and it gets more precise with the iterations.

There exist several approaches to the numerical analysis of PGD, [84]. They combine the existence of a best approximation and a greedy algorithm. The idea of using greedy algorithms to construct successive approximations was considered in [85] in the context of the numerical solution of multidimensional PDEs. In [86] the use of a Greedy Rank-One Update Algorithm to construct a rank-r approximate solution for a full rank linear system is proposed. Also, different greedy tensor decompositions have been used in order to compute low-rank approximations, see [87, 88, 89, 90, 91]. The principle of these methods is to approximate a function which depends on a large number of variates by a sum of tensor product functions, each term of which is iteratively computed via a greedy algorithm.

Between the main applications of the PGD method we can find: uncertainty quantification, where sometimes is called Generalized Spectral Decomposition, [92], stochastic parametric analyses and numerical analysis of separated representation and its associated constructors. See more applications and details in [93, 94].

Let us now introduce the concept of *unfoldings* of a tensor. Assume a $N-$th order tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_N}$. The matrix unfolding $X^{(n)} \in \mathbb{R}^{\mathcal{N}_n \times (\mathcal{N}_{n+1} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1})}$ contains the element $x_{i_1 \ldots i_p}$ at the row position i_n and column number equal to:

$$(i_{n+1} - 1)\mathcal{N}_{n+2} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1} + (i_{n+2} - 1)\mathcal{N}_{n+3} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1} + \ldots$$
$$\ldots + (i_N - 1)\mathcal{N}_1 \ldots \mathcal{N}_{n-1} + (i_1 - 1)\mathcal{N}_2 \ldots \mathcal{N}_{n-1} + \ldots + i_{n-1}$$

The *slices* of a tensor are introduced as the two-dimensional sections defined by fixing all the indices but two. In a third order tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3}$, the horizontal, lateral, and frontal slices are denoted by $X_{i_1::}$, $X_{:i_2:}$, and $X_{::i_3}$, respectively. They are matrices, and when we refer to the n-th frontal slice of a third order tensor we refer to its unfolding $X^{(n)}$.

In general, we refer to the column and row vectors of a $N-$th order tensor as its *n-mode vectors* defined as the $\mathcal{N}_n-$vectors obtained by varying the index i_n and keeping the rest fixed. The *n-mode vectors* (or just modes) of a certain tensor X are the column vectors of the unfolding $X^{(n)}$.

We define the *n-rank* of a tensor X, denoted by $R_n = rank_n(X)$ as the dimension of the vector space spanned by the $n-$mode vectors.

The $n-$mode multiplication of a tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_n \times \ldots \mathcal{N}_d}$ by a matrix $A \in \mathbb{R}^{\mathcal{I}_n \times \mathcal{N}_n}$ is defined by

$$[X \times_n A]_{i_1 \ldots i_{n-1} \mathcal{I}_n i_{n+1} \ldots i_d} = \sum_{i_n=1}^{\mathcal{N}_n} X_{i_1 \ldots i_{n-1} i_n i_{n+1} \ldots i_d} a_{\mathcal{I}_n \times i_n} \tag{1.6}$$

leading to a tensor $X \times_n A \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{I}_n \times \ldots \mathcal{N}_d}$. With the $n-$mode product, the POD decomposition introduced in Equation 1.1 is written as: $X = U \Sigma V^T = \Sigma \times_1 U \times_2 V$.

1.1.3 Tucker decomposition

The Tucker decomposition was first introduced by Tucker in 1963 in [95] and then refined in [96, 97]. The Tucker decomposition is a form of higher-order PCA, see [98, 99]. The Tucker model seeks a $d-$dimensional tensor $F \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_d}$ as mode products of a core tensor $G \in \mathbb{R}^{\mathcal{I}_1 \times \ldots \times \mathcal{I}_d}$ and d mode matrices $U^{(n)} \in \mathbb{R}^{\mathcal{I}_n \times \mathcal{N}_1}$ It decomposes a tensor into a core tensor multiplied by a matrix along each mode. For a certain tensor F it reads:

$$F \approx \sum_{i_1=1}^{\mathcal{I}_1} \ldots \sum_{i_d=1}^{\mathcal{I}_d} G_{i_1 \ldots i_d} u_{i_1}^{(1)} \otimes \ldots \otimes u_{i_d}^{(d)} \tag{1.7}$$

where $u^{(j)}$ are the corresponding modes. G is a real $(\mathcal{I}_1 \times \ldots \times \mathcal{I}_d)$-tensor with the property of "all orthogonality" called the core tensor. When $\mathcal{I}_n = \mathcal{N}_n$, for all $n = 1, \ldots, d$, the approximation becomes an equality. The Tucker format [97, 100] is stable but has exponential in d number of parameters, $d\mathcal{I}R + R^d$ (for all equal ranks and degrees of freedom per dimension namely R and $\mathcal{I}$ respectively) parameters while the initial array has $\mathcal{N}^d$ entries. It is suitable for "small" dimensions, especially for the three-dimensional case [101, 102, 103]. Therefore, in some sense Tucker representations suffer from the curse of rank due to the need of storing the core tensor, whose size scales exponentially with the number of dimensions. For large d it is not suitable. The CP decomposition can be viewed as a special case of Tucker where the core tensor is superdiagonal and $\mathcal{N}_1 = \ldots = \mathcal{N}_d$. The Hierarchical Tucker format (H-Tucker), can be seen as a multi-level variant of the Tucker format, it was introduced in [104] as a hierarchical SVD method for tensors of $d \geq 2$.

Tucker representation is often computed by a generalization of the SVD, **Higher-Order SVD (HOSVD)** that produces quasi optimal approximations of the tensor, see [82, 105, 104]. For compression, it is also commonly used the HOSVD:

$$F = \sum_{i_1=1}^{R_1} \ldots \sum_{i_d=1}^{R_d} G_{i_1 \ldots i_d} u_{i_1}^{(1)} \otimes \ldots \otimes u_{i_d}^{(d)} \tag{1.8}$$

in which $R_1, \ldots, R_d$ are the $n-$ranks defined above and they are smaller that $\mathcal{N}_1, \ldots, \mathcal{N}_d$, respectively. See more details in Algorithm 2. The core tensor $G \in \mathbb{R}^{R_1 \ldots R_d}$ is computed by the $n-$mode multiplication defined in Equation 1.6 of the initial tensor and the mode matrices.

Algorithm 2 Truncated HOSVD algorithm

1: **Require:** The tensor $W(x_1, \ldots, x_d) \in L^2(\Omega)$. The set of ranks $\{R_k\}_{1 \leq k \leq d}$.
2: **Output:** The set of truncated HOSVD matrices $\{U_k\}_{1 \leq k \leq d}$. The HOSVD core tensor $G \in \mathbb{R}^{R_1 \ldots R_d}$.
3: **for** $i = 1, \cdots, d$ **do**
4: Compute the SVD of $W^{(i)}$:

$$W^{(i)} = \bar{U}_i \Sigma_i V_i^T$$

5: Set U_i as the R_i first column vectors of $\bar{U}_i$.
6: **end for**
7: Compute the core tensor: $G = W \times_1 U_1^T \ldots \times_d U_d^T$

The Tucker tensors of fixed rank being an embedded manifold provide a stable parametrization, reflected in reliable practical computations. The Tucker-ALS algorithm convergence properties are very satisfactory [106, 99, 98, 107], even if a formal analysis is not yet available.

The principle of these methods is to construct a hierarchy of optimal subspaces that results in a final tensor product of subspaces in which the function F is projected, to define its final approximation. Under strong assumptions on the estimation error made in the determination of subspaces, it is shown in [82] that with a number of evaluations scaling in the storage complexity of the tree-based tensor format, see more

in [108, 109], the approximation satisfies the desired error up to constants depending on some projection operators, which are not quantified.

For d-dimensional tensors with possibly large $d > 3$, a hierarchical data structure, called the Tree-Tucker format, is presented in [110] as an alternative to the canonical decomposition CP. It has smaller or equal number of representation parameters and viable stability properties. The approach involves a recursive construction described by a tree with the leafs corresponding to the Tucker decompositions of three-dimensional tensors, and is based on a sequence of SVDs for the recursively obtained unfolding matrices and on the auxiliary dimensions added to the initial "spatial" dimensions.

Several examples of using the Tucker decomposition in chemical analysis are provided in [111] as part of a tutorial on N-way PCA, also applications in signal processing [112], as well as several tasks in computer vision related with facial expression [113, 114].

1.1.4 Hierarchical Tucker (H-Tucker) format

The Hierarchical Tucker (H-Tucker) format from [104, 115] is a variant of the Tucker format. Hierarchical Tucker representations are an attempt to keep the generality of the Tucker representation but reducing at the same time its complexity, which was greatly affected by the size of the core tensor. In order to define it, we need to introduce a hierarchy between the modes. To do so, some preliminary concepts are needed.

Let us define the indices of a $d-$order tensor as $D := \{1, \ldots, d\}$. And let $t \subset D$, $t \neq \emptyset$.
Definition. A dimension tree or mode-cluster tree T for dimension d is a finite tree with root $Root(T) = D$ and depth $p := \{i \in \mathbb{N}_0 | i \leq log_2(d)\}$ such that each node $t \in T$ is either:

- a leaf and singleton $t = \{\mu\}$, with $\mu \in D$.
- the union of two disjoint successors $S(t) = \{t_1, t_2\}$ and $t = t_1 \dot{\cup} t_2$.

The level l of the tree is defined as a set of nodes having a distance l to the root. We denote the set of nodes in the level l as: $T^l := \{t \in T | \text{level}(t) = l\}$. A node of the tree is called a so-called mode cluster or just cluster (a set of modes). A Canonical dimension tree is a particular case in which each node has two successors.

Definition. Let T be a dimension tree. The hierarchical rank (H-rank) $(r_t)_{t \in T}$, see [116], of a tensor $F \in \mathbb{R}^{\mathcal{N}}$, being $\mathcal{N} := \mathcal{N}_1 \times \ldots \times \mathcal{N}_d$, is defined by:

$$\forall t \in T, \quad r_t := \text{rank}(F^{(t)})$$

the set of tensors of hierarchical rank (nodewise) at most $(r_t)_{t \in T}$ are the so calles H-Tucker tensors and they are denoted by:

$$\text{H-Tucker}(T, (r_t)_{t \in T}) := \{F \in \mathbb{R}^{\mathcal{N}} | \forall t \in T : \text{rank}(F^{(t)}) \leq r_t\}$$

From Definition of the H-rank of a tensor F based on a dimension tree T, one can obtain representation of the tensor. For this, we first notice that a representation matrix unfolding $F^{(t)}$ in the form

$$F^{(t)} = U_t V_t^T$$

is an exact representation of F for every node t of the tree T. Since for $S(t) = t_1, t_2$ the column-vectors $(U_t)_i$ of $U_t \in \mathbb{R}^{\mathcal{N}_t \times r_t}$ fulfil the nestedness property, there exists for every $i \in \{1, \ldots, r_t\}$ a matrix $(B_t)_{i,\cdot,\cdot} \in \mathbb{R}^{r_{t_1} \times r_{t_2}}$ such that

$$(U_t)_i = \sum_{j=1}^{r_{t_1}} \sum_{l=1}^{r_{t_2}} (B_t)_{i,j,l} (U_{t_1})_j \otimes (U_{t_2})_l$$

In the root and the leaves $t = \{\mu\}$, the matrices are small enough to be stored in full tensor format but in the rest of the nodes we need to store sparse matrices. A tensor stored or represented in this form is said to be given in H-Tucker format.

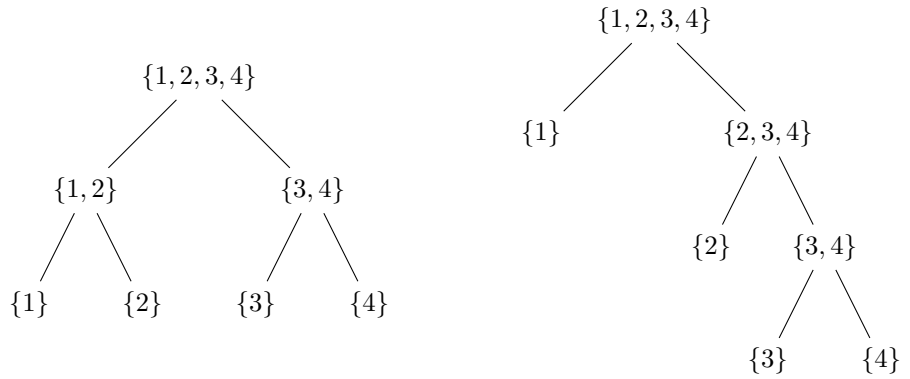


Figure 1.1: In this schematic trees we can see two 4-dimensional examples of dimension trees. In the right, a hierarchical binary tree, H-Tucker. In the left the representation of the TT format as a binary dimension tree.

The computational complexity is in the order of $\mathcal{O}(dr^3 + dNr)$, where $r = \max_{t\in T} r_t$ and $N = \max_{k\in D} \#\mathcal{N}_k$, which improves Tucker's complexity. However, operating with H-Tucker tensors is not always straightforward. For instance, the complexity of the inner product between tensors in H-Tucker format is in the order of $\mathcal{O}(dNr^2 + dr^4)$.

The basic idea of the Hierarchical Tucker decomposition is that a tensor F can be represented in the H-Tucker format if it allows for a recursive construction out of lower-dimensional subspaces. This recursion is completely defined by the dimension tree introduced before. In this framework, a tensor is partitioned into a dimension tree with matrices at its leaves containing the modes of the tensor. These matrices are linked by the so-called transfer tensors, which are located at all inner nodes of the partition tree. Typical choices for such dimension trees are those obtained by a balanced splitting of the index set, as used, e.g., by Grasedyck [104], or a front-to-back splitting which leads to the Tensor Trains (TT) format, [117], proposed by Oseledets and Tyrtyshnikov in [118, 110].

In [119], a projection method in a linear space is proposed. This method constructs the approximation of a function in tree-based tensor format, proposing a strategy to choose the tree structure in order to reduce ranks of the approximation at given precision and therefore its complexity and the required number of samples.

The use of rank-structured set of functions in tree-based tensor formats or tree tensor networks can be seen as a particular class of neural networks. It includes the Tucker format for a trivial tree, the tensor-train format for a linear binary tree [110] and the more general hierarchical format for a general dimension partition tree [115]. Any function in $T^l_{\mathcal{I}}$ admits a multilinear parametrization with parameters forming a tree network of low-order tensors, hence the name tree tensor networks, [120]. Considering dimension trees gives nice topological and geometrical properties [121, 122]. Tree tensor networks are particularly relevant for high-dimensional approximation, because the complexity of the parametrization of a function in tree-based tensor format is linear in the dimension d and polynomial in the ranks [123]. It can also be used for one-dimensional approximation [124, 125].

The tensor tree structure contains the Tucker format and the Hierarchical Tucker format (including the Tensor Train format). In the following scheme, Figure 1.1, we can see a 4-dimensional example of the front-to-back dimension trees representations for H-Tucker format on the left, and on the right special variant of the general H-Tucker format, the so-called Tensor Train (TT) format.

In the literature, some algorithms constructing approximations in tree-based tensor formats using points evaluations of functions have already been proposed. In [123], by using the tree structure and the dimensions of the associated minimal subspaces, it is proved that there exist a set of tensors that can be approximated by a tree-based format with bounded tree-based rank. There are learning approaches that use

random evaluations of the functions [126, 127]. In the numerical experiments, robustness and effectiveness of such algorithms are observed. On the other side, there are algorithms that use adaptive and structured evaluations of tensors for tree-based tensor formats, see [128].

1.1.5 Tensor Train format

Let us now continue presenting the tensor format introduced before as a concrete case of the Tucker decomposition, that is going to be one of the key methods in the contributions of this manuscript. It was introduced in [118]. The same format was introduced in the computational chemistry community under the name Matrix Product States (MPS), [129]. The function F is said to belong to the **Tensor Train (TT)** format with ranks $r_1, \ldots, r_{d-1} \in \mathbb{N}^*$ if and only if

$$F(x_1, x_2, ..., x_d) = \sum_{i_1=1}^{r_1} ... \sum_{i_{d-1}=1}^{r_{d-1}} u_{i_1}^{(1)}(x_1) u_{i_1,i_2}^{(2)}(x_2) u_{i_2,i_3}^{(3)}(x_3) \cdots u_{i_{d-2},i_{d-1}}^{(d-1)}(x_{d-1}) u_{i_{d-1}}^{(d)}(x_d) \tag{1.9}$$

with $u_{i_{j-1},i_j}^{(j)} \in L^2(\Omega_j)$ for $1 \leq i_{j-1} \leq r_{j-1}$ and $1 \leq i_j \leq r_j$ for all $1 \leq j \leq d$ (with $r_0 = r_d = 1$).

It combines two main advantages: On the one hand, it is stable from an algorithmic point of view; on the other, it is computationally affordable provided that the TT ranks of the tensors used stay reasonably small. The ranks characterize the complexity to store a tensor in the TT format. Supposing that $r_1 = \ldots = r_{d-1} = r$, its storage complexity is $\mathcal{O}(dr^2N)$. In [116] bound for these TT ranks are provided.

One of the schemes used for the optimization of a TT tensor with a given rank is the ALS method, see details in [130, 67, 131]. In [67] some numerical examples that concern the stability of the TT decomposition and of ALS are shown as well as how high TT ranks are required during the iterative approximation of low-rank tensors, showing some potential of improvement. These ranks are chosen *a priori* and that makes the method less robust. As we will recall in Algorithm 3, other ways of computing the TT decomposition of a tensor is via the well-known TT-SVD algorithm.

We will also consider three different tensor representations which are based on slight modifications of the TT format, namely the quantized tensor-train format (QTT), (see more in [132]), the block tensor-train format (BTT) (see [133]), and the cyclic tensor-train format (CTT) (see [24]).

Let us make a reference also to the approximation of high-dimensional functions in a statistical learning setting: using model classes of functions in tree-based tensor format. These are particular classes of rank-structured functions that admit explicit and numerically stable representations, parametrized by a tree-structured network of low-order tensors. These nonlinear model classes can be seen as deep neural networks, see [126].

1.1.6 Approximations in local subdomains

Nowadays, high-dimensional problems arise in a wide range of fields with practical interest such as quantum chemistry, molecular dynamics, uncertainty quantification, finance... or mathematical applications that naturally lead to high-dimensional or multivariate approximation problems. One of the main challenges that these high-dimensional and complex systems present is the already introduced curse of dimensionality [4, 134, 5], that refers to the need of using a number of degrees of freedom exponentially increasing with the dimension to describe them.

Dimensionality reduction methods naturally arise from this problem. They consist in finding a compressed representation of a multivariate function or dataset while limiting as much as possible the loss of information. In many applications, high-dimensional data have a low intrinsic dimension or present a certain correlation in the hidden structure of the data such as symmetry, periodicity, etc. In the case in which the method is able to detect these aspects, the curse of dimensionality is alleviated.

A more efficient approach will involve approximating the function locally using a series of subspaces of smaller dimension or with less degrees of freedom on it. This idea of computing local approximations is extensively developed in the field of Model Order Reduction, see Chapter 3 of [7] for more details.

Algorithm 3 TT-SVD algorithm

Require: Prescribed tolerance $\epsilon > 0$, the tensor $W \in L^2(\Omega)$

Output: $K_1, \cdots, K_{d-1} \in \mathbb{N}^*$ TT-ranks, $R_1 \in L^2\left(\Omega_1, \mathbb{R}^{1\times K_1}\right)$, $R_d \in L^2\left(\Omega_d, \mathbb{R}^{K_{d-1}\times 1}\right)$ and for all $i = 2, \cdots, d-1$, $R_i \in L^2\left(\Omega_i, \mathbb{R}^{K_{i-1}\times K_i}\right)$ so that the Tensor Train $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := R_1(x_1)R_2(x_2)\cdots R_d(x_d) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|^2_{L^2(\Omega)} \leq \epsilon^2$.

Define $K_0 := 1$, $D_0 = \{1\}$ and define $\overline{W}_0 \in L^2(D_0 \times \Omega)$ such that $\overline{W}_0(1, \cdot) = W$, $\mathcal{I}_0 := \{1, \cdots, d\}$, $\widehat{\Omega}_0 := \Omega$.

for $j = 1, \cdots, d-1$ **do**

Since $D_{j-1} \times \widehat{\Omega}_{j-1} = (D_{j-1} \times \Omega_j) \times \widehat{\Omega}_j$ with $\widehat{\Omega}_j = \Omega_{j+1} \times \cdots \times \Omega_d$, compute the SVD decomposition of $\overline{W}_{j-1}$ according to the separation of variables $(D_{j-1} \times \Omega_j, \widehat{\Omega}_j)$ so that

$$\overline{W}_{j-1} = \sum_{k\in\mathbb{N}^*} \sigma_{j,k} U_{j,k} \otimes V_{j,k}.$$

Select $K_j \in \mathbb{N}^*$ such that $K_j = \inf\left\{ K \in \mathbb{N}^*, \quad \sum_{k\geq K} |\sigma_{j,k}|^2 \leq \frac{\epsilon^2}{d-1} \right\}$.

Define $D_j := \{1, \cdots, K_j\}$ and define $\overline{W}_j \in L^2\left(D_j \times \widehat{\Omega}_j\right)$ by

$$\overline{W}_j(k_j, y_j) = \sigma_{j,k_j} V_{j,k_j}(y_j)$$

for all $(k_j, y_j) \in D_j \times \widehat{\Omega}_j$.

Define $R_j \in L^2\left(\Omega_j, \mathbb{R}^{K_{j-1}\times K_j}\right)$ as

$$R_j(x_j) = \left(U_{j,k_j}(k_{j-1}, x_j)\right)_{\substack{1\leq k_{j-1}\leq K_{j-1} \\ 1\leq k_j \leq K_j}}$$

for all $x_j \in \Omega_j$.

end for

Define $R_d \in L^2\left(\Omega_d, \mathbb{R}^{K_{d-1}\times 1}\right)$ by

$$R_d(x_d) = \left(\sigma_{d-1,k_{d-1}} V_{d-1,k_{d-1}}(x_d)\right)_{1\leq k_{d-1}\leq K_{d-1}}.$$

The Kolmogorov $N-$widths describe the rate of the worst-case error arising from a projection onto the best-possible linear subspace of dimension $N \in \mathbb{N}$, [135]. It has been proven that for certain linear, coercive, parameterized problems, the Kolmogorov $N-$widths decay exponentially or polynomially with high exponent as N grows, [136]. This extremely fast decay is the base of any model reduction strategy since it guarantees that even choosing a very moderate dimension N, it is possible to achieve good approximations.

In the last years, the problem of finding a dimensionality reduction model has also been considered as a classification problem and, treated like one, studied with supervised learning algorithms, [137, 138].

Projection-based model order reduction consists in computing an approximate solution in a low-dimensional subspace of the solution space, which can give accurate predictions provided that the solution manifold is embedded in a low-dimensional space, see [139]. Projection-based model reduction techniques reduce the computational cost associated by restricting the solution space to a subspace of much smaller dimension. Among them, we can find POD Galerkin method [140] and Reduced Basis method [141, 142, 143]. They enclose the family of models in which the approximated solution is obtained by solving the physics equations with the Galerkin method on a well chosen ROB, [144].

Find in [7] some cases in which the Kolmogorov $N-$widths decay very slowly. In these cases, it may be very difficult to build a linear model that provides a certified approximation for a small tolerance and an affordable numerical complexity. For these classes of problems, an efficient strategy for model reduction requires to look for nonlinear models that capture the geometry of the manifold where the data lies better than linear spaces. In the situations in which we face moderate order functions is sometimes advantageous to apply a nonlinear dimensionality reduction method [145, 146, 147, 148, 149]. Theoretically, this is equivalent to map the data into a higher-dimensional space and then applying POD. Some methods like Kernel PCA (a nonlinear extension of PCA, see [150, 151, 152]) or the use of neural networks, [153, 154] are proposed in the literature for nonlinear problems.

Local reduced basis methods

Most MOR techniques build a single subspace that approximates the solution manifold. However, when the solution manifold is highly nonlinear and as a result a subspace of large dimension is required to approximate it globally, a more efficient approach involves approximating the manifold locally using a series of subspaces of smaller dimension or with less degrees of freedom on it. Each of the subspaces is associated with a local reduced-order basis that can be eventually stored. This methods works in such a way that the parametric domain is partitioned into subregions. Then, for a new parameter value, a local basis can be computed by simply choosing the region where the parameter belongs to. See [155] for a detailed analysis of nonlinear model reduction using local basis.

In [156, 157, 155], the local subspace is selected by finding the closest cluster to the solution. The distance used is the Euclidean distance.

The MOR method introduced in [156] (and applied in [157]) alleviates the problem of the low-efficiency of reduced models in high dimensions through the use of multiple local bases. In this approach, a local ROB is selected at each time step of the ROM simulation based on the current state of the system. Such a concept is particularly well-suited for the POD method in which the basis is built from snapshots of the system taken at various locations of the state-space.

In practice, the set of reduced bases $\{V_j\}_{j=1}^{N_V}$, where N_V is the number of subspaces, is built using a POD algorithm based on the method of snapshots. The proposed method for constructing a database of local bases consists of three steps:

- First, the states are clustered into N_V subsets using an unsupervised learning algorithm, such as the k-means. More important than the choice of algorithm is the choice of distance metric used, the Euclidean distance in this case.
- Second, the clusters are made to overlap with one another by sharing a small number of states between neighboring clusters.
- Third, the snapshots are formed from the state clusters and the individual local reduced order basis are computed.

Transitions between local subspaces require special care and updating the reduced bases associated with each subspace increases the accuracy of the reduced-order model.

Let us recall the already introduced Generalized principal component analysis (GPCA), [47], where the alternative extension of PCA to the case of data lying in a union of subspaces is exploited. Subspace segmentation is a fundamental problem in many applications. If the segmentation of the data was known, one could simply assign a subspace to each set of points using PCA (although its effectiveness is limited by its global linearity). On the other hand, if the subspaces and its basis were known, the problem reduces to find the points that best fit the subspace. Normally, neither the subspaces or the segmentation of the data are known, most existing methods randomly choose a base for each subspace and then iterate between data segmentation and subspace estimation. In the literature, it has been done as K-subspaces, in [158], the extension of K-means to the case of subspaces; subspace growing or subspace selection, [159]; or Expectation Maximization ([160]) fore mixtures of PCAs, see more in [161].

Splitting a non-reducible problem into multiple reducible ones can be achieved with cluster analysis. Cluster analysis belongs to unsupervised learning tasks and can be defined as the search of groups (or clusters) of similar objects in a database. The choice of the clustering algorithm depends on the underlying motivation and thus on the clusters topological properties that are expected.

1.2 State of the art of clustering methods

Clustering is a common technique for statistical data analysis, which is used in many fields, including machine learning, data mining, pattern recognition, image analysis between others. This reflects its broad appeal and usefulness as one of the steps in exploratory data analysis. Clustering is the process of grouping similar objects into different groups, or more precisely, the partitioning of a data set into subsets, so that the data in each subset according to some defined distance measure. Clustering problem is not a trivial task, especially in the case of high-dimensional data that is exactly the case that we face in most applications, and it is where conventional methods usually fail. The number of data required to define correctly the system increases exponentially with its dimension. This phenomenon is referred to as the already introduced curse of dimensionality. The difference with classification tasks is that the set of categories (or clusters) is not known a priori. Hence, clustering methods are included in the family of unsupervised learning models, where the data to be classified are not labeled. Dimensionality reduction methods are also included.

Therefore, clustering can simply be defined as the task of grouping entities in terms of a similarity measure. Here, the critical issue is to understand what is meant by “similar”. Similarity is in a sense the inverse of a distance metric between two entities. The shorter the distance, the more similar the entities, and vice versa. It is important hence to note that, clustering results will be crucially dependent on the similarity notion chosen. Determining the distance between both clustering solutions is non-trivial and still subject to discussion. A conventional distance metric is the Euclidean distance. Deriving the Euclidean distance between two data points involves computing the square root of the sum of the squares of the differences between corresponding values. Many other similarity measures, e.g., [162, 163], could be used to tackle the broad range of domain specific clustering problems.

Clustering methods are usually categorized under four main groups: hierarchical, partitioning, overlapping, and ordination algorithms.

- The first group is based on the cluster formation methodology. Hierarchical algorithms can be agglomerative (bottom-up) or divisive (top-down), and this choice is done via analytic optimization techniques (see more in [164]). Agglomerative algorithms begin with each element as a separate cluster and merge them in successively larger clusters while divisive algorithms starts from large clusters and splits it succesively in smaller ones.

- A second group lists methods depending on the cluster model acquired such as hierarchical [165], representative (such that each cluster is associated to a partitioning representative, i.e. a reference

point that well represents the cluster's members, they include k-means [166, 167] when the optimal representatives are the clusters means or centroids; k-medians [168] when the optimal representatives are the medians, k-medoids [169] when the optimal representatives are the medioids, in this case the representatives must be taken among the elements of the dataset; and many variants of k-means), distribution such as expectation maximization, density [170], subspace, group, and graph-based models, [171].

- Thirdly, depending on the relationship type between entities and clusters, hard or soft clustering can be distinguished by defining binary or fuzzy relations, respectively.

- A final clustering group, based on the nature of cluster-cluster relations, defines the distinction between overlapping versus disjoint partition groups in general.

Let us make a remark on hierarchical clustering. Hierarchical techniques produce a nested sequence of partitions, with a single, inclusive cluster at the top and singleton clusters of individual points at the bottom. Each intermediate level can be viewed as combining two clusters from the next lower level (or splitting a cluster from the next higher level). The result of a hierarchical clustering algorithm can be graphically displayed as tree. This tree graphically displays the merging process and the intermediate clusters. For document clustering, this tree provides a hierarchical index. The two basic approaches to generate a hierarchical clustering:
a) Agglomerative: Start with the points as individual clusters and, at each step, merge the most similar or closest pair of clusters. This requires a definition of cluster similarity or distance.
b) Divisive: Start with one, all-inclusive cluster and, at each step, split a cluster until only singleton clusters of individual points remain. In this case, we will need to decide, at each step, which cluster to split and how to perform the division.

Hierarchical method suffers from the fact that once the merge/split is done, it can never be undone. This rigidity is useful in that it leads to smaller computation costs by not worrying about a combinatorial number of different choices. Divisive algorithms begin with the whole set and proceed to divide it into successively smaller clusters.

See [172, 173], for more information of families of clustering methods. The clustering process is considerably more complicated when dealing with clusterings of overlapping clusters. In this work we are going to restrict ourselves to the situation in which they are not overlapping. More information and an analysis of this situation can be found in [174]. The validation of such algorithms refers to the problem of determining the ability of the methods to recover cluster configurations which are known to exist in the data. Validation approaches include mathematical derivations, analyses of empirical datasets, and Monte Carlo simulation methods.

Some unsupervised learning tasks such as clustering and dimensionality reduction can be seen as data mining tasks. Dimensionality reduction is used for the construction of reduced-order models, the compression of simulation data, and the reduction of the number of variables to be processed by some predictive models; cluster analysis is used for the identification of groups of data with similar physical or mechanical behaviors, enabling the construction of dictionaries of cluster-specific models.

See [175] for an extensive study in model order reduction methods and its machine learning-based variants (ROM-net) and applications. In this work, local ROMs are combined with a classifier for automatic model recommendation. The classifier recommends a suitable reduced-order model. Nevertheless, as the classes are given by a clustering algorithm the computational cost is very high. The time-consuming operations required for model selection are avoided by replacing by an approximation the theoretical perfect classifier.

1.2.1 Physics-informed clustering

Physics-informed clustering strategies were firstly introduced in [176, 177]. The focus is on finding a clustering strategy that is appropriate for model order reduction purposes. Physics-informed cluster analysis consists in clustering the parameter space by means of a dissimilarity measure which involves physical

quantities obtained when solving the problem. In practice, this means that a clustering algorithm is applied in the solution space.

When using a clustering algorithm to compute partitions of the solution manifold, the quality of the partition is related with the choice of the clustering method and the measure used to group similar solutions on the manifold. Among physics-informed clustering strategies, it is common to use K-means algorithm (introduced in what follows), that measures similarities with Euclidean distances in the solution space or in a subspace of the solution space found by PCA. In [178], it was noticed that clustering based on the Euclidean distance in the solution space was not adapted for the construction of local ROMs, which led to the definition of projection-error based local ROMs where the solution space is hierarchically partitioned using the projection error as a dissimilarity criterion. In [176] an alternative based on the use of relative errors is proposed. In other works like [138] and [179], the Grassmann distance between subspaces is used.

1.2.2 Other classification methods

Many algorithms exist for nonlinear classification problems, each of them having its own advantages and drawbacks. Naive Bayes classifiers [180] are well-known for their interpretability as well as Decision trees [181]. Decision trees are supervised learning algorithms that model the correspondence between inputs and outputs by means of simple decision rules, leading to a recursive partition of the input space. They are known to be interpretable in the sense that classification rules can be visualized as paths in a tree structure.

Other nonlinear classifiers include the k-nearest neighbors algorithm (kNN [182]) and quadratic discriminant analysis. See more detailed information in [183, 184]. In [185] a greedy method to reduce dimensions in classification problems is investigated. The k-nearest neighbors method was first introduced in [186], and more developed in [187] and [188]. The kNN classifier [182] belongs to the family of instance-based learning algorithms, which store training data in memory and compare test data with training examples to make predictions. The label for a given observation is obtained by a majority vote among the k-nearest training examples, understanding nearest as the smallest Euclidean distance.

K-means algorithm is one of the best-known and simplest clustering algorithms, it is based on kNN method and it is mostly applied to solve classification problems, grouping the given data in a certain number K of clusters defined a priori.

1.3 State of the art of Deep Learning-based Schemes

Neural networks produce structured parametric families of functions that have been studied and used for almost 70 years. In the last several years, however, their popularity has surged as they have achieved state of the art in large variety of problems. We have seen an increase in the application of Deep Neural Networks (DNNs) in a wide range of disciplines. In scientific computing, computational experiments with DNNs for the numerical solution of Partial Differential Equations (PDEs) have been reported to be strikingly successful in a wide range of applications [189, 190, 191, 192]. Moreover, deep learning-based reduced order models (DL-ROMs) have been recently proposed to overcome common limitations shared by conventional reduced order models, [193]. All these encouraging results show that DNNs are an interesting candidate for efficiently emulating other (linear or nonlinear) approximation methods.

Underlying this success, deep learning based schemes are a way to approximate functions. They embrace schemes from an additive construction commonly used in approximation theory to a compositional construction used in deep neural networks. The compositional construction seems to be particularly powerful in high dimensions. This suggests that deep neural network based models can be an interesting choice in function approximation tasks. This includes solving partial differential equations [194], molecular modeling [195] and as mentioned before, model reduction [138], among others.

The idea of working with neural network functions to solve PDEs is by far not novel, and countless contributions have been proposed on this front in recent years, [194, 191, 189].

In high dimensional equations, the number of degrees of freedom grows exponentially fast as the dimension of PDE increases, hence we face the curse of dimensionality. One striking advantage of DNNs over classical numerical methods is that the number of degrees of freedom "only" grows (at most) polynomially. Therefore, DNNs are particularly suitable for solving high-dimensional PDEs, see [196, 197, 191, 198].

Neural networks produce certified approximations of continuous functions, [199, 200], in the sense that given a function f and a prescribed tolerance $\varepsilon > 0$, a neural network (with the right parameters) produces an approximation of f within an error smaller or equal than ε, [201].

The motivation to study the approximation of parametric PDEs by DNNs stems from the similarities between parametric problems and statistical learning problems. Assume given domains $\mathcal{X} \subset \mathbb{R}^{d_\mathcal{X}}$, where $d_\mathcal{X} \in \mathbb{N}$ and $\mathcal{Y} \subset \mathbb{R}^{d_\mathcal{Y}}$, $d_\mathcal{Y} \in \mathbb{N}$. Let $\mathcal{X}$ to be the vector space of all possible inputs and $\mathcal{Y}$ the vector space for the possible outputs. Further assume that there exists an unknown probability distribution ρ on $\mathcal{X} \times \mathcal{Y}$. Let us call loss function to $\mathcal{J} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^+$. It measures (or penalizes, that's why it is called loss) the disparity between the estimated values and the true values for an instance of the data.

The goal of a statistical learning problem is to find a function $f : \mathcal{X} \to \mathcal{Y}$ such that $f(x) \sim y$ and that minimizes the loss function $\mathcal{J}(f(x), y)$

$$\min_{f \in \mathcal{Y}} \mathcal{J}(f(x), y) \tag{1.10}$$

Since the probability measure is unknown, we have no direct access to the expected loss. Instead, we assume that we are given a set of training data, i.e., N pairs of points $(x_i, y_i)_{i=1}^N$, $N \in \mathbb{N}$, called collocation points and distributed independently with respect to ρ. Then one finds f by minimizing the so-called empirical loss function:

$$\min_{f \in \mathcal{Y}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{J}(f(x_i), y_i) \tag{1.11}$$

We will call optimizing the empirical loss to the learning procedure. The approximation model provides a solution at a given collocation point. It will be detailed later on.

1.3.1 Deep learning scheme

Let us start by recalling basic concepts and notions related with neural networks. We begin by introducing feedforward neural networks and their elementary properties. The feedforward neural networks are those whose connections between neurons go from the input layer to the output layer without forming loops in the hidden layers (see [202] for general references). Let us have the case of a classical multilayer perceptron, that is a feedforward network composed of fully connected layers. While the latter networks are generally not the architecture of choice in most targeted applications, their architecture provides the most convenient way to understand the trade-offs between approximation efficiency and the complexity of the network.

Let $\mathcal{X} \subset \mathbb{R}^{d_\mathcal{X}}$ and $\mathcal{Y} \subset \mathbb{R}^{d_\mathcal{Y}}$ be some input and output sets of finite dimensions $d_\mathcal{X}, d_\mathcal{Y} \in \mathbb{N}^*$. A feedforward neural network ψ is a function

$$\psi : \mathcal{X} \to \mathcal{Y}$$

which reads as

$$\psi(x) = T_L(\sigma(T_{L-1}(\sigma(\dots \sigma(T_0(x))))), \quad \forall x \in \mathcal{X}, \tag{1.12}$$

being, for every $\ell \in \{0, \dots, L\}$,

$$T_\ell : \left\{ \begin{array}{ccc} \mathbb{R}^{p_\ell} & \to & \mathbb{R}^{p_{\ell+1}} \\ x_\ell & \mapsto & T_\ell(x_\ell) = A_\ell x_\ell + b_\ell \end{array} \right. \tag{1.13}$$

an affine function which can be expressed through a matrix $A_\ell \in \mathbb{R}^{p_{\ell+1} \times p_\ell}$, and an offset vector $b_\ell \in \mathbb{R}^{p_{\ell+1}}$. $\sigma : \mathbb{R} \to \mathbb{R}$ is called the (nonlinear) activation function. The smoothness of the activation function plays a key role in the accuracy of the optimization part of the algorithm, [26]. In [194] it is stated that, under some technical assumptions, there exists a DNN that approximates the discretized solution map.

Note that since ψ maps $\mathcal{X}$ onto $\mathcal{Y}$, it is necessary that the number of neurons in the input layer is equal to the input dimension, $p_0 = d_{\mathcal{X}}$, and equivalently in the output layer, where $p_{L+1} = d_{\mathcal{Y}}$. The layers numbered from 1 to L are usually called the hidden layers of the neural network.

The main component of the neural network based methods is a nonlinear transformation with respect to its parameters. To define a class of feedforward neural networks, its architecture needs to be fixed by prescribing a given activation function σ, depth $L \in \mathbb{N}$, and layer widths $\boldsymbol{p} = (p_0, \dots, p_{L+1}) \in (\mathbb{N}^*)^{L+2}$. Once the values of σ, L and $\boldsymbol{p}$ have been chosen, the coefficients $(A_\ell, b_\ell)_{0\leq\ell\leq L}$ of the affine mappings $T_0, \cdots, T_L$ are viewed as parameters. We gather these coefficients in the vector of parameters, typically known as the weights in the neural network and named θ, that helps to define the transformation from x_ℓ to $T_\ell(x_\ell)$. They read:

$$\theta = \{(A_\ell, b_\ell)\}_{\ell=0}^{L},$$

and assume that θ takes values in a set

$$\Theta \subseteq \bigtimes_{\ell=0}^{L} \left(\mathbb{R}^{p_\ell \times p_{\ell+1}} \times \mathbb{R}^{p_{\ell+1}}\right).$$

For any $\theta \in \Theta$, we define by $\psi_\theta : \mathcal{X} \to \mathcal{Y}$ the function ψ with $\theta = \{(A_\ell, b_\ell)\}_{\ell=0}^{L} \in \Theta$.

The class of neural network functions with architecture $(\sigma, L, \boldsymbol{p})$ and coefficient sets Θ is then defined as

$$\mathcal{N}(\sigma, L, \boldsymbol{p}, \Theta) := \{\psi_\theta : \theta \in \Theta\}. \tag{1.14}$$

Let us name $\mathcal{K}$ the set of neural networks for concrete values of the parameters, $\mathcal{K} \subset \mathcal{N}$. In the following scheme, Figure 1.2, let us show the possible architecture of a feedforward neural network with one hidden layer composed by n neurons and input and output dimension $d_{\mathcal{X}}$ and $d_{\mathcal{Y}}$, respectively.

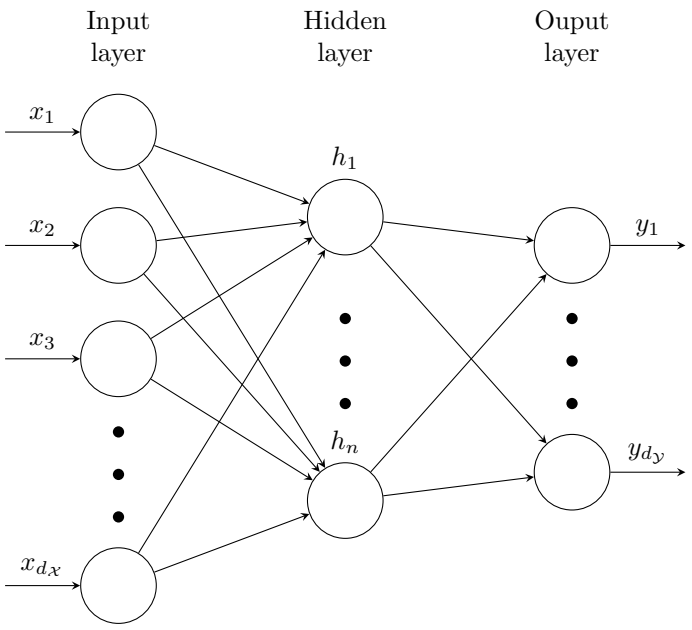


Figure 1.2: Schematic example of the architecture of a feedforward neural network with one hidden layer composed by n neurons and input dimension $d_{\mathcal{X}}$ and output dimension $d_{\mathcal{Y}}$.

Having made the choices of architecture and activation function of the network, the problem of design an approximation method for a target function f can be reduced to the one of choosing parameters θ.

Nevertheless, one must first specify the activation functions σ and initialize the parameters θ. The proper choice of initialization is important, because one tends to run into the problem of vanishing gradients whenever initialization is done poorly. Vanishing gradients occur when activation become saturated and thus

these nodes no longer contribute to training, i.e. when the inputs x to the activation function $\sigma(x)$ become so small, or large, that its gradient becomes vanishingly small. Activation functions introduce non-linearities into the neural network. Common activation functions include tanh, ReLU, swish and sigmoid. A deeper study of the use of the ReLU activation function is done in [26, 203, 204, 205].

1.3.2 Physics-Informed Neural Networks (PINNs)

Being motivated by solving PDEs in the forward and inverse sense in regimes where data is sparse, combining the generality and power of machine-learning techniques with physics and domain knowledge has been deeply studied. To this end, Physics-Informed Neural Networks (PINNs, [206]) were developed. The method of PINNs is based on embedding prior knowledge about the physical systems into a neural network. The major innovation with PINNs is the introduction of a residual network that encodes the governing physics equations, takes the output of a deep-learning network, and calculates a residual value in order to minimize the loss function. In such a way, it is possible to build the space of possible solutions only with the solutions which we are interested in. Then, the information that the model obtains from the available data is magnified. This is done through a regularizer in the form of the governing equations which enforces the physical laws into the solution. As such, the model is not only more efficient with the dealt data, but also it is able to infer the solution by using the physical laws. This is especially useful in the instances where collecting the data is difficult or expensive.

PINNs methods approximate PDE solutions by training a neural network to minimize a loss function (mostly based on residuals of the equations); it includes terms reflecting the initial and boundary conditions along the space-time domain's boundary and the PDE residual at selected points in the domain (collocation points). The method retrieves the approximated function evaluated on the collocation points. One finds the coefficients of the neural network solution by minimizing a discretized version of the $L^2(\Omega)$ norm of the strong form of the residual of the PDE. This is normally done by Gradient Descent (GD) based algorithms. One of the main advantages of this method is that it is very easily implementable but a lot of regularity in the solutions is implicitly assumed.

The basic concept behind PINN training is that it can be thought of as an unsupervised strategy that does not require labelled data, such as results from prior simulations or experiments. The PINN algorithm is essentially a mesh-free technique that finds PDE solutions by converting the problem of directly solving the governing equations into a loss function optimization problem. It works by integrating the mathematical model into the network and reinforcing the loss function with a residual term from the governing equation, which acts as a penalizing term to restrict the space of acceptable solutions.

Beyond the initial collocation version of PINNs expressed above, these methods to conservative PINNs (cPINNs, [207]), variational PINNs (vPINNS, [208, 209, 210]), stochastic PINNs (sPINNs, [211]) between the developed variants.

Neural networks require a large amount of training data to perform well. For example, the image classification problem can only really be solved effectively by gathering sufficiently many labelled image examples for the neural network to train on. With today's powerful and efficient computers, training the neural network does not constitute a huge hurdle anymore, even with large data-sets and large networks. However, for many scientific applications, there can be a great difficulty and expense in gathering such great amount of data, especially for large scale systems. In those cases, data may be quite sparse in both space and time and it may be corrupted by noise. Hence, obtaining a good performance may be prohibitively expensive.

Physics-informed neural networks can address problems that are described by few data, or noisy experiment observations. They can use known data while adhering to any given physical law specified by general nonlinear partial differential equations, PINNs can also be considered as neural networks that deal with supervised learning problems. Let a differential equation on its more general form be:

$$\begin{aligned} R(u(x), y) &= f(x), \quad x \in \Omega \\ S(u(x)) &= g(x), \quad x \in \partial\Omega \end{aligned} \tag{1.15}$$

Where $x := [x_1, \ldots, x_{d-1}, t]$ indicates the space-time coordinate vector. The function u represents the unknown solution, y are the parameters related to the physics, f is the function identifying the data of the problem and R is the non linear differential operator. We denote S as the operator indicating arbitrary initial or boundary conditions related to the problem and g the boundary function.

In classical numerical methods, boundary conditions can be exactly enforced for mesh points at the boundary. Typically boundary conditions include Dirichlet, Neumann, Robin, and periodic boundary conditions [212]. However, it is very difficult to impose exact boundary conditions for a DNN representation. Therefore, in the loss function, it is common to add a penalty term which penalizes the difference between the DNN representation on the boundary and the exact boundary condition, typically in the sense of L^2 norm.

PDEs can describe numerous physical systems including both forward and inverse problems. The goal of forward problems is to find the function $u(x)$ for every x, where y are specified parameters.

The forward and inverse PINNs wish to solve for two very different problems, but there are only very slight differences between how the two formulations are constructed. Recall that the forward problem is concerned with solving for u when the PDE and initial/boundary conditions are fully specified. The inverse problem however, is concerned with using data measured from the approximation of the solution and then estimating the unknown parameters in the PDE. The main difference comes from how the supervised loss function is defined, and which parameters we set as trainable variables. In this work, we are working with forward PINNs.

The usage of a DNNs to solve PDE can be summarized on three main parts:

1. Assembly of the loss function.
2. Neural network structure.
3. Solving the discretized minimization problem.

In the following, we introduce two of the most popular different methods to solve PDE's with the PINN method. The Deep Ritz Method and the Deep Galerkin Method.

The Deep Ritz Method

Let us consider the boundary value problem over a bounded domain $\Omega \subset \mathbb{R}^d$ introduced in Equation 1.15. To proceed, we assume that the problem is well-posed. The basic idea of solving a PDE using DNNs is to seek an approximated solution represented by a DNN in a certain sense [200].

Let us start by recalling the Deep Ritz Method, [190], named so since it is based on using the neural network representation of functions in the context of the Ritz method, [213]. The Deep Ritz method is naturally nonlinear, naturally adaptive and its moderate computational cost gives the method the potential to work in rather high dimensions [214, 215, 216]. The framework is quite simple and fits well with the stochastic gradient descent method used in deep learning. It leverages the fact that the solution of certain PDEs is the unique minimizer to a certain energy functional.

The method is based on the following set of ideas:

- Deep neural network based approximation of the trial function.
- A numerical quadrature rule for the functional.
- An algorithm for solving the final optimization problem.

Any neural-network based numerical scheme for the solution of the kind of problems we are interested in relies on the use of a variational formulation of this problem which let us write the approximation of the solution u (or another function defined from u) as the solution of the minimization problem

$$u = \arg\min_{v \in \mathcal{V}} \mathcal{J}(v), \tag{1.16}$$

where $\mathcal{V}$ is a particular set of real-valued functions defined on a certain domain Ω.

The Ritz approximation, is defined by a solution u_k such that for all $v_k \in \mathcal{V}_k \subset \mathcal{V}$, being $\mathcal{V}_k$ the set of admissible functions, it satisfies Equation 1.15.

The minimization problem set in Equation 1.16 reads:

$$u_k = \arg \min_{v_k \in \mathcal{V}_k} \mathcal{J}(v_k), \tag{1.17}$$

and it is proved to converge. Hence, the loss function associated to Equation 1.15 reads:

$$\mathcal{J}(v_k) = \int_\Omega |W(v_k) - f(x)v_k|^2 dx$$

where we have named $W(\cdot)$ to the variational expression of the left-hand-side of the given PDE, $R(v_k)$, and $\mathcal{V}_k$ denotes the space of functions considering the boundary conditions.

Following the work of Raissi et al. [217], we approximate the solution of a PDE $u(x,t)$ with a neural network parametrized by θ and with input variables (x,t). From now on, let us call it $\psi_\theta(x,t)$. The form of this parametrized function is directly related to the architecture of the network and the activation functions used. Recall that in the PINN case, we are required to use activation functions which can be differentiated more than once. In some contemporary papers [218, 219] the $\tanh(x)$ activation function was used.

Let the test function be now a NN, $\psi_\theta(x)$, the loss function associated to the bulk term reads:

$$\mathcal{I}(\psi_\theta) = \int_\Omega |W(\psi_\theta) - f(x)\psi_\theta|^2 dx$$

Where $W(\psi_\theta)$ is the variational expression of the left-hand-side part of the PDE, $R(\psi_\theta)$. DRM works in a weak sense that the variation of $\mathcal{I}(\psi_\theta)$ with respect to ψ_θ yields the associated Euler-Lagrange equation $R(\psi_\theta) = f(x)$.

Since the NN doesn't necessarily belong to the space and it may not satisfy the boundary conditions, we impose them by adding them as a penalty term in the loss function

$$\mathcal{B}(\psi_\theta) = \int_{\partial\Omega} |Z(\psi_\theta) - g(x)\psi_\theta|^2 dx$$

Where $Z(\psi_\theta)$ is the variational expression of the left-hand-side part of the boundary conditions of the PDE, $S(\psi_\theta)$.

And then the total loss function reads:

$$\mathcal{J}(\psi_\theta) = \mathcal{I}(\psi_\theta) + \lambda\mathcal{B}(\psi_\theta) \tag{1.18}$$

where $\lambda \in [0,1]$ is the penalization parameter. The optimal solution for the PDE is given by the solution of the minimization problem introduced in Equation 1.19.

Then, the problem of Equation 1.16 in the case in which the minimization process is computed by a Neural Network, often reads as:

$$\theta^* = \arg\min_{\theta\in\Theta} \mathcal{J}(\psi_\theta) := \frac{1}{N}\sum_{i=1}^{N} \mathcal{J}_i(\psi_\theta) \tag{1.19}$$

where θ^* denotes the parameters of the NN that provide the approximation and each term at the right-hand side corresponds to one data point $(x_i, y_i)_{i=1}^N$, $N \in \mathbb{N}$. Conjointly, $\mathcal{J}_i(\psi_\theta) := \mathcal{J}(\psi_\theta(x_i), y_i)$.

The minimization of the loss function evaluated in the collocation points $(x_j, y_j)_{j=1}^M$, for $M \in \mathbb{N}$, will provide the values for the parameters that we await. In this expression, N, the number of data points, and M the number of evaluation points, are typically very large. Quadrature schemes for the high-dimensional

integral run into the curse of dimensionality, and Monte-Carlo method can overcome this issue.

In order to solve the minimization problem defined in Equation 1.19, we make use of the Stochastic Gradient Descent algorithm (SGD, [220, 221]). The SGD algorithm and its variants play a key role in deep learning training. It is a first-order optimization method which naturally incorporates the idea of Monte-Carlo sampling and thus alleviates the curse of dimensionality. At each iteration, SGD updates the neural parameters by evaluating the gradient of the loss function at a batch of samples $P \in \mathbb{N}$, $P \leq N$, as:

$$\theta^{k+1} = \theta^k - \epsilon_k \frac{1}{P} \sum_{i=1}^{P} \nabla_\theta j_i(\theta^k) \tag{1.20}$$

Here θ^k are the parameters of neural network at the k-th iteration, ϵ_k is the learning rate, and $j_i(\theta^k)$ is used to approximate the loss function. The points x_i are randomly generated with uniform distribution over the domain Ω and the boundary domain $\partial\Omega$. This is the stochastic version of the gradient descent algorithm (GD). The key idea is that instead of computing the sum when evaluating the gradient of $\mathcal{J}$, we simply randomly choose one term in the sum. Even though SGD requires more iterations to converge than GD, each iteration is much less computationally expensive.

Other strategies where less regular solutions are allowed are proposed in the literature. Usually, they start from its variational formulation. Let us cite the deep Galerkin method (DGM, [222]) that is based on a least-squares formulation, and the vPINNs (mentioned above), based on the Galerkin method. In this kind of methods, the quality of the approximation depends on the architecture of both the trial and the test neural network classes. In addition, numerous evaluations for multiple test functions need to be performed. In what follows, a brief introduction of Deep Galerkin Method is done in order to see other way to approximate the solution.

The Deep Galerkin Method

The Deep Galerkin Method, was proposed by Sirignano and Spiliopoulos [192]. The Galerkin method is a widely used computational method which seeks a reduced-form solution to a PDE as a linear combination of basis functions. The deep learning algorithm, DGM, uses a deep neural network instead of a linear combination of basis functions. The deep neural network is trained to satisfy the differential operator, initial condition, and boundary conditions using stochastic gradient descent at randomly sampled spatial points. In this case, the loss function is defined as the PDE residual in the least-squares sense.

In this case, the DNN approximation is substituted directly in the strong form of the equation 1.15, $R(\psi_\theta) = f(x)$. The loss function associated to the bulk term is:

$$\mathcal{I}(\psi_\theta) = \int_\Omega |R(\psi_\theta) - f(x)|^2 dx$$

DGM aims to minimize the imbalance when the approximate DNN solution is substituted into the strong form in the least-squares sense.

The inclusion of the boundary conditions in the loss function is done by adding the following penalty term. In an analogous way, the DNN is substituted in the strong form of the boundary conditions.

$$\mathcal{B}(\psi_\theta) = \int_{\partial\Omega} |S(\psi_\theta) - g(x)|^2 dx$$

And then the total loss function reads:

$$\mathcal{J}(\psi_\theta) = \mathcal{I}(\psi_\theta) + \lambda \mathcal{B}(\psi_\theta) \tag{1.21}$$

where $\lambda \in \mathbb{R}$ is the penalization parameter. The optimal solution for the PDE is given by the solution of the minimization problem introduced in Equation 1.19.

Then, the problem of Equation 1.16 is solved then in an analogous way as in DRM and the minimization process is solved by SGD method of Equation 1.20.

The highest derivative in the loss function in DRM is lower than that in DGM, thus it is thought that DGM works better for smooth solutions while DRM works better for low-regularity solutions. However, DRM can outperform DGM with a clear dependence of dimensionality even for smooth solutions and DGM can also outperform DRM for low-regularity solutions, [223].

1.4 Contributions of the thesis

In this work, the aim is to find numerical methods in order to approximate multivariate functions. They can be derived from datasets, multiway arrays, or solutions of PDEs coming from physics, biology, economy or other disciplines.

The task of approximating multivariate functions is done from three different perspectives in this thesis. The original contributions are the following:

1. A tensor approximation method, Sum of Tensor Trains (SoTT), that retrieves the approximated solution in a sum of TTs format.

2. A local tensor approximation method based on clustering, that retrieves an approximated solution in the introduced local HOSVD format.

3. A deep learning-based method that computes the approximation of the solution of a convection diffusion PDE when the diffusion parameter is very small.

Find in the following a brief overview of the contributions that can be found in the manuscript.

1.4.1 Contribution on Tensor Formats: The Sum of Tensor Trains method

In the first chapter of the thesis, we propose a numerical scheme, **Sum of Tensor Trains (SoTT)**, that constructs an approximation of a tensor as a sum of tensor train decompositions, [224].

Among the different existing tensor formats in the literature, two of them present specific importance with respect to applications: the Canonical Polyadic (CP) and Tensor Train (TT) format. As we can see in Equation 1.4, the CP format presents a low memory cost but may be ill-posed and present instabilities. On the other hand, recalling Equation 1.9, the TT format is stable from an algorithmic point of view and it is computationally affordable (provided that the TT ranks of the tensors remain reasonably small) but the order of the variables needs to be fixed a priori.

The aim of this work is to find a method that is efficient in memory and that doesn't fix a priori the order of the variables or the ranks. This is done in such a way that the resulting approximation presents a Sum of Tensor Trains (SoTT) format.

The advantages of this procedure are:

- It selects in an adaptive way the order of the variables in each term so as to obtain favorable compressing rates with respect to pure TT approximations with an a priori prescribed order of variables.

- When the values of the ranks of the terms are fixed to be equal to one, the procedure provides a new scheme for the computation of a CP approximation of a given tensor (namely CP-TT), which appears to be more efficient than ALS for high-order tensors.

In the proposed SoTT method, the order of the variables and the ranks are greedily chosen by the method and they can vary from one term to another. They are the result of an optimization step based on the idea of an energy-balance between the accuracy of the approximation of a tensor and the computational cost needed to store it. This is done by defining a certain minimization problem in which the functional to minimize has two terms, one represents the memory spent to store the approximation and the other one is

related with the energy (information) kept from the system.

The main goal of the SoTT algorithm is to compute, after n iterations, an approximation of a certain tensor F as a sum of n TTs. At iteration n, the SoTT computes an approximation of F under the form

$$F \approx \widetilde{F}^{n-1} + R_1^n(x_{\tau^n(1)})R_2^n(x_{\tau^n(2)}) \cdots R_d^n(x_{\tau^n(d)}) \tag{1.22}$$

where $\widetilde{F}^{n-1}$ is the approximation obtained after $n-1$ iterations of the algorithm, $\tau_n \in \{1, \ldots, d\}$ denotes the permutation of the variables on the iteration n and, for all $1 \leq j \leq d$, let $R_j^n \in L^2\left(\Omega_{\tau^n(j)}, \mathbb{R}^{K_{j-1}^n \times K_j^n}\right)$, be the modes obtained in the SVD of the tensor:

$$R_j^n(x_{\tau^n(j)}) = \left(U_{j,k_j}^{\tau_n(j),n}(k_{j-1}, x_{\tau^n(j)})\right)_{1 \leq k_j \leq K_j^n, 1 \leq k_{j-1} \leq K_{j-1}^n}$$

where $K_0^n = K_d^n = 1$.

The aim of the n^{th} iteration is to choose the permutation τ_n and the values of the ranks $(K_j^n)_{1 \leq j \leq d-1}$ in an appropriate way, which is done here using a greedy procedure.

Fixing the ranks as one in all the steps of the algorithm, a CP approximation of the given tensor is obtained. We name this rank-1 particular instance as the **Canonical Polyadic - Tensor Train (CP-TT)** method. As an output, after n iterations of the CP-TT algorithm, the method greedily produces an approximation of the tensor F under the CP format

$$F \approx \sum_{k=1}^{n} R_1^k(x_{\tau_k(1)}) \cdots R_d^k(x_{\tau_k(d)}) \tag{1.23}$$

where for all $1 \leq k \leq n$ and all $1 \leq i \leq d$, $R_i^k \in L^2(\Omega_{\tau_k(i)})$.

We propose a proof of convergence in the general case of the SoTT algorithm, which can be extended to the case of the CP-TT algorithm.

In the particular case of the CP-TT method, we take advantage of the simplicity of the CP format and the efficiency of the TT. CP-TT is exempt of the ill-posedness of the solving methods present in the canonical format and it has the advantage of not choosing the order of the variables a priori like in the TT format. Thus, the CP-TT method is a hybrid rank-1 approximation method that gets the best of both worlds from the TT and the CP methods.

In order to check the performance of the proposed techniques, several numerical experiments were conducted. First, we compared the CP-TT to other rank-one update methods (ALS, ASVD, TTr1). Although a single iteration of CP-TT is more expensive in terms of number of operations, it presents a stability that makes it a promising candidate to compress high-dimensional tensors in CP format. We propose some tests in which we compressed the numerical solution of a parametric partial differential equation of reaction-diffusion type as well as other functions coming from different applications. In particular, we compared SoTT with the TT-SVD obtained by testing all the possible permutations of the indices. Although SoTT is suboptimal with respect to the best TT-SVD, it is independent of the order of the variables and its performances are comparable to the average TT-SVD. In this test, the SoTT method outperforms CP-TT.

The SoTT method and its rank-1 particular case show preliminary yet encouraging results in view of applications in scientific computing and compression of high order tensors. Nevertheless, the SoTT method shows some shortcomings to be addressed in further investigations: while a greedy method is appealing in view of computational tasks in which fixing the rank a priori could be cumbersome, it might be featured by a saturation effect, slowing down its convergence.

1.4.2 Contribution in local tensor methods: The local HOSVD method

In the second chapter of the thesis, we present a method that computes an approximation of a tensor in a local HOSVD format.

In the classical compression methods, one of the most common ways to compute the approximation of a tensor is through projection. Usually, one must choose onto which space to project and normally the basis of the space consists of functions whose support is the whole domain. Nevertheless, the choice of an appropriate projection space is crucial to obtain a good approximation.

The tensor to be approximated is a multivariate function $F(x_1, \ldots, x_d)$ of dimension d defined in $\Omega := \Omega_1 \times \ldots \times \Omega_d$. The aim of the proposed method is to provide an approximation of the tensor on the local HOSVD format with good compression rate with respect to the HOSVD decomposition.

The main idea of the method is that a tensor defined in a global domain can be expressed as the sum of its approximations on local subdomains $\Omega^{(i)}$:

$$F \approx \tilde{F}(x_1, \ldots, x_d) = \sum_{i=1}^{N} \tilde{F}(x_1, \ldots, x_d)\mathbf{I}^{(i)}(x_1, \ldots, x_d) = \sum_{i=1}^{N} \tilde{F}_i(x_1, \ldots, x_d) \tag{1.24}$$

where $\mathbf{I}^{(i)}(x)$ are the characteristic functions that restrict the tensor into a certain subdomain.

$$\mathbf{I}^{(i)}(x) = \begin{cases} 1 & \text{if } \mathbf{I}^{(i)}(x) = 1 \quad if \quad x \in \Omega^{(i)} \\ 0 & \text{otherwise} \end{cases}$$

The method is constructed in such a way that on each of the partitions of the domain, a local approximation of the function (restricted to the partition of the domain) is computed by applying the HOSVD method. Then, the approximation reads, for $1 \leq i \leq N$:

$$\tilde{F}_i = \sum_{i_1=1}^{R_1} \ldots \sum_{i_d=1}^{R_d} G_{i_1 \ldots i_d} b_{i_1}^{(1)} \times \ldots \times b_{i_d}^{(d)}$$

in which $R_1, \ldots, R_d$ are the $n-$ranks, b are the modes and G is the tensor core.

A crucial point is to determine the subdomain partition. In the proposed method, we don't impose the partition *a priori*. The aim is to come up with a method that computes automatically the partition. This is done with an agglomerative clustering algorithm. The clustering process is computed in 2 different phases and it is applied independently to every direction k, with $1 \leq k \leq d$, exploiting that the subdomains $\Omega^{(i)}$ are separable. Henceforward, the problem of finding the partition of Ω is reduced to the problem of partitioning each of the domains per direction.

The clustering method computes a dyadic hierarchical tree of the possible partitions of the domains $\left(\Omega_k^{(i)}\right)_{1 \leq k \leq d}$.

1. It starts by computing the clusters of fibers that will be held as the leaves of the tree. This is done by grouping the fibers considered as similar and finding the 2-dimensional basis that represents well all of them.

   The partition of the set of fibers in smaller subsets is mutually exclusive (no fiber in the set is in more than one subset) and jointly exhaustive (every fiber is in some subset). The clustering provides the set of indices of the fibers that belong to each cluster, the set of bases that define them and the set of errors that we are making when we group the fibers.

2. It computes the pairwise Ward's distance [225], between the clusters in the leaves. Find the neighbor clusters and merge them. The resulting cluster encloses the fibers of its two sons.

The algorithm starts pairing the similar leaves and it carries on pairing till every node in the tree is fused. Each one of the levels l_k of the tree denotes a possible partition of the domain on the direction k.

In the root of the tree, the point in which every cluster is fused, the POD decomposition of the unfolding transposed in the direction k is recovered.

Each of the nodes defines a partition of the domain. The total number of possible partitions in all the directions reads as $\prod_{k=1}^{d} l_k$. The closest the nodes are to the root of the tree, the biggest the clusters are and the less the domain is divided.

In the present work, in order to find the best partition, an extensive search between the possible partition combinations is done. Then, a HOSVD approximation of the tensor on each one of the partitions is computed. The selected criterion in order to choose one partition over the others is the memory needed to store the approximation. For every possible combination of the partitions of the domain obtained, the method computes the basis that generates the subspace defined by the partition performing HOSVD on the truncated space. The one that requires less amount of memory to be stored is said to be the *best* of the partitions of the domain and it is the one that we are going to use to compute the approximation of the tensor.

Some tests have been performed for moderate order tensors. In the light of the numerical results obtained, we can say that the method achieves a good compression rate with respect to the HOSVD method. This fact is specially remarkable for the systems that present different "regimes" and the local approximation seems to be more advantageous with respect to a global approximation. Computing partitions of the domain per direction allows the method to detect the different patterns on each direction and obtain more information spending the same amount of memory.

1.4.3 Contribution in Deep Learning-based Schemes for Singularly Perturbed Convection-Diffusion Problems

In the third chapter of the thesis, we propose a deep learning-based method in the context of finding an approximated solution of the convection-diffusion PDE, when the diffusion parameter tends to zero and classical methods like FEM are not able to solve it correctly, [226].

Let us consider the following singularly perturbed convection-diffusion equation on a given domain $\Omega \subset \mathbb{R}^d$, with $d \in \mathbb{N}^*$. Let $F : \Omega \to \mathbb{R}^d$ be a given force field, $0 < \epsilon \ll 1$ a small parameter, and $f : \Omega \to \mathbb{R}$ be a given right-hand side function. Our goal is to find a solution $u : \Omega \to \mathbb{R}$ to

$$-\epsilon(\Delta u)(x) + \nabla \cdot (Fu)(x) = f(x), \quad \forall x \in \Omega, \tag{1.25}$$

together with general Robin boundary conditions

$$\alpha(\nabla u \cdot n)(x) + \kappa u(x) = g(x) \quad \forall x \in \partial\Omega, \tag{1.26}$$

where n refers to the outward unit vector of $\partial\Omega$, α, $\kappa \geq 0$ and g is a real-valued function defined on $\partial\Omega$. In the following, we assume that the force field F derives from a potential function $V : \Omega \to \mathbb{R}$, in the sense that

$$F(x) = -\nabla V(x), \quad \forall x \in \Omega.$$

Under appropriate assumptions on F, f and g, which are assumed to be smooth functions for the sake of simplicity, the problem can be proved to have a unique solution [227, 228, 229].

The problem introduced in Equation (1.25) is challenging for classical numerical methods because it presents numerical instabilities when ϵ is small. The aim of this work is to propose and test various neural network-based schemes in order to find a strategy that is robust for small values of ϵ in the convection-diffusion problem.

Any neural-network based numerical scheme for the solution of the kind of problems we are interested in, relies on the use of a variational formulation of this problem which makes it possible to write the approximation of the solution u (or another function defined from u) as the solution of the minimization problem

$$\min_{v \in \mathcal{V}} \mathcal{J}(v), \tag{1.27}$$

where $\mathcal{V}$ is a particular set of real-valued functions defined on a certain domain Ω. The loss function $\mathcal{J} : \mathcal{V} \to \mathbb{R}$ is usually of the form

$$\mathcal{J}(v) := \lambda \int_{\Omega} \mathcal{R}(v)(x) d\rho(x) + (1-\lambda) \int_{\partial\Omega} \mathcal{S}(v)(r) d\tau(r), \quad \forall v \in \mathcal{V}, \tag{1.28}$$

where λ is the penalization parameter and for every $v \in \mathcal{V}$, $\mathcal{R}(v)$ and $\mathcal{S}(v)$ are real-valued functions defined on Ω and $\partial\Omega$ respectively. They are assumed to be measurable with respect to ρ and τ, which are defined on the domain Ω and its boundary $\partial\Omega$ respectively.

In order to assemble the loss function from the PDE and its boundary conditions, we propose different variants and we compare among the results obtained which one is more suitable for our problem. Let us start by the the most classical formulation used in neural network-based numerical schemes, in which the loss function is directly the strong formulation of the problem 1.25,

$$\mathcal{R}(v)(x) = | - \epsilon(\Delta v)(x) + \nabla \cdot (Fv)(x) - f(x)|^2$$

analogously for the boundary conditions

$$\mathcal{S}(v)(x) = |\alpha(\nabla v \cdot n)(x) + \kappa u(x) - g(x)|2$$

and therefore we refer to it as **Vanilla (V)** formulation. Remark that the approach requires the solution u to be more regular than the usual solution. For this reason, we develop an approach that requires less regularity in the solutions. It is computed by introducing an exponential change of variable in the vanilla formulation and solving the minimization problem. We call it the **Vanilla-z (Vz)** formulation. This method does not fully exploit the change of variable. Since the elliptic nature of the problem allows us to easily build a weak formulation of this equation, other strategy is then developed in which the loss function is written on its weak form, the **Weak-z (Wz)** formulation. Last, we introduce another formulation based on a change of scale in the original problem. We refer to this approach as the **rescaled-weak-z (RWz)** formulation.

The proposed method is based on Physics-Informed Neural Networks (PINN, [206]). PINN is a collocation-based method, [230], in which the problem defined in a PDE is encoded as a part of the neural network. These models transform the problem of finding a solution for the PDE (1.25) into a minimization problem, and its solving process relies on the neural network. The PINNs method finds the coefficients of the neural network solution by minimizing a discretized version of the $L^2(\Omega)$ norm of the strong form of the residual of the PDE. This method is very easily implementable but it implicitly assumes that solutions are very regular.

One of the objectives of this work is to see how the introduced alternatives for the shape of the loss function and the strategy adopted to solve the problem affecting the neural network approach, as well as a comparison between them. Moreover, a survey on other factors that also impact the results is done. These factors include the number of K training points used, the architecture of the NN and the precision of the machine, between others.

The last step of the study is a comparison between the merits and limitations of using deep learning-based approaches (for the different formulations studied) and the classical Finite Elements Method (FEM) when the solutions become low regular. See more about FEM techniques applied to convection-diffusion equations in [231, 232, 233]. Some hybrid models between NNs and FEM are proposed as well in [234, 235].

For large values of ϵ when solutions are rather regular, some PINNs perform clearly better than FEM regarding the generalization errors. The superiority is particularly remarkable for very small number of training points. However, the shapes of PINN solutions are sometimes not as satisfactory as the ones given by FEM. For the case where ϵ becomes small and solutions become less regular (which was the main motivation of our study), the accuracy of the variational neural-network methods is essentially comparable or worse to the one given by FEM in terms of generalization errors. Some PINN variational approaches become too unstable and the errors blow up. Only FEM and the vanilla PINN approach seem to be able to recover the correct shape of the exact function. The latter one has however the risk of sometimes falling into local minima with bad shapes. The runtimes are clearly in favor to PINN methods as well as the simplicity of implementation.

1.5 Organization of the manuscript

The first chapter of this work, Chapter 2, is devoted to the development of the Sum of Tensor Trains (SoTT) algorithm in which the decomposition of a tensor is obtained with the shape of a sum of Tensor Trains. It is also shown its particular case for a rank-1 approximation, the so called Canonical Polyadic - Tensor Train (CP-TT), in which the Canonical Polyadic is recovered. Some numerical tests are shown, some of them showing the compression of the algorithm in comparison with other methods and others showing its efficiency in terms of cost and memory. In Appendix A.1, some results and tests are included.

In the second chapter of the manuscript, Chapter 3 a local method for tensor compression is proposed. The approximation of the tensor is retrieved in a local HOSVD format. It is a clustering-based method and it performs in order to compute local subdomains in which the function restricted to the subspace is approximated. This is done with the aim of approximating solutions of tensor in which the global approximation was not very effective. By construction, if the best way to approximate the tensor is the global one, it recovers the HOSVD method. Some numerical examples are included in which we can see diverse applications of the method.

In the third chapter, Chapter 4, an analysis of Deep Learning-based Schemes for Singularly Perturbed Convection-Diffusion Problems is done. Specifically, the stationary Fokker-Plank equation. In order to approximate this particular equation, we make use of Physics-Informed Neural Networks (PINNs). A survey of some possible variations of the method and an anaysis of how their input format affects the results is studied. The corresponding tests in order to illustrate the different behaviors of the obtained approximation are also included in Appendix B.1.

Chapter 2

Sum of Tensor Trains: SoTT

Contents

2.1 Introduction 29
2.1.1 Organization of the chapter 30
2.2 Notation and preliminaries 30
2.2.1 Tensor spaces 31
2.2.2 Object definition and POD decomposition 32
2.2.3 Ranks and tensor formats: Canonical Polyadic (CP) and Tensor Train (TT) 33
2.3 The Sum of Tensor Trains (SoTT) algorithm 36
2.3.1 Presentation of the SoTT algorithm 36
2.3.2 Exponential convergence of the SoTT algorithm in finite dimension 40
2.3.3 Complexity estimate of the SoTT algorithm 41
2.4 CP-TT: fixed-rank SoTT algorithm with rank 1 42
2.5 Numerical Experiments 43
2.5.1 Comparison between CP-TT and other rank-one update methods 43
2.5.2 SoTT method for the compression of multivariate functions 47
2.6 Conclusions and perspectives 53

2.1 Introduction

Nowadays, applications such as artificial intelligence and data mining algorithms are becoming more and more important in our society. The task of analyzing large volume, multi-relational and multi-modal datasets, is gaining importance too. These datasets are often represented as multiway arrays or tensors [1, 2, 3].

The main challenge in dealing with such data is the already cited *curse of dimensionality*, that refers to the need of using a number of degrees of freedom exponentially increasing with the dimension [5]. This problem can be alleviated by using various tensor formats, achieved by low-rank tensor approximations, for the compression of the full tensor as described for instance in [29, 30, 31, 32]. The definition of these different tensor formats relies on the well-known separation of variables principle. Some of the most used tensor formats have been already presented in the Introduction: Canonical Polyadic (CP), Tensor Train (TT), Tucker and High Order Singular Value Decomposition (HOSVD); and they are defined in Equation 1.4, 1.9, 1.7, 1.8, respectively. We refer the reader to [24] and [42] for extensive reviews on tensor theory and extended analysis of tensor decompositions and their numerous applications. Tensor formats are also used for solving time-dependent and stochastic/parametric PDEs ([43], [44]). Other families of methods called Proper Generalized Decomposition (PGD) methods use tensor decomposition in high dimensional problems [49], [50].

Among the different existing tensor formats, two of them are of specific importance with respect to applications: the Canonical Polyadic (CP) and Tensor Train (TT) format. The main advantage of these decompositions is the low memory cost needed to store them. In the case of the CP format, this cost only scales linearly with the order of the tensor, whereas the memory cost for the storage of a full tensor scales

exponentially with its order. However, the problem of finding a best approximation of a tensor in CP format may be ill-posed [63] and it leads to numerical instabilities. The most classical algorithm in order to compute an approximation of a tensor in the CP format is the so-called Alternating Least Square (ALS) method, which sometimes may be quite slow to converge [236] especially for high-order tensors. Some alternative methods [237, 238] have been proposed in order to obtain more efficient algorithms.

The Tensor Train format is probably one of the most used tensor formats in realistic applications [239, 240, 241], due to a good trade off between optimality and numerical stability. The TT format combines two advantages to take into consideration: on the one hand, it is stable from an algorithmic point of view; on the other, it is computationally affordable provided that the TT ranks of the tensors remain reasonably small. The computation of the approximation of a tensor in the TT format is usually done via the so-called TT-SVD algorithm. One of the drawback of the TT-format is that it requires a priori the choice of a particular order in the variables of the tensor, and the quality of the resulting approximation computed by a TT-SVD algorithm strongly depends on this particular choice. Even if the number of entries could be larger than in CP, the main advantage of the TT format is its ability to provide stable quasi-optimal rank reduction, obtained, for instance, by truncated singular value decompositions.

In the literature, hybrid formats combining CP with other methods have been proposed in [242], and described in [27]. Also, CP has been combined with TT in [243], where it was highlighted that the combination of both methods yields interesting improvements. Fast algorithms for the rank truncation in the canonical input tensors with large CP-ranks and large mode size, have been introduced and analyzed in [101]. Some other optimization-based algorithms could be seen in [244].

The main contribution of this chapter is a numerical scheme that constructs an approximation of a tensor as a sum of TTs, called the Sum of Tensor Trains (SoTT) scheme, where the order of the variables and the values of the ranks can vary from one term to another and can be adaptively chosen by an algorithm which combines the TT-SVD algorithm together with a greedy procedure (see [245]). The interest of such a procedure is two-fold:

- It enables to select in an adaptive way the order of the variables in each term so as to obtain favorable compressing rates with respect to pure TT approximations with an a priori prescribed order of variables.
- When the values of the ranks of the terms computed are fixed to be equal to one, the procedure provides a new scheme for the computation of a CP approximation of a given tensor (namely CP-TT), which appears to be more efficient than ALS for high-order tensors.

We also observe numerically that this algorithm performs well in practice in the sense that it provides a more accurate approximation of a given tensor, at fixed memory storage cost, than a TT-SVD algorithm, in average, when the order of the variables in the TT decomposition is chosen randomly.

We also consider a particular version of the SoTT algorithm, named CP-TT, which consists in adding pure rank-1 tensor-product at each iteration of the scheme. This procedure can be used in order to compute a CP approximation of a given tensor. Such a scheme gives interesting results in comparison with other rank-1 update methods such as ALS for instance, especially when the order of the tensor is high.

2.1.1 Organization of the chapter

This chapter is structured as follows. In Section 2.2, some preliminary concepts about tensors are recalled or introduced. The SoTT algorithm is presented and discussed in Section 2.3. The CP-TT version of SoTT is discussed in Section 2.4 and compared with other numerical methods used for the construction of CP decompositions. Numerical experiments and results illustrating the efficiency of the approach are given in Section 2.5.

2.2 Notation and preliminaries

This section is devoted to the presentation of some concepts and notations that are going to be used in the following chapter. Let us recall some of the concepts introduced before and particularize them to its concrete use on this section.
Tensors are well-studied objects in the standard mathematics literature ([33], [34], [35]) and more specifically

in multilinear algebra ([36], [37]). In this section, we introduce the tensor spaces for multivariate functions and some of the objects that concern tensors. Furthermore, we recall the concept of tensor rank and some of the tensor formats more relevant in the literature. We begin by introducing some notations together with the well-known Singular Value Decomposition in Section 2.2.1. We then recall some basic facts about the Canonical Polyadic (CP) and Tensor Train (TT) format in Section 2.2.3. The most common algorithm to compute the CP format ALS and one of its variations, the Alternating Singular Value Decomposition (ASVD) are introduced in Section 2.2.3. The classical TT-SVD algorithm is then presented in Section 2.2.3.

2.2.1 Tensor spaces

Let D be a subset of $\mathbb{R}^p$ with a product structure $D := D_1 \times \ldots \times D_d$. Let us consider $d \in \mathbb{N}^*$. Then, for all $1 \leq i \leq d$, D_i is an open bounded subset of $\mathbb{R}^{p_i}$ for some $p_i \in \mathbb{N}^*$. We call a *tensor* (with a slight language abuse) any real-valued function $F \in L^2(D_1 \times \cdots \times D_d)$.
For each $1 \leq i \leq d$, a certain $\mathcal{H}_i$ is a Hilbert space of univariate functions defined of D_i, equipped with the corresponding inner product $(\cdot,\cdot)_{\mathcal{H}_i}$ and the associated norm $\|\cdot\|_{\mathcal{H}_i}$.

The elementary tensor product $u^{(1)} \otimes \ldots \otimes u^{(d)}$ of d univariate functions $u^{(i)} \in \mathcal{H}_i$ has associated a multilinear mapping $D_1 \times \ldots \times D_d \to D_1 \otimes \ldots \otimes D_d$ and is defined such that for a multivariate function F and $x = (x_1, \ldots, x_d) \in D$,

$$F(x) = (u^{(1)} \otimes \ldots \otimes u^{(d)})(x_1, \ldots, x_d) = u^{(1)}(x_1) \otimes \ldots \otimes u^{(d)}(x_d).$$

The span of these elementary tensor products is the algebraic tensor space $\mathcal{H}_1 \otimes \ldots \otimes \mathcal{H}_d$ where belong all the multivariate functions that can be expressed as a finite linear combination of elementary tensors (see more in [246]).

For any $u^{(1)} \in L^2(D_1), \cdots, u^{(d)} \in L^2(D_d)$, we denote $u^{(1)} \otimes \cdots \otimes u^{(d)} \in L^2(D)$ the *pure tensor product function* defined by

$$u^{(1)} \otimes \cdots \otimes u^{(d)} : \left\{ \begin{array}{ccc} D = D_1 \times \cdots \times D_d & \to & \mathbb{R} \\ (x_1, \cdots, x_d) & \mapsto & u^{(1)}(x_1) \cdots u^{(p)}(x_d). \end{array} \right.$$

A canonical inner product over the tensor space between F and other multivariate function namely $V \in L^2(D_1 \times \ldots \times D_d)$, whose elementary tensors are denoted by $v^{(i)}$, with $i \leq i \leq d$, is defined by:

$$(u^{(1)} \otimes \ldots \otimes u^{(p)}, v^{(1)} \otimes \ldots \otimes v^{(d)}) = (u^{(1)} \otimes v^{(1)})_{\mathcal{H}_1} \ldots (u^{(d)} \otimes v^{(d)})_{\mathcal{H}_d}.$$

This tensor product holds bilinearity properties and in the literature can be referred as *Kronecker product*. Its associated canonical norm $\|\cdot\|_{\mathcal{H}_i}$, verifies that: $\|u^{(1)} \otimes \ldots \otimes u^{(d)}\| = \|u^{(1)}\|_{\mathcal{H}_1} \ldots \|u^{(d)}\|_{\mathcal{H}_d}$. A tensor Hilbert space $\mathcal{H}$ of dimension d is defined as the completion of this algebraic tensor space $\mathcal{H} = \mathcal{H}_1 \otimes \ldots \otimes \mathcal{H}_d$.

Moreover, we make use of the following abuse of notation. For any nonempty subset $\mathcal{I} \subset \{1, \cdots, d\}$ such that $\mathcal{I}^c = \{1, \cdots, d\} \setminus \mathcal{I}$ is non-empty, and any $F \in L^2(D_1 \times \cdots D_d)$, we still denote by F the function $\widetilde{F} \in L^2\left((\bigtimes_{i \in \mathcal{I}} D_i) \times (\bigtimes_{j \in \mathcal{I}^c} D_j)\right)$ defined by

$$\widetilde{F} : \left\{ \begin{array}{ccc} (\bigtimes_{i \in \mathcal{I}} D_i) \times (\bigtimes_{j \in \mathcal{I}^c} D_j) & \to & \mathbb{R} \\ ((x_i)_{i \in \mathcal{I}}, (x_j)_{j \in \mathcal{I}^c}) & \mapsto & F(x_1, \cdots, x_d). \end{array} \right.$$

For each $1 \leq i \leq d$, we consider $L^2(D_i)$, the Hilbert space of sqare integrable, real-valued univariate functions defined on D_k. Its natural norm $\|\cdot\|_{L^2(D_i)}$ is defined for $u^{(i)} \in L^2(D_i)$ by

$$\|u^{(i)}\|^2_{L^2(D_i)} = \int_{D_i} u^{(i)}(x_i)^2 dx_i.$$

The space $L^2(D)$ of square-integrable, real-valued functions defined on D equipped with the canonical norm $\|\cdot\|_{L^2(D)}$ that is defined for $F \in L^2(D)$ by

$$\|F\|^2_{L^2(D)} = \int_D F(x)^2 dx \quad \text{and} \quad \|F\|_{L^2(D)} := \left(\int_D F^2(x) dx \right)^{1/2}$$

and it can be identified with the completion of the algebraic tensor spaces $L^2(D_1) \otimes \ldots \otimes L^2(D_p)$. The scalar product for all $F, G \in L^2(D)$,

$$\langle F, G \rangle_D := \int_D F(x)G(x)dx$$

From now on, in order to simplify the notation of the manuscript (every time that there's no ambiguity) the norm $\|\cdot\|_{L^2(D)}$ will be denoted just as $\|\cdot\|_D$ and its associated inner product as $(\cdot,\cdot)_D$.

Let us have a discrete $d-$dimensional subset $D_h \subset \mathbb{R}^{\mathcal{N}_1 \times \mathcal{N}_2 \times \ldots \times \mathcal{N}_d}$. For each direction i we fix a index set $\mathcal{N}_i = \{1, \ldots, N_i\}$, that denotes the degrees of freedom per direction. The Cartesian product of these index sets is denoted as $\mathcal{N} = \mathcal{N}_1 \times \ldots \times \mathcal{N}_d$.
Being d the dimension of the discrete tensor. Let us refer to discrete tensor (or just tensor) to the discretization of the multivariate function introduced before. A dth-order tensor is an element of the tensor product of d vector spaces $D_1 \times \ldots \times D_d$. Up to a certain choice of basis on the spaces (each has its own coordinate system), such an element has to be represented as a $d-$way array of real numbers. Tensors will carry d sub indices $i_1, \ldots, i_d \in \{1, \ldots, d\}$ to denote them by elements. A certain discrete tensor $x \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_d}$ is defined by its entries, $x_{i_1 \ldots i_d}$.

The *Frobenius norm*, or the l^2 norm in a discrete d-dimensional tensor space is the square root of the sum of the squares of all its elements. For a certain $F \in L^2(D_h)$:

$$\|F\|_{l^2}^2 = \sum_{k=1}^{d} |F_i|^2$$

if the norm is not explicitly specified, we assume that is the Frobenius norm.

2.2.2 Object definition and POD decomposition

The so called *fibers* all along the manuscript are the higher-order analogue of matrix rows and columns, defined by fixing every index of a tensor but one. In a 3-dimensional tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3}$, for a certain value of two of its indices $i_2 \in \{1, \ldots, \mathcal{N}_2\}$ and $i_3 \in \{1, \ldots, \mathcal{N}_3\}$, a fiber on the i_1-th direction is given by $x^{(i_1)} = X_{:i_2 i_3}$, with $i_1 \in \{1, \ldots, \mathcal{N}_1\}$. When fibers are extracted from tensors, they are assumed to be column vectors.

Assume a $N-$th order tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_N}$. The matrix *Unfolding* $X^{(n)} \in \mathbb{R}^{\mathcal{N}_n \times (\mathcal{N}_{n+1} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1})}$ contains the element $x_{i_1 \ldots i_p}$ at the row position i_n and column number equal to:

$$(i_{n+1} - 1)\mathcal{N}_{n+2} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1} + i_{n+2} - 1)\mathcal{N}_{n+3} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1} + \ldots$$

$$\ldots + (i_N - 1)\mathcal{N}_1 \ldots \mathcal{N}_{n-1} + (i_1 - 1)\mathcal{N}_2 \ldots \mathcal{N}_{n-1} + \ldots + i_{n-1}$$

In a third order tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3}$, the horizontal, lateral, and frontal *slices* are denoted by $X_{i_1::}$, $X_{:i_2:}$, and $X_{::i_3}$, respectively. When we refer to the n-th frontal slice of a third order tensor, we obtain its unfolding $X^{(n)}$.

Proper orthogonal decomposition (POD)

The Proper Orthogonal Decomposition (POD) [38] is a popular dimensionality reduction method used in model reduction (see [39], [40], [41]) to define the trial subspace.

Definition. For any domain $D = D_x \times D_y$, where D_x and D_y are open subdomains of $\mathbb{R}^{d_x}$ and $\mathbb{R}^{d_y}$ for some $d_x, d_y \in \mathbb{N}^*$ respectively, and any $W \in L^2(D)$, it holds that there exists an orthonormal basis $(U_k)_{k \in \mathbb{N}^*}$ of $L^2(D_x)$, an orthonormal basis $(V_k)_{k \in \mathbb{N}^*}$ of $L^2(D_y)$ and a non-increasing sequence $(\sigma_k)_{k \in \mathbb{N}^*}$ of non-negative real numbers which converges to 0 as k goes to ∞, such that

$$W = \sum_{k \in \mathbb{N}^*} \sigma_k U_k \otimes V_k. \tag{2.1}$$

A decomposition of W under the form (2.1) is called a Singular Value Decomposition (SVD) (or Proper Orthogonal Decomposition (POD)) of W according to the separation of variables (D_x, D_y). The sequence $(\sigma_k)_{k\in\mathbb{N}^*}$ is known to be unique and is called the sequence of singular values of W associated to the separation of variables (D_x, D_y) of the set D. The orthonormal basis $(U_k)_{k\in\mathbb{N}^*}$ (respectively $(V_k)_{k\in\mathbb{N}^*}$) may not be unique but is called a sequence of left (respectively right) singular vectors of W associated to this partitioning.

For any set E, we denote in the sequel by $\#E$ the cardinality of E. Assuming that $\mathcal{N}_x := \#D_x < +\infty$ and $\mathcal{N}_y := \#D_y < +\infty$, the complexity of the computation of an SVD decomposition of the form (2.1) scales like

$$\mathcal{O}\left(\max(\mathcal{N}_x, \mathcal{N}_y)\min(\mathcal{N}_x, \mathcal{N}_y)^2\right). \tag{2.2}$$

2.2.3 Ranks and tensor formats: Canonical Polyadic (CP) and Tensor Train (TT)

Let us define the *rank of a tensor* F, denoted $rank(F)$ or just r, as the smallest number of rank-one tensors that generate F as their sum.

$$rank(F) = min\{r : F \in \mathcal{R}_r\}$$

where $\mathcal{R}_r$ is the set of tensors of rank not exceeding r.

The *tensor rank decomposition* factorizes a tensor into a minimum-length linear combination of rank-one tensors. Let F be a real tensor obtained by the discretization of a certain $d-$variate function $F \in D_1 \times \cdots \times D_d$. Every order-$d$ tensor in this space may then be represented with a suitably large $r \in \mathbb{N}^*$, as a linear combination of r rank-one tensors.

More details about the concepts of tensor rank and the decomposition of a tensor into a sum of outer products of elementary tensors introduced in [52] and [53], can be found in [247].

Although we focus on the previous concept of rank, there is a simpler notion of *multilinear rank* which directly generalizes the column and row ranks of a matrix to higher order tensors. Let us illustrate this concept with a $3-$dimensional example. Let us have a certain tensor $F \in \mathbb{R}^{\mathcal{N}_1\times\mathcal{N}_2\times\mathcal{N}_3}$. Note that the space can be viewed as $\mathbb{R}^{\mathcal{N}_1\times\mathcal{N}_2\mathcal{N}_3}$ (similarly in the other directions). Then $r_1(F)$ is simply the rank of the first one of its unfoldings. The same for $r_2(F)$ and $r_3(F)$. Then, the multilinear rank of F, is the tuple $(r_1(F), r_2(F), r_3(F))$ and it was also introduced in [52], [53].

The Canonical Polyadic (CP) decomposition

As it is detailed in [51], the idea of the polyadic form of a tensor (expressing a tensor as the sum of a finite number of elementary tensors) arose initially in [52], [53]. The form of CANDECOMP (canonical decomposition), was proposed in [54] and the form of parallel factors in [55]. We refer to the CP decomposition as proposed independently in [56] and [57].

Let us now introduce some notation which will be used in all the sequel. From now on, we fix some $d \in \mathbb{N}^*$ and for all $1 \leq j \leq d$, let Ω_j be an open subset of $\mathbb{R}^{p_j}$ for some $p_j \in \mathbb{N}^*$. We define $\Omega := \Omega_1 \times \cdots \times \Omega_d$.

Let $F \in L^2(\Omega_1 \times \cdots \times \Omega_d)$. The function F is said to belong to the **Canonical Polyadic (CP)** format [52, 29, 58] with rank $r \in \mathbb{N}^*$ if it reads as:

$$F(x_1, x_2, ..., x_d) = \sum_{i=1}^{r} u_i^{(1)}(x_1)u_i^{(2)}(x_2)\cdots u_i^{(d)}(x_d) \tag{2.3}$$

for some functions $u_i^{(j)} \in L^2(\Omega_j)$ for $1 \leq i \leq r$ and $1 \leq j \leq d$. The CP decomposition factorizes a tensor into a sum of component rank-one tensors.

The *rank* of a tensor can be expressed as the minimum number of terms in an exact CP. A CP decomposition of a tensor F with $r = rank(F)$ number of components is called the *rank decomposition*.

More details about this decomposition and its uniqueness can be found in [59], [51], [60] and [61].

Given a certain tensor F, find the best rank approximation is equivalent to determine d rank-1 tensors that approximate well it. This can be written as a minimization problem: Find $u_i^{(1)} \in \mathbb{R}^{\mathcal{N}_1}, \ldots, u_i^{(d)} \in \mathbb{R}^{\mathcal{N}_d}$, with $i = 1, \ldots, r$, that minimize

$$\|F - \sum_{i=1}^{r} u_i^{(1)}(x_1) \otimes u_i^{(2)}(x_2) \otimes \ldots \otimes u_i^{(d)}(x_d)\|$$

where $\|\cdot\|$ denotes some choice of norm on $\mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_d}$. When $d = 2$, the result of the minimization problem is given by the first r terms in the sum of its SVD decomposition, see [248], for unitarily invariant norms. The SVD decomposition in two variables is then optimal. For higher order tensors this is a problem of central importance. It can be seen in [247] and [249] that the problem is ill-posed for many r regardless the choice of norm. Moreover, if one randomly picks a tensor F in a suitable tensor space, there's a non-zero probability that F will fail to have a best rank-r approximation for some $r < rank(F)$. It is also known as the phenomenon underlying the concept of *border rank*, see [250], [251], [252], and it is related to (but with different from) "CP degeneracy" ([253]).

The main advantage of the CP decomposition is the low memory cost needed to store it. Indeed, if $\mathcal{N}$ degrees of freedom are used per variable ($\mathcal{N}_1 = \ldots = \mathcal{N}_d = \mathcal{N}$), the storage cost of a general function $F \in L^2(\Omega_1 \times \cdots \times \Omega_d)$ is $\mathcal{O}\left(\mathcal{N}^d\right)$. On the other hand, the storage cost of a CP tensor with rank r reduces to $\mathcal{O}(d\mathcal{N}r)$, which scales linearly in the tensor order d and size $\mathcal{N}$. However, the problem of finding a best approximation of a tensor in CP format may be ill-posed [63] and leads to numerical instabilities. The most classical algorithm in order to compute an approximation of a tensor in the CP format is the so-called Alternating Least Square (ALS) method.

Alternating Least Square (ALS) and Alternating Singular Value Decomposition (ASVD)

In general the determination of a tensor rank is an NP-hard problem. Previous works revealed us that there isn't any finite algorithm that determines the rank of a tensor. Because of that, the first problem that we face in computing a CP decomposition is how to choose the number of rank-one components. The CP decomposition with r components that best approximates F via the ALS method will be the one solving the minimization problem:

$$min_{\widetilde{F}}(\|F - \widetilde{F}\|)$$

where $\widetilde{F}$ is the rank$-r$ approximation of F given by Equation 2.3. We introduce and detail the ALS algorithm in Algorithm 4 for a certain tensor namely W and a given accuracy ϵ. In order to explain the method, let us introduce some notation. In the following, $\widetilde{W}^{n-1}$ is the approximation of the tensor W obtained after $n-1$ iterations of the algorithm, and for all $1 \leq i \leq d$, $R_i^n \in L^2(\Omega_i)$ and its components: $R_i^{n,m} \in L^2(\Omega_i)$.

As we can see in [64], the ALS method for the computation of a CP approximation of a tensor can take many iterations to converge [236], especially when the order of the tensor d is high. Moreover, it is not guaranteed to converge to a global minimum or even a stationary point, only to a solution where the objective function stops decreasing. The convergence properties of the ALS algorithm have been abundantly studied. We refer the reader for more details to the following series of works [64, 131, 254, 255, 256]. The final solution can be heavily dependent on the starting guess as well. The reason for the popularity of this algorithm lies in the fact that it is very simple, conceptually and numerically, while still delivering astonishingly good results in many cases in low dimension. For higher ranks most of the difficulties with the global behavior of ALS seem to be intimately related to the fact that the approximation problem itself can be ill-posed. Some alternative methods [237, 238] have been proposed in order to obtain more efficient algorithms. The Alternating Singular Value Decomposition (ASVD) method is proposed in [257].

For the presentation of the ASVD algorithm, we need to introduce some additional notation. We denote by $\mathcal{J} := \{\{i, j\},\ 1 \leq i < j \leq d\}$ the set of all possible pairs of indices between 1 and d. An ordering of the elements of $\mathcal{J}$ is chosen so that $\mathcal{J} = (J_l)_{1 \leq l \leq L}$ with $L := \#\mathcal{J}$. We introduce the ASVD algorithm in Algorithm 5 for W and the tolerance ϵ.

Algorithm 4 ALS algorithm

Require: $\epsilon > 0$, $W \in L^2(\Omega)$

Output: $N \in \mathbb{N}^*$, for all $1 \leq n \leq N$ and all $1 \leq i \leq d$, $R_i^n \in L^2(\Omega_i)$ so that the CP tensor $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := \sum_{n=1}^{N} R_1^n(x_1) R_2^n(x_2) \cdots R_d^n(x_d) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|^2_{L^2(\Omega)} \leq \epsilon$.

Set $W^0 = W$, $n = 1$.

while $\|W^{n-1}\|^2_{L^2(\Omega)} > \epsilon$ **do**

  For all $1 \leq i \leq d$, select randomly $R_i^{n,0} \in L^2(\Omega_i)$ and set $\eta := \epsilon$ and $m = 1$.

  while $\eta > \frac{1}{10}\epsilon$ **do**

    for $j = 1, \cdots, d$ **do**

      Compute $R_j^{n,m} \in L^2(\Omega_j)$ solution to

$$R_j^{n,m} \in \underset{R_j \in L^2(\Omega_j)}{\operatorname{argmin}} \left\| W_{n-1} - R_1^{n,m} \otimes \cdots \otimes R_{j-1}^{n,m} \otimes R_j \otimes R_{j+1}^{n,m-1} \otimes \cdots \otimes R_d^{n,m-1} \right\|^2_{L^2(\Omega)}$$

    end for

    Compute $\eta := \left\| R_1^{n,m} \otimes \cdots \otimes R_d^{n,m} - R_1^{n,m-1} \otimes \cdots \otimes R_d^{n,m-1} \right\|^2_{L^2}$. Set $m := m + 1$.

  end while

  Define $R_i^n = R_i^{n,m-1}$ for all $1 \leq i \leq d$.

  Compute $W_n(x_1, \cdots, x_d) = W_{n-1}(x_1, \cdots, x_d) - R_1^n(x_1) R_2^n(x_2) \cdots R_d^n(x_d)$ for all $(x_1, \cdots, x_d) \in \Omega$.

  $n = n + 1$

end while

$N = n - 1$

The Tensor Train (TT) decomposition

Let us now continue presenting other tensor format that is going to be present in this manuscript. It was introduced in [118].

The function F is said to belong to the **Tensor Train (TT)** format with ranks $r_1, \ldots, r_{d-1} \in \mathbb{N}^*$ if and only if

$$F(x_1, x_2, \ldots, x_d) = \sum_{i_1=1}^{r_1} \cdots \sum_{i_{d-1}=1}^{r_{d-1}} u_{i_1}^{(1)}(x_1) u_{i_1,i_2}^{(2)}(x_2) u_{i_2,i_3}^{(3)}(x_3) \ldots u_{i_{d-2},i_{d-1}}^{(d-1)}(x_{d-1}) u_{i_{d-1}}^{(d)}(x_d)$$

with $u_{i_{j-1},i_j}^{(j)} \in L^2(\Omega_j)$ for $1 \leq i_{j-1} \leq r_{j-1}$ and $1 \leq i_j \leq r_j$ for all $1 \leq j \leq d$ (with $r_0 = r_d = 1$).

It combines two main advantages: On the one hand, it is stable from an algorithmic point of view; on the other, it is computationally affordable provided that the TT ranks of the tensors used stay reasonably small.

One of the schemes used for the optimization of a TT tensor with a given rank is the ALS method, see details in [130]. In [67] some numerical examples that concern the stability of the TT decomposition and of ALS are shown as well as how high TT ranks are required during the iterative approximation of low-rank tensors, showing some potential of improvement. These ranks are chosen *a priori* and that makes the method less robust. As we will recall in Algorithm 6, the most used way of computing the TT decomposition of a tensor is via the well-known TT-SVD algorithm. Let us make a special remark in this chapter to the TTr1 algorithm, proposed in [258]. As we will see, this is one of the closest alternatives to the proposed *rank*-1 particular case of the method.

TT-SVD algorithm

Let now choose and fix some $W \in L^2(\Omega)$. We recall in Algorithm 6 the well-known TT-SVD algorithm for computing an approximation of the tensor W with prescribed accuracy $\epsilon > 0$ in a TT format.

Algorithm 5 ASVD algorithm

Require: $\epsilon > 0$, $W \in L^2(\Omega)$

Output: $N \in \mathbb{N}^*$, for all $1 \leq n \leq N$ and all $1 \leq i \leq d$, $R_i^n \in L^2(\Omega_i)$ so that the CP tensor $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := \sum_{n=1}^{N} R_1^n(x_1 R_2^n(x_2) \cdots R_d^n(x_d) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|^2_{L^2(\Omega)} \leq \epsilon$.

Set $W^0 = W$, $n = 1$.

while $\|W^{n-1}\|^2_{L^2(\Omega)} > \epsilon$ **do**

  For all $1 \leq i \leq d$, select randomly $R_i^{n,0} \in L^2(\Omega_i)$ and set $\eta := \epsilon$ and $m = 1$.

  while $\eta > \frac{1}{10}\epsilon$ **do**

    Set $R_i^{n,m} = R_i^{n,m-1}$ for all $1 \leq i \leq d$

    for $l = 1, \cdots, L$ **do**

      Let $1 \leq i_l < j_l \leq d$ so that $J_l = (i_l, j_l)$.

      Compute $U_l^{n,m} \in L^2(\Omega_{i_l} \times \Omega_{j_l})$ solution to

$$U_l^{n,m} \in \underset{U_l \in L^2(\Omega_{i_l} \times \Omega_{j_l})}{\operatorname{argmin}} \left\| W_{n-1} - U_l \otimes \bigotimes_{i \in \{1, \cdots, d\} \setminus J_l} R_i^{n,m} \right\|^2_{L^2(\Omega)}.$$

      Compute $\left(R_{i_l}^{n,m}, R_{j_l}^{n,m}\right) \in L^2(\Omega_{i_l}) \times L^2(\Omega_{j_l})$ solution to

$$\left(R_{i_l}^{n,m}, R_{j_l}^{n,m}\right) \in \underset{\left(R_{i_l}, R_{j_l}\right) \in L^2(\Omega_{i_l}) \times L^2(\Omega_{j_l})}{\operatorname{argmin}} \|U_l^{n,m} - R_{i_l} \otimes R_{j_l}\|^2_{L^2(\Omega_{i_l} \times \Omega_{j_l})}.$$

    end for

    Compute $\eta := \left\| R_1^{n,m} \otimes \cdots \otimes R_d^{n,m} - R_1^{n,m-1} \otimes \cdots \otimes R_d^{n,m-1} \right\|^2_{L^2}$.

  end while

  Define $R_i^n = R_i^{n,m}$ for all $1 \leq i \leq d$.

  Compute $W_n(x_1, \cdots, x_d) = W_{n-1}(x_1, \cdots, x_d) - R_1^n(x_1) R_2^n(x_2) \cdots R_d^n(x_d)$ for all $(x_1, \cdots, x_d) \in \Omega$.

  $n = n + 1$

end while

$N = n - 1$

2.3 The Sum of Tensor Trains (SoTT) algorithm

The aim of this section is to present the Sum of Tensor Trains (SoTT) algorithm. We propose a method in order to greedily construct an approximation of a given tensor as a sum of Tensor Trains (TTs), where the order of the variables and the values of the ranks can be different from one term to another. The algorithm is presented in Section 2.3.1 and in a more detailed way in Algorithm 7. It is proved to converge exponentially fast in finite dimension in Section 2.3.2. Lastly, a discussion about the complexity of the method is given in Section 2.3.3.

2.3.1 Presentation of the SoTT algorithm

In the following, we denote $\mathcal{S}_d$ the set of permutations of the set $\{1, \cdots, d\}$.
Let us introduce a detailed scheme of the SoTT method in Algorithm 7. The different steps are explained in the following.

The aim of the SoTT algorithm is to compute, after n iterations, an approximation of a certain tensor W as a sum of n TTs. At iteration n, the SoTT computes an approximation of W under the form

$$\widetilde{W}^{n-1} + R_1^n(x_{\tau^n(1)}) R_2^n(x_{\tau^n(2)}) \cdots R_d^n(x_{\tau^n(d)}),$$

where $\widetilde{W}^{n-1}$ is the approximation obtained after $n-1$ iterations of the algorithm, $\tau_n \in \mathcal{S}_d$ denotes a well-chosen permutation of the variables, and for all $1 \leq j \leq d$, $R_j^n \in L^2\left(\Omega_{\tau^n(j)}, \mathbb{R}^{K_{j-1}^n \times K_j^n}\right)$, where $K_0^n = K_d^n = 1$. The aim of the n^{th} iteration is to choose the permutation τ_n and the values of the ranks $(K_j^n)_{1 \leq j \leq d-1}$ in an appropriate way, which is done here using a greedy procedure.

Algorithm 6 TT-SVD algorithm

Require: $\epsilon > 0$, $W \in L^2(\Omega)$

Output: $K_1, \cdots, K_{d-1} \in \mathbb{N}^*$ TT-ranks, $R_1 \in L^2\left(\Omega_1, \mathbb{R}^{1\times K_1}\right)$, $R_d \in L^2\left(\Omega_d, \mathbb{R}^{K_{d-1}\times 1}\right)$ and for all $i = 2, \cdots, d-1$, $R_i \in L^2\left(\Omega_i, \mathbb{R}^{K_{i-1}\times K_i}\right)$ so that the Tensor Train $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := R_1(x_1)R_2(x_2)\cdots R_d(x_d) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|^2_{L^2(\Omega)} \leq \epsilon^2$.

Define $K_0 := 1$, $D_0 = \{1\}$ and define $\overline{W}_0 \in L^2(D_0 \times \Omega)$ such that $\overline{W}_0(1, \cdot) = W$, $\mathcal{I}_0 := \{1, \cdots, d\}$, $\widehat{\Omega}_0 := \Omega$.

for $j = 1, \cdots, d-1$ **do**

Since $D_{j-1} \times \widehat{\Omega}_{j-1} = (D_{j-1} \times \Omega_j) \times \widehat{\Omega}_j$ with $\widehat{\Omega}_j = \Omega_{j+1} \times \cdots \times \Omega_d$, compute the SVD decomposition of $\overline{W}_{j-1}$ according to the separation of variables $(D_{j-1} \times \Omega_j, \widehat{\Omega}_j)$ so that

$$\overline{W}_{j-1} = \sum_{k\in\mathbb{N}^*} \sigma_{j,k} U_{j,k} \otimes V_{j,k}.$$

Select $K_j \in \mathbb{N}^*$ such that $K_j = \inf\left\{ K \in \mathbb{N}^*, \quad \sum_{k\geq K} |\sigma_{j,k}|^2 \leq \frac{\epsilon^2}{d-1} \right\}$.

Define $D_j := \{1, \cdots, K_j\}$ and define $\overline{W}_j \in L^2\left(D_j \times \widehat{\Omega}_j\right)$ by

$$\overline{W}_j(k_j, y_j) = \sigma_{j,k_j} V_{j,k_j}(y_j)$$

for all $(k_j, y_j) \in D_j \times \widehat{\Omega}_j$.

Define $R_j \in L^2\left(\Omega_j, \mathbb{R}^{K_{j-1}\times K_j}\right)$ as

$$R_j(x_j) = \left(U_{j,k_j}(k_{j-1}, x_j)\right)_{\substack{1\leq k_{j-1}\leq K_{j-1} \\ 1\leq k_j \leq K_j}}$$

for all $x_j \in \Omega_j$.

end for

Define $R_d \in L^2\left(\Omega_d, \mathbb{R}^{K_{d-1}\times 1}\right)$ by

$$R_d(x_d) = \left(\sigma_{d-1,k_{d-1}} V_{d-1,k_{d-1}}(x_d)\right)_{1\leq k_{d-1}\leq K_{d-1}}.$$

Algorithm 7 SoTT algorithm

Require: $\epsilon > 0$, $W \in L^2(\Omega)$

Output: $N \in \mathbb{N}^*$, for all $1 \leq n \leq N$, $\tau^n \in \mathcal{S}_d$, $K_1^n, \cdots, K_{d-1}^n \in \mathbb{N}^*$ TT-ranks, $R_1^n \in L^2\left(\Omega_{\tau^n(1)}, \mathbb{R}^{1\times K_1^n}\right)$, $R_d^n \in L^2\left(\Omega_{\tau^n(d)}, \mathbb{R}^{K_{d-1}^n \times 1}\right)$ and for all $i = 2, \cdots, d-1$, $R_i^n \in L^2\left(\Omega_{\tau^n(i)}, \mathbb{R}^{K_{i-1}^n \times K_i^n}\right)$ so that the sum of Tensor Trains $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := \sum_{n=1}^{N} R_1^n(x_{\tau^n(1)}) R_2^n(x_{\tau^n(2)}) \cdots R_d^n(x_{\tau^n(d)}) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|^2_{L^2(\Omega)} \leq \epsilon$.

Set $W^0 = W$, $n = 1$.

while $\|W^{n-1}\|^2_{L^2(\Omega)} > \epsilon$ **do**

Define $K_0^n := 1$, $D_0^n = \{1\}$ and define $\overline{W}_0^n \in L^2(D_0 \times \Omega)$ such that $\overline{W}_0^n(1, \cdot) = W^{n-1}$, $\mathcal{I}_0^n := \{1, \cdots, d\}$.

for $j = 1, \cdots, d-1$ **do**

For all $i \in \mathcal{I}_{j-1}^n$, since $D_{j-1}^n \times \bigtimes_{i\in\mathcal{I}_{j-1}^n} \Omega_i = \left(D_{j-1}^n \times \Omega_i\right) \times \bigtimes_{i'\in\mathcal{I}_{j-1}^n \setminus \{i\}} \Omega_{i'}$, compute the SVD decomposition of $\overline{W}_{j-1}^n$ according to the separation of variables $(D_{j-1}^n \times \Omega_i, \bigtimes_{i'\in\mathcal{I}_{j-1}^n \setminus \{i'\}} \Omega_{i'})$ so that

$$\overline{W}_{j-1}^n = \sum_{k\in\mathbb{N}^*} \sigma_{j,k}^{i,n} U_{j,k}^{i,n} \otimes V_{j,k}^{i,n}.$$

Select $i_j^n \in \mathcal{I}_{j-1}^n$ and $\overline{K}_j^n \in \mathbb{N}^*$ so that

$$\left(i_j^n, \overline{K}_j^n\right) \in \underset{i\in\mathcal{I}_{n-1}, r\in\mathbb{N}^*}{\operatorname{argmax}} \sum_{k=1}^{r} \left(\sigma_{j,k}^{i,n}\right)^2 - \beta_{i,j}^n r,$$

where for all $i \in \mathcal{I}_{j-1}^n$, $\beta_{i,j}^n > 0$ is chosen according to (2.4).

Define $\tau^n(j) = i_j^n$.

Select $K_j^n \in \mathbb{N}^*$ such that $K_j^n = \min\left(\overline{K}_j^n, \inf\left\{K \in \mathbb{N}^*, \quad \sum_{k\geq K} \left|\sigma_{j,k}^{\tau^n(j),n}\right|^2 \leq \frac{\epsilon^2}{d-1}\right\}\right)$.

Define $\mathcal{I}_j^n := \mathcal{I}_{j-1}^n \setminus \{\tau^n(j)\}$ so that $\#I_j^n = d - j$.

Define $D_j^n := \left\{1, \cdots, K_j^n\right\}$ and define $\overline{W}_j^n \in L^2\left(D_j^n \times \bigtimes_{i\in\mathcal{I}_j^n} \Omega_i\right)$ by

$$\overline{W}_j^n(k_j, y_{\tau^n(j)}) = \sigma_{j,k_j}^{\tau^n(j),n} V_{j,k_j}^{\tau^n(j),n}(y_{\tau^n(j)})$$

for all $(k_j, y_{\tau^n(j)}) \in D_j^n \times \bigtimes_{i\in\mathcal{I}_j^n} \Omega_i$.

Define $R_j^n \in L^2\left(\Omega_{\tau_n(j)}, \mathbb{R}^{K_{j-1}^n \times K_j^n}\right)$ as

$$R_j^n(x_{\tau^n(j)}) = \left(U_{j,k_j}^{\tau_n(j),n}(k_{j-1}, x_{\tau^n(j)})\right)_{1\leq k_j \leq K_j^n, 1\leq k_{j-1}\leq K_{j-1}^n}$$

for all $x_{\tau^n(j)} \in \Omega_{\tau^n(j)}$.

end for

Since $\#I_{d-1}^n = 1$, let $i_d^n \in \{1, \cdots, n\}$ such that $I_{d-1}^n = \{i_d^n\}$. Define $\tau^n(d) = i_d^n$.

Define $R_d^n \in L^2\left(\Omega_{\tau^n(d)}, \mathbb{R}^{K_{d-1}^n \times 1}\right)$ by

$$R_d^n(x_{\tau^n(d)}) = \left(\sigma_{d-1,k_{d-1}}^{\tau^n(d-1),n} V_{d-1,k_{d-1}}^{\tau^n(d-1),n}(x_{\tau^n(d)})\right)_{1\leq k_{d-1}\leq K_{d-1}^n}.$$

Compute $W^n(x_1, \cdots, x_d) = W^{n-1}(x_1, \cdots, x_d) - R_1^n(x_{\tau^n(1)}) R_2^n(x_{\tau^n(2)}) \cdots R_d^n(x_{\tau^n(d)})$ for all $(x_1, \cdots, x_d) \in \Omega$.

$n = n + 1$

end while

$N = n - 1$

The idea behind the SoTT procedure is the following: the order of the variables is chosen so that it enables to obtain an interesting trade-off between accuracy and memory storage.

For instance, $\tau_n(1)$ is chosen as follows. Let us denote by $W^{n-1} := W - \widetilde{W}^{n-1}$ the difference between the tensor and its approximation in the $(n-1)^{th}$ iteration and by $\overline{W}_0^n := W^{n-1}$. The POD decomposition of $\overline{W}_0^n$ is computed with respect to all the unfoldings, i.e. the partitioning of the variables of the form $\Omega = \Omega_i \times (\bigtimes_{1\leq j\neq i\leq d} \Omega_j)$ for all $i \in \{1, \cdots, d\} = \mathcal{I}_0^n$. Denoting by $\left(\sigma_{1,k}^{i,n}\right)_{k\in\mathbb{N}^*}$ the sequence of singular values associated to the i^{th} partitioning of the variables, for all $r \in \mathbb{N}$, let us define the functional

$$L_{i,1}^n(r) = \sum_{k=1}^{r} \left(\sigma_{1,k}^{i,n}\right)^2 - \beta_{i,1}^n r$$

where $\beta_{i,1}^n$ is a positive real number. The function $L_{i,1}^n : \mathbb{N} \to \mathbb{R}$ represents the trade-off between accuracy and memory storage on every iteration. It reads as the sum of two terms: on the one hand, $\sum_{k=1}^{r} \left(\sigma_{1,k}^{i,n}\right)^2$ is equal to the ℓ^2 norm of the rank-r truncated POD of $\overline{W}_0^n$ and increases with r; on the other hand, $\beta_{i,1}^n r$ is a term which reflects the memory need related to the storage of a rank-r truncated POD of $\overline{W}_0^n$. An integer $r_{i,1}^n \in \mathbb{N}$ solution to

$$r_{i,1}^n \in \operatorname*{argmax}_{r\in\mathbb{N}} L_{i,1}^n(r)$$

is a value of rank which enables to obtain a reasonable trade-off between the accuracy of the truncated POD and its memory storage. Then, $\tau_n(1)$ is chosen as the optimum index $i \in \mathcal{I}_0^n$ such that

$$\tau_n(1) = \operatorname*{argmax}_{i\in\mathcal{I}_0^n} L_{i,1}^n(r_{i,1}^n) = \operatorname*{argmax}_{i\in\mathcal{I}_0^n} \max_{r\in\mathbb{N}} L_{i,1}^n(r),$$

and gives the index of the first variable in the TT computed at the n^{th} iteration of the SoTT algorithm. A preliminary value of the rank $\overline{K}_1^n$ is then chosen so that $\overline{K}_1^n = r_{\tau_n(1),1}^n$.

An additional step is used at line 9 for the definition of the final value of the rank K_1^n which ensures that if

$$\sum_{k\geq\overline{K}_1^n} \left(\sigma_{1,k}^{\tau^n(1),n}\right)^2 \leq \frac{\epsilon^2}{d-1},$$

then the final value of K_1^n is the lowest possible rank which guarantees that

$$\sum_{k\geq K_1^n} \left(\sigma_{1,k}^{\tau^n(1),n}\right)^2 \leq \frac{\epsilon^2}{d-1}.$$

Now we need to select the values for the rest of the permutations $\tau_n(2), \cdots, \tau_n(d)$. In order to choose the complete order of the variables entering the definition of the n^{th} TT term, one uses a similar iterative procedure applied to the $d-1$-order tensor $\overline{W}_1^n$ which reads as the projection of the tensor $\overline{W}_0^n$ onto the K_1^n first POD modes obtained from the $\tau_n(1)^{th}$ partitioning of variables.

We observe that the choice of the values of $\beta_{i,j}^n > 0$ at Step 8 of the SoTT algorithm is critical for its efficiency. In practice, in the case where for all $1 \leq j \leq d$, $\#\Omega_j = \mathcal{N}_j < +\infty$ (i.e. when the tensor is defined on a discrete domain), we make the following choice:

$$\beta_{i,j}^n = \frac{\mathcal{N}_i + \Pi_{i'\in\mathcal{I}_j^n\setminus\{i\}}\mathcal{N}_{i'}}{\Pi_{i'\in\mathcal{I}_j^n}\mathcal{N}_{i'}} \sum_{k=1}^{+\infty} \left(\sigma_{j,k}^{i,n}\right)^2 = \frac{\mathcal{N}_i + \Pi_{i'\in\mathcal{I}_j^n\setminus\{i\}}\mathcal{N}_{i'}}{\Pi_{i'\in\mathcal{I}_j^n}\mathcal{N}_{i'}} \|\overline{W}_{j-1}^n\|_{\ell^2}^2. \tag{2.4}$$

Introducing the function $\mathbb{N} \ni r \mapsto L_{i,j}^n(r) := \sum_{k=1}^{r} \left(\sigma_{j,k}^{i,n}\right)^2 - \beta_{i,j}^n r$, for $1 \leq j \leq d$, it holds that $L_{i,j}^n(0) = 0$, and there exists at least one $r_{i,j}^n \in \left\{0, \cdots, \min\left(\mathcal{N}_i, \Pi_{i'\in\mathcal{I}_j^n\setminus\{i\}}\mathcal{N}_{i'}\right)\right\}$ so that

$$r_{i,j}^n \in \operatorname*{argmax}_{r\in\left\{0,\cdots,\min\left(\mathcal{N}_i,\Pi_{i'\in\mathcal{I}_j^n\setminus\{i\}}\mathcal{N}_{i'}\right)\right\}} L_{i,j}^n(r),$$

and

$$L_{i,j}^n(r_{i,j}^n) \geq 0.$$

2.3.2 Exponential convergence of the SoTT algorithm in finite dimension

The aim of this section is to prove that the SoTT algorithm converges exponentially fast with the number of iterations in finite dimension.

Proposition 2.3.1 *Let us assume that for all $1 \leq j \leq d$, $\#\Omega_j < +\infty$. Then, there exists $0 < \alpha < 1$ such that for all $n \in \mathbb{N}^*$,*

$$\|W^n\|^2_{L^2(\Omega)} \leq \alpha^n \|W\|^2_{L^2(\Omega)}, \tag{2.5}$$

with

$$\alpha \leq 1 - \frac{1}{\mathcal{N}^{\lceil d/2 \rceil (\lceil d/2 \rceil + 1)}},$$

where $\mathcal{N} := \max_{1 \leq i \leq d} \#\Omega_i$.

We observe in practice that the upper bound on the convergence rate of the SoTT algorithm given by Proposition 2.3.1 is pessimistic. We refer the reader to Section 2.5 for numerical results which illustrate this fact.

Proof 1 *Let us begin by proving that for all $n \in \mathbb{N}^*$,*

$$\|W^n\|^2_{L^2} \leq \left(1 - \frac{1}{\mathcal{N}^{\lceil d/2 \rceil (\lceil d/2 \rceil + 1)}}\right) \left\|W^{n-1}\right\|^2_{L^2}. \tag{2.6}$$

Indeed, for all $n \in \mathbb{N}^*$, let us denote by

$$U^n(x_1, \cdots, x_d) := R^n_1(x_{\tau^n(1)}) R^n_2(x_{\tau^n(2)}) \cdots R^n_d(x_{\tau^n(d)}), \quad \forall (x_1, \cdots, x_d) \in \Omega_1 \times \cdots \times \Omega_d.$$

Note that, by construction and definition of the SoTT algorithm, $\left\langle W^{n-1} - U^n, U^n \right\rangle_{L^2(\Omega)} = 0$, so that

$$\|W^n\|^2_{L^2} = \left\|W^{n-1} - U^n\right\|^2_{L^2} = \left\|W^{n-1}\right\|^2_{L^2} - \|U^n\|^2_{L^2}. \tag{2.7}$$

By definition of the algorithm, it holds that for all $1 \leq j \leq d-1$, K^n_j is lower than the minimum of the cardinality of $D^n_{j-1} \times \Omega_i$ and the cardinality of $\bigtimes_{i' \in \mathcal{I}^n_{j-1} \setminus \{i'\}} \Omega_{i'}$. Hence, we have

$$K^n_j \leq \min\left(\mathcal{N} K^n_{j-1}, \mathcal{N}^{d-j}\right),$$

where $K^n_0 = 1$. Thus, by induction, we obtain that for all $1 \leq j \leq d-1$,

$$K^n_j \leq \min\left(\mathcal{N}^j, \mathcal{N}^{d-j}\right).$$

As a consequence, for all $n \in \mathbb{N}^*$, and all $1 \leq j \leq d-1$, we obtain that for all $i \in \mathcal{I}^n_{j-1}$,

$$\#D^n_{j-1} \times \Omega_i \leq \mathcal{N} \min\left(\mathcal{N}^{j-1}, \mathcal{N}^{d+1-j}\right) = \min\left(\mathcal{N}^j, \mathcal{N}^{d+2-j}\right) \quad \text{and} \quad \#\bigtimes_{i' \in \mathcal{I}^n_{j-1} \setminus \{i\}} \Omega_{i'} \leq \mathcal{N}^{d-j}.$$

Thus, for all $i \in \mathcal{I}^n_{j-1}$,

$$\min\left(\#D^n_{j-1} \times \Omega_i, \#\bigtimes_{i' \in \mathcal{I}^n_{j-1} \setminus \{i\}} \Omega_{i'}\right) \leq \min\left(\mathcal{N}^j, \mathcal{N}^{d-j}\right). \tag{2.8}$$

As a consequence, for all $i \in \mathcal{I}^n_{j-1}$, it holds that

$$\left(\sigma^{i,n}_{j,1}\right)^2 \geq \frac{\left\|\overline{W}^n_{j-1}\right\|^2_{L^2}}{\min\left(\mathcal{N}^j, \mathcal{N}^{d-j}\right)}.$$

Moreover, denoting by $\widehat{W}^n_j := \sum_{k_j=1}^{K^n_j} \sigma^{\tau^n(j),n}_{j,k_j} U^{\tau^n(j),n}_{j,k_j} \otimes V^{\tau^n(j),n}_{j,k_j}$, it holds that

$$\left\|\overline{W}^n_{j-1} - \widehat{W}^n_j\right\|^2_{L^2\left(D^n_{j-1} \times \bigtimes_{i \in \mathcal{I}^n_{j-1}} \Omega_i\right)} \leq \left(\sigma^{i,n}_{j,1}\right)^2.$$

Thus, using the fact that $\overline{W}^n_{j-1} - \widehat{W}^n_j$ *is orthogonal to* $\widehat{W}^n_j$*, we obtain that*

$$\begin{aligned}\left\|\widehat{W}^n_j\right\|^2_{L^2\left(D^n_{j-1}\times \mathbb{X}_{i\in\mathcal{I}^n_{j-1}}\Omega_i\right)} &= \left\|\overline{W}^n_{j-1}\right\|^2_{L^2\left(D^n_{j-1}\times \mathbb{X}_{i\in\mathcal{I}^n_{j-1}}\Omega_i\right)} - \left\|\overline{W}^n_{j-1} - \widehat{W}^n_j\right\|^2_{L^2\left(D^n_{j-1}\times \mathbb{X}_{i\in\mathcal{I}^n_{j-1}}\Omega_i\right)} \\ &\geq \left\|\overline{W}^n_{j-1}\right\|^2_{L^2\left(D^n_{j-1}\times \mathbb{X}_{i\in\mathcal{I}^n_{j-1}}\Omega_i\right)} \left(1 - \frac{1}{\min\left(\mathcal{N}^j, \mathcal{N}^{d-j}\right)}\right).\end{aligned}$$

Lastly, using the fact that $\left(U^{\tau^n(j),n}_{j,k_j}\right)_{1\leq k_j\leq K^j_n}$ *is an orthonormal family of* $L^2\left(\Omega_{\tau^n(j)}\right)$ *and* $\left(V^{\tau^n(j),n}_{j,k_j}\right)_{1\leq k_j\leq K^j_n}$ *is an orthonormal family of* $L^2\left(\mathbb{X}_{i\in\mathcal{I}^n_j}\Omega_i\right)$*, it holds that*

$$\left\|\widehat{W}^n_j\right\|^2_{L^2\left(D^n_{j-1}\times \mathbb{X}_{i\in\mathcal{I}^n_{j-1}}\Omega_i\right)} = \left\|\overline{W}^n_j\right\|^2_{L^2\left(D^n_j\times \mathbb{X}_{i\in\mathcal{I}^n_j}\Omega_i\right)} = \sum_{k_j=1}^{K^n_j}\left(\sigma^{\tau^n(j),n}_{j,k_j}\right)^2.$$

As a consequence, we obtain that for all $n\in\mathbb{N}^*$ *and for all* $1\leq j\leq d-1$*,*

$$\left\|\overline{W}^n_j\right\|^2_{L^2\left(D^n_j\times \mathbb{X}_{i\in\mathcal{I}^n_j}\Omega_i\right)} \geq \left\|\overline{W}^n_{j-1}\right\|^2_{L^2\left(D^n_{j-1}\times \mathbb{X}_{i\in\mathcal{I}^n_{j-1}}\Omega_i\right)}.$$

In addition, it can easily be checked that $\|U^n\|_{L^2(\Omega)^2} = \left\|\overline{W}^n_{d-1}\right\|^2_{L^2\left(D^n_{d-1}\times\Omega_{\tau_n(d)}\right)}$*. Thus, by induction over* $1\leq j\leq d-1$*, we obtain that for all* $n\in\mathbb{N}^*$*,*

$$\begin{aligned}\|U^n\|_{L^2(\Omega)^2} &\geq \left\|\overline{W}^n_0\right\|^2_{L^2(\Omega)} \frac{1}{\Pi_{1\leq j\leq d-1}\min(\mathcal{N}^j,\mathcal{N}^{d-j})} \\ &= \left\|W^{n-1}\right\|^2_{L^2(\Omega)} \frac{1}{\Pi_{1\leq j\leq d-1}\min(\mathcal{N}^j,\mathcal{N}^{d-j})} \\ &\geq \left\|W^{n-1}\right\|^2_{L^2(\Omega)} \frac{1}{\mathcal{N}^{\sum_{j=1}^{\lceil d/2\rceil} j+\sum_{j=\lfloor d/2\rfloor}^{d-1} d-j}} \\ &\geq \left\|W^{n-1}\right\|^2_{L^2(\Omega)} \frac{1}{\mathcal{N}^{2\sum_{j=1}^{\lceil d/2\rceil} j}} \\ &= \left\|W^{n-1}\right\|^2_{L^2(\Omega)} \frac{1}{\mathcal{N}^{\lceil d/2\rceil(\lceil d/2\rceil+1)}}.\end{aligned}$$

Collecting this estimate with (2.7), we obtain (2.6). Thus, by induction, we easily obtain the desired result (2.5).

2.3.3 Complexity estimate of the SoTT algorithm

The aim of this section is to provide some estimates on the complexity of the computational cost of the SoTT algorithm. Let us assume here that there exists $\mathcal{N}\in\mathbb{N}^*$ such that $\#\Omega_i\leq\mathcal{N}$ for all $1\leq i\leq d$.

We detail the computational cost of each iteration $n\in\mathbb{N}^*$. The computational cost is concentrated in the computation of the different POD decompositions of the tensor $\overline{W}^n_{j-1}$ for each $1\leq j\leq d-1$ (l.7 of SoTT algorithm), which can be estimated using (2.2). In this process the most expensive part is the one in which we compute the SVD of the unfoldings. Exploting the computation presented in Appendix A.1.2 this cost can be reduced significantly. We consider two different cases.

Case 1: Unbounded ranks

Let us begin by giving a very pessimistic bound in the case where no upper bound on the ranks K^n_j is imposed for all $1\leq j\leq d$. From (2.8), it holds that the computational cost of each POD decomposition scales like

$$\mathcal{O}\left(\max\left(\mathcal{N}^j,\mathcal{N}^{d-j}\right)\min\left(\mathcal{N}^j,\mathcal{N}^{d-j}\right)^2\right).$$

Thus, the total computational cost of the POD decompositions of one iteration of the SoTT algorithm is of the order of

$$\mathcal{O}\left(\sum_{j=1}^{d-1}(d-j+1)\max\left(\mathcal{N}^j,\mathcal{N}^{d-j}\right)\min\left(\mathcal{N}^j,\mathcal{N}^{d-j}\right)^2\right) \approx \mathcal{O}\left(d^2\mathcal{N}^{3\lceil d/2\rceil}\right).$$

Case 2: Bounded ranks

Now, let us assume that there exists $R \in \mathbb{N}^*$ such that $R < \mathcal{N}$ and such that for all $1 \leq j \leq d-1$, $K_j^n \leq R$. Then, for $j = 1$, the computational cost of each POD decomposition scales like

$$\mathcal{N}^{d+1}.$$

Besides, for $2 \leq j \leq d-2$, the computational cost of each POD decomposition scales like

$$R^2\mathcal{N}^{d-j+1}.$$

Lastly, for $j = d-1$, there is only one POD decomposition to compute, the cost of which scales like

$$R\mathcal{N}^3.$$

Thus, the total cost of the POD decompositions of one SoTT iteration scales like

$$\mathcal{O}\left(d\mathcal{N}^{d+1} + \sum_{j=2}^{d-2}(d-j+1)R^2\mathcal{N}^{d-j+1} + R\mathcal{N}^3\right) \approx \mathcal{O}\left(d\mathcal{N}^{d+1} + dR^2\mathcal{N}^{d-1} + R\mathcal{N}^3\right).$$

Remark The computational cost of the SoTT algorithm is in general larger than the computational cost of the TT-SVD, as we do not fix a priori the order of the variables. In the first step of the SoTT iteration, the cost is similar to the one of the HOSVD method, in which we compute the POD for all the unfoldings. However, there is a difference in terms of the amount of memory needed to carry out the computations. In the practical implementation of SoTT, it is not necessary to store POD decompositions of all the unfoldings of the tensor, only a POD decomposition of the best one. In addition, we do not need to store the potentially dense core tensor. In the case in which the values of the ranks in SoTT are a priori fixed (for instance to 1, as will be the case in the forthcoming section), the computational cost of a POD decomposition can be reduced.

Storage

Let d be the dimension of the tensor. Let us suppose that the SoTT ranks K_j^n are equal for all the values of j, $1 \leq j \leq d$, and for all the n terms of the SoTT sum. For simplicity in the notation, in the following we will denote them by K. The total storage of the SoTT tensor format is then

$$\mathcal{O}(n \times \mathcal{N} \times ((d-2)K^2 + 2K)).$$

Comparing this value with the storage of a tensor on its full format $\mathcal{O}(\mathcal{N}^d)$, we obtain a compression rate of order:

$$\mathcal{O}(n^{-1} \times \mathcal{N}^{d-1} \times ((d-2)K^2 + 2K)^{-1}).$$

2.4 CP-TT: fixed-rank SoTT algorithm with rank 1

We make here a focus on a particular variant of the SoTT algorithm where all the ranks K_j^n are a priori chosen to be fixed and equal to 1 for all $1 \leq j \leq d$ and all iterations $n \in \mathbb{N}^*$. We refer the reader to [259] for a review on the stability properties of rank-1 tensor decompositions. As an output, the SoTT algorithm then computes an approximation of the tensor W in a CP format and we refer to the resulting procedure as the CP-TT algorithm. More precisely, for all $1 \leq j \leq d$ and $n \in \mathbb{N}^*$, Step 8 of the Algorithm 7 is replaced

by the following step:
Select $i_j^n \in \mathcal{I}_{j-1}^n$ so that

$$i_j^n \in \underset{i \in \mathcal{I}_{n-1}}{\operatorname{argmax}} \left(\sigma_{j,1}^{i,n}\right)^2$$

and then Step 10 is not performed.

As an output, after n iterations of the CP-TT algorithm, the method greedily produces an approximation of the tensor W under the CP format

$$W \approx \sum_{k=1}^{n} R_1^k(x_{\tau_k(1)}) \cdots R_d^k(x_{\tau_k(d)})$$

where for all $1 \leq k \leq n$ and all $1 \leq i \leq d$, $R_i^k \in L^2(\Omega_{\tau_k(i)})$. Find in the following Algorithm 8 the detailed method.

We make a specific focus on this particular case because we numerically observed that this algorithm owns interesting stability and approximation properties in comparison to other more classical numerical methods like ALS or ASVD for the computation of a CP approximation of a tensor, especially when the order of the tensor d is high. For the sake of comparison, we recall the ALS and ASVD algorithm in Algorithm 4 and Algorithm 5 respectively. For more details and convergence properties of the ALS method, we refer the reader to the following series of works [64, 131, 254, 255, 256]. The ASVD method is proposed in [257].

The closest method to CP-TT we found in the literature is the so called TTr1 algorithm, proposed in [258]. In this method, a TT-SVD decomposition with fixed ranks equal to one is computed at every stage of the algorithm. All the possible rank-1 terms are computed, stored and ordered according to the singular values of the corresponding POD decompositions. Favorable orthogonality properties of the method enable to truncate the obtained CP decomposition in order to fulfill a prescribed accuracy. The main differences of the TTr1 algorithm with respect to CP-TT are the following: first, the order of the variables is not fixed a priori in CP-TT, but is fixed beforehand in TTr1. Second, a greedy procedure is used in CP-TT computing one term at a time, whereas in TTr1 all the terms are computed at the same time before truncation. Hence, the computational cost per term is larger in CP-TT, but the storage need is much less significant, which seems beneficial for the compression of higher order tensors.

2.5 Numerical Experiments

In this section, several numerical experiments are proposed. In the first part, we compare several rank-1 update methods: ALS, ASVD, TTr1 and CP-TT, on the compression of a set of random functions belonging to certain classes of regularity. Then in the second part, we illustrate the behavior of the SoTT method with respect to the standard TT-SVD algorithm on several test cases which consist in compressing the solution of a parametric Partial Differential Equation (PDE), as well as other functions arising in different applications.

2.5.1 Comparison between CP-TT and other rank-one update methods

The aim of this section is to compare the efficiency of the CP-TT algorithm for the computation of approximations of a tensor in a CP format with other rank-1 methods, ALS, ASVD and TTr1.

Let $(x_1, \ldots, x_d) \in \Omega = [0,1]^d$. Let $(k_1, ..., k_d) \in \mathbb{N}^d$ be the wave numbers. The function to be compressed is assumed to be given in a Tucker format :

$$W(x_1, \ldots, x_d) = \sum_{k_1=1}^{l_1} \sum_{k_2=1}^{l_2} \cdots \sum_{k_d=1}^{l_d} a_{k_1 \ldots k_d} \sin(\pi k_1 x_1) \times \ldots \times \sin(\pi k_d x_d) \tag{2.9}$$

First, the values of $(l_i)_{1 \leq i \leq d} \in \mathbb{N}^*$ are chosen to be a family of independent random integers uniformly distributed between 1 and 6.

Second, the coefficients $(a_{k_1 \ldots k_d})_{1 \leq k_1 \leq l_1, \cdots, 1 \leq k_d \leq l_d}$ are randomly chosen as follows: Let $\beta > 0$. Let $(\alpha_{k_1 \ldots k_d})_{1 \leq k_1 \leq l_1, \ldots, 1 \leq k_d \leq l_d}$ be a family of independent random variables uniformly distributed in $[-1, 1]$. For

Algorithm 8 CP-TT algorithm

Require: $\epsilon > 0$, $W \in L^2(\Omega)$

Output: $N \in \mathbb{N}^*$, for all $1 \leq n \leq N$, $\tau^n \in \mathcal{S}_d$ and for all $i = 1, \cdots, d$, $R_i^n \in L^2\left(\Omega_{\tau^n(i)}, \mathbb{R}^2\right)$ so that the sum of Tensor Trains $\widetilde{W} \in L^2(\Omega)$ defined by

$$\widetilde{W}(x_1, \cdots, x_d) := \sum_{n=1}^{N} R_1^n(x_{\tau^n(1)}) R_2^n(x_{\tau^n(2)}) \cdots R_d^n(x_{\tau^n(d)}) \quad \forall (x_1, \cdots, x_d) \in \Omega,$$

satisfies $\|W - \widetilde{W}\|^2_{L^2(\Omega)} \leq \epsilon$.

Set $W^0 = W$, $n = 1$.

while $\|W^{n-1}\|^2_{L^2(\Omega)} > \epsilon$ **do**

Define $K_0^n := 1$, $D_0^n = \{1\}$ and define $\overline{W}_0^n \in L^2(D_0 \times \Omega)$ such that $\overline{W}_0^n(1, \cdot) = W^{n-1}$, $\mathcal{I}_0^n := \{1, \cdots, d\}$.

for $j = 1, \cdots, d-1$ **do**

For all $i \in \mathcal{I}_{j-1}^n$, since $D_{j-1}^n \times \underset{i \in \mathcal{I}_{j-1}^n}{\times} \Omega_i = \left(D_{j-1}^n \times \Omega_i\right) \times \underset{i' \in \mathcal{I}_{j-1}^n \setminus \{i\}}{\times} \Omega_{i'}$, compute the SVD decomposition of $\overline{W}_{j-1}^n$ according to the separation of variables $(D_{j-1}^n \times \Omega_i, \times_{i' \in \mathcal{I}_{j-1}^n \setminus \{i'\}} \Omega_{i'})$ so that

$$\overline{W}_{j-1}^n = \sum_{k \in \mathbb{N}^*} \sigma_{j,k}^{i,n} U_{j,k}^{i,n} \otimes V_{j,k}^{i,n}.$$

Select $i_j^n \in \mathcal{I}_{j-1}^n$ so that

$$i_j^n \in \underset{i \in \mathcal{I}_{n-1}}{\operatorname{argmax}} \left(\sigma_{j,1}^{i,n}\right)^2,$$

Define $\tau^n(j) = i_j^n$.

Define $\mathcal{I}_j^n := \mathcal{I}_{j-1}^n \setminus \{\tau^n(j)\}$ so that $\# I_j^n = d - j$.

Define $D_j^n := \left\{1, \cdots, K_j^n\right\}$ and define $\overline{W}_j^n \in L^2\left(D_j^n \times \times_{i \in \mathcal{I}_j^n} \Omega_i\right)$ by

$$\overline{W}_j^n(k_j, y_{\tau^n(j)}) = \sigma_{j,k_j}^{\tau^n(j),n} V_{j,k_j}^{\tau^n(j),n}(y_{\tau^n(j)})$$

for all $(k_j, y_{\tau^n(j)}) \in D_j^n \times \underset{i \in \mathcal{I}_j^n}{\times} \Omega_i$.

Define $R_j^n \in L^2\left(\Omega_{\tau_n(j)}, \mathbb{R}^2\right)$ as

$$R_j^n(x_{\tau^n(j)}) = \left(U_{j,1}^{\tau_n(j),n}(x_{\tau^n(j)})\right)$$

for all $x_{\tau^n(j)} \in \Omega_{\tau^n(j)}$.

end for

Since $\# I_{d-1}^n = 1$, let $i_d^n \in \{1, \cdots, n\}$ such that $I_{d-1}^n = \{i_d^n\}$. Define $\tau^n(d) = i_d^n$.

Define $R_d^n \in L^2\left(\Omega_{\tau^n(d)}, \mathbb{R}^2\right)$ by

$$R_d^n(x_{\tau^n(d)}) = \left(\sigma_{d-1,1}^{\tau^n(d-1),n} V_{d-1,1}^{\tau^n(d-1),n}(x_{\tau^n(d)})\right).$$

Compute $W^n(x_1, \cdots, x_d) = W^{n-1}(x_1, \cdots, x_d) - R_1^n(x_{\tau^n(1)}) R_2^n(x_{\tau^n(2)}) \cdots R_d^n(x_{\tau^n(d)})$ for all $(x_1, \cdots, x_d) \in \Omega$.

$n = n + 1$

end while

$N = n - 1$

all $1 \leq k_1 \leq l_1$, ..., $1 \leq k_d \leq l_d$, the value $a_{k_1 \ldots k_d}$ is then defined as:

$$a_{k_1 \ldots k_d} = \frac{\alpha_{k_1 \ldots k_d}}{\left(\sqrt{k_1^2 + \ldots + k_d^2}\right)^{\beta}}.$$

For different random samples and different values of the coefficient β, we obtain different families of functions W given by (2.9). Let us remark that more detailed rank-r CP approximation of the orthogonal Tucker tensor could be seen in [260], [101].

We are testing how the four methods behave for the compression of 32 different functions generated by the random procedure described above for values of d ranging from 4 to 16. Let us point out that ALS and ASVD are both fixed point based methods, in contrast to CP-TT and TTr1, and the tolerance for the fixed point procedure has been set as $\eta = 1.0 \times 10^{-4}$. The maximum number of iterations of the method $it_{max} = 100$. A uniform discretization grid of Ω with 25 degrees of freedom per direction is used for the discretization of W.

Description and analysis of Sobolev regularity in the d-torus

Let us establish the modes $R_1(x_1), \ldots, R_d(x_d)$ contained in the space $L^2(\Omega)$. Let $\Omega : [0,1]^d$ be the domain. By simplicity of notation, let us name the product: $R(x_1, \ldots, x_d) = R_1(x_1) \ldots R_d(x_d)$.

We want to prove that the norm of the modes is finite: $\|R(x_1, \ldots, x_d)\|_{L^2(\Omega)} < \infty$.

To do so, we are going to work with the square of the norm. Writing the modes in the continuous space on the exponential representation of the sinus

$$\int_\Omega \|R(x_1, \ldots, x_d)\|^2 d\Omega = \int_\Omega \left(a_{k_1 \ldots k_d} e^{i2\pi k_1 x_1} \ldots e^{i2\pi k_d x_d}\right)^2 d\Omega$$

Extending to a certain values of the wave number also the amplitude:

$$\int_\Omega \|R(x_1, \ldots, x_d)\|^2 d\Omega = \int_\Omega \left(\sum_{k_1=1}^{l_1} \cdots \sum_{k_d=1}^{l_d} \frac{\alpha_{k_1,\ldots,k_d}}{\sqrt{k_1^2 + \ldots + k_d^2}^\beta} e^{i2\pi k_1 x_1} \ldots e^{i2\pi k_d x_d} \right)^2 d\Omega =$$

$$= \sum_{k_1=1}^{l_1} \cdots \sum_{k_d=1}^{l_d} \sum_{m_1=1}^{l_1} \cdots \sum_{m_d=1}^{l_d} \frac{\alpha_{k_1,\ldots,k_d} \alpha_{m_1,\ldots,m_d}}{\sqrt{(k_1^2 + \ldots + k_d^2)(m_1^2 + \ldots + m_d^2)}^\beta} \int_\Omega e^{i2\pi(k_1 - m_1)x_1} \ldots e^{i2\pi(k_d - m_d)x_d} d\Omega =$$

$$= \sum_{k_1=1}^{l_1} \cdots \sum_{k_d=1}^{l_d} \frac{\alpha^2_{k_1,\ldots,k_d}}{(k_1^2 + \ldots + k_d^2)^\beta} \leq \sum_{k_1=1}^{l_1} \cdots \sum_{k_d=1}^{l_d} \frac{1}{(k_1^2 + \ldots + k_d^2)^\beta} = \sum_{k_1=1}^{l_1} \cdots \sum_{k_d=1}^{l_d} \frac{1}{\sum_{i=1}^d (k_i^2)^\beta} < \infty$$

where we have used the orthogonality of the functions inside the integral. Solving this inequality in order to establish boundaries to the value of β for the functions to be regular in $L^2(\Omega)$, we obtain a relation between the dimension and the parameter: $d - 2\beta < 0 \implies \beta > \frac{d}{2}$.

If we want to extend the regularity of the functions to $H^1(\Omega)$ the calculus is completely analogue. The relation obtained in this case: $\beta - 1 > \frac{d}{2} \implies \beta > 1 + \frac{d}{2}$.

In the view of this results, from the dimension of the problem we can select a value for the parameter β that will determine the regularity of the functions.

Results for functions with $\beta = \frac{d}{2} + 0.1$

We begin by presenting here some numerical tests obtained with functions generated with $\beta = \frac{d}{2} + 0.1$.

We first present numerical experiments comparing the compression of the function given by Equation 2.9 computed by CP-TT, TTr1, ALS and ASVD in cases where $d = 4$ in Figure 2.1. Note that the memory required by the TTr1 method has prevented us from being able to carry out the method for higher values of d. Hence, for higher values of d, we only compare CP-TT with ALS and ASVD methods in Figure 2.2 for $d = 12$ and $d = 16$. The mean and standard deviation of the L^2 norm of the difference between the exact function W and its approximation given by any method is plotted as a function of the rank of the approximation. Let us make a remark: the negative values of the squared norm of the residual appear by subtracting the standard deviation to the mean of the results in Figure 2.1 and Figure 2.2.

Note that in the case $d = 4$ (Figure 2.1), ALS outperforms ASVD and CP-TT. The ALS also outperforms TTr1 for small values of the rank. However, in cases where $d = 12$ and $d = 16$ (Figure 2.2), CP-TT has a better numerical behavior when considering the decay of the norm of the residual with respect to the number of terms. In particular, the compression rate is better on average and the norm decay of the error with respect to the rank of the approximation is less subject to statistical noise.

Table 2.1 summarizes the numerical results obtained with the CP-TT, ALS and ASVD methods. In particular, the mean and the standard deviation of the error (on the 32 random functions) are reported for ranks equal to $25, 50, 75$ and tensor orders $d = 4, 6, 8, 10, 12, 14, 16$. We see here again that for low-order tensors (here when $d = 4$) ALS has better performances, whereas for higher order tensors CP-TT outperforms the other methods both in terms of mean and standard deviation.

Similar numerical tests have been performed in the case where the value of the parameter β is chosen to be equal to $\frac{d}{2} + 1.1$. We refer the reader to the Appendix A.2 to see them in more detail. The results obtained

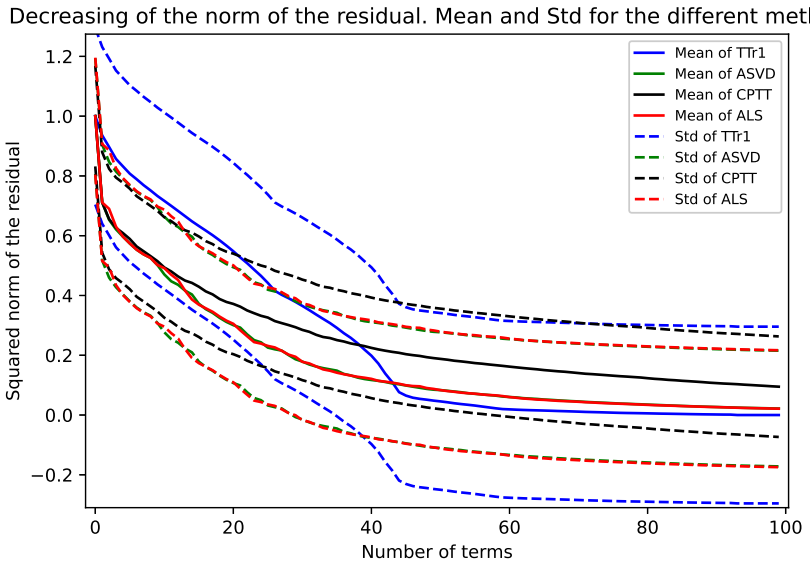


Figure 2.1: Case $d = 4$ and $\beta = \frac{d}{2} + 0.1$. Mean and standard deviation of the L^2 norm of the difference between the exact function W and its approximation given by ALS (red), TTr1 (blue), CP-TT (black) and ASVD (green) as a function of the number of terms. See Table 2.1 for more detailed information.

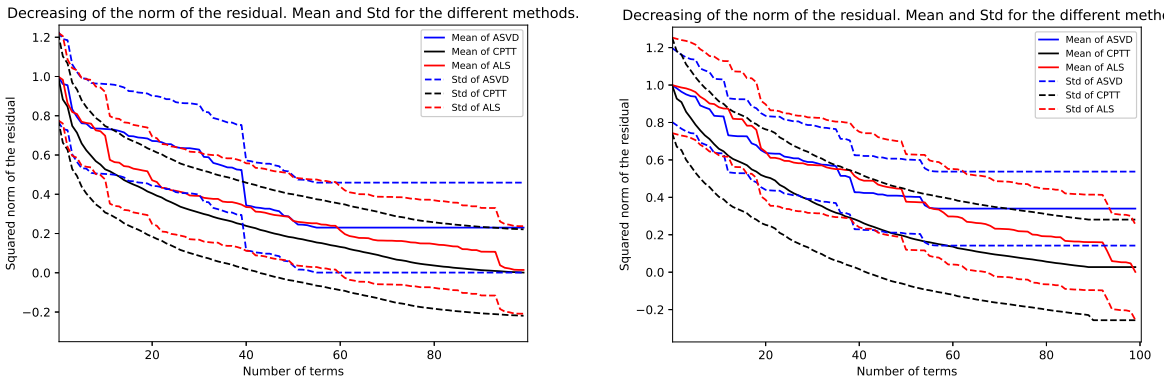


Figure 2.2: Case $\beta = \frac{d}{2} + 0.1$. Mean and standard deviation of the L^2 norm of the difference between the exact function W and its approximation given by ALS (red), ASVD (blue) and CP-TT (black) as a function of the number of terms. Left: case $d = 12$. Right: case $d = 16$. See Table 2.1 for more detailed information.

on $H^1(\Omega)$ functions are equivalent to the ones shown for $L^2(\Omega)$ functions, showing that the decrease in the error norm with the approximation rank is quite regular in CP-TT and behaves in a quite stable way also for higher order tensors. As a summary of the tests done we add Table A.1 as the counterpart of Table 2.1 for $\beta = \frac{d}{2} + 1.1$.

Comparison of the norm of the residual with respect to computational time

It is clear that one iteration of CP-TT is in general more costly in terms of computational time than one ALS iteration. As a consequence, even if the norm of the residual given by the CP-TT algorithm seems to decrease faster as a function of the number of terms in the approximation than with any other rank-1 update methods for high values of d, it is legitimate to compare the norm of the residual given by any method with respect to the computational time needed to compute the corresponding approximations.

This is the aim of Figure 2.3, where the norm of the residual is plotted for CP-TT, ALS and ASVD as a function of the computational time.

		Mean				Std			
Dimension (d)	Rank (r)	ALS	CPTT	ASVD	TTr1	ALS	CPTT	ASVD	TTr1
4	25	**0.2942**	0.3826	0.3118	0.4948	**0.0702**	0.0850	0.0843	0.0644
	50	**0.1082**	0.2433	0.1257	0.2092	**0.0326**	0.0568	0.0664	0.0981
	75	**0.0508**	0.1681	0.0689	0.0928	**0.0180**	0.0408	0.0666	0.0720
6	25	0.4479	**0.3771**	0.4806		0.1099	**0.0826**	0.1074	
	50	0.2705	**0.1982**	0.2883		0.0752	**0.0485**	0.0675	
	75	0.1232	**0.0806**	0.1369		0.0325	**0.0252**	0.0368	
8	25	0.5341	**0.3707**	0.5532		0.1183	**0.0592**	0.1238	
	50	0.3060	**0.1909**	0.3415		0.0722	**0.0341**	0.0932	
	75	0.1592	**0.0682**	0.1807		0.0435	**0.0160**	0.0625	
10	25	0.5023	**0.3598**	0.5451		0.0879	**0.0643**	0.1055	
	50	0.3191	**0.1826**	0.3797		0.0643	**0.0342**	0.0774	
	75	0.1714	**0.0655**	0.2792		0.0453	**0.0162**	0.1265	
12	25	0.5170	**0.3246**	0.5639		0.1117	**0.0576**	0.1250	
	50	0.3249	**0.1623**	0.4206		0.0824	**0.0286**	0.1579	
	75	0.1543	**0.0579**	0.3498		0.0369	**0.0113**	0.2057	
14	25	0.4443	**0.2336**	0.4783		0.1712	**0.1064**	0.1585	
	50	0.2407	**0.1004**	0.3307		0.0937	**0.0588**	0.1737	
	75	0.1411	**0.0321**	0.2230		0.0541	**0.0235**	0.1821	
16	25	0.5529	**0.3160**	0.6150		0.1305	**0.0818**	0.1656	
	50	0.3487	**0.1448**	0.4424		0.0849	**0.0389**	0.1942	
	75	0.1946	**0.0616**	0.3678		0.0905	**0.0289**	0.2354	

Table 2.1: Mean and standard deviation of the norm of the residual for 32 random functions in the case where $\beta = \frac{d}{2} + 0.1$. For TTr1, due to memory issues, it wasn't possible to obtain definite results for order higher than four.

We observe in these tests that, in terms of mean of the decay of the norm of the residual as a function of the computational time, the three methods perform similarly in the sense that they are uniformly decreasing even if they present small jumps or flat intervals. However, we observe that CP-TT has a lower stochastic variability than ALS and ASVD. For functions in $H^1(\Omega)$ the behavior is similar.

SoTT method

The goal of this section was to propose, on a simple synthetic test case, a comparison between rank-one update methods. We have also run the tests presented above by using the SoTT method. For the sake of brevity, we will synthetically comment the results in this section. For all dimensions, the SoTT method outperforms its rank-one particularization (CP-TT), as well as all the other rank-one update methods. Furthermore, it can reach a relative accuracy of 10^{-3} by using 3-4 TT terms. This has two important consequences. In terms of memory, SoTT outperforms in general rank-one update methods. Concerning the computational cost: the cost per iteration of SoTT is larger than the cost of all the other methods. It is similar to the one of CP-TT (with some overheads due to the need of computing more singular vectors and values when performing the SVD, instead of just one). However, the large computational cost per iteration is counterbalanced by the fact that the method just needs three of four iterations to converge, as opposed to hundreds of iterations needed for rank-one update methods. These encouraging results motivate further investigation of the SoTT method. In the following section, we propose four different test-cases in which we compare the performances of SoTT with the TT-SVD method. Moreover, we will report the comparison between SoTT and two rank-one update methods, namely CP-TT and ALS, which is the rank-one update method which seems to perform the best for the functions of interest in the present work.

2.5.2 SoTT method for the compression of multivariate functions

In this section we present some numerical experiments to assess the performances of the SoTT method in compressing functions arising in different applications. Two main comparisons are shown: first, we will com-

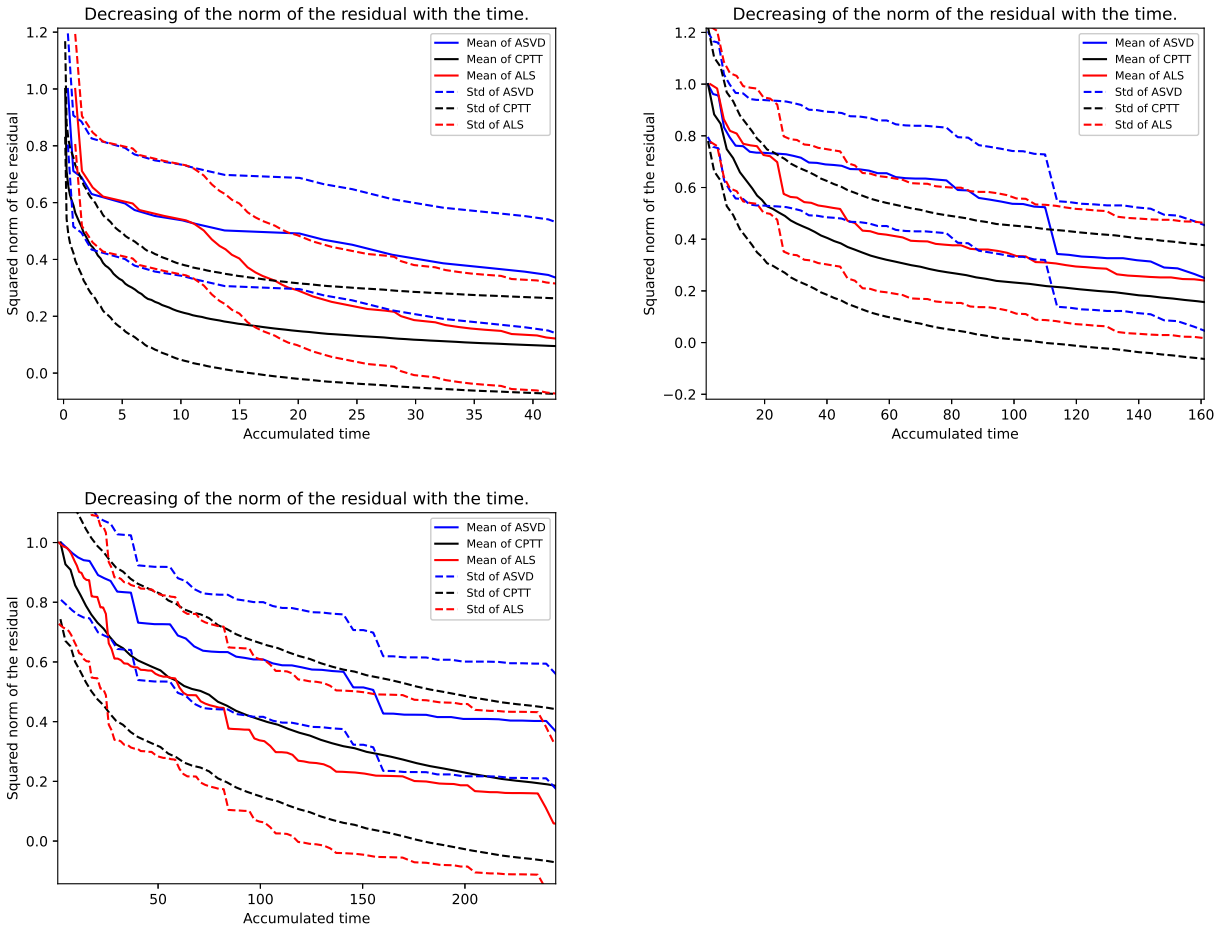


Figure 2.3: Functions with $\beta = \frac{d}{2} + 0.1$. From left to right in the top $d = 4, 12$. In the bottom: $d = 16$. Mean and standard deviation of the norm of the residual as a function of the accumulated time of computation for ALS (red), ASVD (blue) and CP-TT (black).

pare the SoTT method with the classical TT-SVD method, for different values of accuracy and by considering all the possible permutation of the indices (for TT-SVD); second, we will compare the performances (in terms of memory) of SoTT with its rank-one particularization, CP-TT, and the ALS method.

SoTT for the compression of the solution of a parametric reaction diffusion equation

The aim of this section is to illustrate the numerical behavior of the SoTT algorithm where the ranks are not fixed a priori but chosen according to Algorithm 7.

We consider here a fourth-order tensor obtained by solving numerically a 1D-1D parametric Fischer-Kolmogorov-Petrovsky-Piskunov (FKPP) equation. Let $\Omega_1 := [0, 1]$ be the space domain, and $\Omega_2 := [0, 0.25]$ be the time domain. Let $\alpha \in \Omega_3 := [25, 100]$ be the reaction coefficient, and $\beta \in \Omega_4 := [0.25, 0.75]$ be a parameter defining the initial condition. The equation reads: for all $(\alpha, \beta) \in \Omega_3 \times \Omega_4$, find $u_{\alpha,\beta} : \Omega_1 \times \Omega_2 \ni (x, t) \mapsto u_{\alpha,\beta}(x, t) \in \mathbb{R}$ solution to

$$\begin{cases} \partial_t u_{\alpha,\beta} &= \partial_x^2 u_{\alpha,\beta} + \alpha u_{\alpha,\beta}(1 - u_{\alpha,\beta}), & \forall (x, t) \in \Omega_1 \times \Omega_2 \\ u_{\alpha,\beta}(0, t) &= u_{\alpha,\beta}(1, t) = 0, & \forall t \in \Omega_2, \\ u_{\alpha,\beta}(x, 0) &= \exp(-200(x - \beta)^2), & \forall x \in \Omega_1. \end{cases} \tag{2.10}$$

We then define, for all $(x_1, x_2, x_3, x_4) \in \Omega_1 \times \Omega_2 \times \Omega_3 \times \Omega_4$,

$$W(x_1, x_2, x_3, x_4) := u_{x_3,x_4}(x_1, x_2).$$

Equation 2.10 is discretized and solved by means of a classical centered finite difference scheme. Examples of the space-time portrait of the solution for different values of the parameters are shown in Fig.2.4.

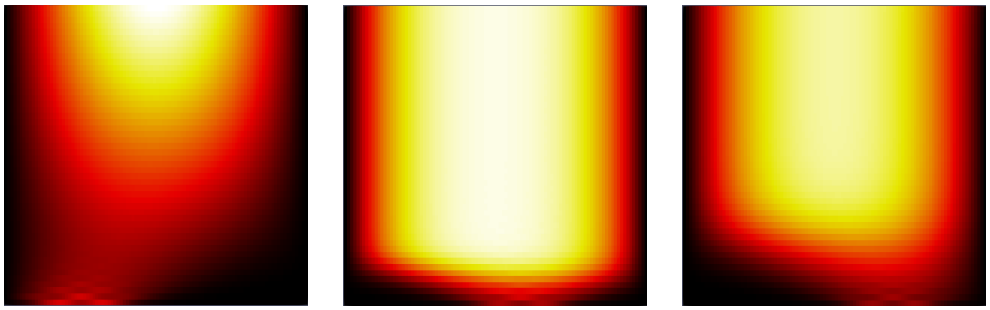


Figure 2.4: Three slices of the full tensor used in Section 2.5.2. The horizontal axis is the space coordinate, the vertical axis is the time coordinate, the color represents the solution value, from 0 (black), to 1 (white) for different values of the parameters determining the initial condition and the reaction coefficient.

We consider uniform discretization grids of Ω_1, Ω_2, Ω_3 and Ω_4 of size $\mathcal{N}_1 = 100$, $\mathcal{N}_2 = 50$, $\mathcal{N}_3 = 10$ and $\mathcal{N}_4 = 10$ respectively.

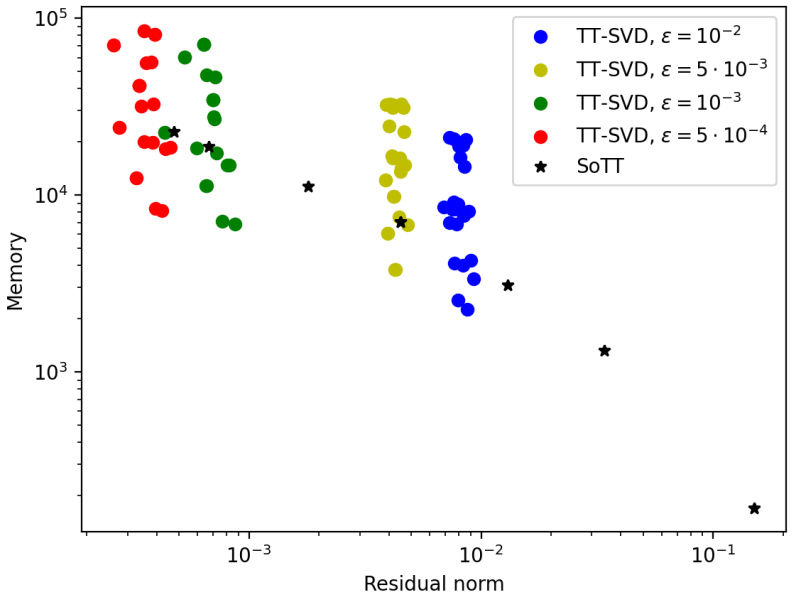


Figure 2.5: Compression test performed in Section 2.5.2, double logarithmic plot of the memory as function of the residual norm for the TT-SVD runs (obtained by considering all the possible permutations of the indices), for several tolerances, and the SoTT approximations.

In Figure 2.5, the memory of the computed approximation (i.e. the number of stored double precision numbers) is plotted as a function of the residual norm, for the TT-SVD approximations corresponding to all the possible 24 choices of permutations of the variable indices and the SoTT approximation. These approximations are computed in all cases for several residual tolerances, ranging from 10^{-2} to $5{\cdot}10^{-4}$. Remark here that the results obtained by the TT-SVD algorithm heavily depend on the order of the variables chosen. The difference in memory between the best and the worst TT-SVD is roughly one order of magnitude, for all the tolerances tested.

We observe in this test case that the SoTT method produces a sub-optimal compression with respect to the best TT-SVD compression. However, it performs better than the average TT-SVD and in general better than the canonical order $1, 2, 3, 4$. The first term computed is a rank-1 update, for the second term the TT

ranks are $[5, 5, 4]$, for the third $[7, 7, 5]$, and in general we observe that the order of the variables change.

In Figure 2.6, we compare the performance of SoTT with CP-TT, its particularization to rank-1 updates. More precisely, the logarithm of the memory is plotted as a function of the logarithm of the residual norm, for 5 iterations of SoTT and approximately 360 iterations of CP-TT and about 300 iterations of ALS. We observe that the performance of SoTT is better than the one of CP-TT and ALS.

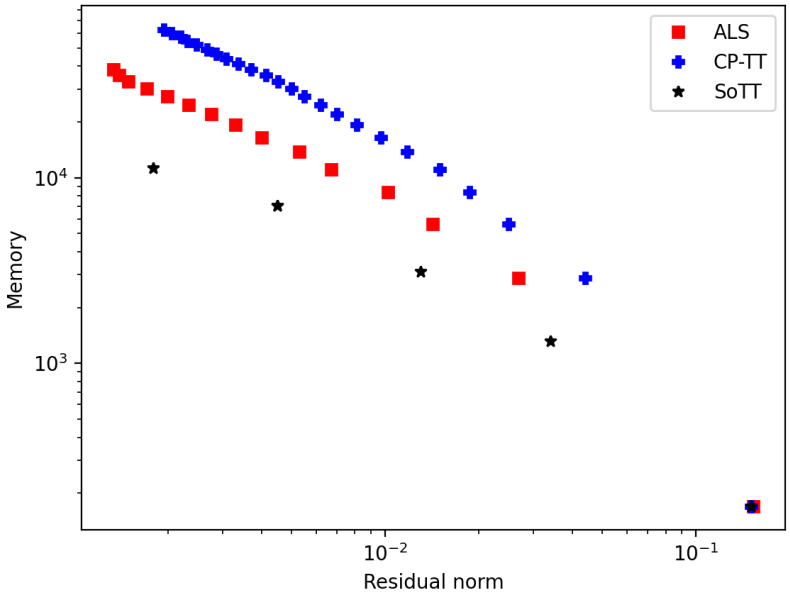


Figure 2.6: Compression test performed in Section2.5.2, double logarithmic plot of the memory as function of the residual norm for the SoTT, the CP-TT and ALS algorithms. For the rank-1 update methods we plotted the result every 16 iterations for the sake of clarity in the graphical representation.

SoTT for the compression of the parametric displacement of a cantilever beam

In this section we consider, as a 4-variate function, the displacement of the tip of a cantilever beam subjected to a uniform load. More details can be found in [261]. Let $L = 1$ be the beam length, perfectly attached in $x = 0$, let E be the Young modulus, $J_x = J_y = J$ be the inertia momentum of the beam (supposed to be symmetric), f_y, f_z the components along the y and z coordinates of a uniform in space load. Let $u_y(x), u_z(x) : [0, 1] \to \mathbb{R}$ be the displacement fields along the y and z directions respectively. It holds:

$$u_y(x) = \frac{f_y}{4EJ}x^2 - \frac{f_y}{6EJ}x^3 + \frac{f_y}{24EJ}x^4, \tag{2.11}$$

$$u_z(x) = \frac{f_z}{4EJ}x^2 - \frac{f_z}{6EJ}x^3 + \frac{f_z}{24EJ}x^4. \tag{2.12}$$

The maximal displacement reads: $u = \left(u_y^2 + u_z^2\right)^{1/2}$, which equals:

$$u(E, J, f_y, f_z) = \frac{\left(f_y^2 + f_z^2\right)^{1/2}}{8EJ}. \tag{2.13}$$

For the present case we considered $E \in [10^8, 2 \cdot 10^8]$, $J \in [0.5, 1]$, $f_y, f_z \in [0, 1]$ and $N_i = 32,\ 1 \leq i \leq 4$.

In Fig. 2.7.a) we show a comparison, in terms of memory as function of the residual norm, between SoTT and the 24 possible TT-SVD, for 3 different levels of tolerance. Due to the function symmetries, the 24 TT-SVD group into 3 clusters of indices permutations. We observe that SoTT is suboptimal if compared to the best possible TT-SVDs, but it is better than the average TT-SVD. In terms of the comparison between SoTT and rank-one update methods, shown in Fig.2.7.b), we observe, that SoTT is performing better than CP-TT and ALS, although the difference is not large for the function considered.

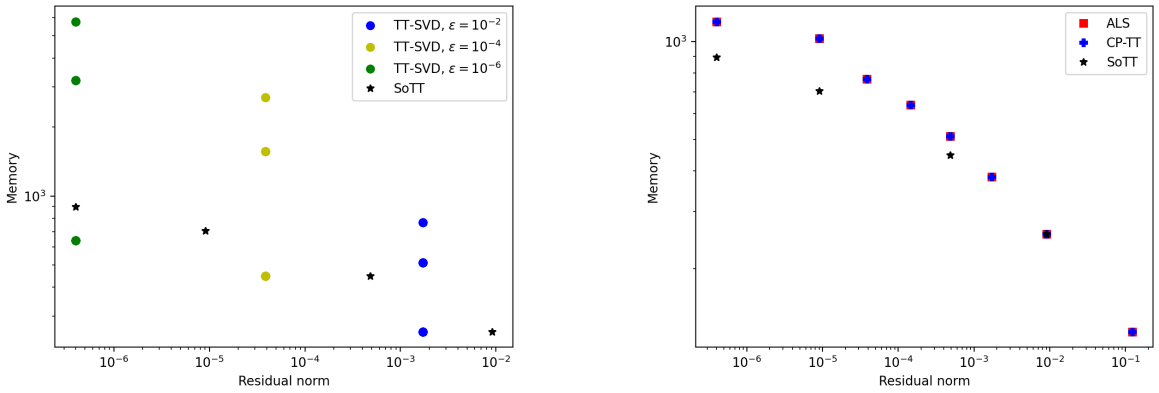


Figure 2.7: Cantilever beam case, presented in Section 2.5.2, memory as function of the residual norm, in logarithmic scale for: a) and b).

SoTT for the compression of the Friedman function

The Friedman function emulates the outcome of a computational process.[1] Let $x \in [0,1]^5$, the Friedman function $f : [0,1]^5 \to \mathbb{R}$ reads:

$$f(x) = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5. \tag{2.14}$$

In terms of discretization, we consider $N_i = 20, 1 \leq i \leq 5$. As for the other test cases, we compared the

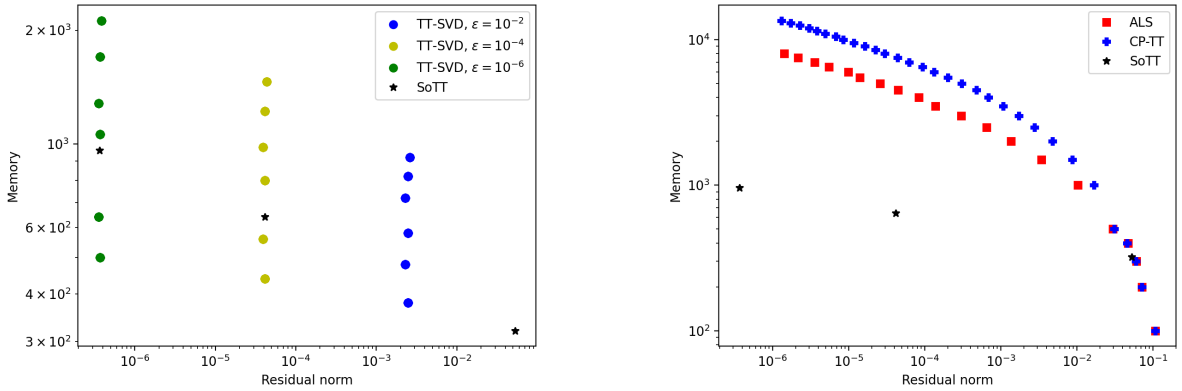


Figure 2.8: Friedman function case, presented in Section 2.5.2, memory as function of the residual norm, in logarithmic scale for: a) and b).

SoTT method with the TT-SVD for all 120 possible permutations of the indices, for 3 different values of tolerance, namely $\varepsilon = \{10^{-2}, 10^{-4}, 10^{-6}\}$. The results are shown in Figure 2.8.a). We observe that, due to the symmetries of the function, the 120 permutations clusters into 6 groups. SoTT is suboptimal with respect to the best TT-SVD, but it performs better than the average, requiring roughly half the memory than the worse TT-SVD. In Figure 2.8.b) we compare SoTT with its rank-1 version, CP-TT and the ALS method. As expected, SoTT outperforms both the rank-one update methods by roughly on order of magnitude in terms of storage for an accuracy of 10^{-6}.

[1] We consider the function as reported in `http://www.sfu.ca/~ssurjano/fried.html`.

SoTT for the compression of the OTL-circuit function

In this section we consider a function relying the values of resistances and gain of a transformerless push-pull circuit (OTL) to its output[2]. Let the variables be $R_1 \in [50, 150], R_2 \in [25, 70], R_f \in [0.5, 3], R_3 \in [1.2, 2.5], R_4 \in [0.25, 1.2], \beta \in [50, 300]$. The circuit output V reads as follows:

$$V_1 = \frac{12R_2}{R_1 + R_2}, \tag{2.15}$$

$$\gamma = \frac{\beta(R_4 + 9) + R_f}{\beta(R_4 + 9)}, \tag{2.16}$$

$$V(R_1, R_2, R_f, R_3, R_4, \beta) = \frac{(V_1 + 0.74)}{\gamma} + \frac{11.35R_f}{\gamma\beta(R_4 + 9)} + \frac{0.74R_f}{\gamma R_3}. \tag{2.17}$$

The results are shown in Fig.2.9. On the left, we show the comparison, in terms of memory as function of

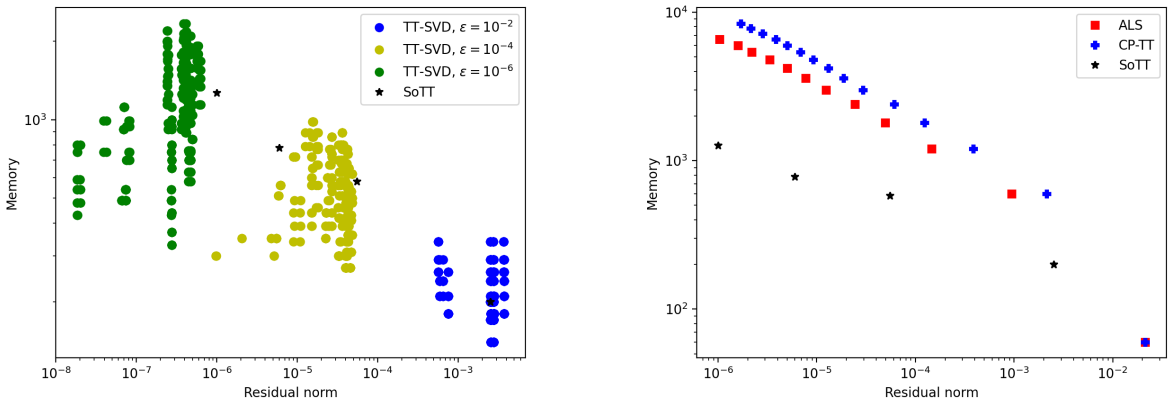


Figure 2.9: OTL circuit case, presented in Section 2.5.2, memory as function of the residual norm, in logarithmic scale for: a) SoTT and the 720 possible TT-SVD, for 3 different values of tolerance, b) ALS, CP-TT and SoTT methods.

the residual, between the TT-SVD performed by considering all the 720 permutations of the indices and the SoTT iterations. On the right, we show the results for the SoTT method compared to its rank one version, CP-TT, and the ALS method. The observed behavior is similar to the one observed and commented in previous test cases.

Computational cost

We summarize some observations on the computational cost, for all the test cases presented in this section. The computational cost per single iteration of SoTT is comparable to the one of CP-TT; both of them are in general larger than the computational cost of a single iteration of ALS, although the latter is featured by a large variance (due to the convergence of the fix point). The overall cost of SoTT to reach a given result in terms of accuracy is significantly smaller if compared to the cost of both ALS and CP-TT. This is due to the fact that, in order to achieve the target accuracy, we need to perform few iterations of SoTT (typically we computed not more than 5-6 SoTT terms on the test cases considered), contrary to ALS and CP-TT, which need a significantly larger amount of iterations. Compared to TT-SVD, the computational cost per iteration of SoTT is about d^2 times larger than the one of the TT-SVD. This computational burden could be compensated by the fact that, choosing a bad order of variables for TT-SVD could result in extra costs in terms of storage and further computational tasks.

[2]The function can be found in `http://www.sfu.ca/~ssurjano/otlcircuit.html`.

2.6 Conclusions and perspectives

In this chapter, we proposed a method to compress a given tensor as a sum of Tensor Trains (SoTT). Neither the order of the variables nor the ranks are fixed a priori. Instead, they are the result of an optimization step. A particular instance of this method, consisting in fixing the ranks equal to one in all the steps of the algorithm, produces a CP approximation of a given tensor. A proof of convergence is proposed in the general case of the SoTT algorithm, which can be extended to the case of the CP-TT algorithm. Several numerical experiments are proposed to illustrate the properties of the methods. First, we compared the CP-TT to other rank-one update methods (ALS, ASVD, TTr1). Although a single iteration of CP-TT is more expensive in terms of number of operation, its stability makes it a promising candidate to compress high-dimensional tensors in CP format. We proposed some tests in which we compressed the numerical solution of a parametric partial differential equation of reaction-diffusion type as well as other functions coming from different applications. In particular, we compared SoTT with the TT-SVD obtained by testing all the possible permutations of the indices. Although SoTT is suboptimal with respect to the best TT-SVD, it is independent of the order of the variables and its performances are comparable to the average TT-SVD. In this test, the SoTT method outperforms CP-TT. Both methods showed preliminary yet encouraging results in view of applications in scientific computing and compression of high order tensors. The method presented shows some shortcomings, to be addressed in further investigations: while a greedy method is appealing in view of computational tasks in which fixing the rank a priori could be cumbersome, it might be featured by a saturation effect, slowing down its convergence.

Chapter 3

Local tensor methods

Contents

3.1 Introduction 55
3.1.1 Organization of the chapter 59
3.2 Notation and preliminaries 59
3.2.1 Partitioning of tensors 60
3.2.2 The High Order Singular Value (HOSVD) decomposition 61
3.3 Local tensor spaces 61
3.3.1 Construction of local subdomains 61
3.3.2 Computation of the leaves 62
3.3.3 Merging local subdomains 65
3.4 Computing local HOSVD method 67
3.5 Cost and complexity of the algorithm 71
3.6 Summary 73
3.7 Numerical results 74
3.7.1 Compression of a Gaussian function 74
3.7.2 Compression of the solutions of the Fitz-Hugh-Nagumo equation 75
3.8 Conclusions and perspectives 80

3.1 Introduction

In this chapter of the thesis, we propose a "local" HOSVD method to approximate tensors. Let $d \in \mathbb{N}^*$, $p_1, \cdots, p_d \in \mathbb{N}^*$ and $\Omega_1 \subset \mathbb{R}^{p_1}, \ldots, \Omega_d \subset \mathbb{R}^{p_d}$ be open finite sets. Let $\Omega := \Omega_1 \times \ldots \times \Omega_d$ and $F \in L^2(\Omega)$.

The tensor to be approximated is a multivariate function $F(x_1, \ldots, x_d)$. This object is defined in all the domain, namely $\Omega := \Omega_1 \times \ldots \times \Omega_d$. Let us introduce the domain partition.

Let $N \in \mathbb{N}^*$ be the number of non-overlapping local domains $\Omega^{(i)}$. We have:

$$\Omega = \bigcup_{i=1}^{N} \Omega^{(i)}.$$

To each subdomain $\Omega^{(i)}$, we can associate the characteristic functions $\mathbf{I}^{(i)}(x_1, \ldots, x_d)$, which will be useful in what follows. These functions are one in the local subdomain and zero elsewhere. They read:

$$\mathbf{I}^{(i)}(x) = \begin{cases} 1 & \text{if} \quad x \in \Omega^{(i)} \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

When multiplying a function by the characteristic function of a subdomain we can restrict its values to the local domain that we are considering.

Henceforward, a tensor defined in the global domain can be expressed as the sum of its values on N local domains as follows:

$$F(x_1,\dots,x_d)=\sum_{i=1}^{N}F_i(x_1,\dots,x_d)=\sum_{i=1}^{N}F(x_1,\dots,x_d)\mathbf{I}^{(i)}(x_1,\dots,x_d) \tag{3.2}$$

where we have denoted $F_i(x_1,\dots,x_d)$ the function $F(x_1,\dots,x_d)$ whose values are restricted to the specified subdomain $\Omega^{(i)}$.

Let $\tilde{F}(x_1,\dots,x_d)$ denote the approximation of $F(x_1,\dots,x_d)$. The goal is to provide a certified approximation such that

$$\|F-\tilde{F}\|_{L^2(\Omega)}\leq\epsilon$$

where $\epsilon>0$ is the prescribed accuracy. The approximation can be expressed by using the characteristic functions:

$$\tilde{F}(x_1,\dots,x_d)=\sum_{i=1}^{N}\tilde{F}(x_1,\dots,x_d)\mathbf{I}^{(i)}(x_1,\dots,x_d)=\sum_{i=1}^{N}\tilde{F}_i(x_1,\dots,x_d) \tag{3.3}$$

The local HOSVD method to approximate F consists in approximating F_i in local domains and applying the HOSVD method on each one of the partitions. Then, the approximation reads, for $1\leq i\leq N$:

$$F_i\approx\tilde{F}_i=\sum_{i_1=1}^{R_1}\dots\sum_{i_d=1}^{R_d}G_{i_1\dots i_d}b_{i_1}^{(1)}\times\dots\times b_{i_d}^{(d)}$$

in which $R_1,\dots,R_d$ are the $n-$ranks, b are the modes and G is the tensor core. Their computation will be detailed later on.

The goal of this chapter is to investigate whether the local HOSVD will provide a better compression rate with respect to the HOSVD decomposition.

A crucial point is to determine the subdomain partition. In the proposed method, we don't impose the partition *a priori*, the aim is to come up with a method that computes automatically the partition.

Let us illustrate the partitioning on a 2-dimensional example in Figure 3.1. As $\Omega=\Omega_1\times\Omega_2$ the local partition reads $\Omega^{(i)}=\Omega_1^{(i)}\times\Omega_2^{(i)}$, where $1\leq i\leq N$. In this case, let us say that $N=2$, and let the subdomains originated by dividing the domain of x_1 be $\Omega_1^{(1)}$ and $\Omega_1^{(2)}$, and the ones obtained by dividing x_2, $\Omega_2^{(1)}$ and $\Omega_2^{(2)}$. For $x_1\in(0,1)$ and $x_2\in(0,1)$.

As we can see in the schematic Figure 3.1, the partition of the left is done in the axis x_1 and the partition of the right is done in the axis x_2. One would like to guess which one of the partitions provides a better local approximation of the tensor.

Let us call $N_1\in\mathbb{N}^*$ the number of the partitions in x_1 and $N_2\in\mathbb{N}^*$ the partitions in x_2. The total number of subdomains is $N=N_1\times N_2$. In this case, $N=4$. When partitioning at the same time on x_1 and x_2 we have a set of N local domains. This corresponds to the partition shown in Figure 3.2.

At this point the question of how to make the choice of the local domains arises. The main idea we investigate in this chapter is how to perform the subdivision per direction by using a clustering method. We would like to exploit the separability of the domain $\Omega=\Omega_1\times\dots\times\Omega_d$ in such a way that the partitions in the different directions provides a partition of the whole domain. The way of computing the local domains with a clustering algorithm is by applying it to the unfoldings of the different directions of the tensor. In this work we are going to restrict ourselves to the situation in which they are not overlapping. See [172, 173] for details of clustering methods.

Being N_k the number of partitions of the domain in the $k-$th direction, with $k=1,\dots,d$. Let $\mathbf{J}_k^{(i)}(x_k)$ be the characteristic functions in the direction k, $\mathbf{J}_k^{(i_k)}$.For $i_k=1,\dots,N_k$ and $i=(i_1,\dots,i_d)$, they read:

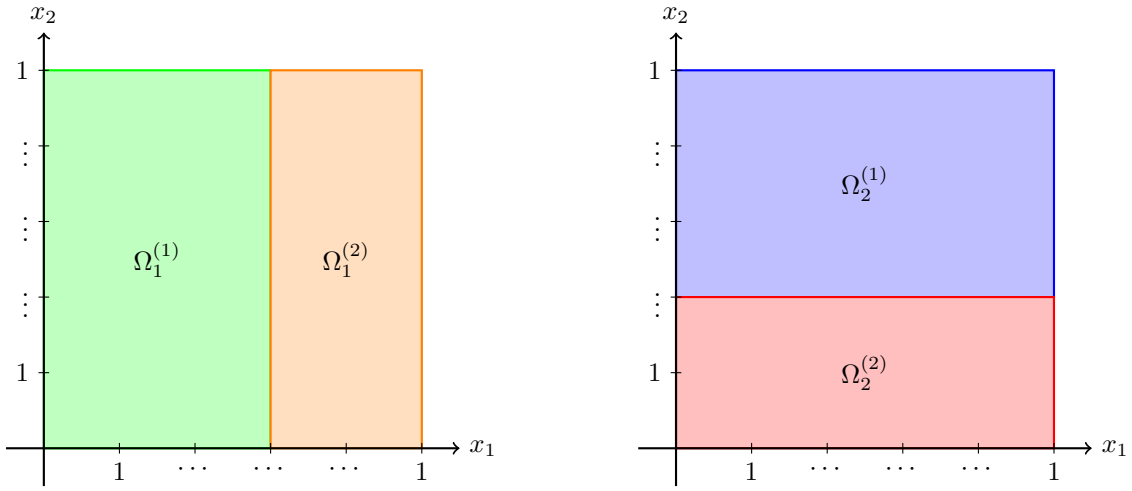


Figure 3.1: 2-dimensional schematic example of a possible partition of the domain with respect to its 2 directions.

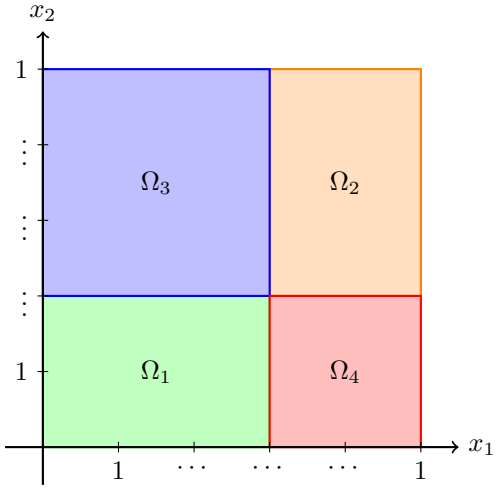


Figure 3.2: 2-dimensional schematic example of the combination of the partitions of Figure 3.1.

$$\mathbf{J}_k^{(i_k)}(x_k) = \begin{cases} 1 & \text{if} \quad x \in \Omega_k^{(i_k)} \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

Where the domain Ω_k is partitioned as:

$$\Omega_k = \bigcup_{i_k=1}^{N_k} \Omega_k^{(i_k)}.$$

The method performs independently direction per direction. Let us recall that the characteristic functions that restrict the tensor of Equation 3.2 to the local domain read

$$\mathbf{I}^{(i)}(x_1, \ldots, x_d) = \prod_{k=1}^{d} \mathbf{J}_k^{(i_k)}(x_k) \tag{3.5}$$

The domain Ω is separable and then, the subdomain $\Omega^{(i)}$ can be also separated $\Omega^{(i)} = \Omega_1^{(i_1)} \times \ldots \times \Omega_d^{(i_d)}$. Hence, the problem of partitioning the domain Ω can be reduced to the problem of partitioning Ω_k. The

partition of Ω_k into local subdomains is done by a clustering method.

Clustering problem is not a trivial task, especially in the case of high-dimensional data that is exactly the case that we face in most applications, and it is where conventional methods usually fail [262, 263]. The number of data required to define correctly the system increases exponentially with the dimension of the system. This phenomenon is referred to as the already introduced curse of dimensionality.

Clustering is the process of grouping similar objects into different groups, or more precisely, the partitioning of a data set into subsets. This can be achieved by grouping entities in terms of a similarity (or dissimilarity) measure. Here, the critical issue is to understand what is meant by "similar". In this work, we consider similarity as the inverse of a distance metric between two entities. The shorter the distance, the more similar the entities, and vice versa. Clustering results will be crucially dependent on the similarity notion chosen. In this work, we make the choice of using a L^2 distance, and when discretizing a l^2 distance.

The clustering method proposed in this chapter is agglomerative. As mentioned, it works independently for each one of the directions of the tensor. Let us recall that the *n-mode vectors* (or just modes), in this work also called fibers, of a certain unfolding $F^{(k)}$ are its column vectors, when $k \in \{1, \dots, d\}$. In the new partitioning of Ω_k, we will apply the clustering to the fibers of the unfolding transposed. Let us now introduce some notation which will be used in all the sequel. Let x_k and $y_k = (x_1, \dots, x_j, \dots, x_d)_{j \neq k}$. The set of fibers of $F^{(k)T}$ reads as: $\{f_p^k(y_k)\}_{1 \leq p \leq \mathcal{N}_k}$.

If we face a $d-$dimensional tensor, $F \in \mathbb{R}^{\mathcal{N}_1 \times \dots \times \mathcal{N}_d}$ and we compute the unfolding transposed corresponding to the first one of the directions $F^{(1)} \in \mathbb{R}^{(\mathcal{N}_2 \times \dots \times \mathcal{N}_d) \times \mathcal{N}_1}$, the system presents a set of $\mathcal{N}_1$ mode vectors: $f_1^1(y_1), \dots, f_{n_1}^1(y_1)$ of size $\mathcal{N}_2 \times \dots \times \mathcal{N}_d$. Then, for the k-th one of the unfoldings of the tensor, $k \in \{1, \dots, d\}$, the $\mathcal{N}_k$ fiber modes read as: $f_1^k(y_k), \dots, f_{\mathcal{N}_k}^k(y_k)$, of size $\mathcal{N}_{yk} = \prod_{k \neq j}^d \mathcal{N}_j$.

This clustering method will allow us to subdivide the fibers of a tensor in different groups (clusters). The intuition suggest that in a physical context, we are grouping the fibers that belong to the same "regime".

Remark: In this case, we are going to work from the unfoldings of the tensor transposed, $F^{(k)T}$. If we want to work with the unfoldings directly, let us remark that the partitions that we are computing are done not in the direction k but in the rest of the directions $1 \leq j \neq k \leq d$. The subdomains in which we divide the multivariate functions reads as $\left(\Omega_{j \neq k}^{(i)}\right)_{1 \leq j \leq d}$ instead of $\left(\Omega_k^{(i)}\right)_{1 \leq k \leq d}$. It is indeed, an equivalent way of proceeding. In this work, we have chosen to work with the transpose of the unfoldings.

The output of the clustering method is a hierarchical tree of the possible partitions of the domains $\left(\Omega_k^{(i)}\right)_{1 \leq k \leq d}$. It starts pairing the similar leaves and it carries on pairing the nodes till everything is fused. Each one of the levels l_k of the tree denotes a possible partition of the domain on the direction k. In the root of the tree, the point in which every cluster is fused, the POD decomposition of the unfolding transposed in the direction k is recovered. Each of the nodes defines a partition of the domain. The closest the nodes are to the root of the tree, the biggest the clusters are.

Let $\#l_k$ be the number of levels of the tree T_{I_k} on the direction k. Each level l_k denotes a possible partition of the domain Ω_k on that direction. The total number of possible combinations of partitions in all the directions reads as $\prod_{k=1}^d \#l_k$. On each level l_k, where $l_k = 0, \dots, \#l_k - 1$, of the tree, the number of subdomains in which the domain Ω_k is divided is denoted by N_{k,l_k}.

In this work, in order to find the best partition we perform an extensive search between the possible partition combinations and we perform the HOSVD on each of them. In order to determine which partition of the domain is the best we need to establish a criterion in order to select one partition over the others. This is the memory needed to store the approximation. For every possible combination of the partitions of the domain obtained, the method computes the basis that generates the subspace defined by the partition performing HOSVD on the truncated space. The one that requires less amount of memory to be stored is said to be the *best* of the partitions of the domain.

Clustering methods that focus on finding a clustering strategy that is appropriate for model order reduction purposes have been already used and can be found in [176, 177], where a dictionary of clusters is assembled and when a new instance of the data needs to be classified, the method finds the cluster to which it belongs. In [156, 157] a multiple local bases method is introduced. The main difference with the method proposed in this work is that in [156], the local ROB are computed by the method of snapshots in the intersection between overlapping clusters.

3.1.1 Organization of the chapter

After the presented overview of the method proposed in this chapter, it will be studied in detail in the following sections. The organization followed in the chapter reads as follows:

Find in Section 3.2 some notation and preliminary concepts that are used in the present chapter. In Section 3.3 the process of construction of the local domains is detailed. In Section 3.4, the computation of the approximation of the tensor in the local subspaces obtained is computed. It is in this section where the local HOSVD method is presented. Section 3.5 breaks down the cost and complexity of the proposed algorithm. A summary recalling the important steps of the method can be found in Section 3.6. To finish, the numerical results of the method applied to practical cases appear in Section 3.7.

3.2 Notation and preliminaries

The notation here recalls the one used in Section 2.2. Let us look back on some of the concepts introduced before and particularize them to its concrete use on this section. New notation and concepts are introduced as well.

Let us begin by recalling some definitions. Let us start by the definition of a tensor as a multivariate function and its associated preliminary concepts. Let p be the dimension and let D be a subset of $\mathbb{R}^p$ with a product structure $D := D_1 \times \ldots \times D_d$. Let us consider $d \in \mathbb{N}^*$. Then, for all $1 \leq i \leq d$, D_i is an open bounded subset of $\mathbb{R}^{p_i}$ for some $p_i \in \mathbb{N}^*$. We call a *tensor* any real-valued multivariate function $F \in L^2(D_1 \times \cdots \times D_d)$.
For each $1 \leq i \leq d$, a certain $\mathcal{H}_i$ a Hilbert space of univariate functions defined of D_i, equipped with the corresponding inner product $(\cdot,\cdot)_{\mathcal{H}_i}$ and the associated norm $\|\cdot\|_{\mathcal{H}_i}$.

The space $L^2(D)$ of square-integrable, real-valued functions defined on D equipped with a scalar product and the corresponding canonical norm $\|\cdot\|_{L^2(D)}$ that is defined for $F, G \in L^2(D)$ by

$$\langle F, G\rangle_D := \int_D F(x)G(x)dx \quad \text{and} \quad \|F\|_{L^2(D)} := \left(\int_D F^2(x)dx\right)^{1/2}.$$

From now on, in order to simplify the notation of the manuscript (every time that there's no ambiguity) $L^2(D)$ will be denoted as D, equipped with the norm $\|\cdot\|_D$ and its associated inner product $(\cdot,\cdot)_D$. In the cases in which we work with discrete objects in $l^2(D)$, the norm used will be the Euclidean norm and will be denoted by $\|\cdot\|_{l^2}$.

We recall here some well-known definitions and introduce some notation about tensor fibers and tensor unfoldings:

The so called *fibers* are the higher-order analogue of matrix rows and columns, defined by fixing every index of a tensor but one. In a 3-dimensional tensor $F \in \mathbb{R}^{\mathcal{N}_1 \times \mathcal{N}_2 \times \mathcal{N}_3}$, for a certain value of two of its indices $i_2 \in \{1, \ldots, \mathcal{N}_2\}$ and $i_3 \in \{1, \ldots, \mathcal{N}_3\}$, a fiber on the i_1-th direction is given by $f^{i_1} = F_{:i_2 i_3}$, with $i_1 \in \{1, \ldots, \mathcal{N}_1\}$. When fibers are extracted from tensors, they are assumed to be column vectors.

Unfoldings. Assume a $N-$th order tensor $X \in \mathbb{R}^{\mathcal{N}_1 \times \ldots \times \mathcal{N}_N}$. The matrix unfolding $X^{(n)} \in \mathbb{R}^{\mathcal{N}_n \times (\mathcal{N}_{n+1} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1})}$ contains the element $x_{i_1 \ldots i_p}$ at the row position i_n and column number equal to:

$$(i_{n+1} - 1)\mathcal{N}_{n+2} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1} + i_{n+2} - 1)\mathcal{N}_{n+3} \ldots \mathcal{N}_N \mathcal{N}_1 \ldots \mathcal{N}_{n-1} + \ldots$$

$$\ldots + (i_N - 1)\mathcal{N}_1 \ldots \mathcal{N}_{n-1} + (i_1 - 1)\mathcal{N}_2 \ldots \mathcal{N}_{n-1} + \ldots + i_{n-1}$$

The unfolding transpose of a $N-$th order tensor $X \in \mathbb{R}^{\mathcal{N}_1 \dots \mathcal{N}_d}$ tensor is the matrix $X^{(n)T} \in \mathbb{R}^{(\mathcal{N}_{n+1}\dots\mathcal{N}_N\mathcal{N}_1\dots\mathcal{N}_{n-1})\times\mathcal{N}_n}$ that contains the element $x_{i_1\dots i_p}$ at the column position i_n and row number equal to:

$$(i_{n+1}-1)\mathcal{N}_{n+2}\dots\mathcal{N}_N\mathcal{N}_1\dots\mathcal{N}_{n-1}+i_{n+2}-1)\mathcal{N}_{n+3}\dots\mathcal{N}_N\mathcal{N}_1\dots\mathcal{N}_{n-1}+\dots$$
$$\dots+(i_N-1)\mathcal{N}_1\dots\mathcal{N}_{n-1}+(i_1-1)\mathcal{N}_2\dots\mathcal{N}_{n-1}+\dots+i_{n-1}$$

As a remark, notice that the fibers are the column vectors of the unfolding matrices.

Given two fibers, $f_1^{i_1} \in \mathbb{R}^{\mathcal{N}_1}$ and $f_2^{(i_1)} \in \mathbb{R}^{\mathcal{N}_1}$, for $i_1 \in \{1,\dots,\mathcal{N}_1\}$, let us define the concatenation of them as a matrix $F \in \mathbb{R}^{\mathcal{N}_1\times 2}$:

$$F = (f_1^{i_1} f_2^{i_1}) \tag{3.6}$$

such that its first column $F_{:,1}$, is the fiber $f_1^{i_1}$ and its second column $F_{:,2}$, is $f_2^{i_1}$.

Following this definition, we extend it to a concatenation of matrices. Let $F \in \mathbb{R}^{\mathcal{N}_1\times\mathcal{N}_2}$ and $X \in \mathbb{R}^{\mathcal{N}_1\times\mathcal{N}_3}$. The concatenation of them results other matrix $A \in \mathbb{R}^{\mathcal{N}_1\times(\mathcal{N}_2+\mathcal{N}_3)}$ reads:

$$A = (FX)$$

such that: for $1 \le i \le \mathcal{N}_2$, $A_{:,i} = F_{:,i}$ and for $\mathcal{N}_2 \le j \le \mathcal{N}_4$, $A_{:,j} = X_{:,j-\mathcal{N}_2}$.

3.2.1 Partitioning of tensors

In this chapter, the notation followed in order to explain the concepts related with local tensor decomposition is the one introduced in [264].

Let $d \in \mathbb{N}^*$ be the dimension of the system and $p \in \mathbb{N}^*$. Let $D_1,\dots,D_p$ be open subsets of $\mathbb{R}^{d_i}$ with $d_i \in \mathbb{N}^*$ and $1 \le i \le p$. We denote $D := D_1 \times \dots \times D_p$.

A domain partition $\{D^{(k)}\}_{1\le k\le K}$ of D satisfies:

- For all $1 \le k \le K$, there exists $D_1^{(k)} \subset D_1,\dots,D_d^{(k)} \subset D_d$ open subsets such that $D^{(k)} := D_1^{(k)} \times \dots \times D_d^{(k)}$.
- For all $1 \le k \ne l \le K$, $D^{(k)} \cap D^{(l)} = \emptyset$.
- $D = \bigcup_{k=1}^K D^{(k)}$.

A domain partition of D for $1 \le j \le d$ can be: a set of subdomains $P_j = D_j^{(k_j)} \in D_j$, with $1 \le k_j \le K_j$, that satisfies the properties enhanced before, such that the partition reads

$$P = \left(\times_{j=1}^d D_j^{(k_j)}\right)_{1\le k_1\le K_1,\dots,1\le k_d\le K_d}$$

In [264] we can also see that under appropriate assumptions, it can be proved that there exists an admissible domain partition $D^{(k)} \subset D$ such that the restricted tensor $F(x_1^k,\dots,x_d^k)$, with x_j^k the restricted values of the variable x_j, can be represented in some tensor formats with low ranks.

Partition trees

Let T_I be a tree with nodes $\mathcal{V}(T_I)$ and edges $\mathcal{E}(T_I)$. For all $J \in \mathcal{V}(T_I)$ we denote the set of sons of the node J as:

$$\mathcal{S}_J(T_I) := \{J' \in \mathcal{V}(T_I), (J,J') \in \mathcal{E}(T_I)\}$$

and equivalently we can define the set of sons of J of the $k-$th generation, $\mathcal{S}_J^k(T_I)$ and the nodes in the $k-$th generation as $\mathcal{V}^k(T_I)$. The set of leaves reads,

$$\mathcal{L}(T_I) := \{J \in \mathcal{V}(T_I), \mathcal{S}_J(T_I) = \emptyset\}$$

The set of parents of leaves of a tree T_I is defined as $\mathcal{L}^p(T_I) \subset \mathcal{V}(T_I)$ which have at least one son that is a leaf of T_I.

With all this concepts set we are in position to recall the definition of partition tree. A tree T_I is called a partition tree if:

- I is the root of T_I.
- For all $J \in \mathcal{V}(T_I)\backslash\mathcal{L}(T_I)$, $\mathcal{S}_J(T_I)$ is an admissible partition and $|\mathcal{S}_J(T_I)| \geq 2$.
- For all $J \in \mathcal{V}(T_I), J \neq \emptyset$.

In this work we are interested in the particular case in which T_I is a dyadic partition tree. In this case, for all $J \in \mathcal{V}(T_I)\backslash\mathcal{L}(T_I)$, $\mathcal{S}_J(T_I)$ is an admissible partition and $|\mathcal{S}_J(T_I)| = 2$.

3.2.2 The High Order Singular Value (HOSVD) decomposition

Let us recall from Equation 1.8 the already introduced HOSVD format altogether with the concept of $n-$rank of a tensor. It is the generalization of the SVD decomposition presented in Equation 2.1.

Let us refer to the column, row, ... vectors of a $N-$th order tensor as its *n-mode vectors* defined as the $\mathcal{N}_n-$vectors obtained by varying the index i_n and keeping the rest fixed. The *n-mode vectors* (or just modes) of a certain tensor X are the column vectors of the unfolding $X^{(n)}$.
We define the *n-rank* of a tensor X, denoted by $R_n = \text{rank}_n(X)$ as the dimension of the vector space spanned by the $n-$mode vectors.
The HOSVD of a certain tensor F reads:

$$F = \sum_{i_1=1}^{R_1} \dots \sum_{i_d=1}^{R_d} G_{i_1 \dots i_d} u_{i_1}^{(1)} \times \dots \times u_{i_d}^{(d)} \tag{3.7}$$

in which $R_1, \dots, R_d$ are the $n-$ranks defined above and they are smaller than $\mathcal{N}_1, \dots, \mathcal{N}_d$, respectively. For compression, it is also commonly used the Truncated HOSVD, in which the summation is truncated when the approximation reaches a certain tolerance. The representation of a tensor in this format requires to store $\mathcal{N}^d + \mathcal{N}dR$ elements, supposing that all the ranks $R_1 = \dots = R_d = R$ are equal.

3.3 Local tensor spaces

The objective is to propose a partition of the domain in which the function is well approximated by HOSVD method. This is done automatically by the method and it doesn't require the choice of the ranks of the approximation a priori.

From now on, we fix some $d \in \mathbb{N}^*$ and for all $1 \leq j \leq d$, let Ω_j be an open subset of $\mathbb{R}^{p_j}$ for some $p_j \in \mathbb{N}^*$. We define $\Omega := \Omega_1 \times \dots \times \Omega_d$.

3.3.1 Construction of local subdomains

Recalling Equation 3.2, a multivariate function $F(x_1, \dots, x_d)$ can be expressed as a sum on N local domains:

$$F(x_1, \dots, x_d) = \sum_{i=1}^{N} F_i(x_1, \dots, x_d) = \sum_{i=1}^{N} F(x_1, \dots, x_d)\mathbf{I}^{(i)}(x_1, \dots, x_d)$$

where, for N number of local subdomains and $i = 1, \dots, N$, $F_i(x_1, \dots, x_d)$ is the function whose values are restricted to the specified subdomain defined by the characteristic functions defined in Equation 3.1.

In order to exploit that Ω is a separable domain, the clustering method is applied independently to each one of the unfoldings of the tensor $F^{(1)}, \dots, F^{(d)}$. On each direction, a partition tree is computed in an agglomerative way. It will provide the possible partitions of the domain that will be considered.
Let N_{k,l_k} be the number of subdomains. The approximation of the unfolding transposed reads:

$$F^{(k)T}(x_k, y_k) = \sum_{i_k=1}^{N_{k,l_k}} F_{i_k}^{(k)T}(x_k, y_k)\mathbf{I}_k^{(i_k)}(x_k)$$

where the characteristic functions per direction are defined in Equation 3.4.

As introduced before, in this part of the work for u and v given fibers of the unfolding transposed, we set the distance between them as $\text{dist}(u,v) = \|u-v\|_{L^2(\bigtimes_{j\neq k}\Omega_j)}$, being $\bigtimes_{j\neq k}\Omega_j$ with $j=1,\dots,d, j\neq k$ the product of the spaces different from Ω_k, when we are dealing with continuous functions and $\text{dist}(u,v) = \|u-v\|_{l^2}$, where l^2 is on $\mathbb{R}^{\mathcal{N}_{yk}}$, when we are working with discrete vectors. Let us recall that we have denoted $\mathcal{N}_{yk} = \prod_{j\neq k}^{d}\mathcal{N}_j$.

The method introduced in this work proceeds by computing an agglomerative tree. Let us start introducing how to compute the leaves. They are clusters of fibers represented by 2-dimensional bases. This grouping process relies on the notion of distance between the fibers on the different unfoldings of the tensor.

3.3.2 Computation of the leaves

Let us now explain how the method proceeds for one of the directions of the system. The extension to others is identical. As we have introduced before, for a certain direction k of the tensor we have a set of fibers depending of the rest of the variables, $f_1^k(y_k),\dots,f_{\mathcal{N}_k}^k(y_k)$ of size $\mathcal{N}_{yk}$. Let us define a set of indices per direction $\mathcal{T}_k = \{1,\dots,\mathcal{N}_k\}$ that denotes the index of the fibers. At the beginning of the method, it denotes the whole set and at the end an empty set $\mathcal{T}_k = \emptyset$.

Each one of the fibers is then a vector of $\mathcal{N}_{yk}$ components. Choosing randomly one of the fibers, let us say without loss of generality the first of them $f_1^k(y_k)$, we take its index out of the set, $\mathcal{T}_k$ is updated by $\mathcal{T}_k \setminus \{1\}$, and we compute its nearest neighbor.

$$q_* = \arg\min_{q\in\mathcal{T}_k} \text{dist}(f_1^k, f_q^k)$$

where

$$\text{dist}(f_1^k(y_k), f_{q_*}^k(y_k)) = \|f_1^k - f_{q_*}^k\|_{l^2}$$

This pairing of two fibers is the first step in the grouping process. A SVD of the matrix composed by the two fibers as its columns will allow us to compute the basis.

Let us suppose that q_* is such that $f_{q_*}^k(y_k)$ is linearly independent from $f_1^k(y_k)$. Then, let us define the matrix $F^{k(1,q_*)} \in \mathbb{R}^{\mathcal{N}_{yk}\times 2}$ as introduced in 3.6:

$$F^{k(1,q_*)} = (f_1^k(y_k) f_{q_*}^k(y_k)).$$

Recalling the previously defined SVD decomposition, it holds that there exists an orthonormal basis $(U_i)_{i\in\mathbb{N}^*} \in \mathbb{R}^{\mathcal{N}_{yk}\times\mathcal{N}_{yk}}$, an orthonormal basis $(V_i)_{i\in\mathbb{N}^*} \in \mathbb{R}^{2\times 2}$, and a decreasing sequence $(\sigma_i)_{i\in\mathbb{N}^*} \in \mathbb{R}^{\min(\mathcal{N}_{yk},2)\times\min(\mathcal{N}_{yk},2)}$ of non-negative real numbers, such that

$$F^{k(1,q_*)} = U^{k(1,q_*)}\Sigma^{k(1,q_*)}V^{k(1,q_*)T} = \sum_{i\in\mathbb{N}^*}\sigma_i^{k(1,q_*)}U_i^{k(1,q_*)}\otimes V_i^{k(1,q_*)}. \tag{3.8}$$

If the fibers f_1^k and $f_{q_*}^k$ are linearly dependent, one of the singular values is going to be zero and a one-dimensional basis will be retrieved. If that occurs, we look for the next neighbor to perform this step.

The orthonormal basis of the joint subspace is given by the matrix $U^{k(1,q_*)}$. From now on, let us call it $B_k^{(1)}$, where the superindex of the basis is the first selected fiber. Let us define now a certain set of indices $\mathcal{I}_k^{(1)} = \{1, q_*\}$ that is composed by the indices of the fibers in a certain cluster. The superindex corresponds to the one used for the basis. Every time that one of the fibers is added to a cluster, we take its index out of $\mathcal{T}_k$ in order to avoid repetition, and $\mathcal{T}_k$ is updated by $\mathcal{T}_k \setminus \mathcal{I}_k^{(j)}$, being $\mathcal{I}_k^{(j)}$ the set of indices of the fibers in the $j-$th cluster in the $k-$th direction that is being used to construct a cluster.

If some neighbors with zero distance had been found when constructing the 2-dimensional basis process, its indices need to be included also in the set because it is obvious that they are well represented by the basis. An example of this situations is the case in which the solution of a PDE is periodic and some of the fibers are consecutively repeated.
If they are linearly dependent, the index of the neighbor will be also added to the set.

In the clusters, not only the fibers that we used to compute the basis or the ones that are linearly dependent (and that belong to the subspace) can be included, but also the ones that are close enough to the subspace. In this case, a fiber is considered close enough to the subspace if it can be represented on it with an error smaller than the one needed to guarantee the prescribed accuracy.

The error for a certain fiber $f_p^k(y_k)$ that is approximated in the space spanned by a basis $B_k^{(h)}$ is defined as the difference between the projection of the fiber and the fiber itself:

$$\|f_p^k(y_k) - \int_{\times_{j\neq k}\Omega_j} B_k^{(h)T} f_p^k(y_k) dy_k\|_{L^2} = \epsilon_k^{(h)} \tag{3.9}$$

Let $\mathcal{N}_k$ be the number of fibers in the k-th direction. In order to consider a certain fiber of the direction k, $f_p^k(y_k)$, where $p \notin \mathcal{I}_k^{(1)}$, well represented by the basis $B_k^{(1)}$, the error defined in Equation 3.9 needs to be smaller than the squared tolerance ε^2 (prescribed) over the number of fibers $(\delta_k^{(1)})^2 = \varepsilon^2/\mathcal{N}_k$.

Let us define the error made in the cluster q of the direction k, namely $\epsilon_{k,l}^{(q)}$, as the sum of the errors made each time that a fiber is added to it: $\sum_{h\in\#\mathcal{I}_k^{(q)}} \epsilon_k^{(h)} = \epsilon_{k,l}^{(q)}$. In order to obtain a guaranteed approximation, we need that $\epsilon_k^{(h)} < \delta_k^{(1)}$. At the beginning, this error is zero because the cluster consists only in the two vectors themselves, the fibers with indices $\{1, q_*\}$ in the previous example. When we add fibers to the cluster, they are represented by the basis with an error $\epsilon_k^{(1)}$. This error is such that it guarantees that the global error made in all the clusters respects the prescribed tolerance.

When a certain fiber is included in a cluster, we modify its basis in order to reduce the approximation error in the cluster. When just the two neighbor fibers were considered, the error was zero but when we include other fiber in the cluster it may be represented for the basis of the cluster with a certain error given by Equation 3.9. With the objective of minimizing the error in the cluster, an update of the basis is computed. In [265, 266] the author develops an identity for additive modifications of a singular value decomposition (SVD) to reflect updates, downdates, shifts, and edits of the data matrix. This situation has our particular interest because it allows us to compute the new SVD decomposition by adapting the old one instead of recomputing the whole SVD. In our concrete case, we are interested on the SVD of the matrix after a rank-1 modification. Let a real matrix $F \in \mathbb{R}^{\mathcal{N}_i \times p}$, with $\mathcal{N}_i$, degrees of freedom in the direction i and $p \in \mathbb{N}^*$, have rank R and SVD $F = U\Sigma V^T$ with $\Sigma \in \mathbb{R}^{R\times R}$ the diagonal matrix of singular values. Let $A \in \mathbb{R}^{\mathcal{N}_i \times c}$, $B \in \mathbb{R}^{p\times c}$ be arbitrary matrices of rank c. We are interested in the SVD of the real matrix $\hat{F} \in \mathbb{R}^{\mathcal{N}_i \times (p+1)}$ resulting from the sum:

$$\hat{F} = F + AB^T = \begin{pmatrix} U & A \end{pmatrix} \begin{pmatrix} \Sigma & 0 \\ 0 & \mathbb{I} \end{pmatrix} \begin{pmatrix} V & B \end{pmatrix}^T \tag{3.10}$$

Where $\mathbb{I}$ is the identity matrix. In our case the application of this method for the joint function $F^{k(1,q_*,p)}$ as a rank-1 modification of $F^{k(1,q_*)}$ reads:

$$F^{k(1,q,p)} = F^{k(1,q)} + f_p^k b^T = \begin{pmatrix} B_k^{(1)} & f_p^k \end{pmatrix} \begin{pmatrix} \Sigma^{k(1,q_*)} & 0 \\ 0 & \mathbb{I} \end{pmatrix} \begin{pmatrix} V^{k(1,q_*)} & b \end{pmatrix}^T \tag{3.11}$$

where $b^T = [0, 0, \ldots, 1]$ is the rank-1 value of B and A on its particular rank-1 case is chosen as f_p^k. Find more details of the choice of the values for A and B and of the method itself in the cited work, [265]. In our case, the new basis of the cluster with the new fiber included obtained by the SVD of the expression 3.11 will update the old $B_k^{(1)}$. The error budget, that at the beginning of the method was equal to the tolerance squared, $b_k = \varepsilon^2$, will be also updated because making some error in the projection means spending part of the error budget. It becomes now $b_k = b_k - (\epsilon_k^{(1)})^2$. The quantity $(\delta_k^{(1)})^2$ used to determine if a fiber is well represented or not by a certain basis, is updated as: $(\delta_k^{(1)})^2 = \frac{b_k}{\#\mathcal{T}_k}$.

This process is repeated till there are no fibers left to be checked and $\mathcal{T}_k = \{0\}$. We have obtained a set of 2-dimensional basis that well represents all the fibers in a certain direction.

It could happen that at the end of the clustering process, a fiber is not well represented by any of the existing bases. The fiber is then the generator of its own 1-dimensional space.

Let us introduce $N_{k,0}$ as the number of clusters that we obtain as the leaves for the tree and let us remark that $1 \leq N_{k,0} \leq \lceil \frac{\mathcal{N}_k}{2} \rceil$. The case in which, $N_{k,0} = \lceil \frac{\mathcal{N}_k}{2} \rceil$ is the one in which all the fibers of the direction k have been paired two by two. The other endpoint, $N_{k,\#l_k-1} = 1$ represents the case in which the first basis originated by two fibers represents well the rest of the fibers on the set. Let us name N_k the set of all the partitions of the domain Ω_k, $N_k = \sum_{j=0}^{\#l_k-1} N_{k,j}$.

A necessary condition to guarantee that we are obtaining a certified approximation is that sum of all the errors squared is smaller or equal that the prescribed tolerance: $\sum_{q=1}^{N_{k,0}} (\epsilon_{k,l}^{(q)})^2 \leq \varepsilon^2$. The error satisfies also that $\sum_{q=1}^{N_{k,0}} (\epsilon_{k,l}^{(q)})^2 + b_k = \varepsilon^2$. Where b_k denotes the part of the error squared that hasn't been made in the clustering process, or what is the same, the difference between the prescribed tolerance and the errors made on each cluster.

The first part of the method, developed till here, can be found more detailed in Algorithm 9.

Algorithm 9 Computation of the leaves algorithm

Require: Unfolding of a tensor in the direction k, $F^{(k)T}(x_1, \ldots, x_d) \in \mathbb{R}^{\left(\prod_{j=1, j\neq k}^{d} \mathcal{N}_j\right) \times \mathcal{N}_k}$, for $1 \leq k \leq d$. Tolerance ε.
Output: Set of indices $\{\mathcal{I}_k^{(q)}\}_{1 \leq q \leq N_{k,0}}$. Set of bases $\{B_k^{(q)}\}_{1 \leq q \leq N_{k,0}}$. Set of local error budgets $\{\epsilon_{k,b}^{(q)}\}_{1 \leq q \leq N_{k,0}}$ and an error budget b_k.
Compute the set of $\mathcal{N}_k$ fibers $f_1^k(y_k), \ldots, f_{\mathcal{N}_k}^k(y_k)$
Initalize $n = 1$
Initialize $\mathcal{T}_k^{(1)} = \{1, \ldots, \mathcal{N}_k\}$
Initialize $i = \mathcal{T}_{k,1}^{(1)}$
Initialize the error budget: $b_k = \varepsilon^2$
while $\mathcal{T}_k^{(i)} \neq \{0\}$ **do**
  for $n > 1$ **do**
    Set $i = \mathcal{T}_{k,1}^{(i)}$
  end for
  for $q \in \mathcal{T}_k^{(i)}$ **do**
    Compute the distance between the fibers and select the closest one:

$$q_* = \arg \min_{q \in \mathcal{T}_k^{(i)}, q \neq i} \operatorname{dist}(f_i^k(y_k), f_q^k(y_k)), \quad \text{where} \quad \operatorname{dist}(f_i^k(y_k), f_q^k(y_k)) = \|f_{i,j}^k - f_{q,j}^k\|_{l^2}$$

  end for
  Set $\mathcal{I}_k^{(i)} = \{i, q_*\}$
  Update $\mathcal{T}_k^{(i)} = \mathcal{T}_k^{(i)} - \{q_*\}$
  Compute the joint matrix $F^{k(i,q_*)} = (f_i^k(y_k) f_{q_*}^k(y_k))$
  SVD of $F^{k(i,q_*)} = \sum_{j \in \mathbb{N}^*} \sigma_j^{k(i,q_*)} U_j^{k(i,q_*)} \otimes V_j^{k(i,q_*)}$
  Set $B_k^{(i)} = U^{k(i,q_*)}$
  Set $\delta^2 = b_k / \#\mathcal{T}_k^{(i)}$
  Initialize local error: $\epsilon_{k,l}^{(i)} = 0$
  for $p \in \mathcal{T}_k^{(i)}$ **do**
    if $|f_p^k(y_k) - \int_{\times_{j \neq k} \Omega_j} B_k^{(i)T} f_p^k(y_k) dy_k|^2 = (\epsilon_k^{(i)})^2 < \delta^2$ **then**
      The fiber is well represented by the basis
      Update $\mathcal{I}_k^{(i)} = \mathcal{I}_k^{(i)} + \{p\}$
      Update $\mathcal{T}_k^{(i)} = \mathcal{T}_k^{(i)} - \{p\}$
      Recompute the SVD of $F_i^{k(i,q_*,p)} = (F^{k(i,q_*)} f_p^k(y_k))$.
      Update the basis $B_k^{(i)} = U_i^{k(i,q,p)}$
      Update the error budget: $b_k = b_k - (\epsilon_k^{(i)})^2$
      Update the local error $(\epsilon_{k,l}^{(i)})^2 = (\epsilon_{k,l}^{(i)})^2 + (\epsilon_k^{(i)})^2$
    end if
  end for
  n = n+1
end while

This process is computed independently for each of the directions. For $1 \leq k \leq d$, the output of the method is:

1. The sets of indices $\{\mathcal{I}_k^{(q)}\}_{1 \leq q \leq N_{k,0}}$
2. The sets of local error budgets $\{\epsilon_{k,b}^{(q)}\}_{1 \leq q \leq N_{k,0}}$

3. The set of bases $\{B_k^{(q)}\}_{1\leq q\leq N_{k,0}}$
4. The global error budget squared b_k

where $N_{k,0}$ is the number of the leaves.

Let us remark that for the moment the basis obtained are 2-dimensional ones. We now consider that option and let us remember that we may still have some budget that we can spend in compressing more the tensor.

3.3.3 Merging local subdomains

In this section we are going to introduce a second phase of the proposed method: starting from the leaves obtained from Algorithm 9, a merging process between clusters is performed. The merging criteria will rely on the notion of similarity between clusters. This process provides a hierarchical bottom-to-top binary partition tree with an agglomerative structure, [165], per direction. This procedure is computed independently for each of the directions of the tensor.

Let us denote the set of clusters obtained from Algorithm 9 as $\{C_k^{(q)}\}_{1\leq q\leq N_{k,0}}$. In order to start the merging phase, the notion of distance between clusters needs to be set.

The computation of the distances between clusters have been widely studied and some methods are proposed in the literature, [174]. Between the alternatives we find in the literature, let us cite some works on distances between subspaces in [267, 268]. In this work, the notion of distance used is the so called Ward's distance, denoted by Δ, (see [225] for more details). Find more choices of distances and its analysis in [269].

Ward's method is used in order to find the nearest neighbor of a centroid c_p^k. This is performed by finding the smaller distance with respect to the others centroids weighted with a certain constant.
Let us name the partition tree that we are computing per direction k as T_{I_k}. This part of the algorithm begins by computing the centroids or baricenters of the clusters. These centroids of the leaves of the T_{I_k} denoted as $c_1^k,\dots,c_{N_{k,0}}^k$ are computed as the mean of the fibers well represented by the basis of the clustering, they are the barycenters of the $C_1^k,\dots,C_{N_k,0}^k$ clusters. For $q=1,\dots,N_{k,0}$:

$$c_q^k=\frac{\sum_{p\in\mathcal{I}_k^{(q)}} f_p^k(y_k)}{\#\mathcal{I}_k^{(q)}}$$

Let us introduce the set of indices $\mathcal{C}_k^{(q)}$ of clusters that are not merged (as the analogous set to $\mathcal{T}_k^{(q)}$ for the fibers). At the beginning of this merging phase, the set is complete $\mathcal{C}_k^{(q)}=\{1,\dots,N_{k,0}\}$ and while we are merging, we are removing the indices of the clusters that are being paired. For an intermediate iteration let us have $\mathcal{C}_k^{(q)}=\{1,\dots,N_k\}$. The process finishes when $\mathcal{C}_k=\{0\}$.

$$q_*,p_*=\arg\min_{q,p\in\mathcal{C}_k^{(q)}}\Delta(C_p^k,C_q^k)$$

Ward's method says that the distance between two clusters,C_p^k and C_q^k, is how much the sum of squares will increase when we merge them:

$$\Delta(C_p^k,C_q^k)=\sum_{i\in\mathcal{I}_k^{(p)}\cup\mathcal{I}_k^{(q)}}\|f_i^k-c_{C_p^k\cup C_q^k}^k\|^2-\sum_{i\in\mathcal{I}_k^{(p)}}\|f_i^k-c_p^k\|^2-\sum_{i\in\mathcal{I}_k^{(q)}}\|f_i^k-c_q^k\|^2=\frac{\#\mathcal{I}_k^{(p)}\#\mathcal{I}_k^{(q)}}{\#\mathcal{I}_k^{(p)}+\#\mathcal{I}_k^{(q)}}\|c_p^k-c_q^k\|^2$$

where $\#\mathcal{I}_k^{(p)},\#\mathcal{I}_k^{(q)}$ is the number of elements of each set of indices and then of each cluster. Let us remark that in this case, the coefficient that multiplies the distance between the centroids promotes to the fusion of small clusters, considering them closer than if we were computing the Euclidean distance. Or what is the same, given two pairs of clusters whose centers are equally far apart, Ward's method will prefer to merge the smaller ones.

Other difference between the computation of the leaves and the merging process apart from the notion of distance chosen, is that in this case instead of selecting the nearest neighbor between one fiber and the rest, we perform a pairwise computation of the distance. This means that we compute the Ward's distances between all the clusters: for all $1 \leq p, q \leq N_{k,0}$, the distances $\Delta(C_p^k, C_q^k)$ are computed, and among all of them we select the smallest. We select the pair that provides it as the neighbor clusters.

In the leaves level of the tree $l_k = 0$, in order to compute the pairwise distance operation, $(N_{k,0} - 1) \times (N_{k,0} - 2) \times \ldots \times 1$ distances need to be computed, taking into account that we don't recompute distances between the same two clusters. As we go up in the levels of the tree, some of the clusters are merged and this quantity diminishes. In Algorithm 10, this procedure is explained in a more detailed way.

Merging the two neighbor clusters $C_{q_*}^k$ and $C_{p_*}^k$ found, they become sons of a certain node of the tree of the level $l_k = 1$. This node defines a new cluster C_{p_*,q_*}^k that holds the fibers whose indices are in $\mathcal{I}_k^{(q_*,p_*)}$. This set of indices $\mathcal{I}_k^{(q_*,p_*)}$ is the joint set of the ones of the sons of the node, such that $\mathcal{I}_k^{(q_*,p_*)} = \mathcal{I}_k^{(q_*)} \cup \mathcal{I}_k^{(p_*)}$. Then, it is possible to compute its basis by a POD, let us name it $A^{k(p_*,q_*)}$. In this case, we truncate the SVD, in order for our approximation to have an error squared smaller or equal to the sum of the local errors squared obtained in both clusters.

Let $e_{k,l}^{(q)}$ be the error for the cluster C_k^q. Before starting the merging process, it is equal to the one that the computation of the leaves outputted, $e_{k,l}^{(q)} = \epsilon_{k,l}^{(q)}$. Let us recall that at the end of the previous grouping given by Algorithm 9, a global error budget b_k such that $(\epsilon_{k,l}^{(1)})^2 + \ldots + (\epsilon_{k,l}^{(N_{k,0})})^2 + b_k = \varepsilon^2$, may be left. Now, this budget can be used by equally distributing it between the clusters. A proportional part of the budget, $b_k/N_{k,0}$, is added to the local budgets for each cluster. Hence, the error budget that we have for a cluster C_k^q reads as $e_{k,l}^{(q)} = \epsilon_{k,l}^{(q)} + b_k/N_{k,0}$.

Notice that the error budget has been distributed between the clusters but the errors keep satisfying that $(e_{k,l}^{(1)})^2 + \ldots + (e_{k,l}^{(N_{k,0})})^2 \leq \varepsilon^2$. Respecting the prescribed accuracy is a necessary condition and this inequality will held in all the directions and for all the iterations of the method.

The error of the resulting merged cluster $e_k^{(q_*)}$ is given by the tail of the singular values of its SVD, the sum of the singular values that we depreciate in the truncation,

$$(e_k^{(q_*)})^2 = \sum_{h \in \mathbb{N}^*, h > j} (\sigma_h^{k(q_*,p_*)})^2$$

being j the number of terms that we need to keep in the truncated SVD in order to ensure that $(e_k^{(q_*)})^2 \leq (e_{k,l}^{(q_*)})^2 + (e_{k,l}^{(p_*)})^2$. At this stage of the algorithm, let us have the part of the error budget that has not been spent: $g_k^{(q)} = (e_{k,l}^{(q)})^2 + (e_{k,l}^{(p)})^2 - (e_k^{(q)})^2$. This value is equally redistributed into all the errors $(e_{k,l}^{(q)})^2$. They are updated by its value plus a factor $\frac{g_k^{(q)}}{\#\mathcal{C}_k^{(q)}}$.

Now that the two clusters are merged, the centroid of this new cluster needs to be recomputed as well as the values for the distances between clusters. The process of finding its nearest neighbor and merging the clusters they refer to is performed again. This process originates a new merge. Notice that this procedure is agglomerative and it is repeated till we have consecutively fused all the clusters in one. In this situation, the POD of the unfolding transposed is recovered. Find more details about the computation of the joint bases in Algorithm 10, where the process of merge clusters is shown in a schematic way.

Once the method explained in Algorithm 9 and Algorithm 10 finishes for all the different directions, we obtain a set of d partition trees. Each one of the trees corresponds to the succession of $\#l_k$ possibilities of partitions of the domain with respect to one direction. In a certain level l_k of the tree, we have N_{k,l_k} partitions of the domain. A node h, with $1 \leq h \leq N_k$, of the tree has associated sets of indices $\mathcal{I}_k^{(h)}$ that denote synthetically which fibers are well represented by the basis of a certain cluster and the error $\epsilon_{k,l}^{(h)}$ that we are making when we compute the approximation of a fiber by projecting it into the space generated by the basis of the cluster. We have obtained the different subdomains, $\Omega_k^{(i)}$, for $1 \leq i \leq N_k$.

Algorithm 10 Merging clusters algorithm

Require: Unfolding of a tensor in the direction k, $F^{(k)}(x_1,\dots,x_d) \in \mathbb{R}^{\left(\prod_{j=1,j\neq k}^{d} \mathcal{N}_j\right)\times \mathcal{N}_k}$, for $1 \leq k \leq d$. Tolerance ε. Set of indices $\mathcal{I}_k^{(1)},\dots,\mathcal{I}_k^{(N_k)}$. Set of local errors $\epsilon_{k,l}^{(1)},\dots,\epsilon_{k,l}^{(N_k)}$ and error budget b_k.
Output: Set of indices on each node $\mathcal{I}_k^{(1)},\dots,\mathcal{I}_k^{(N_k)}$. Local errors $\epsilon_{k,l}^{(1)},\dots,\epsilon_{k,l}^{(N_k)}$ for $1 \leq k \leq d$.
Compute the set of $\mathcal{N}_k$ fibers $f_1^k(y_k),\dots,f_{\mathcal{N}_k}^k(y_k)$, with $1 \leq k \leq d$
Initialize the set of indices of centroids: $\mathcal{C}_k^{(0)} = \{1,\dots,N_k\}$
Initialize counter of iterations $n = 1$
for $i = 1,\dots,N_k$ **do**
    Compute centroid: $c_i^k = \dfrac{\sum_{p\in\mathcal{I}_k^{(i)}} f_p^k(y_k)}{\#\mathcal{I}_k^{(i)}}$
    Distribute the budget equally in the local errors: $(e_{k,l}^{(i)})^2 = (\epsilon_{k,l}^{(i)})^2 + b_k/N_k$
end for
for $i = 1,\dots,N_k$ **do**
    Compute the Ward's distance between clusters and select the closest one:

$$q_* = \arg\min_{q\in\mathcal{C}_k^{(i)}} \Delta(C_i^k(x_k), C_q^k(x_k))$$

    where

$$\Delta(C_i^k, C_q^k) = \frac{\#\mathcal{I}_k^{(i)}\#\mathcal{I}_k^{(q)}}{\#\mathcal{I}_k^{(i)} + \#\mathcal{I}_k^{(q)}} \|c_{C_i^k}^k - c_{C_q^k}^k\|^2$$

    Compute joint basis matrix: $A^{k(i,q_*)} = (B_k^{(i)} B_k^{(q_*)})$
    Compute its basis by the truncated SVD:

$$A^{k(i,q_*)} = \left(\sum_{j\in\mathbb{N}^*} \sigma_j^{k(i,q_*)} U_j^{k(i,q_*)} \otimes V_j^{k(i,q_*)}\right) + (e_k^{(i)})^2, \quad \text{, where} \quad (e_k^{(i)})^2 \leq (e_{k,l}^{(i)})^2 + (e_{k,l}^{(q)})^2$$

    Merge clusters and set the basis of the joint $A_k^{(i)} = U^{k(i,q_*)}$
    Set the error budget of the joint: $(e_k^{(i)})^2 = \sum_{h\in\mathbb{N}^*, h>j} (\sigma_h^{k(i,q_*)})^2$
    Update the local error: $e_{k,l}^{(i)} = e_k^{(i)}$
    Compute the non used error budget: $g_k^{(i)} = (e_{k,l}^{(i)})^2 + (e_{k,l}^{(q)})^2 - (e_k^{(i)})^2$
    Update the set of indices of centroids: $\mathcal{C}_k^{(i)} = \mathcal{C}_k^{(i-1)} - q$
    Redistribute $g_k^{(i)}$: $(e_{k,l}^{(i)})^2 = (e_{k,l}^{(i)})^2 + g_k^{(i)}/\#\mathcal{C}_k^{(i)}$
    Update the centroid: $c_i^k = \dfrac{\sum_{p\in\mathcal{I}_k^{(i)}\cup\mathcal{I}_k^{(q)}} f_p^k(y_k)}{\#\mathcal{I}_k^{(i)}\cdot\#\mathcal{I}_k^{(q)}}$
end for

3.4 Computing local HOSVD method

In the following, in order to determine the best partitioning to approximate the tensor, we perform an extensive search for all the possible combinations. If we have l_k different possibilities of partitions of the domain in the direction k, with $1 \leq k \leq d$, the whole set of combinations is then $\prod_{k=1}^{d} l_k$. Each of this combinations provides a certain partitioning for computing a certified approximation of the tensor F in the local domain. Between all these combinations, we select as the best the one that provides the most compressed approximation.

The cost of computing and comparing the HOSVD of the $\prod_{k=1}^{d} l_k$ possible combinations of partitions is what makes this alternative very costly and in some situations not feasible.

Let us explain the method in more detail. As we have seen along the chapter, the clustering method proposed in this work, performs independently for each one of the directions. From the clustering method, d binary partition trees are obtained $\{T_{I_k}\}_{1\leq k\leq d}$. Being l_k the number of partitions of the domain in the $k-$th dimension, with $k = 1,\dots,d$.

Each level l_k of the partition tree obtained, T_{I_k} in the direction k, defines a set of sets of indices that determine a possible partition of the domain. The possible combinations of indices are given by the number of nodes in the different generations of the tree, N_k. The number of nodes per generation is equal to the total number of levels $\#l_k$ minus the level of the generation l_k for this binary tree structure.

We recall that the proposed clustering method provides N_k sets of indices per direction namely $\mathcal{I}_k^{(i)}$, with $i = 1, \ldots, N_k$, that denote the fibers that belong to the cluster C_k^i. During the merging process, the number of clusters, and its associated set of indices, decrease in one unit as we get closer to the root of the tree. See in the following Figure 3.3 , an schematic partition tree T_I in the $k-$th direction for $\#l_k = 4$. As we can see, in the leaves $N_{k,0} = 4$, and with the first fusion, $N_{k,1} = 3$ and successively till arriving to the root.

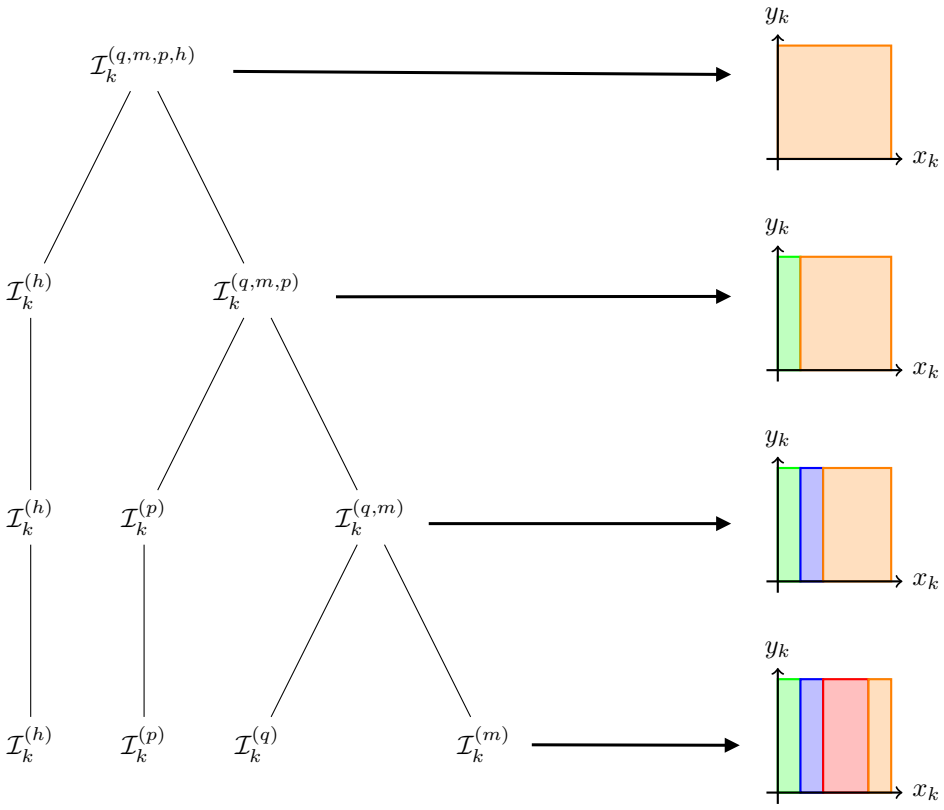


Figure 3.3: In the left an example of the partition tree in the direction k with $N_{k,0} = 4$ and in parallel on the right, the evolution of the partitions of the domain with the mergings.

As we can see in Figure 3.3, as the method goes on and the levels of the tree increase, the number of partitions of the domain in one direction changes. The number of subdomains is reduced in one after each merge.

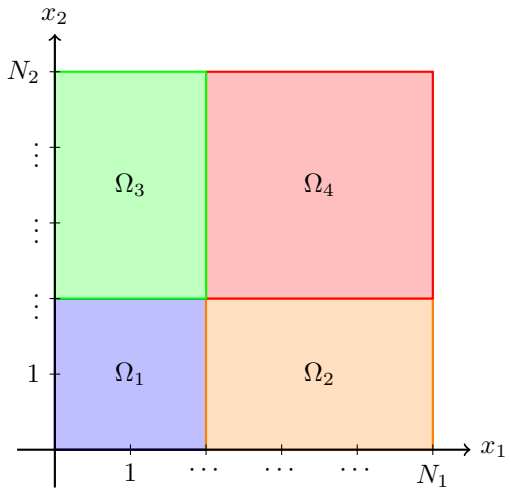
For each one of the trees, each level defines one possible partition of the domain. As we have seen, each tree presents $\#l_k$ levels. As we can easily see in the scheme of the partition tree, the level $l_k = 0$ corresponding to the leaves, provides one possible partition of the domain, in which $\mathcal{L}(T_{I_k}) = \mathcal{I}_k^{(h)} \cup \mathcal{I}_k^{(p)} \cup \mathcal{I}_k^{(q)} \cup \mathcal{I}_k^{(m)}$. In the previous generation a different partition of the domain is provided, $\mathcal{V}^1(T_{I_k}) = \mathcal{I}_k^{(h)} \cup \mathcal{I}_k^{(p)} \cup \mathcal{I}_k^{(q,m)}$, in which the partitions have decreased in one unit. It continues till we arrive to the root.

Remark: In general, the best partition of the domain is different from the intersection of the best partitions per direction.

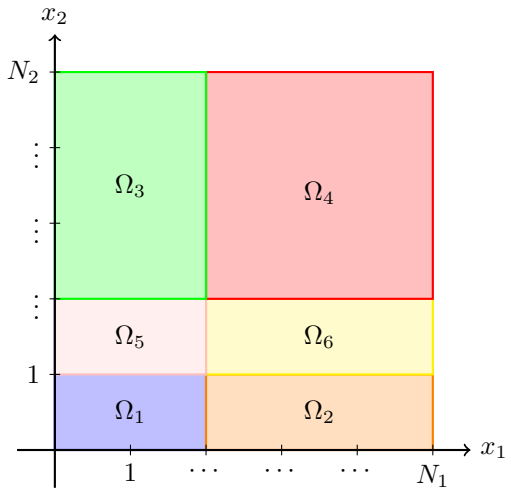
The sets of indices $\mathcal{I}_k^{(i)}$ are closely related with the characteristic functions $\mathbf{J}_k^{(i)}$ defined in Equation 3.4. They denote which fibers of the unfolding belong to the same cluster and hence, they can be well represented on a certain subdomain. Let us recall from Equation 3.5 that the characteristic functions per direction are related with the characteristic functions that restrict the tensor of Equation 3.2 to the local domain as

$$\mathbf{I}^{(i)}(x_1, \ldots, x_d) = \prod_{k=1}^{d} \mathbf{J}_k^{(i)}(x_k) \tag{3.12}$$

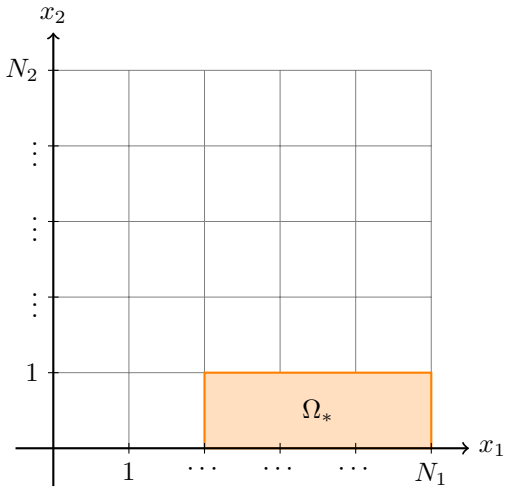
To illustrate the main idea of the method, let us start with a 2-dimensional example. The merging process provides two trees T_{I_1} and T_{I_2}. Let us say that, a certain partition of the domain provides $N_1 = 2$ and $N_2 = 2$. The local domains that appear look as follows: The combination of the partitions $\mathbf{J}_1^{(1)}$ and $\mathbf{J}_2^{(1)}$ originates the blue domain Ω_1; the combination of $\mathbf{J}_1^{(1)}$ and $\mathbf{J}_2^{(2)}$ the orange subdomain, Ω_2, the partitions $\mathbf{J}_1^{(2)}$ and $\mathbf{J}_2^{(1)}$ originate Ω_3 in green, and $\mathbf{J}_1^{(2)}$ and $\mathbf{J}_2^{(2)}$ the red domain Ω_4. In the figure below we can see a schematic example of how these domains can be placed in the 2-dimensional set.



In a different iteration of the method, we place ourselves in different levels of the trees. Let us say that they provide a partition in which the domain is more divided than before (the case is in a generation close to the leaves of the tree) and we have that $N_1 = 2$ and $N_2 = 3$ and for a certain combination of partitions, the local domains that appear in this case look as follows: As before, The combination of the partitions $\mathbf{J}_1^{(1)}$ and $\mathbf{J}_2^{(1)}$ originates the blue domain Ω_1; the combination of $\mathbf{J}_1^{(1)}$ and $\mathbf{J}_2^{(2)}$ the orange subdomain, Ω_2, but then the partitions $\mathbf{J}_1^{(2)}$ and $\mathbf{J}_2^{(1)}$ originate Ω_5 in pink, and $\mathbf{J}_1^{(2)}$ and $\mathbf{J}_2^{(2)}$ the yellow domain Ω_6. The combination between $\mathbf{J}_1^{(1)}$ and $\mathbf{J}_2^{(3)}$ provides now the green domain Ω_3 and the red domain Ω_4 is given by $\mathbf{J}_1^{(1)}$ and $\mathbf{J}_2^{(3)}$. In the schematic figure below we can see how these domains are distributed.



Among all the possible partitions, we pick the best one in terms of memory. That means that we compute the POD basis of all the non-zero subdomains and we keep the partition per direction that provides the subdomain with the smaller amount of memory needed. We have named it Ω_*. In the following schematic figure, an example of the local domains obtained is presented following the simple 2-dimensional case. Without loss of generality we can say that in this case, $\Omega_* = \Omega_2$ and then, the domain in which we are considering our approximation will look:



The aim of the method will be to find out the partitions of the domain given by $\mathbf{J}_1^{(i_1*)}(x_1)\dots\mathbf{J}_d^{(i_d*)}(x_d)$ that provide the best approximation $\tilde{F}_{i_1*\dots i_d*}(x_1,\dots,x_d)$ in terms of memory. This means that for all $1\leq i_1\leq \#l_1,\dots,1\leq i_d\leq \#l_d$, between all the $\tilde{F}_{i_1\dots i_d}$ we select as the best $\tilde{F}_{i_1*\dots i_d*}$, the one that requires less memory to store its HOSVD basis. For the sake of simplicity, in the following let us denote the set $\{i_1*\dots i_d*\}$ just $*$. Then, the best one of the approximations becomes $\tilde{F}_*(x_1,\dots,x_d)$.

After computing the memory needed to store the basis on each of the $\#l_1\times\dots\times\#l_d$ possible partitions of the domain and selecting the best in terms of memory, we define the partition of the domain associated to the best approximation. The best of the possible partitions of the domain is given by the set of indices:

$$\mathbf{I}^{(*)}(x_k,y_k)=\prod_{k=1}^{d}\mathbf{J}_k^{(i_k*)}(x_k)$$

where $\mathbf{J}_k^{(i_k*)}(x_k)$, with $1\leq i_k*\leq N_k$ is the value of the characteristic function in the direction k that appears in the best combination of partitions. Then, the number of different subdomains in which the function is locally approximated reads as $N=\prod_{k=1}^{d}N_{k,l_k*}$.

The best approximation obtained by the proposed method is given by Equation 3.2 and it reads:

$$\tilde{F}_*(x_k,y_k)=\tilde{F}(x_k,y_k)\mathbf{J}_1^{(i_1*)}(x_1)\dots\mathbf{J}_d^{(i_d*)}(x_d) \tag{3.13}$$

where $\tilde{F}$ is the HOSVD approximation given by:

$$\tilde{F}(x_k,y_k)=\sum_{i_1=1}^{R_1}\dots\sum_{i_d=1}^{R_d}G_{i_1\dots i_d}b_{i_1}^{(1)}\otimes\dots\otimes b_{i_d}^{(d)} \tag{3.14}$$

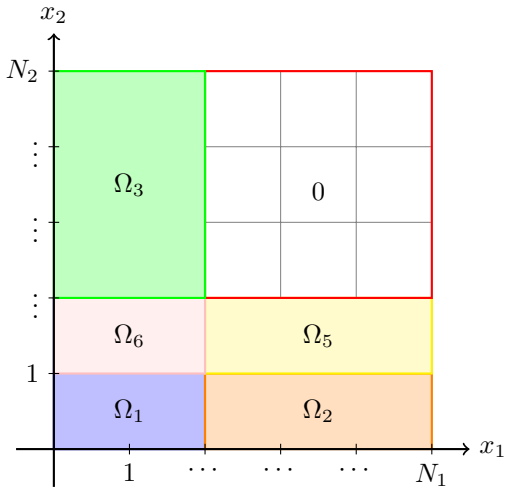
in which $R_1,\dots,R_d$ are the $n-$ranks and they are smaller that $\mathcal{N}_1,\dots,\mathcal{N}_d$, respectively. Given the function restricted in the local subdomain F_*, the ranks of the approximation are chosen by truncating the HOSVD approximation with a certain error that has been uniformly distributed in the local domains $\frac{\varepsilon}{N_k}$. In this way, the approximation is certified under the prescribed tolerance.

The core tensor $G\in\mathbb{R}^{R_1\times\dots\times R_d}$ is stored in full format. The basis functions are written by terms, introducing the set of R_k modes of each basis B_k as: $b_{k,1},b_{k,2}\dots,b_{k,R_k}$. These modes are the columns of the basis: $B_k=(b_{k,1}\quad b_{k,2}\quad\dots\quad b_{k,R_k})$.

The approximation in local HOSVD format of the tensor reads:

$$F(x_1,\dots,x_d)\approx\left(\tilde{F}_{i*,\dots,j*}\right)\mathbf{J}_1^{(i*)}\dots\mathbf{J}_d^{(j*)}=\left(\sum_{i_d=1}^{R_d}\dots\left(\sum_{i_1=1}^{R_1}G_{i_1,\dots,i_d}^{(i*,\dots,j*)}\otimes b_{1,i_1}^{(i*)}\otimes\dots\otimes b_{d,i_d}^{(j*)}\right)\right)\mathbf{J}_1^{(i*)}\dots\mathbf{J}_d^{(j*)} \tag{3.15}$$

As a remark, if we observe that in some region, the function is zero (or close to zero), we don't spend memory on that region. We consider it well approximated by the zero function. The approximation $\|F_i\|_{L^2(\Omega_i)} < \frac{\varepsilon}{N_k}$ and then $\tilde{F}_i = 0$. Imagine that following with the previous example, the function in the region Ω_4 in the example turns out to have values equal or very close to zero and its best approximation is the zero function. Then, the domain can be considered as:



Discretization

If we proceed with the discretization of the function by points, the variables will read: $y_{k,p_k} \in y_k$ for $p_k = 1, \ldots, \mathcal{N}_{yk}$. The approximated discretized function then reads:

$$F(x_{1,p_1}, \ldots, x_{d,p_d}) \approx \left(\sum_{i_d=1}^{R_d} \cdots \sum_{i_1=1}^{R_1} G^{(i*,\ldots,j*)}_{i_1,\ldots,i_d}(x_{1,p_1}, \ldots, x_{d,p_d}) \times b^{(i*)}_{1,i_1}(x_{1,p_1}) \times \ldots \times b^{(j*)}_{d,i_d}(x_{d,p_d}) \right) \mathbf{J}^{(i*)}_1(x_{1,p_1}) \ldots \mathbf{J}^{(j*)}_d(x_{d,p_d})$$

3.5 Cost and complexity of the algorithm

Let us compute the complexity of the algorithm with respect to one of the unfoldings of the input tensor. Taking into account that there are d of them, the final result will be d times the result obtained for the unfolding k, where $k \in [1, \ldots, d]$. In the following we will consider the number of degrees of freedom of the fibers equal for all of them and let us call it $\mathcal{N}$ such that $\mathcal{N} = \mathcal{N}_{yk}$.

Cost of computing the leaves

We start by computing the cost of its first phase of clustering/regrouping the fibers. The number of operations of this part will depend on how long it takes to find the "closest" fiber, computing its basis and then checking again the similarity between basis in the sense of the l^2 distance that we are considering.

In the first iteration, in order to compute the distance to find the nearest neighbors, the computation requires $\mathcal{O}(\mathcal{N}\mathcal{N}_k^2)$ operations. Then, on each iteration the number is decreasing by at least 2 units. To write so, let us use the already introduced cardinality of the set of indices of the fibers that are not yet included in a cluster $\#\mathcal{T}^{(it)}$. At the beginning $\#\mathcal{T}^{(0)} = \mathcal{N}_k$, and it will decrease with the iterations.

This numbers are specific for the case in which the partition of the set of fibers in smaller subsets is mutually exclusive (no fiber in the set is in more than one subset) and jointly exhaustive (every fiber is in some subset).

It would be possible to accelerate this first phase constructing $k - d$ trees (this data structure splits the space and allows faster search for neighbors), or with the ball tree method (that also divides the space but with in contrast to k-d trees, which divides the space with median value "cuts", ball tree groups points into

"balls" organized into a tree structure). See more details and alternatives in [270].

After the computation of the nearest neighbors and the SVD in order to compute the basis that generates the subspace needs to be computed. Computing a full SVD of a matrix $M \in \mathbb{R}^{p\times q}$ is fundamentally an $\mathcal{O}(pq \cdot \min(p,q))$-time problem. Normally, we are facing "tall and skinny" matrices, because the number of columns is equal to two and the number of rows is equal to the degrees of freedom $\mathcal{N}$ (normally larger than 2). The cost of an SVD of one of them is equal to $4\mathcal{N}$, i.e. linear on the degrees of freedom $\mathcal{O}(\mathcal{N})$.

The next step is checking if some of the fibers left can be well represented by the basis. This operation is composed by a scalar product between the $2d$ basis, a subtraction and a squared. To compute the scalar product, we must perform $2\mathcal{N}$ multiplications and $2(\mathcal{N}-1)$ additions. The cost reads $\mathcal{O}(\mathcal{N})$.

The last operation that eventually we need to compute is the modification of the basis with the already introduced Brand's method. In the concrete case in which we are computing an additive rank-1 modification, the new SVD can be computed in $\mathcal{O}(R_b^2)$ time, being R_b the rank of the matrix.

In a certain iteration of the method in which we have included some fiber into the cluster, it, the cost of the subdivision in 2-dimensional domains will be:

$$\mathcal{O}(\#\mathcal{T}^{(it)}\mathcal{N}\mathcal{N}_k^2) + \mathcal{O}(\mathcal{N}) + \mathcal{O}((\#\mathcal{T}^{(it)} - 2)\mathcal{N}) + \mathcal{O}(R_b^2) =$$

$$= \mathcal{O}((\#\mathcal{T}^{(it)}(\mathcal{N}_k^2 + 1) - 1)\mathcal{N} + R_b^2)$$

In the case in which the method doesn't compute the tilting part, the expression won't have the last term.

This process is done d times, one per direction. Considering all $\mathcal{N}_k$ equal. The total cost of this phase reads:

$$\mathcal{O}((\#\mathcal{T}^{(it)}(\mathcal{N}^2 + 1) - 1)d\mathcal{N}_k + dR_b^2)$$

Cost of the merging

In this second part of the proposed method, a partition tree is computed. Starting from the clusters set as leaves and finishing when all the local domains have been fused and the HOSVD is recovered.

In order to compute the neighbors between centroids we compute its pairwise distance. In the leaves level of the tree, iteration $it_l = 0$, $(N_{k,0} - 1) \times (N_{k,0} - 2) \times \ldots \times 1$ distances need to be computed, taking into account that we don't recompute distances between the same two clusters. let us name the quantity of distances that we need to compute on each iteration it_l as $\#\mathcal{D}_k^{(it_l)}$. In the level l_k, reached at $it_l = l_k$, the number of distances: $(l_k - 1) \times (l_k - 2) \times \ldots \times 1$. As we iterate, we go up in the levels of the tree, and this quantity diminishes because a pair of clusters are merged. The distance computation requires $\mathcal{O}(\#\mathcal{D}_k^{(it_l)^2}\mathcal{N}_k)$

The next operation in the algorithm is the computation of the SVD of the joint basis matrix. In this case the matrices are still "tall and thin" because the number of columns is normally smaller that $\mathcal{N}$. In the first merging, the number of columns of the matrix is at most 4 (due to the fact that in the previous part of the method the bases were at most 2-dimensional), in the second is at most 6, etc. For each iteration it_l on the tree, the number of columns of the matrix obtained by SVD has at most $2 \cdot it_l + 2$ columns. Hence, the order is $\mathcal{O}((2 \cdot it_l + 2)^2\mathcal{N})$, linear on the degrees of freedom.

This way, the cost per direction reads as:

$$\mathcal{O}((\#\mathcal{D}_k^{(it_l)^2} + (2 \cdot it_l + 2)^2)\mathcal{N}_k)$$

As before, the merging process is done independently for each one of the d directions. If we consider the value of $\#\mathcal{D}_k^{(it_l)}$ equivalent in all the directions, the total cost:

$$\mathcal{O}((\#\mathcal{D}_k^{(it_l)^2} + (2 \cdot it_l + 2)^2)d\mathcal{N}_k)$$

Cost of the basis computation

The last operation in the algorithm is the computation of the SVD of the $\prod_{k=1}^{d} l_k$ possible partitions of the domain.

Compute the cost of the computation of the basis is difficult a priori because it depends on the partitions produced by the method. Let us estimate this value considering generic partitions of the domains. Let us suppose that for the $k-$th direction, the partition of the subdomain Ω_k that provides the best partition of the whole domain Ω involves $\frac{\mathcal{N}_k}{p_k}$ fibers of size $\mathcal{N}$. Let p_k be a generic fractional coefficient to describe the partition. The cost of one computation of the HOSVD for the local domain is:

$$\mathcal{O}(\prod_{k=1}^{d}\left(\frac{\mathcal{N}_k}{p_k}\right))$$

The total cost of the algorithm depending on the iterations of the first and second phase results:

$$\mathcal{O}((\#\mathcal{T}^{(it)}(\mathcal{N}_k^2+1)-1+\#\mathcal{D}^{(it_l)2}+(2\cdot it_l+2)^2)d\mathcal{N}_k+dR_b^2+\prod_{k=1}^{d}\left(\frac{\mathcal{N}_k}{p_k}\right)) \tag{3.16}$$

Let us also remark that in the subdomains in which the value for the tensor is zero, or very close to zero, no memory is spent, neither the computation of the SVD is done once we have identified the null regions. Even though, the cost is very high. In this work a first approach to a good local approximation is studied by computing all the possibilities, computing their memories and keeping the most compressed one.

3.6 Summary

The clustering method proposed performs independently for each one of the unfolding matrices on the different directions. Its aim is to compute a partition tree per direction that encloses the different partitions of the domain in subdomains. Among them, the one in which the compression ratio for a given tensor is larger is selected.

In broad terms, the process can be divided in three different phases: the initial one that computes 2-dimensional subdomains, leaves of the partition trees; the second phase in which the tree is computed that merges consecutively them and saves information on every merging; and the last one in which, once the two previous phases of the method have finished for all the different dimensions, between all the possible partitions of the domain, selects the one in which the tensor is more compressible. Let us recapitulate the main steps performed by the method.

For $1 \leq k \leq d$, let $F^{(k)}$ be the unfolding of the $k-$th direction of the tensor F.

- Computation of the leaves
  - Starting from the fibers (column vectors) of the unfolding, find the nearest neighbor.
  - Compute 2-dimensional basis of the space spanned by the two neighbor fibers as the matrix of modes of a POD of the two vectors.
  - Check if other fibers are well represented by the basis. If it is so, modify the POD with Brand's method of Equation 3.11.

  At this point, a set of clusters defined by 2-dimensional basis that well represents some of the fibers (with a certain error) is obtained. They are the leaves of the partition tree constructed. In the following phase, the tree is constructed.

- Merging clusters
  - Compute centroids of each cluster.
  - Compute pairwise distances between clusters and select the smallest.
  - Compute the joint subspace, a new cluster resulting from the merging of two clusters.

- Recompute centroids and distances for the new cluster.

This process is repeated till all the clusters have been fused. On each fusion, the resulting partition of the domain is saved.

- Selecting the best partition of the domain

The task of selecting the best partition of the domain is done by computing all the possible combinations of partitions of the domain between the different directions. The method computes all the HOSVD and selects as the best the one that requires less memory to store the approximation, i.e. the one that provides higher compression ratio with respect to the HOSVD of the initial tensor.

3.7 Numerical results

In the present section, we propose several test cases to asses the properties of the proposed method. The tensors we are going to approximate are of moderate order because, as we have already introduced, the computational complexity of the method is high.

3.7.1 Compression of a Gaussian function

The initial test consists in computing the approximation of a Gaussian function. The function to approximate reads as:

$$f(x,t) = e^{-500*(x-0.05-t)^2} \tag{3.17}$$

where $x \in [0,1]$ and $t \in [0,0.9]$. The resolution for the variable x is fixed to $\mathcal{N}_x = 500$ points and the resolution in the variable t, $\mathcal{N}_t$ is going to be changed in order to see how its value affects the results. We consider a finite difference approach of the function f.

The compression ratio (Cr) shown in the following results is defined as the memory employed to store the compressed tensor obtained by the local HOSVD method over the memory needed to store the tensor in full format:

$$Cr = \frac{\text{Memory}(F_{locHOSVD})}{\text{Memory}(F)}$$

Let us show in Figure 3.4 the compression rate for different values of the tolerance of the approximation of the tensor computed by HOSVD method and the local HOSVD alternative. The values from the tolerance in this test hold $\epsilon = \{1.0 \cdot 10^{-2}, \epsilon = 1.0 \cdot 10^{-3}, \epsilon = 1.0 \cdot 10^{-4}\}$. And the values for the time resolution $\mathcal{N}_t = \{200, 400, 600, 800, 1000\}$.

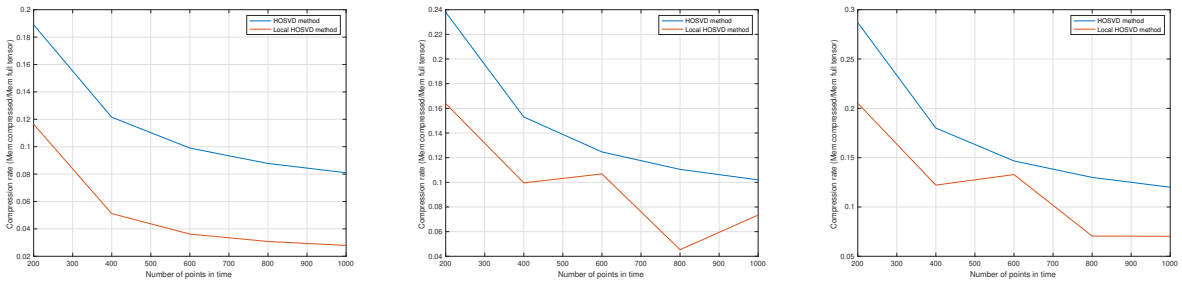


Figure 3.4: The compression rate for the function computed by the local HOSVD approximation method (in orange) and also the compression rate for HOSVD method (in blue). From left to right, the tolerance: $\epsilon = 1.0 \cdot 10^{-2}$, $\epsilon = 1.0 \cdot 10^{-3}$ and $\epsilon = 1.0 \cdot 10^{-4}$. In all of them the resolution in space is fixed to $\mathcal{N}_x = 500$ and in time $\mathcal{N}_t = 200, 400, 600, 800, 1000$ points.

Let us show in the next Figure 3.5, the reconstruction of the Gaussian function of Equation 3.17 with an error of $\epsilon = 1.0 \cdot 10^{-2}$ and resolution $\mathcal{N}_x = 500$ and $\mathcal{N}_t = 400$. In the figure, the exact function is

reconstructed with the HOSVD method and with its local version. It is known that the HOSVD method is not optimal for the reconstruction of this function, leading to some oscillations that are not present in the exact function. We can observe that in the reconstruction made via the local HOSVD method, these oscillations have disappeared.

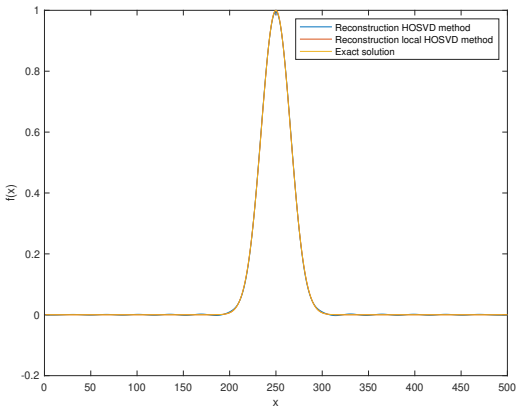


Figure 3.5: Local reconstruction for the Gaussian function (in orange) in comparison with the HOSVD (in blue) and the analytic expression (in yellow) for an error of $\epsilon = 1.0 \cdot 10^{-2}$ and resolution $\mathcal{N}_x = 500$ and $\mathcal{N}_t = 400$.

In Figure 3.6, the solution of the transport function is shown for the same values of error and resolution. The value for the error is $\epsilon = 1.0 \cdot 10^{-2}$ and for the resolution the values taken are $\mathcal{N}_x = 500$ and $\mathcal{N}_t = 400$.

The exact solution of the Equation 3.17 is shown, as well as its reconstructions by local HOSVD and HOSVD methods.

As we can see, the reconstruction with local HOSVD presents some ondulatory interferences, this can be due to the large value for the error that we are considering. On the other hand, the reconstruction computed with HOSVD presents several oscillations is all the domain, more concentrated close to the solution and wider as we are going further.

In the following table, Table 3.1 some values of the lowest memory storage required obtained from the extensive research between the possible partitions of the domain in which the HOSVD method has been applied. They are computed for different resolutions in the domain, changing the number of degrees of freedom in the variable x, $\mathcal{N}_x$, and in t, $\mathcal{N}_t$, as well as the values for the error. In this test, the error adopts values from $1.0 \cdot 10^{-4}$ to $5.0 \cdot 10^{-2}$. It is shown as well, the compression ratio. In this case, the ratio that appears in the table is not with respect to the memory needed to store the function in the full format but to the one needed to store the HOSVD (in this 2-dimensional case, SVD) decomposition of the function.

Moreover, the number of partitions per direction that provides the best compression ratio is shown.

As we can see in Table 3.1, the compression ratio obtained with the proposed method varies with relation to the HOSVD, providing a larger compression of the data with an approximately factor two. This ratio stays stable even when changing the tolerance or the resolution. Nevertheless, the optimal number of partitions of the domain changes with the prescribed tolerance and also with the resolution.

3.7.2 Compression of the solutions of the Fitz-Hugh-Nagumo equation

The Fitz-Hugh–Nagumo model (FHN), named after Richard FitzHugh (1922–2007) who suggested the system in 1961 and J. Nagumo et al. who created the equivalent circuit the following year, describes a prototype of an excitable system (e.g., a neuron).

The FHN Model is an example of a relaxation oscillator because, if the external stimulus I_{ext} exceeds a certain threshold value, the system will exhibit a characteristic excursion in phase space, before the variables u and w relax back to their rest values. This behavior is typical for spike generations (a short, nonlinear

$(\mathcal{N}_x, \mathcal{N}_t)$	ϵ	Best Memory	Mem POD/ Best Mem	Best # clusters (x-t)
(250,200)	$5 \cdot 10^{-2}$	4576	2,07	3-4
	$1 \cdot 10^{-2}$	5015	2,43	7-9
	$5 \cdot 10^{-3}$	6087	2,15	12-15
	$1 \cdot 10^{-3}$	6866	2,23	8-8
	$5 \cdot 10^{-4}$	7326	2,21	6-8
	$1 \cdot 10^{-4}$	8438	2,13	6-8
(500,400)	$5 \cdot 10^{-2}$	6477	2,92	6-7
	$1 \cdot 10^{-2}$	9433	2,58	9-9
	$5 \cdot 10^{-3}$	12013	2,17	9-13
	$1 \cdot 10^{-3}$	13894	2,20	11-17
	$5 \cdot 10^{-4}$	14931	2,17	6-8
	$1 \cdot 10^{-4}$	18144	1,98	7-8
(750,600)	$5 \cdot 10^{-2}$	9768	2,90	7-8
	$1 \cdot 10^{-2}$	12806	2,85	10-10
	$5 \cdot 10^{-3}$	15899	2,46	8-10
	$1 \cdot 10^{-3}$	23477	1,96	11-14
	$5 \cdot 10^{-4}$	21883	2,22	7-11
	$1 \cdot 10^{-4}$	25134	2,15	8-9
(1000,800)	$5 \cdot 10^{-2}$	14559	2,60	6-6
	$1 \cdot 10^{-2}$	17367	2,80	7-9
	$5 \cdot 10^{-3}$	19723	2,65	9-10
	$1 \cdot 10^{-3}$	29826	2,05	7-7
	$5 \cdot 10^{-4}$	34050	1,90	11-13
	$1 \cdot 10^{-4}$	35581	2,02	7-10

Table 3.1: Comparison between the memory obtained when computing the POD and the memory computed for the best of the possible partitions of the domain for different values of the error tolerance and different resolutions in the domain.

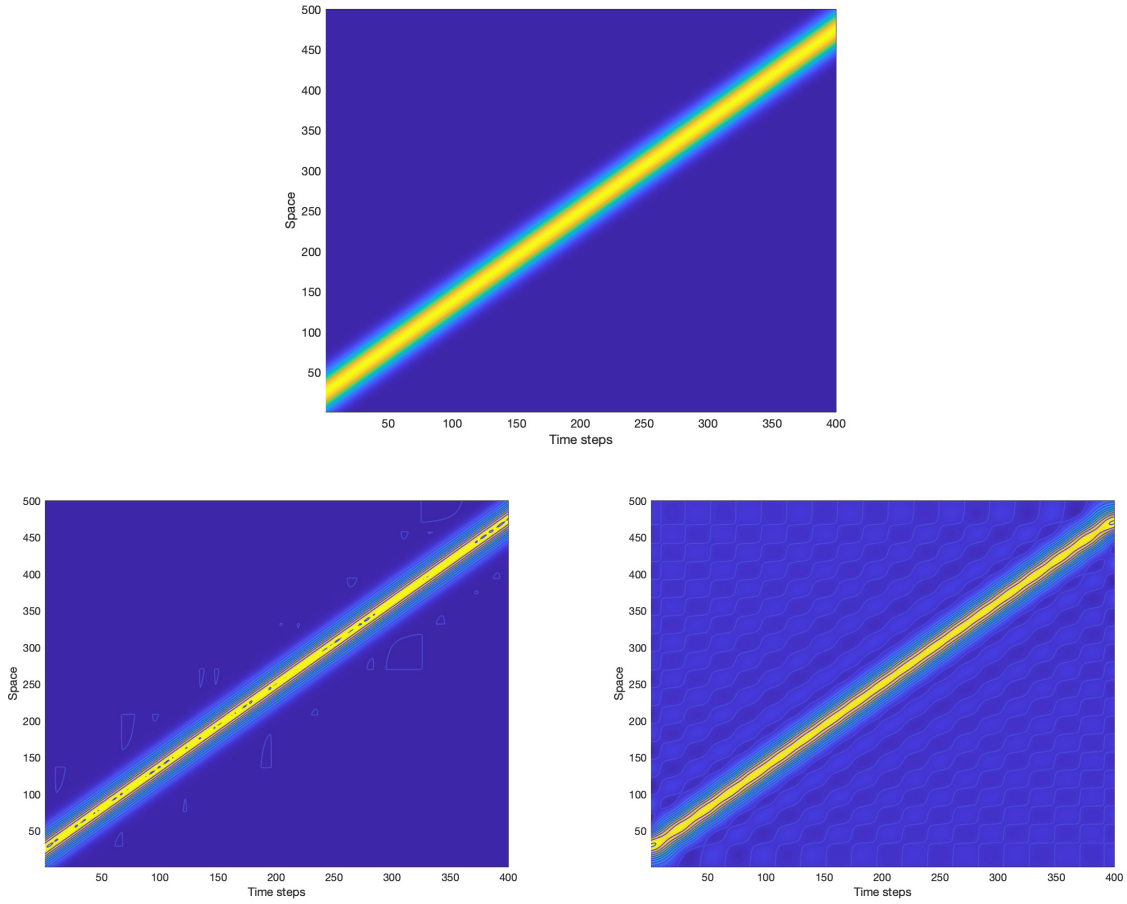


Figure 3.6: Reconstruction of the exact solution with an error of $\epsilon = 1.0 \cdot 10^{-2}$ and resolution $\mathcal{N}_x = 500$ and $\mathcal{N}_t = 400$. In the left, the reconstruction of the exact solution is computed with the local HOSVD method and in the right with the HOSVD method.

elevation of membrane voltage u, diminished over time by a slower, linear recovery variable w) in a neuron after stimulation by an external input current.

The pair of equation describing the phenomena:

$$\partial_t u(x,t) = -ku(x,t)(u(x,t)-a)(u(x,t)-1) - w(x,t) + \nu\Delta u(x,t) + I_{ext} \tag{3.18}$$

$$\partial_t w(x,t) = \epsilon(u(x,t) - \gamma w(x,t))$$

where $\Delta := \frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y}$ the Laplace operator. The domain is defined as $\Omega = ([0,1] \times [0,1], T)$. Being T the final time whose value change depending on the test performed. The boundary conditions have been set as Neumann homogeneous boundary conditions (zero-flux boundary conditions.): $\frac{\partial u(x,t)}{\partial \vec{n}} = 0$ in $\partial\Omega$.

The parameters $\epsilon > 0$, the time of the pulse, and $\gamma \geq 0$, the strength of the recovery of the system. The different choices of the parameters in the equations make the solution $u(x,t)$ adopting different shapes: from the stationary wave to a labyrinth structure, passing trough certain values that produce a spiral. See [271] for more information about the values of the parameters and the different patterns that they produce. In the tests performed, we are facing a 2-dimensional space variable, the time variable and other variable, let us call it θ, that refers to the parameters of the system. In this case, the variable θ encloses a linear combination of the parameters k and ϵ. In some tests, the variable θ will adopt a fixed value and it will behave as a parameter of the system. In other tests, it will adopt values inside a certain domain and it will be treated as a third variable.

The system is simulated numerically by a finite differences scheme on a (256×256) two-dimensional spatial grid. The number of time steps is being changed on the different tests performed. The final time has been set as $T = 800$. The initial conditions from which we have recovered the spiral shape have been: $u(x, 0) = 1$ for $x \in ([0, 0.5] \times [0, 0.5])$ and 0 in the rest, being the left down quarter of the square 1 and the rest 0. The values for the parameters chosen in all the simulations are: $\gamma = 1$, $a = 0.1$, $\nu = 1/62500$, $k = 0.96$ and $\epsilon = 0.0015$. In the case in which they are treated as variables and take values in the intervals they read $k \in [0.96, 1.20]$ and $\epsilon \in [0.0015, 0.0060]$ respectively.

In Figure 3.7 we can see how from the initial conditions the system evolves in time. In order to compute this simulation, the number of time steps is $\mathcal{N}_t = 400$.

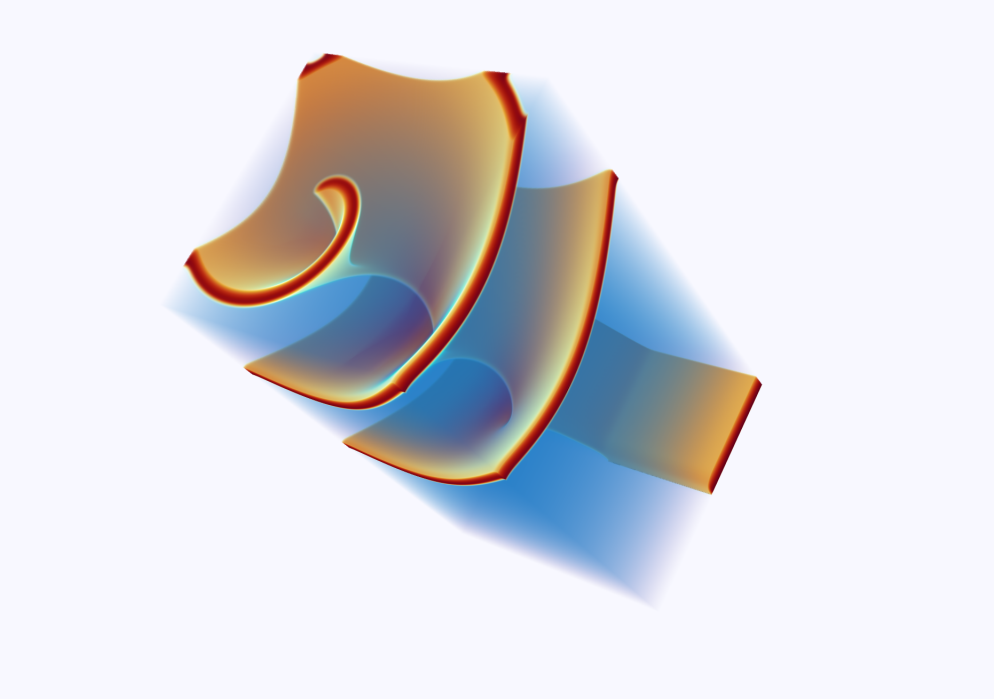


Figure 3.7: Solution of the Fitz-Hugh-Nagumo equation, 3.18, for $\mathcal{N}_x = 256 \times 256$ square and time step equal to 2, $\mathcal{N}_t = 400$. The initial conditions are set as: 1 for $x \in ([0, 0.5] \times [0, 0.5])$. The values for the parameters $k = 1$ and $\epsilon = 0.005$.

The part of the algorithm shown in Algorithm 9 returns a set of 2-dimensional basis with respect to one of the dimensions of the system. Let us show an example of one of the basis obtained for the unfolding in x in Figure 3.8. We can see the spiral structure that we identify in the Figure 3.7 in the spatial domain.

Let us show graphically how the merging algorithm operates. In Figure 3.9 three different screenshots of the simulation in the spacial domain are shown. We can see in the exact solution the transition from the initial solution till the initial part of the spiral for three different values of time. The values have been enough spaced in order to give an intuition of how the function behaves. In the next Figure 3.10, three sets of partitions of the spatial domain corresponding to the beginning of the simulation are shown. Each partition of the domain is colored differently to visualize well them. They refer to three consecutive nodes of the partitioning tree. It is clear how after each merging (iteration of the merging phase), the number of groups is reduced in one. The plots are for a $T = 200$, what means that just the first part of the dynamical system is considered.

From Figure 3.10, we can see how the local HOSVD method has divided the spatial domain in order to compute the approximation. We can also see how the shape of the partitions is adapted to the initial condition and its initial translation. The Figure 3.11 shows three analogous plots for three consecutive partitions on the spatial domain in a longer time $T = 800$. In this case, the system has evolved and we can identify the spiral structure that characterizes the solution of the equation. We can

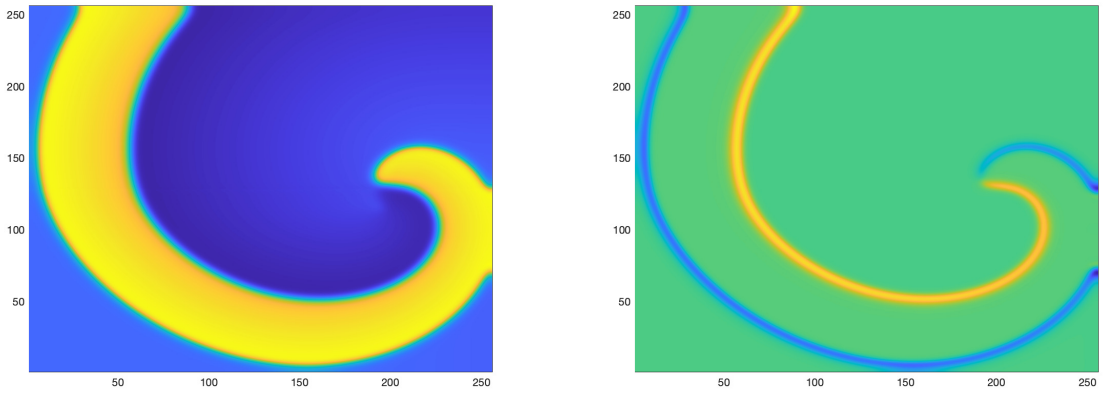


Figure 3.8: One of the 2-dimensional basis obtained for 3.18 with respect to the variable x. From left to right, the first and the second modes of the basis. For $\mathcal{N}_x = 256 \times 256$ square and time step equal to 2. The values for the parameters are: 2 values of $k = [0.96, 0.98]$ and 5 values of ϵ with $\epsilon \in [0.0050, 0.00375]$.

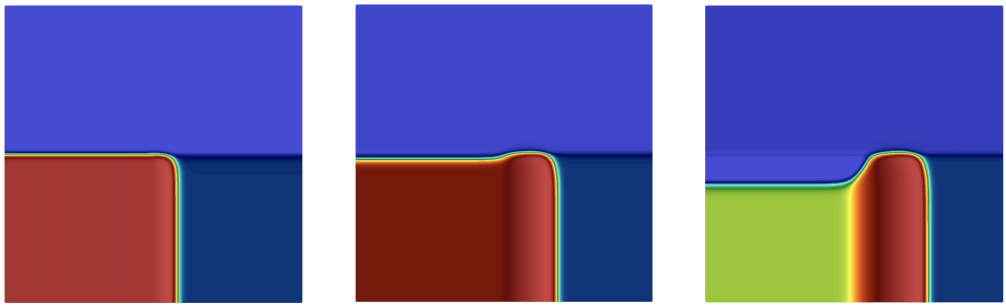


Figure 3.9: From left to right, three different time consecutive captures of the initial condition square moving to the right and vanishing into a vertical structure that will originate the spiral shape. The simulation is produced for final time $T = 200$.

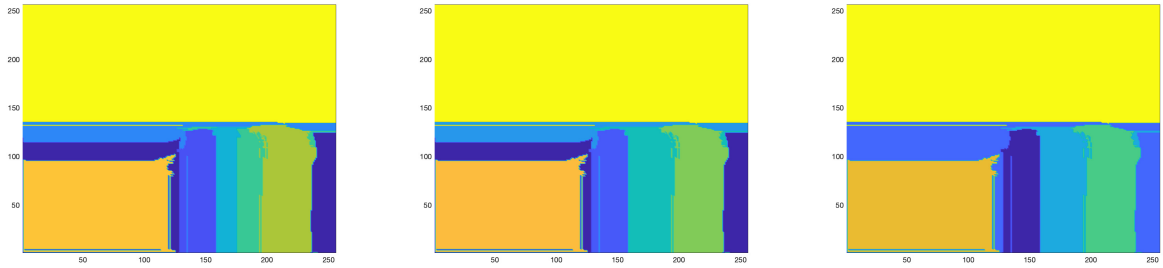


Figure 3.10: Three different consecutive partitions of the spatial domain. Each partition is colored different. From left to right, the number of color blocks is reduced by one. Plots for $T = 200$. These partitions show the domains that represent the sequence of Figure 3.9

see that the method is capable of capturing the behavior of the system and adapting the partitions made to it.

In Table 3.2 and Table 3.3 a summary of the memory compression for the obtained best number of divisions of the space is included. In Table 3.2 we start by changing the final time of the experiment, $T = \{200, 400, 800\}$ and fixing the time step $n_t = 1$ in such a way that the degrees of freedom of the time variable change with the final time.

In the following table, Table 3.3, let us see how the values for the memory have changed if we change the

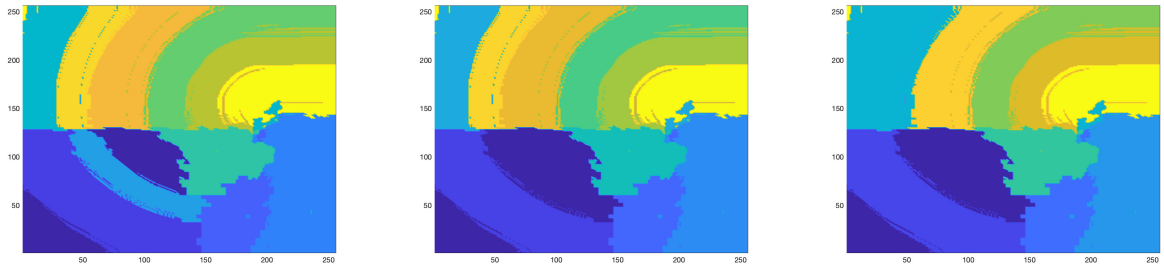


Figure 3.11: Three different consecutive partitions of the spatial domain. Each partition is colored differently. From left to right, the number of color blocks is reduced by one. Plots for $T = 800$.

T	ϵ	Best Memory	Mem POD/ Best Mem	Best # clusters (x-t)
200	$5 \cdot 10^{-2}$	239082	3,30	1-19
	$1 \cdot 10^{-2}$	305741	5,59	1-20
	$5 \cdot 10^{-3}$	315965	6,45	1-18
	$1 \cdot 10^{-3}$	459811	6,43	1-30
	$5 \cdot 10^{-4}$	497865	6,87	1-70
	$1 \cdot 10^{-4}$	604648	7,28	1-42
400	$5 \cdot 10^{-2}$	204283	2,16	1-29
	$1 \cdot 10^{-2}$	483252	6,55	1-42
	$5 \cdot 10^{-3}$	535816	7,14	1-30
	$1 \cdot 10^{-3}$	697919	8,03	1-44
	$5 \cdot 10^{-4}$	884585	7,23	1-30
	$1 \cdot 10^{-4}$	1017435	8,17	1-69
800	$5 \cdot 10^{-2}$	1605626	2,02	1-28
	$1 \cdot 10^{-2}$	2410667	2,96	1-60
	$5 \cdot 10^{-3}$	2450132	3,46	1-60
	$1 \cdot 10^{-3}$	3367752	3,51	1-68
	$5 \cdot 10^{-4}$	2952861	5,22	1-50
	$1 \cdot 10^{-4}$	3022160	6,15	1-50

Table 3.2: Comparison between the memory obtained when computing the POD and the memory computed for the best of the possible partitions of the domain for different values of the error tolerance and different resolutions in the domain. In this case the final time $T = \mathcal{N}_t$, i.e. what we are reconstructing are parts of the whole simulation with a times tep $n_t = 1$. The resolution in x, is $\mathcal{N}_x = 256 \times 256$.

resolution in time. Let us fix the final time $T = 800$ and let us change the time step $n_t = \{4, 2, 1\}$ in a way that the degrees of freedom in this variable also change as $\mathcal{N}_t = \{200, 400, 800\}$, respectively.

As we can see in the previous results, the compression rate with respect to the HOSVD is notorious. Starting from a factor two of compression and reaching so far a factor eight in some cases. The results are very optimistic. When the prescribed tolerance decreases, more costly the HOSVD becomes, and the advantage of the use of the local approximation method arises. The number of optimal partitions of the domain is augmenting when decreasing the tolerance for the same resolution what was expected because the number of partitions obtained before starting the merging phase was also larger.

3.8 Conclusions and perspectives

In this chapter a method that computes the local HOSVD approximation of a tensor is introduced. This is done by computing a HOSVD approximation of the tensor restricted to local subdomains, obtained by dividing the whole domain. A key point of the method is how to compute the subdomain in which the function is well approximated. What we propose in this work is that it is obtained automatically by applying

$(\mathcal{N}_x, \mathcal{N}_t)$	ϵ	Best Memory	Mem POD/ Best Mem	Best # clusters (x-t)
$(256 \times 256{,}200)$	$5 \cdot 10^{-2}$	1258991	2,51	1-46
	$1 \cdot 10^{-2}$	1431429	3,48	1-50
	$5 \cdot 10^{-3}$	1571165	4,98	1-50
	$1 \cdot 10^{-3}$	2334709	4,67	1-50
	$5 \cdot 10^{-4}$	2424712	4,56	1-50
	$1 \cdot 10^{-4}$	3204997	4,02	1-50
$(256 \times 256{,}400)$	$5 \cdot 10^{-2}$	1392122	2,27	1-30
	$1 \cdot 10^{-2}$	1528037	3,15	1-30
	$5 \cdot 10^{-3}$	1859227	4,27	1-50
	$1 \cdot 10^{-3}$	1893692	6,16	1-50
	$5 \cdot 10^{-4}$	1942698	6,07	1-80
	$1 \cdot 10^{-4}$	3915820	4,39	1-70
$(256 \times 256{,}800)$	$5 \cdot 10^{-2}$	1605626	2,02	1-28
	$1 \cdot 10^{-2}$	2410667	2,96	1-60
	$5 \cdot 10^{-3}$	2450132	3,46	1-60
	$1 \cdot 10^{-3}$	3367752	3,51	1-68
	$5 \cdot 10^{-4}$	2952861	5,22	1-70
	$1 \cdot 10^{-4}$	3022160	6,15	1-70

Table 3.3: Comparison between the memory obtained when computing the POD and the memory computed for the best of the possible partitions of the domain for different values of the error tolerance and different resolutions in the domain. In this case the final time $T = 800$ and the resolution $\mathcal{N}_t = \{200, 400, 800\}$, i.e. what we are reconstructing is the whole simulation changing the time step.

a clustering method independently on each direction. This can be done due to the separability of the domain and it provides a certain partition of the domains on each direction.

In this work a first approach is obtained by computing all the possible partitions of subdomains and keeping the one that provides the most compressed approximation. The main negative aspect of the proposed method is that the cost of the algorithm is very high, even though the numerical tests have been restricted to moderate order tensors. In further works a less expensive approach to the current method will be studied and eventually a good trade-off between memory and computational cost will be studied.

From the numerical results of the tests computed, we can infer that the partitions of the domain obtained with the clustering method are able to provide a certified approximation by spending less memory. This aspect increases notoriously when the tensor we want to approximate presents different regimes.

Chapter 4

Deep Learning-based schemes

Contents

4.1 Introduction 83
4.1.1 Organization of the chapter 85
4.2 A singularly perturbed convection-diffusion equation 85
4.2.1 Problem definition 85
4.2.2 General formulation 86
4.2.3 Vanilla (V) formulation 86
4.2.4 Weak variational (W) formulation 87
4.2.5 Rescaled formulation 88
4.2.6 Summary of the methods 89
4.3 Neural networks based numerical schemes 89
4.3.1 General principle 90
4.3.2 Neural Network classes of functions 90
4.3.3 Sampling schemes 91
4.3.4 Comparison with finite element schemes 92
4.4 Numerical Results 93
4.4.1 Test case and comparison criteria 93
4.4.2 Our code and practical implementation details 94
4.4.3 Discussion 94
4.4.4 Conclusions from the numerical experiments 95
4.5 Future research directions and extensions 97

4.1 Introduction

Singularly perturbed differential equations are typically characterized by a small parameter $\epsilon > 0$ multiplying some of the highest order terms in the differential equation. In general, the solutions to such equations exhibit multiscale phenomena, and this raises significant challenges to classical numerical methods such as finite elements or finite volumes. To build accurate and robust approximations with these methods as ϵ decreases, it is necessary to develop elaborate numerical discretizations. In addition to the mathematical difficulties of the formulation, the resulting numerical schemes are often not entirely trivial to implement: they often require mesh adaptation, and working on complicated geometries is challenging. These difficulties motivate the search for new discretization schemes, hopefully mesh-free, with potential to deliver good quality approximations with easier implementation techniques. In this chapter, we explore this research direction, and consider strategies based on deep learning techniques. Our main goal is to test various neural network-based schemes, so as to design a strategy which should be robust when $\epsilon \to 0$, easily implementable even for complicated geometries, and with potential to scale in high dimension.

The idea of working with neural network functions to solve PDEs is by far not novel, and countless contributions have been proposed on this front in recent years. The strategies can be roughly classified into two categories:

1. In the first category, deep neural networks are employed to assist classical numerical methods by improving some limitations, or accelerating certain steps (see, e.g., [272, 273]). It has notably been used to assist in the construction of numerical fluxes adapted to Finite Volumes (see, e.g., [274]). They can also be used in order to compute reduced-order models of parametric problems: for each value of the parameters, the solution (or other quantities of interest) of the model of interest is computed by means of a standard numerical scheme, and the values of the solutions are interpolated by mean of a deep neural network over the whole range of parameter values [275, 276].

2. In the second category, neural networks are used to directly approximate the solution of PDEs. The solution schemes become in this case an optimization problem where it is crucial to design appropriate loss functions. The loss functions are mostly based on residuals of the equations, and yield to different methods depending on the specific choice:

   (a) Physics-informed neural networks (PINNs, [206]) is a collocation-based method. One finds the coefficients of the neural network solution by minimizing a discretized version of the L^2 norm of the strong form of the residual of the PDE. This method is very easily implementable but it implicitly assumes that solutions are very regular.

   (b) Other strategies leverage weak variational formulations where less regular solutions are allowed. On this front, most of the classical methods originally formulated for piecewise polynomial functions have by now been tested with trial and test spaces of neural network functions. In this respect, the deep Galerkin method (DGM, [222]) is based on a least-squares formulation, and the variational PINNS (vPINNs, [209, 210]) is based on the Galerkin method. The main drawback of this approach is that the approximation quality depends on the architecture of both the trial and the test neural network classes. In addition, numerous evaluations for multiple test functions need to be performed. Also, strategies involving the minimization of weak-form residuals are usually not trivial to implement because they involve the computation of norms in very weak spaces which necessitate extra discretization steps.

   (c) Another approach based on weak variational formulations is the so-called deep Ritz method (DRM, [215]). It leverages the fact that the solution of certain PDEs is the unique minimizer to a certain energy functional. When possible, this approach seems the most appealing: the loss function is naturally given by the problem, it can accommodate low regular solutions, and the computational cost is moderate in the sense that it only requires to handle test functions (no trial functions). It also carries potential to address high dimensional problem as illustrated in [214, 215, 216].

The goal of this part of the work is to develop and compare several neural network schemes for singularly perturbed problems when $\epsilon \to 0$. We focus more particularly on convection-diffusion (or stationary Fokker-Planck) problems with vanishing diffusion for which we explore schemes from the second category according to the above distinction. In other words, we approximate solutions of singular PDEs with feed-forward neural network functions. When $\epsilon \to 0$, the regularity of the solutions is deteriorated because of local or boundary thin layers.

The classical formulation commonly used in neural network based schemes is constructed from the strong formulation of the problem, where the analytical solution is approximated by the one generated by evaluations on the sampling points. It is referred in the following as vanilla PINNs, which has been introduced in the (2)-(a) subcategory on the introduction. More details could be found in Section 4.2.3. Therefore, it is expected to perform poorly for small values of ϵ because it commits a "variational crime" (and this is actually confirmed in our numerical experiments). By "variational crime", we mean that the norm of the residual which is traditionally used in vanilla PINNs methods requires that the solution of the PDE has to be more regular than what would naturally be expected from the theory.

In our case (convection-diffusion problems), the Vanilla-PINN method requires the solution to belong to an H^2 space, whereas it is more natural from a theoretical point of view to look for an approximation of the solution in a set of H^1 functions.

Methods based on weak variational formulations seem better adapted, and on that front, it is desirable to work with the deep Ritz method. However, finding energy formulations is not straightforward due to the

non-symmetric nature of convective effects. We show how this method can be applied in this context thanks to a change of variable. We compare its numerical robustness with respect to the PINNs method, and a naive finite element discretization with a uniform grid. In the present study, our tests are performed on a 1D example. Despite its simplicity, the example exhibits all the features that are challenging for numerical schemes. For our purposes, the example also presents the important advantage of having analytic solutions which we can leverage in our error analysis, and our validations. Higher-dimensional tests involving also more elaborate sampling strategies are left for future work.

4.1.1 Organization of the chapter

This chapter is organized as follows. In Section 4.2, various formulations of the convection-diffusion problem we are interested in are introduced. In Section 4.3, we introduce various neural networks-based schemes which are inspired from the various formulations introduced in Section 4.2. The reader is encouraged to observe that an expert mathematical insight is required in order to build formulations that do not incur in variational crimes. In Section 4.4, these various schemes are compared for one-dimensional problem. To conclude, in Section 4.4.4, after the presentation of the numerical results, it is summarized how each one of the PINN methods behave for small values of ϵ and its comparison with the FEM method.

4.2 A singularly perturbed convection-diffusion equation

The aim of this section is to introduce the singularly perturbed convection-diffusion equation we consider in this work, and various formulations of the problem which will be used in Section 4.3 so as to design various neural networks-based schemes for its numerical solution.

4.2.1 Problem definition

As a prototypical example, we consider the following singularly perturbed convection-diffusion equation on a given domain $\Omega \subset \mathbb{R}^d$, with $d \in \mathbb{N}^*$. Let $F : \Omega \to \mathbb{R}^d$ be a given force field, $0 < \epsilon \ll 1$ a small parameter, and $f : \Omega \to \mathbb{R}$ be a given right-hand side function. Our goal is to find a solution $u : \Omega \to \mathbb{R}$ to

$$-\epsilon(\Delta u)(x) + \nabla \cdot (Fu)(x) = f(x), \quad \forall x \in \Omega, \tag{4.1}$$

together with Robin boundary conditions

$$\alpha(\nabla u \cdot n)(x) + \kappa u(x) = g(x) \quad \forall x \in \partial\Omega, \tag{4.2}$$

where n refers to the outward unit vector of $\partial\Omega$, α, $\kappa \geq 0$ and g is a real-valued function defined on $\partial\Omega$. In the following, we assume that the force field F derives from a potential function $V : \Omega \to \mathbb{R}$, in the sense that

$$F(x) = -\nabla V(x), \quad \forall x \in \Omega.$$

Under appropriate assumptions on F (or V), f and g, which are assumed to be smooth functions for the sake of simplicity, problem (4.1)-(4.2) can be proved to have a unique solution [227, 228, 229]. Note that, more generally, α and κ could also be given as real-valued functions defined on $\partial\Omega$, instead of constants, and our subsequent developments could be easily adapted.

The equation represents the change in the concentration u of a quantity in a given medium in presence of convective and diffusive effects. The force field F represents the drag force while the singular perturbation parameter ϵ represents the diffusivity of the medium. In the limit of an inviscid medium as $\epsilon \to 0$, the equation changes from elliptic to hyperbolic nature, and from second to first order. For Dirichlet boundary conditions $u = 0$ on $\partial\Omega$, the solution can develop sharp boundary layers of width ϵ near the outflow. We refer the reader to [277] for general references on this equation regarding its analysis and numerical methods.

Classical numerical methods are challenged by problem (4.1) when ϵ is small. In the case of the Galerkin finite element method, the poor performance for this problem is reflected in the bound on the error in the finite element solution. For $\Omega = (0, 1)$ and Dirichlet boundary conditions, a standard Galerkin method with a uniform grid of size h delivers a solution u_h on a finite element space $\mathbb{P}_h$ that satisfies

$$\|u - u_h\|_{H^1(0,1)} \leq C(\epsilon) \inf_{w_h \in \mathbb{P}_h} \|u - w_h\|_{H^1(0,1)}, \tag{4.3}$$

where $C(\epsilon) \sim \epsilon^{-1}$, so that the constant blows up as $\epsilon \to 0$ (see [277, Theorem 2.49]). The dependence of C on ϵ is usually referred to as a loss of robustness in the sense that, as ϵ decreases, the Galerkin method is bounded more and more loosely by the best approximation error. As a consequence, on a coarse mesh and for small values ϵ, the Galerkin approximation develops spurious oscillations everywhere in the domain. This very well-known behavior will actually be observed later on in our numerical tests.

Numerous methods have been proposed in order to address this loss of robustness in finite element methods. An important family of methods is based on using residual-based stabilization techniques. Given some variational form, the problem is modified by adding to the bilinear form the strong form of the residual, weighted by a test function and scaled by a stabilization constant τ. The most well-known example of this technique is the streamline upwind Petrov-Galerkin (SUPG) method (see [278]). The addition of the residual-based stabilization term, can be interpreted as a modification of the test functions which means that these methods seek stabilization by changing the test space, and motivates to search for optimal test spaces in the spirit of [279, 280].

Other classical discretization methods such as finite volumes suffer from similar issues, and strategies involving layer-adaptive grids such as Shishkin meshes have been proposed (see, e.g., [281]).

The aim of this work is to explore the potential of approximating solutions of such problems with neural network functions, and the next section presents several options for this, with a discussion on their merits and limitations.

4.2.2 General formulation

Any neural-network based numerical scheme for the solution of (4.1)-(4.2) relies on the use of a variational formulation of this problem which enables to write u (or another function defined from u) as a minimizer of a problem of the form

$$\min_{v \in \mathcal{V}} \mathcal{J}(v), \tag{4.4}$$

where $\mathcal{V}$ is a particular set of real-valued functions defined on Ω. The loss function $\mathcal{J} : \mathcal{V} \to \mathbb{R}$ is usually of the form

$$\mathcal{J}(v) := \int_\Omega \mathcal{R}(v)(x) d\rho(x) + \int_{\partial\Omega} \mathcal{S}(v)(r) d\tau(r), \quad \forall v \in \mathcal{V}, \tag{4.5}$$

where for every $v \in \mathcal{V}$, $\mathcal{R}(v)$ and $\mathcal{S}(v)$ are real-valued functions defined on Ω and $\partial\Omega$ respectively. They are assumed to be integrable with respect to the measures ρ and τ, which are defined on Ω and $\partial\Omega$ respectively. Note that the measures ρ and τ have to be chosen a priori, and they can greatly affect the final result. The question of discovering the optimal weights is beyond the scope of our present investigation.

The aim of the next sections is to introduce various formulations of problem (4.1)-(4.2) under the form (4.4)-(4.5). This requires appropriate definitions of the set $\mathcal{V}$, the functions $\mathcal{R}(v)$ and $\mathcal{S}(v)$ for any $v \in \mathcal{V}$ and the unknown function solution of (4.4). Unless otherwise stated, the measures ρ and τ will be defined as the Lebesgue bulk measure and Lebesgue surface measure respectively.

4.2.3 Vanilla (V) formulation

We begin by introducing the most classical formulation used in neural network-based numerical schemes such as PINNs. For the reasons that we outline next, different aspects of this formulation can be improved, therefore we refer to it as *vanilla* (V) formulation in the following.

The formulation consists in interpreting the solution u of (4.1)-(4.2) as the unique solution of a minimization problem of the form (4.4) with $\mathcal{V} = H^2(\Omega)$ and to define for all $v \in \mathcal{V}$,

$$\begin{cases} \mathcal{R}(v)(x) := \lambda \left| -\epsilon(\Delta v)(x) + \nabla \cdot (Fv)(x) - f(x) \right|^2, & \text{for all } x \in \Omega, \\ \mathcal{S}(v)(x) := (1-\lambda) \left| \alpha(\nabla v \cdot n)(x) + \kappa v(x) - g(x) \right|^2, & \text{for all } x \in \partial\Omega, \end{cases} \tag{4.6}$$

for some $\lambda \in (0, 1)$. In this approach, the parameter λ enables to tune the respective weight of the contributions of the bulk and boundary terms in the total functional $\mathcal{J}$ to be minimized. In practice, in the numerical tests presented in Section 4.4, λ will always be chosen to be equal to $\frac{1}{2}$.

Note that such an approach requires the solution u to belong to $H^2(\Omega)$, which implies that the solution has to be sufficiently regular. When $\epsilon \to 0$, this assumption becomes less and less realistic due to the formation of boundary layers. This raises the question as to whether it is possible to introduce another formulation of problem (4.1)-(4.2) which would allow for less regular solutions. The goal of the next section is to introduce such an alternative formulation.

4.2.4 Weak variational (W) formulation

In this section we develop an avenue based on an energy minimization approach which requires less regularity in the solutions than the vanilla formulation. To this aim, we introduce the change of variable

$$u(x) = e^{cV(x)} z(x), \tag{4.7}$$

where $c \in \mathbb{R}$ is a constant yet to be determined. Taking first and second derivatives in (4.7) yield that for all $x \in \Omega$,

$$\begin{aligned}
\nabla u(x) &= e^{cV(x)} \left(c\nabla V(x) z(x) + \nabla z(x) \right) \\
\Delta u(x) &= e^{cV(x)} \left(c\Delta V(x) z(x) + |c\nabla V(x)|^2 z(x) + 2c\nabla V(x) \cdot \nabla z(x) + \Delta z(x) \right).
\end{aligned}$$

Now, setting the value of c to be

$$c = \frac{1}{2\epsilon},$$

and inserting the change of variable into (4.1), we conclude that u is a solution to (4.1) if and only if z is a solution to the elliptic problem

$$-\Delta z(x) + \left(\frac{\Delta V(x)}{2\epsilon} + \frac{|\nabla V(x)|^2}{4\epsilon^2} \right) z(x) = f(x) \frac{e^{-\frac{V(x)}{2\epsilon}}}{\epsilon}, \quad \forall x \in \Omega, \tag{4.8}$$

with Robin boundary conditions

$$\alpha(\nabla z(x) \cdot n(x)) + \left(\kappa + \frac{\alpha}{2\epsilon} \nabla V(x) \cdot n(x) \right) z(x) = e^{\frac{-V(x)}{2\epsilon}} g(x), \quad \forall x \in \partial\Omega. \tag{4.9}$$

At this stage, one could of course apply the vanilla formulation to solve (4.8)-(4.9) and compute z solution of a minimization problem of the form (4.4) with $\mathcal{V} = H^2(\Omega)$ and the functionals $\mathcal{R}$ and $\mathcal{S}$ defined by

$$\begin{cases}
\mathcal{R}(v)(x) := \lambda \left| \Delta v(x) + \left(\frac{\Delta V(x)}{2\epsilon} + \frac{|\nabla V(x)|^2}{4\epsilon^2} \right) v(x) - f(x) \frac{e^{-\frac{V(x)}{2\epsilon}}}{\epsilon} \right|^2, & \text{for all } x \in \Omega, \\
\mathcal{S}(v)(x) := (1-\lambda) \left| \alpha(\nabla v(x) \cdot n(x)) + \left(\kappa + \frac{\alpha}{2\epsilon} \nabla V(x) \cdot n(x) \right) v(x) - e^{\frac{-V(x)}{2\epsilon}} g(x) \right|^2, & \text{for all } x \in \partial\Omega,
\end{cases} \tag{4.10}$$

for all $v \in \mathcal{V} = H^2(\Omega)$ and some $\lambda \in (0,1)$. The value of λ chosen in our numerical tests is $\lambda = 0.5$. We will refer to this approach as the *vanilla-z* (Vz) formulation.

Note that this method does not fully exploit the change of variables since the elliptic nature of problem (4.8) allows us to easily build a weak formulation of this equation. Testing against a smooth test function v and integrating by parts we obtain the weak formulation

$$\int_\Omega \nabla z \cdot \nabla v - \int_{\partial\Omega} v \nabla z \cdot n + \int_\Omega \left(\frac{\Delta V}{2\epsilon} + \frac{|\nabla V|^2}{4\epsilon^2} \right) zv = \int_\Omega f e^{\frac{V}{2\epsilon}} \frac{v}{\epsilon}$$

Using equality (4.9), we get

$$\begin{aligned}
\int_\Omega \nabla z \cdot \nabla v + \int_\Omega \left(\frac{\Delta V}{2\epsilon} + \frac{|\nabla V|^2}{4\epsilon^2} \right) zv + \int_{\partial\Omega} \left(\frac{\kappa}{\alpha} + \frac{1}{2\epsilon} \nabla V \cdot n \right) zv \\
= \int_\Omega f \frac{e^{-\frac{V}{2\epsilon}}}{\epsilon} v + \int_{\partial\Omega} \frac{1}{\alpha} e^{\frac{-V}{2\epsilon}} gv.
\end{aligned}$$

Therefore the weak formulation of problem (4.8) is to find $z \in H^1(\Omega)$ such that

$$a(z,v) = \ell(v), \quad \forall v \in H^1(\Omega) \tag{4.11}$$

with

$$a(z,v) = \int_\Omega \nabla z \cdot \nabla v + \int_\Omega \left(\frac{\Delta V}{2\epsilon} + \frac{|\nabla V|^2}{4\epsilon^2} \right) zv + \int_{\partial\Omega} \left(\frac{\kappa}{\alpha} + \frac{1}{2\epsilon} \nabla V \cdot n \right) zv$$

$$l(v) = \int_\Omega f \frac{e^{-\frac{V}{2\epsilon}}}{\epsilon} v + \int_{\partial\Omega} \frac{1}{\alpha} e^{\frac{-V}{2\epsilon}} gv.$$

To ensure that the symmetric bilinear form a is continuous and coercive, we assume in the sequel that the following conditions are satisfied:

$$\begin{cases} \left(\frac{\Delta V(x)}{2\epsilon} + \frac{|\nabla V(x)|^2}{4\epsilon^2} \right) \geq a_0 > 0, & \forall x \in \Omega \\ \left(\frac{\kappa}{\alpha} + \frac{1}{2\epsilon} \nabla V(x) \cdot n(x) \right) \geq 0, & \forall x \in \partial\Omega \\ f \in L^2(\Omega),\ g \in L^2(\partial\Omega) \end{cases} \tag{4.12}$$

In that case, all the hypothesis of the Lax-Milgram theorem are satisfied for a and ℓ. Notice that, if V is a regular confining potential (i.e. if $\nabla V(x) \cdot n(x) \geq 0$ for all $x \in \partial\Omega$), then there exists $\epsilon_0 > 0$ such that conditions (4.12) are satisfied for all $\epsilon \leq \epsilon_0$.

If conditions (4.12) are satisfied, z can be equivalently rewritten as the unique solution of a minimization problem of the form

$$z = \underset{v \in H^1(\Omega)}{\operatorname{argmin}} \frac{1}{2} a(v,v) - \ell(v). \tag{4.13}$$

This implies that z can equivalently be recast as the unique solution of a minimization problem of the form (4.4) with $\mathcal{V} = H^1(\Omega)$ and

$$\begin{cases} \mathcal{R}(v)(x) = \frac{1}{2} \left[|\nabla v(x)|^2 + \left(\frac{\Delta V(x)}{2\epsilon} + \frac{|\nabla V(x)|^2}{4\epsilon^2} \right) |v(x)|^2 \right] - f(x) \frac{e^{-\frac{V(x)}{2\epsilon}}}{\epsilon} v(x), \quad \forall x \in \Omega, \\ \mathcal{S}(v)(x) = \frac{1}{2} \left[\left(\frac{\kappa}{\alpha} + \frac{1}{2\epsilon} \nabla V(x) \cdot n(x) \right) |v(x)|^2 \right] - \frac{1}{\alpha} e^{\frac{-V(x)}{2\epsilon}} g(x) v(x), \quad \forall x \in \partial\Omega. \end{cases} \tag{4.14}$$

We will refer to this approach as the *weak-z* (Wz) formulation.

Moreover, using (4.7), we can equivalently express u as a solution of a minimization problem of the form (4.4) with

$$\mathcal{V} := \left\{ v = e^{\frac{V}{2\epsilon}} \overline{v},\ \overline{v} \in H^1(\Omega) \right\}, \tag{4.15}$$

and rewriting the Equation (4.14)

$$\begin{cases} \mathcal{R}(v)(x) = \frac{1}{2} \left[|\nabla \overline{v}(x)|^2 + \left(\frac{\Delta V(x)}{2\epsilon} + \frac{|\nabla V(x)|^2}{4\epsilon^2} \right) |\overline{v}(x)|^2 \right] - f(x) \frac{e^{-\frac{V(x)}{2\epsilon}}}{\epsilon} \overline{v}(x), \quad \forall x \in \Omega, \\ \mathcal{S}(v)(x) = \frac{1}{2} \left[\left(\frac{\kappa}{\alpha} + \frac{1}{2\epsilon} \nabla V(x) \cdot n(x) \right) |\overline{v}(x)|^2 \right] - \frac{1}{\alpha} e^{\frac{-V(x)}{2\epsilon}} g(x) \overline{v}(x), \quad \forall x \in \partial\Omega, \end{cases} \tag{4.16}$$

for all $v \in \mathcal{V}$ with $\overline{v} := v e^{-\frac{V}{2\epsilon}}$ the change of variable suggested above. We will refer to this formulation as the *weak* (W) formulation.

Note that in the non-discretized case, formulations (W) and (W-z) are equivalent up to the exponential change of variable. However, when the minimizer of both formulations is computed by means of a neural network, the corresponding approximations will be different and have different accuracies. The (W) formulation has the advantage to avoid potential machine precision issues linked to the presence of the exponential term when ϵ becomes small.

4.2.5 Rescaled formulation

In this section, we introduce another formulation based on a change of scale in the original problem. More precisely, introducing $\Omega_\epsilon := \frac{1}{\epsilon}\Omega$, we introduce auxiliary functions $\widetilde{u} : \Omega_\epsilon \to \mathbb{R}$, $\widetilde{z} : \Omega_\epsilon \to \mathbb{R}$ and $\widetilde{V} : \Omega_\epsilon \to \mathbb{R}$ defined so that for all $x \in \Omega$,

$$u(x) = \epsilon \widetilde{u}\left(\frac{x}{\epsilon}\right), \quad z(x) = \epsilon \widetilde{z}\left(\frac{x}{\epsilon}\right), \quad V(x) = \epsilon \widetilde{V}\left(\frac{x}{\epsilon}\right). \tag{4.17}$$

Notice that if u and z satisfy (4.7), then

$$\widetilde{z}(y) = \widetilde{u}(y) e^{\frac{1}{2}\widetilde{V}(y)}, \quad \forall y \in \Omega_\epsilon.$$

Denoting by $\widetilde{F}(y) := -\nabla\widetilde{V}(y) = F(\epsilon y)$ for all $y \in \Omega_\epsilon$, it holds that u is solution to (4.1)-(4.2) if and only if $\widetilde{u}$ is solution to

$$-\Delta\widetilde{u}(y) + \nabla\cdot\left(\widetilde{F}\widetilde{u}\right)(y) = \widetilde{f}(y), \quad \forall y \in \Omega_\epsilon \tag{4.18}$$

where $\widetilde{f}(y) := f(\epsilon y)$ for all $y \in \Omega_\epsilon$ with boundary conditions

$$\alpha(\nabla\widetilde{u}\cdot n)(y) + \epsilon\kappa\widetilde{u}(y) = \widetilde{g}(y), \quad \forall y \in \partial\Omega_\epsilon, \tag{4.19}$$

with $\widetilde{g}(y) := g(\epsilon y)$ for all $y \in \Omega_\epsilon$.

Using similar calculations to the ones done in Section 4.2.4, the Lax-Milgram theorem guarantees that $\widetilde{z}$ is the unique solution in $H^1(\Omega_\epsilon)$ of the following variational problem: for all $\widetilde{v} \in H^1(\Omega_\epsilon)$,

$$\int_{\Omega_\epsilon}\nabla\widetilde{z}(y)\cdot\nabla\widetilde{v}(y)\,dy + \int_{\partial\Omega_\epsilon}\left(\frac{\epsilon\kappa}{\alpha} + \frac{1}{2}\nabla\tilde{V}(y)\cdot n(y)\right)\widetilde{z}(y)\widetilde{v}(y)\,dy + \int_{\Omega_\epsilon}\left(\frac{\Delta\tilde{V}(y)}{2} + \frac{|\nabla\tilde{V}(y)|^2}{4}\right)\widetilde{z}(y)\,\widetilde{v}(y)\,dy = \tag{4.20}$$

$$= \int_{\Omega_\epsilon}\widetilde{f}(y)e^{-\frac{1}{2}\tilde{V}(y)}\widetilde{v}(y)\,dy + \int_{\partial\Omega_\epsilon}\frac{1}{\alpha}e^{-\frac{1}{2}\tilde{V}(y)}\widetilde{g}(y)\widetilde{v}(y)dy.$$

The result is valid provided that the following assumptions on the coefficients hold

$$\begin{cases}\Delta\tilde{V}(y) + |\nabla\tilde{V}|^2(y) \geq 0, \ \forall y \in \Omega_\epsilon,\\ \frac{\kappa\epsilon}{\alpha} + \frac{1}{2}\nabla\tilde{V}(y)\cdot n(y) \geq 0, \ \forall y \in \partial\Omega_\epsilon,\\ \widetilde{f} \in L^2(\Omega_\epsilon), \ \widetilde{g} \in L^2(\partial\Omega_\epsilon).\end{cases}$$

The above conditions are equivalent to the assumptions (4.12) stated in Section 4.2.4.

As in the previous section, the function $\widetilde{z}$ is then the unique solution of a minimization problem of the form (4.4) with $\mathcal{V} = H^1(\Omega_\epsilon)$ with

$$\begin{cases}\mathcal{R}(v)(y) = \frac{1}{2}\left[|\nabla v(y)|^2 + \left(\frac{\Delta\widetilde{V}(y)}{2} + \frac{|\nabla\widetilde{V}(y)|^2}{4}\right)|v(y)|^2\right] - \widetilde{f}(y)e^{-\frac{\widetilde{V}(y)}{2}}v(y), \quad \forall y \in \Omega_\epsilon,\\ \mathcal{S}(v)(y) = \frac{1}{2}\left[\left(\frac{\epsilon\kappa}{\alpha} + \frac{1}{2}\nabla\widetilde{V}(y)\cdot n(y)\right)|v(y)|^2\right] - \frac{1}{\alpha}e^{\frac{-\widetilde{V}(y)}{2}}\widetilde{g}(y)v(y), \quad \forall y \in \partial\Omega_\epsilon.\end{cases} \tag{4.21}$$

We will refer to this approach as the *rescaled-weak-z* (RWz) formulation.

4.2.6 Summary of the methods

For the sake of clarity, we summarize here the main features of each method.

Method	Acronym	Unknown	$\mathcal{V}$	$\mathcal{R}$ and $\mathcal{S}$
vanilla	V	u	$H^2(\Omega)$	(4.6)
vanilla-z	Vz	v	$H^2(\Omega)$	(4.10)
weak-z	Wz	z	$H^1(\Omega)$	(4.14)
weak	W	u	(4.15)	(4.16)
rescaled-weak-z	RWz	$\widetilde{z}$	$H^1(\Omega_\epsilon)$	(4.21)

4.3 Neural networks based numerical schemes

In this section we describe the numerical approach used in order to compute an approximation of the solution of a minimization problem of the form (4.4) by means of a neural-network based method. We first present in Section 4.3.1 the general principle of such approaches. The main ingredients to design a neural-network based method consist in the choice of a class of neural network functions and of sampling schemes in order to approximate the integrals involved in the definition of the loss function $\mathcal{J}$ defined by (4.5). These two ingredients are detailed respectively in Section 4.3.2 and Section 4.3.3. Finally, some details on the numerical implementation are given in Section 4.4.2.

4.3.1 General principle

The numerical solution of a minimization problem of the form (4.4) usually requires to consider alternatives to $\mathcal{V}$ and $\mathcal{J}$ that are amenable for practical implementation. The strategy thus consists in formulating a related problem of the form

$$\min_{v \in \mathcal{K}} \widehat{\mathcal{J}}(v), \tag{4.22}$$

where

- $\mathcal{K} \subset \mathcal{V}$ is a set of functions parametrized by a finite number of scalar coefficients. A classical class of functions are finite elements. Here, we consider neural networks (see Section 4.3.2 below);
- $\widehat{\mathcal{J}}$ is an approximation of the loss function $\mathcal{J}$ where the integrals are approximated using some particular quadrature or sampling schemes.

More precisely, for given integers $K, M \in \mathbb{N}^*$, given sets of points $(x_k)_{1\leq k\leq K} \subset \Omega$, $(y_m)_{1\leq m\leq M} \subset \partial\Omega$, and given sets of weights $(\rho_k)_{1\leq k\leq K} \subset \mathbb{R}$ and $(\tau_m)_{1\leq m\leq M} \subset \mathbb{R}$, for all $v \in \mathcal{K}$, the functional $\widehat{\mathcal{J}}(v)$ is defined by

$$\widehat{\mathcal{J}}(v) := \sum_{k=1}^{K} \rho_k \mathcal{R}(v)(x_k) + \sum_{m=1}^{M} \tau_m \mathcal{S}(v)(y_m). \tag{4.23}$$

As a consequence, the definition of a neural-network based numerical scheme for the approximation of a problem of the form (4.4) requires the definition of two ingredients:

- the class $\mathcal{K} \subset \mathcal{V}$ of neural network functions;
- the sampling scheme, i.e. the choice of K, M, $(x_k)_{1\leq k\leq K}$, $(y_m)_{1\leq m\leq M}$, $(\rho_k)_{1\leq k\leq K}$ and $(\tau_m)_{1\leq m\leq M}$ in order to define the approximate functional $\widehat{\mathcal{J}}$ given by (4.23).

The set of neural network functions $\mathcal{K}$ we consider in our numerical experiments is presented in Section 4.3.2. The various sampling schemes tested here are given in Section 4.3.3.

4.3.2 Neural Network classes of functions

In this work, we only consider classes of functions defined by means of feedforward neural networks whose definition was already introduced and we recall next (see [202] for more detailed definitions).

Let $\mathcal{X} \subset \mathbb{R}^{d_\mathcal{X}}$ and $\mathcal{Y} \subset \mathbb{R}^{d_\mathcal{Y}}$ be some input and output sets of finite dimensions $d_\mathcal{X}, d_\mathcal{Y} \in \mathbb{N}^*$. A feedforward neural network is a function

$$\psi : \mathcal{X} \to \mathcal{Y}$$

which reads as

$$\psi(x) = T_L(\sigma(T_{L-1}(\sigma(\dots\sigma(T_0(x))))), \quad \forall x \in \mathcal{X}. \tag{4.24}$$

Recalling Equation 1.13, for every $\ell \in \{0, \dots, L\}$,

$$T_\ell : \left\{ \begin{array}{ccc} \mathbb{R}^{p_\ell} & \to & \mathbb{R}^{p_{\ell+1}} \\ x_\ell & \mapsto & T_\ell(x_\ell) = A_\ell x_\ell + b_\ell \end{array} \right. \tag{4.25}$$

is an affine function which can be expressed through a matrix $A_\ell \in \mathbb{R}^{p_{\ell+1}\times p_\ell}$, an offset vector $b_\ell \in \mathbb{R}^{p_{\ell+1}}$, and the so-called (nonlinear) activation function $\sigma : \mathbb{R} \to \mathbb{R}$. By a slight abuse of notation, for all $p \in \mathbb{N}^*$ and for any vector $w := (w_i)_{1\leq i\leq p} \subset \mathbb{R}^p$, the notation $\sigma(w)$ actually denotes the vector of $\mathbb{R}^p$ with entries $\sigma(w_i)$, that is, $\sigma(w) = (\sigma(w_i))_{i=1}^p$. Note that since ψ maps $\mathcal{X}$ onto $\mathcal{Y}$, it is necessary that $p_0 = d_\mathcal{X}$ and $p_{L+1} = d_\mathcal{Y}$. The layers numbered from 1 to L are the hidden layers of the neural network.

To define a class of feedforward neural networks, we fix an architecture by prescribing a given activation function σ, depth $L \in \mathbb{N}$, and layer widths $\boldsymbol{p} = (p_0, \dots, p_{L+1}) \in (\mathbb{N}^*)^{L+2}$. Once the values of σ, L and $\boldsymbol{p}$ have been chosen, we view the coefficients $(A_\ell, b_\ell)_{0\leq\ell\leq L}$ of the affine mappings $T_0, \cdots, T_L$ as parameters. We gather these coefficients in the vector of parameters

$$\theta = \{(A_\ell, b_\ell)\}_{\ell=0}^{L},$$

and assume that θ takes values in a set

$$\Theta \subseteq \bigtimes_{\ell=0}^{L} \left(\mathbb{R}^{p_\ell \times p_{\ell+1}} \times \mathbb{R}^{p_{\ell+1}}\right).$$

For any $\theta \in \Theta$, we define by $\psi_\theta : \mathcal{X} \to \mathcal{Y}$ the function ψ defined by (4.24) with $\theta = \{(A_\ell, b_\ell)\}_{\ell=0}^{L} \in \Theta$.

The class of neural network functions with architecture $(\sigma, L, \boldsymbol{p})$ and coefficient sets Θ is then defined as

$$\mathcal{N}(\sigma, L, \boldsymbol{p}, \Theta) := \{\psi_\theta : \theta \in \Theta\}$$

In our context, the input and output sets $\mathcal{X}$ and $\mathcal{Y}$ are respectively given by

$$\mathcal{X} = \Omega \text{ (or } \Omega_\epsilon) \quad \text{and} \quad \mathcal{Y} = \mathbb{R},$$

so that $d_{\mathcal{X}} = d$ and $d_{\mathcal{Y}} = 1$. In all the numerical tests presented below, the class $\mathcal{K}$ is chosen as

$$\mathcal{K} := \mathcal{N}(\sigma, L, \boldsymbol{p}, \Theta),$$

with

$$\sigma = \tanh,\ L = 2,\ \boldsymbol{p} = (d, 10, 10, 1) \text{ and } \Theta = \bigtimes_{\ell=0}^{L} \left(\mathbb{R}^{p_\ell \times p_{\ell+1}} \times \mathbb{R}^{p_{\ell+1}}\right).$$

Note that the set $\mathcal{K}$ is then a subset of $\mathcal{V}$ for all the formulations of the convection-diffusion problem we introduced in Section 4.2. Moreover, the solution of the approximate problem (4.22) is equivalent to finding a minimizer $\theta^* \in \Theta$ solution to

$$\min_{\theta \in \Theta} \widehat{\mathcal{J}}(\psi_\theta). \tag{4.26}$$

Remark: In many machine learning applications, the choice of relu activation functions is very common due to its low computational cost when performing evaluation or first order differentiation. However, in our problem, second order derivatives are needed to calculate the loss function. If relu activation functions were used, then the second order derivative terms would be 0, and no good approximation could be learned. This reason motivates our choice of tanh as the activation function. In Figure B.2 some results concerning the choice of the architecture of the network are attached.

4.3.3 Sampling schemes

We detail in this section the various sampling schemes we considered in our numerical tests in order to define the approximate loss function $\widehat{\mathcal{J}}$.

Since we work with one-dimensional examples, we carry the discussion for dimension one. In fact, we consider problem (4.1)-(4.2) with $\Omega = (0, 1)$ so that $\partial\Omega = \{0\} \cup \{1\}$ (and $\partial\Omega_\epsilon = \{0\} \cup \{1/\epsilon\}$). Thus, for all our tests, the domain boundary has $M = 2$ points $y_1 = 0$ and $y_2 = 1$ (or $y_2 = \frac{1}{\epsilon}$ for the RWz method). Taking $\tau_1 = \tau_2 = 1$ for the surface weights, the surface term in (4.23) takes the simple form

$$\sum_{m=1}^{M=2} \tau_m \mathcal{S}(v)(y_m) = \int_{\partial\Omega} \mathcal{S}(v)\, d\tau \quad \forall v \in \mathcal{V} \text{ (or } \int_{\partial\Omega_\epsilon} \mathcal{S}(v)\, d\tau \text{ for the RWz formulation).}$$

We consider three different sampling schemes for the approximation of the bulk term $\int_\Omega \mathcal{R}(v)\, d\rho$:

1. The first choice is a simple *uniform* sampling scheme (labeled $-u$ in our tests). For a given $K \in \mathbb{N}^*$, we set $\rho_k = \frac{1}{K}$ and $(x_k)_{1 \le k \le K}$ as the centers of the intervals given by a uniform discretization grid of the interval $(0, 1)$.

2. The second sampling scheme, called *random* $(-r)$ scheme, consists in choosing $\rho_k = \frac{1}{K}$ and the points $(x_k)_{1 \le k \le K}$ as a collection of random points, identically independently distributed according to the uniform distribution on $(0, 1)$.

3. We lastly consider a third sampling scheme, called *exponential* $(-e)$ scheme, which is specific to the Wz formulation. Recall that in this case, for all $v \in \mathcal{K}$, the expression of $\mathcal{R}(v)$ is given by (4.14), namely

$$\mathcal{R}(v)(x) = \mathcal{R}^{(1)}(v)(x) + \mathcal{R}^{(2)}(v)(x), \quad \forall x \in \Omega$$

with

$$\begin{cases} \mathcal{R}^{(1)}(v)(x) := \frac{1}{2}\left[|\nabla v(x)|^2 + \left(\frac{\Delta V(x)}{2\epsilon} + \frac{|\nabla V(x)|^2}{4\epsilon^2}\right)|v(x)|^2\right] \\ \mathcal{R}^{(2)}(v)(x) := -f(x)\frac{e^{-\frac{V(x)}{2\epsilon}}}{\epsilon}v(x), \quad \forall x \in \Omega. \end{cases}$$

Thus, the bulk integral term:

$$\int_\Omega \mathcal{R}(v)(x)d\rho(x) = \int_\Omega \mathcal{R}^{(1)}(v)(x)d\rho(x) + \int_\Omega \mathcal{R}^{(2)}(v)(x)d\rho(x),$$

and we approximate each component separately as follows. For the first term, we draw a collection of $K_1 \in \mathbb{N}^*$ independent identically distributed (iid) random points $(x_k^{(1)})_{1\leq k\leq K_1}$ from the uniform distribution on $(0,1)$ and for all $1 \leq k \leq K_1$, the weights $\rho_k^{(1)}$ are chosen to be equal to $\frac{1}{K_1}$. For the second term, we draw $K_2 \in \mathbb{N}^*$ iid random points $(x_k^{(2)})_{1\leq k\leq K_2}$ following the probability density

$$\rho^{(2)}(x) := \frac{e^{-\frac{V(x)}{2\epsilon}}}{Z_\epsilon}, \quad x \in \Omega,$$

with

$$Z_\epsilon := \int_\Omega e^{-\frac{V(x)}{2\epsilon}}\,dx.$$

Setting now $\rho_k^{(2)} = \frac{Z_\epsilon}{K_2}$ for all $1 \leq k \leq K_2$, the integral $\int_\Omega \mathcal{R}(v)$ is then approximated by

$$\sum_{k=1}^{K_1}\rho_k^{(1)}\left(\frac{1}{2}\left[|\nabla v(x_k^{(1)})|^2 + \left(\frac{\Delta V(x_k^{(1)})}{2\epsilon} + \frac{|\nabla V(x_k^{(1)})|^2}{4\epsilon^2}\right)|v(x_k^{(1)})|^2\right]\right) - \sum_{k=1}^{K_2}\rho_k^{(2)}\left(f(x_k^{(2)})\frac{1}{\epsilon}v(x_k^{(2)})\right).$$

In the following, we use the notation $-u$ (respectively $-r$ and $-e$), after the name of a formulation, in order to refer to the numerical method obtained by using this formulation, together with a uniform (respectively random or exponential) sampling scheme. For instance, the $V-u$ method refers to the vanilla formulation used in conjunction with a uniform sampling scheme.

Note that to tackle higher-dimensional problems, special sampling techniques such as adaptive Markov-Chain Monte-Carlo or Quasi Monte Carlo would be required.

4.3.4 Comparison with finite element schemes

One important point in the investigation of the merits and limitations of deep learning-based numerical schemes is to understand how they compare with respect to other existing schemes. In our tests, we provide a numerical comparison with a vanilla finite element Galerkin scheme involving a uniform mesh. For the sake of completeness, we briefly recall the main steps of our finite element Galerkin approach.

Integrating the original equation (4.1) against a sufficiently smooth function $v \in \mathcal{C}^\infty(\Omega)$, and integrating by parts, it follows that a weak formulation of problem (4.1) is to find $u \in H^1(\Omega)$ such that

$$a(u,v) = l(v), \quad \forall v \in H^1(\Omega)$$

with

$$a(u,v) = \int_\Omega \nabla u \cdot \nabla v + \epsilon^{-1}\int_\Omega \nabla\cdot(Fu)v + \frac{\kappa}{\alpha}\int_{\partial\Omega} uv \tag{4.27}$$

$$l(v) = \epsilon^{-1}\int_\Omega fv - \int_{\partial\Omega} gv. \tag{4.28}$$

We numerically solve this problem by Galerkin projection. For this, we consider a mesh $(T_n)_{n=1}^N$ of Ω and define the associated $\mathbb{P}_1$ finite element space

$$\mathcal{V}_N := \{v \in \mathscr{C}^0(\mathbb{R}) : \forall 0 \leq s \leq N-1,\ v|_{[x_s,x_{s+1})} \in \mathbb{P}^1([x_s,x_{s+1}])\}$$

with

$$\mathbb{P}^1([x_s,x_{s+1}]) := \{v : [x_s,\ x_{s+1}] \to \mathbb{R},\ v(x) = ax+b,\ (a,b) \in \mathbb{R}^2\}.$$

We then search for a solution $u_N \in \mathcal{V}_N \subset H^1(\Omega)$ by Galerkin projection, that is, we search for $u_N \in \mathcal{V}_N$ such that

$$a(u_N, v) = l(v), \quad \forall v \in \mathcal{V}_N.$$

We next take as a basis of $\mathcal{V}_N$ the set of tent functions defined as

$$\varphi_i(x_j) = \delta_{ij}, \text{ for } 1 \leq i, j \leq N,$$

and we express the solution as $u_N = \sum_{i=1}^N c_i \varphi_i$. Gathering the expansion coefficients in the vector $c = (c_i)_{i=1}^N$, and injecting the expansion of u_N in the variational formulation, we are led to the system of equations

$$Mc = q$$

where

$$M = (M_{i,j})_{1 \leq i,j \leq N}, \quad M_{i,j} := a(\varphi_i, \varphi_j), \quad \text{and} \quad q = (q_i)_{i=1}^N, \quad q_i := l(\varphi_i).$$

4.4 Numerical Results

4.4.1 Test case and comparison criteria

In this section we show the results obtained by approximating the exact solution of the problem described in equation (4.1) using the methods introduced above.

Here, we work in the case when $d = 1$, $\Omega = (0, 1)$, and F, f are assumed to be equal to some constant real numbers. Then, the solution of (4.1)-(4.2) has an analytic expression which is given hereafter. Let us also introduce $g_0, g_1 \in \mathbb{R}$ so that $g(0) = g_0$ and $g(1) = g_1$. The problem then reads as follows: find $u : (0, 1) \to \mathbb{R}$ solution to

$$\begin{cases} -\epsilon u''(x) + F u'(x) = f, & \forall x \in (0, 1), \\ -\alpha u'(0) + \kappa u(0) = g_0, \\ \alpha u'(1) + \kappa u(1) = g_1. \end{cases} \tag{4.29}$$

Then, it can be easily checked that the solution to this equation reads as

$$u(x) = C_1 + C_2 e^{\frac{Fx}{\epsilon}} + \frac{f}{F} x$$

where C_1 and C_2 are constants that are determined with the Robin boundary conditions. They satisfy the system

$$\begin{pmatrix} \kappa & \kappa - \frac{\alpha F}{\epsilon} \\ \kappa & \kappa e^{\frac{F}{\epsilon}} + \alpha \frac{F}{\epsilon} e^{\frac{F}{\epsilon}} \end{pmatrix} \begin{pmatrix} C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} g_0 + \alpha \frac{f}{F} \\ g_1 - \frac{f}{F}(\kappa + \alpha) \end{pmatrix}$$

which is invertible except for

$$\kappa = 0, \quad \text{or} \quad \frac{\alpha}{\kappa} = \frac{\epsilon(1 - e^{\frac{F}{\epsilon}})}{F(1 + e^{\frac{F}{\epsilon}})}.$$

In the following, the values of κ and α are always chosen so that the above system is invertible.

In the numerical tests presented below, we fix $F = 1$, $f = 1$. We choose Robin boundary conditions that mimic Dirichlet conditions and we set $\alpha = 10^{-3}$, $\kappa = 1$, $g_0 = g_1 = 0$. Note that we cannot take $\alpha = 0$ since all variational methods are not well defined for pure Dirichlet boundary conditions. With these choices, the equation reads

$$\begin{cases} -\epsilon u''(x) + u'(x) = 1, & \forall x \in (0, 1), \\ -10^{-3} u'(0) + u(0) = 0, \\ 10^{-3} u'(1) + u(1) = 0. \end{cases} \tag{4.30}$$

Since the exact solution has an analytic form, we can thus easily compare the approximation quality of the output functions $\hat{u}$ from our methods by computing a discrete version of their $L^2(\Omega)$ error norm with respect to the exact solution:

$$e_{L^2}^2 := \|u - \hat{u}\|_{L^2(\Omega)}^2 \approx \frac{1}{\tilde{K}} \sum_{k=0}^{\tilde{K}-1} (u(x_k) - \hat{u}(x_k))^2 =: e_{\ell^2}^2.$$

The points x_k are sampled uniformly as defined in 4.3.3. We use 10 times more points than the ones used for approximating the integral, so $\tilde{K} = 10K$. Similarly, we also compute the error with respect to the $H^1(\Omega)$ semi-norm:

$$e^2_{H^1} := \|u' - \hat{u}'\|^2_{L^2(\Omega)} \approx \frac{1}{\tilde{K}} \sum_{k=0}^{\tilde{K}-1} (u'(x_k) - \hat{u}'(x_k))^2 =: e^2_{h^1}.$$

Note that one can obtain the H^1 error by adding the above error components.

We study the impact on the errors of the following parameters:

- The values of ϵ. They range from $5 \cdot 10^{-3}$ to 10.0 with a logarithmic spacing.
- The number K of training points (or collocation points). We consider $K = 10$, 10^2, 10^3, 10^4.
- The choice of the sampling method for the training points (uniformly spaced or uniformly random, labelled as $-u$ and $-r$).
- The impact of the machine precision (Float16, Float32, Float64).

Due to the randomness in the initialization of weights on the neural networks, for each combination of parameters (ϵ, K, sampling type, and machine precision), we perform 10 repetitions with different initialization. Since we didn't notice a big difference between the l^2 error and the h^1 error, we keep just the second one for clarity and put in the Appendix B.1 the plots in l^2 error. Moreover, in the Section B.3 of the Appendix, some results concerning the training of the NN.

4.4.2 Our code and practical implementation details

All our neural network based numerical tests were performed in Python 3.6 and using the TensorFlow 1.13.1 library [282]. The code provided in the original paper on PINNs [209] was used as the starting point for our own code developments, and we have followed similar guidelines to generalize and enlarge it where needed. In the same way, for each numerical method, derivatives of functions $v \in \mathcal{K}$ are computed using automatic differentiation. The numerical optimization procedure used in order to compute an approximation of θ^* a minimizer of problem (4.26) is given by the quasi-Newton L-BFGS algorithm [283]. The code used to generate the examples shown here is available at

`https://github.com/agussomacal/ConDiPINN`

The interested reader can reproduce our results and test the impact of the variations of certain parameters such as ϵ, K, the sampling method, and the machine precision.

4.4.3 Discussion

Impact of the number K of training points

In this section we discuss the impact of the number K of training points. We fix the machine precision to Float32, and the uniform sampling $-u$.

Figure 4.1 shows the best result obtained in the tests, i.e., the minimum value of the h^1 norm obtained in the 10 different simulations, plotted against the values of ϵ. In Figure 4.2, we fix $\epsilon = 10$, and plot statistics on the accuracy e_{h^1} (left plot) and computation runtimes for different K (right plot), and for the different methods. Find in the Appendix in Section B.1 the equivalent plot for the l^2 error.

From these figures, we first notice that the approximation of FEM degrades when ϵ decreases. However, the accuracy improves when the number of discretization points increases (see, e.g., Figure 4.2 - left plot). The rate of improvement is linear as we can see from the right plot in Figure 4.2, as expected. In addition, when looking at the runtimes (Figure 4.2 - right) we observe the expected linear increase with respect to the number K of discretization points.

We can next study the behavior of Vanilla PINN and compare to FEM. We observe that it performs at an almost constant accuracy for any number of training points until around $\epsilon = 0.027$ where it stops producing reliable approximations (see Figure 4.1). One remarkable observation is that the Vanilla PINN error for large values of ϵ and small number of training points $K = 10$ is comparable to the FEM errors with a much larger number of degrees of freedom $K > 10^3$ (see left plot in Figure 4.2). However, we observe that the running

times of FEM computations remain much lower than the ones of PINN-based methods (right plot in Figure 4.2). Note that the low runtime of the FEM approach is due to the fact that the associated resulting linear system is tridiagonal, and this allows to solve with a linear cost w.r.t. the number of degrees of freedom.

We next comment on the other PINN-based variational methods. For ϵ large enough, we observe that all the variational based methods follow the same error trend as FEM both with respect to ϵ and K and for $K < 10^4$ they even perform marginally better. With respect to the computing time, all the methods perform with almost constant time with respect to K and similarly to a FEM method with $K = 100$ degrees of freedom. However, for $\epsilon < 0.63$, the methods $W - z$, $W - z - e$ and $V - z$ blow up and lose completely their approximation capabilities. We conjecture that this is due to the fact that the neural network is used to approximate the solution z from the transformed problem, and there is an exponential term to go back from z to u (see equation (4.7)). This may lead to machine precision overflows (in the exponential computation) and underflows (the neural network has to learn very small values of z which also are in the limits of precision). To address this issue, we have explored two possible strategies: one was by directly minimizing over u while maintaining the weak formulation which accounts for the method W. The second approach is to perform the re-scaling of the domain $RW - z$. In both cases the blow up caused by the exponential is solved although the re-scaling method $RW - z$ doesn't perform as good as others in the region with large ϵ values.

We finish this section by plotting in Figure 4.3 the best approximated solution for each model, and different values of ϵ. The interested reader may experiment other configurations in our provided code. The most striking observation is that only FEM and the vanilla PINN method recover the final shape of the exact solution when ϵ is small. The other variational PINN methods fail despite that some of them exhibit comparable values to FEM in the generalization errors as Figure 4.1 illustrates. This observations suggests that perhaps other types of error metrics should be introduced in order to be able to better distinguish between "good solution shapes" and "bad ones".

Impact of Machine Precision

Figure 4.4 shows the h^1-error of the different approximated solution by changing the machine precision in the parameters of the neural networks for the different values of ϵ: Float16, Float32 and Float64. There is an improvement when going from Float16 to Float32 in all methods. Interestingly, we did not obtain very satisfactory results when working with Float64 precision. This precision seems to difficult the convergence to good quality minima: even after 10 repetitions, we failed to find good results. However, as the plots show, when a good minimum is found, it delivers slightly better approximation than lower machine precisions. For these reasons we have performed our experiments using the Float32 which seemed the most stable choice.

Impact of Sampling Strategy

Figure 4.5 shows the h^1-error of the different approximated solution by changing the sampling strategy. For all models, the $uniform$ strategy is found to be either as good as the $random$ or slightly better. For this reason we performed all the experiments using the $uniform$ strategy.

4.4.4 Conclusions from the numerical experiments

The above numerical experiments depict a contrasted landscape concerning the merits and limitations of deep learning-based approaches when the solutions become low regular:

- For large values of ϵ when solutions are rather regular, some PINNs perform clearly better than FEM regarding the generalization errors. The superiority is particularly remarkable for very small number K of training points. However, the shapes of PINN solutions are sometimes not as satisfactory as the ones given by FEM.
- For the challenging case where ϵ becomes small and solutions become less regular (which was the main motivation of our study), the accuracy of the variational neural-network methods is essentially comparable or worse to the one given by FEM in terms of generalization errors. Some PINN variational approaches become too unstable and the errors blow up. Only FEM and the vanilla PINN approach seem to be able to recover the correct shape of the exact function. The latter one has however the risk of sometimes falling into local minima with bad shapes.
- The runtimes are clearly in favor to FEM methods as Figure 4.2 illustrates, but the simplicity of implementation is in favor to all PINN methods.

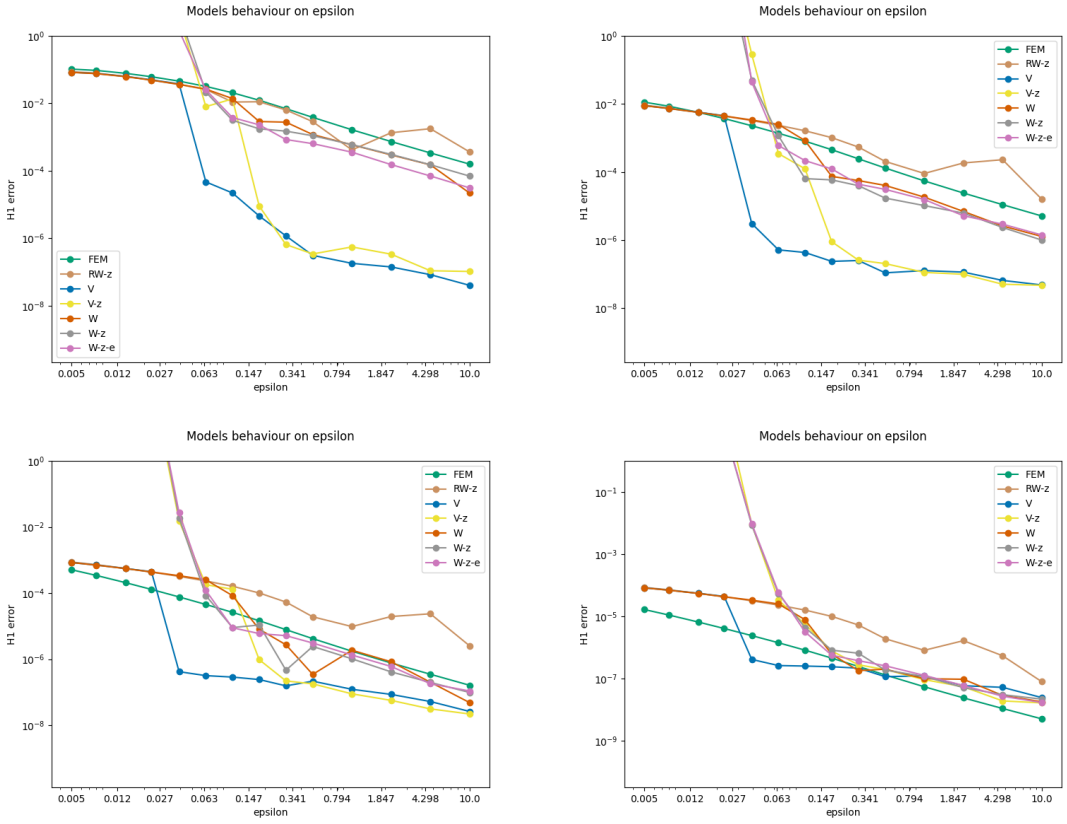


Figure 4.1: Comparison of the behavior of the h^1 error for the different methods and different number of sampling points in training. From top to bottom and from left to right, the first figure is produced for $K = 10$, the second for $K = 100$, the third for $K = 1000$ and the last one for $K = 10000$. The set of points to train and test have been chosen with the *uniform* sampling method. The precision has been fixed to Float32 for all the tests.

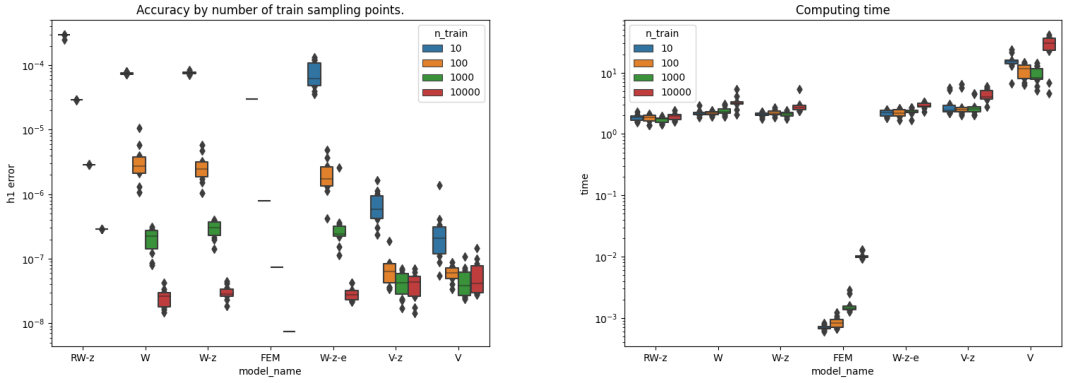


Figure 4.2: For $\epsilon = 10$ (region where all methods work well), we look at the comparison between methods and the difference with respect to the number of training points K. The h^1 error (left) and the computation times (right). The set of points to train and test have been chosen with the *uniform* sampling method. The precision has been chosen as Float32 for all the tests.

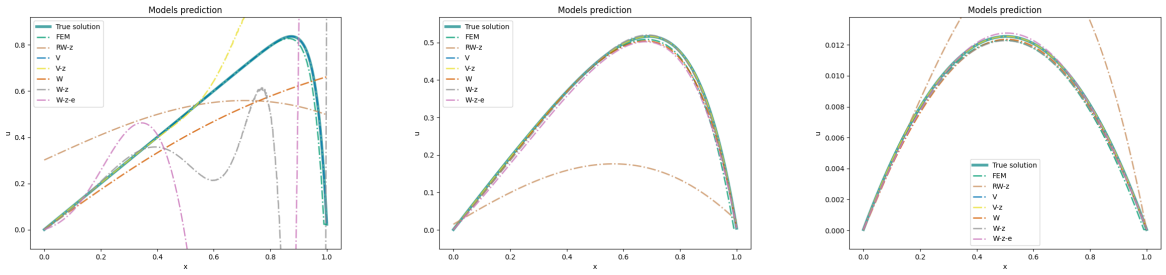


Figure 4.3: The best approximated solution out of 10 repetitions, for each model, and with $K = 100$ training samples. From left to right: $\varepsilon = 0.039, 0.18, 10$. The interested reader may experiment other configurations in our provided code.

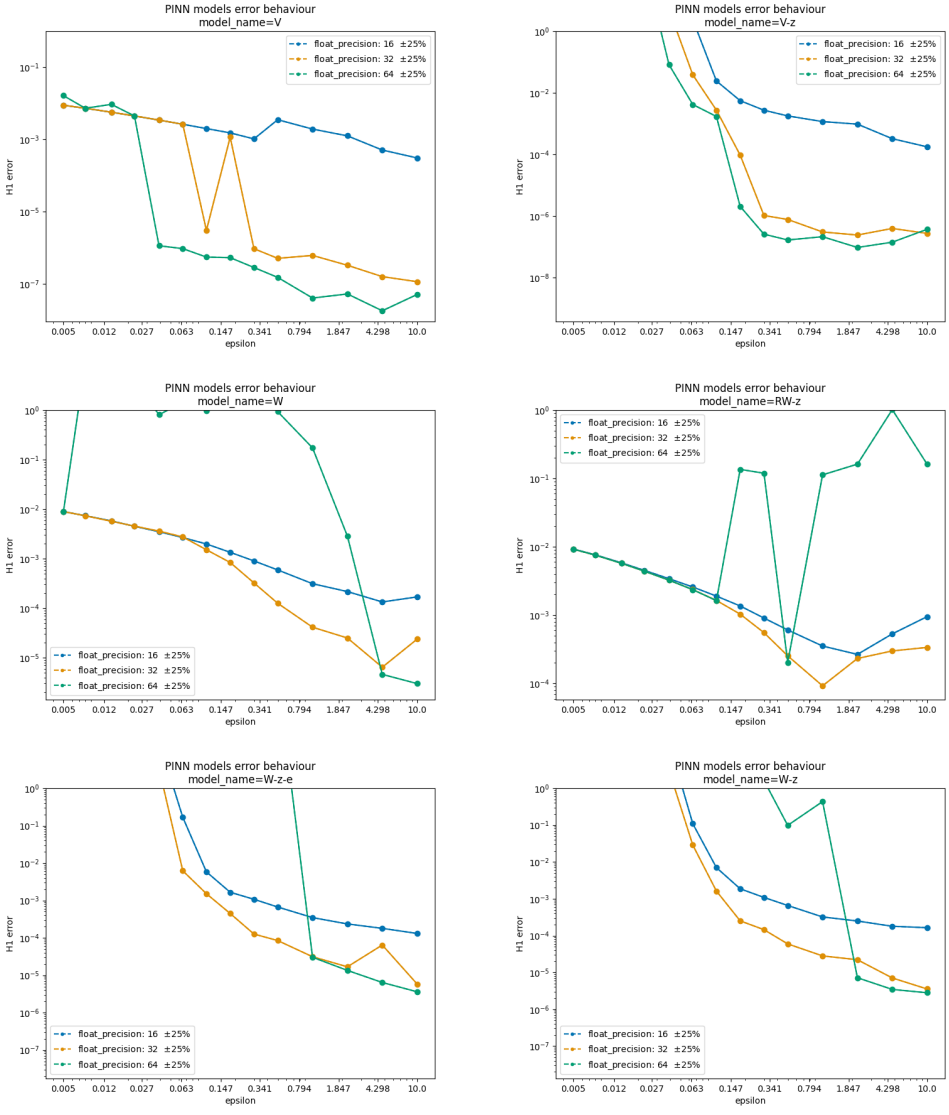


Figure 4.4: Here, the comparison of the behavior of the model for different float precision. The tests have been performed for $K = 100$ and uniform sampling.

4.5 Future research directions and extensions

One important point to explore in future works concerns the choice of the loss function for the training, and also the metric to evaluate generalization errors. It will also be interesting to explore if adaptive sampling

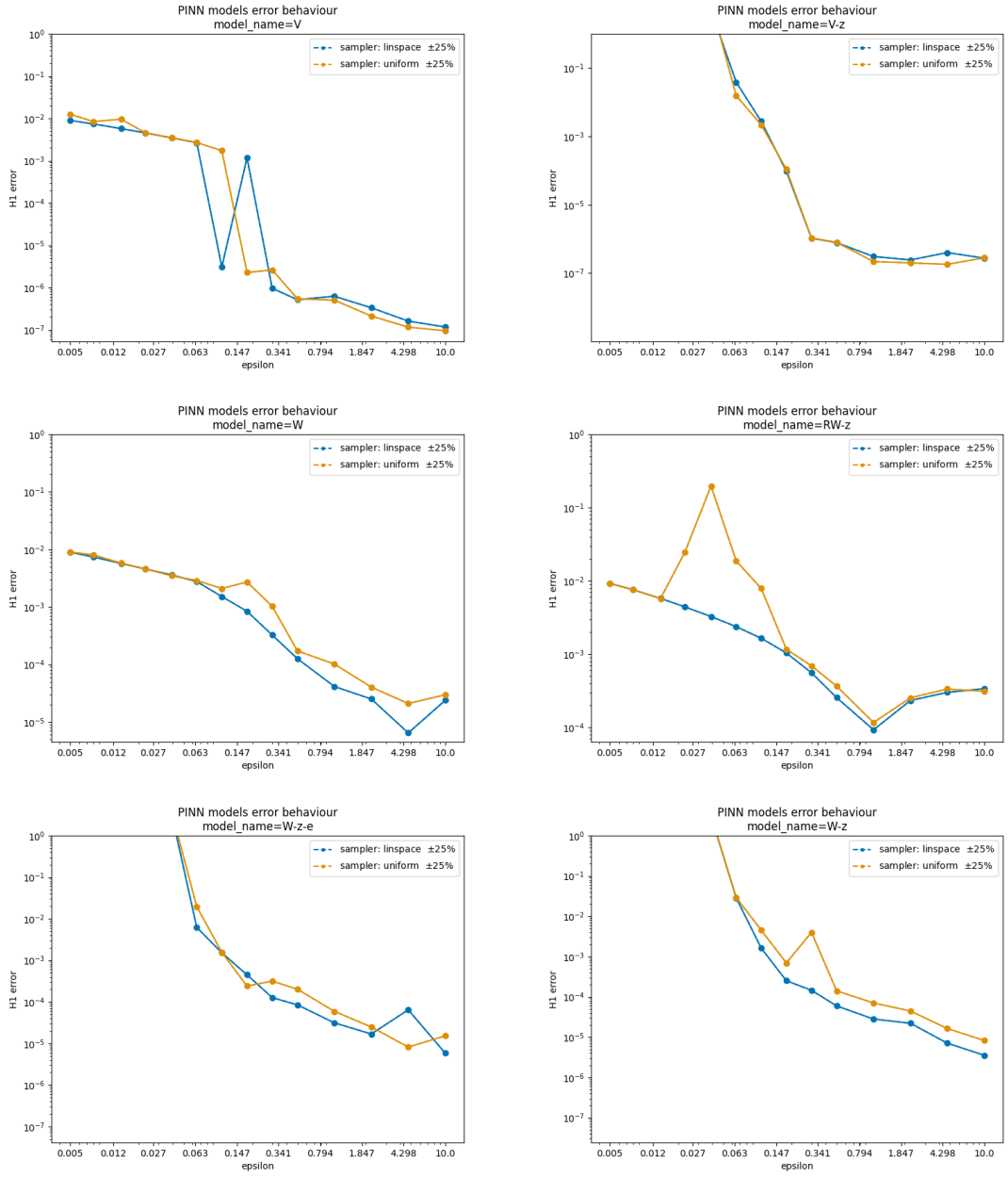


Figure 4.5: Here, the comparison of the behavior of the model for the two different sampling strategies. The tests have been performed for $K = 100$ and the float precision equal to Float32.

strategies during the training could help to recover good solutions in a more stable manner. Finally, the impact of the machine precision in some steps involving exponential transformations seems also to be an important obstacle to retrieving stable solutions. It would be interesting to develop strategies that circumvent this issue. All these developments will play a crucial role in order to address higher dimensional problems with similar characteristics as the one considered here.

Find in Figure B.2 in the Appendix a comparison between different architectures (depth and width) and the activation function of the network that have been tested. In the future, a possible improvement would be test more architectures in order to obtain better results.

We also refer the reader to Figure B.3 and Figure B.4 to see the number of iterations that the network takes to converge.

Chapter 5

Conclusions and perspectives

In the present work, we propose and study three numerical methods to approximate functions from different perspectives.

1. A tensor approximation method, Sum of Tensor Trains (SoTT), whose output is the approximated solution in a sum of TTs format.
2. A local tensor approximation method based on clustering, that retrieves an approximated solution in the introduced local HOSVD format.
3. A deep learning-based method that computes the approximation of the solution of a convection diffusion PDE when the diffusion parameter is very small.

5.1 Conclusions of SoTT

In the first chapter of the thesis, we proposed a method to compress a given tensor as a sum of Tensor Trains (SoTT). Neither the order of the variables nor the ranks are fixed a priori. They are the result of an optimization step. A particular instance of this method, consisting in fixing the ranks equal to one in all the steps of the algorithm, produces a CP approximation of a given tensor, we have introduced it as CP-TT. A proof of convergence is proposed in the general case of the SoTT algorithm, which can be extended to the case of the CP-TT algorithm. Moreover, several numerical experiments are proposed in order to illustrate the properties of the methods:

- First, we compared the CP-TT to other rank-one update methods (ALS, ASVD, TTr1). Although a single iteration of CP-TT is more expensive in terms of number of operations, its stability makes it a promising candidate to compress high-dimensional tensors in CP format.
- Then, we did some tests in which we compressed the numerical solution of a parametric partial differential equation of reaction-diffusion type as well as other functions coming from different applications. In particular, we compared SoTT with the TT-SVD obtained by testing all the possible permutations of the indices. Although SoTT is suboptimal with respect to the best TT-SVD, it is independent of the order of the variables and its performances are comparable to the average TT-SVD. In this test, the SoTT method outperforms CP-TT.

Both methods showed preliminary yet encouraging results in view of applications in scientific computing and compression of high order tensors. The method presented shows some shortcomings, to be addressed in further investigations: while a greedy method is appealing in view of computational tasks in which fixing the rank a priori could be cumbersome, it might be featured by a saturation effect, slowing down its convergence.

5.2 Conclusions of local HOSVD method

In the second chapter of the thesis, a method to compress tensors is proposed. In this case, the approximation is done locally and it is computed by the HOSVD method in a local subdomain of the original domain. The

subdomain is obtained automatically with a clustering method, that exploits the separability of the domain to compute partitions of the domain per direction. The different combinations of these partitions provide local subdomains. The HOSVD approximation of the tensor restricted to the local subdomain is computed. Among the partitions obtained, an extensive search of the subdomain that provides the smaller compression ratio with respect to the classical HOSVD method (in all the domain) is chosen to approximate the function. To the approximation of the tensor with the HOSVD method in this partition of the domain we have called local HOSVD method.

The process is computationally expensive and that is the reason why the numerical tests have been restricted to moderate order tensors. Nevertheless, some numerical tests for different applications and with different input formats have been performed and the results are optimistic.

- The partitions of the domain obtained with the clustering method are able to detect symmetry or periodicity in the system.
- An advantage in memory storage with respect to the HOSVD method is observed. This becomes more remarkable when the tensor we want to approximate presents different regimes.

Some other alternatives of how to select the subspace in which we compute the approximation have been explored but the results were not as encouraging as the ones obtained with the extensive search. An approach to the extensive search reducing the computational cost is a subject under study and it will be addressed in further investigations.

Moreover, there are other alternatives in order to use the idea of computing local approximations produced by the proposed clustering method. Once the clustering part is finished and the partitions of the domains per direction have been obtained, the objects obtained as an output can be used to construct a local tensor approximations with other format different to HOSVD. This idea is still under study but we are encouraged about it.

5.3 Conclusions of Deep Learning-based schemes

In the third chapter of the thesis, the task of approximating a multivariate function is tackled from a deep learning based method. The function to approximate is the solution of a convection-diffusion equation, known to be difficult to solve with the classical methods like Finite Element Method (FEM) when the diffusion coefficient ϵ adopts small values. Some experiments are done in order to see how different parameters affect the neural network (the format of the equation input, number of training points, etc). The experiments done, depict a contrasted landscape concerning the merits and limitations of deep learning-based approaches when the solutions become low regular:

- For large values of ϵ when solutions are rather regular, some PINNs perform clearly better than FEM regarding the generalization errors. The superiority is particularly remarkable for very small number K of training points. However, the shapes of PINN solutions are sometimes not as satisfactory as the ones given by FEM.
- For the challenging case where ϵ becomes small and solutions become less regular (which was the main motivation of our study), the accuracy of the variational neural-network methods is essentially comparable or worse to the one given by FEM in terms of generalization errors. Some PINN variational approaches become too unstable and the errors blow up. Only FEM and the vanilla PINN approach seem to be able to recover the correct shape of the exact function. The latter one has however the risk of sometimes falling into local minima with bad shapes.
- The runtimes and simplicity of implementation are clearly in favor to PINN methods.

In future works, a study of the choice of the loss function for the training and the metric to evaluate generalization errors will be performed. It will also be interesting to explore if adaptive sampling strategies during the training could help to recover good solutions in a more stable manner. Finally, the impact of the machine precision in some steps involving exponential transformations seems also to be an important obstacle to retrieving stable solutions. It would be interesting to develop strategies that circumvent this issue. We refer the reader to Appendix B to see other points in which the method can improve.

Appendix A

Appendix of Sum Of Tensor Trains: SoTT

A.1 Optimization of the computation

A.1.1 Optimization of the coefficients

In this section we try to maximize the coefficients c in every term of the CP decomposition given by Equation 2.3, in order to have a better result. This is possible because we can add some small α such that $|\alpha| \ll 1$ to the coefficient c to reach a better approximation. The goal of the coefficient optimization is to retrieve the best projection on the subspaces selected by the algorithm. Slightly modifying the coefficients c that appear in every term of the CP decomposition, what we are doing is letting the parameters that define our result a small range of movement in order to make the approximation closer to the exact projection of the tensor in the subspace.

The optimum value of the CP coefficients is given by the following minimization problem:

$$c_{opt} = \arg\min_{c}(\frac{1}{2}\|F - \widetilde{F}(c)\|^2)$$

Where $\widetilde{F}$ is the CP approximation of the tensor F given by Equation 2.3. It is possible to rewrite the expression as a minimization problem. Let us define the functional to minimize $J(c) : L^2(\Omega) \to \mathbb{R}^+$ as:

$$J(c) = \frac{1}{2}\int (F - \widetilde{F}(c))^2 d\Omega$$

Then, by finding the minimum of the functional with respect to the coefficients, $\frac{\delta J}{\delta c^{(l)}} = 0$, a linear system for the coefficients has to be solved. For $j, l = 1, \ldots, r$, the system written by terms: $M_{jl}c_j - b_l = 0$.

Where:

$$M_{jl} = \int_\Omega \left[u_j^{(1)}(x_1) \otimes \ldots \otimes u_j^{(d)}(x_d)\right] \left[u_l^{(1)}(x_1) \otimes \ldots \otimes u_l^{(d)}(x_d)\right] d\Omega$$

$$b_l = \int_\Omega \widetilde{F} \left[u_l^{(1)}(x_1) \otimes \ldots \otimes u_l^{(d)}(x_d)\right] d\Omega$$

A.1.2 Computing the SVD of the unfolding without explicitly computing and assembling the unfolding

As we can see in Section 2.3.3, one of the most computationally expensive operations in the computation of a tensor approximation via the SoTT method is the one that computes the unfolding matrices of the tensor and does an SVD of them on each iteration. For this reason, we have developed a numerical scheme that computes the SVD of each one of the unfoldings needed in the SoTT method without computing explicitly all of them.

Let us explain this. By simplicity let us particularize for one of the dimensions of the tensor. Without loss of generality, let us choose the one that has been identified as the first. Let $M \in \mathbb{R}^{\mathcal{N}_1 \times \mathcal{N}_2 \ldots \mathcal{N}_d}$, be an unfolding of a certain tensor F with respect to the first one of the variables, $M^{(1)}$. The CP form of F, defined in Equation 2.3 reads:

$$F = \sum_{i=1}^{r} c_i u_i^{(1)}(x_1) \otimes \ldots \otimes u_i^{(d)}(x_d)$$

A first approximation of the tensor F is given by the projection of F into the subspace generated by the first modes of its SVD decomposition $u_1^{(1)}$ in the first direction.

$$\bar{F}_1 = \int_{\Omega_1} \sum_{i=1}^{r} \left(c_i u_i^{(1)}(x_1) \otimes \ldots \otimes u_i^{(d)}(x_d) \right) u_1^{(1)}(x_1) dx_1 = \sum_{i=1}^{r} c_i \left(\int_{\Omega_1} u_i^{(1)}(x_1) u_1^{(1)}(x_1) dx_1 \right) \otimes u_i^{(2)}(x_2) \otimes \ldots \otimes u_i^{(d)}(x_d)$$

The expression above could be interpreted as a CP tensor decomposition if we consider a new set of coefficients for $1 \leq i \leq r$, namely

$$c_i^{(1)} = c_i \left(\int_{\Omega_1} u_i^{(1)}(x_1) u_1^{(1)}(x_1) dx_1 \right)$$

Multiplying the matrix unfolding and its transpose $M^{(1)} M^{(1)^T}$ and using the change of variable in the coefficients

$$M^{(1)} M^{(1)^T} = c_i^{(1)} c_j^{(1)} \int_{\Omega_2} \ldots \int_{\Omega_d} u_i^{(2)} \otimes \ldots \otimes u_i^{(d)} u_j^{(2)} \otimes \ldots \otimes u_j^{(d)} dx_2 \ldots dx_d$$

We can identify in the result the form $T^{(1)} K T^{(1)^T}$, where $K \in \mathbb{R}^{r \times r}$ is a covariance matrix. By components defined as:

$$K_{ij} = c_i c_j \int_{\Omega_2} \ldots \int_{\Omega_d} u_i^{(2)} \otimes \ldots \otimes u_i^{(d)} u_j^{(2)} \otimes \ldots \otimes u_j^{(d)} dx_2 \ldots dx_d$$

And $T^{(1)} \in \mathbb{R}^r$ reads

$$T_i^{(1)} = \int_{\Omega_1} u_i^{(1)}(x_1) u_1^{(1)}(x_1) dx_1$$

Its SVD is $T^{(1)} = \bar{U}^{(1)} \Sigma V^{(1)^T}$, where we can also define $Z^{(1)} = \Sigma V^{(1)^T}$. Then,

$$M^{(1)} M^{(1)^T} = T^{(1)} K T^{(1)^T} = \bar{U}^{(1)} Z^{(1)} K Z^{(1)^T} \bar{U}^{(1)^T}$$

If we identify the product $\bar{K} = Z^{(1)} K Z^{(1)^T} = W \Lambda W^T$, being Λ the diagonal matrix of eigenvalues, we can compute the largest singular value and its associated singular vector of $\bar{K}$,

$$u_1^{(1)} = \bar{U}^{(1)} w^{(1)}$$

$$\sigma^{(1)} = \sqrt{\lambda_1}$$

By doing this computation, we can obtain all the information of the needed unfolding without explicitly computing it. In terms of the computational cost, instead of computing as much SVD decompositions as unfoldings, we can obtain directly the mode associated to the largest singular value of each one of them.

A.2 Results for functions in H1

Similar numerical tests have been performed to the ones in Section 2.5.1 for different regularity of the functions. In this case, the value of the parameter β is chosen to be $\beta = \frac{d}{2} + 1.1$. The results on the compression of the multivariate function given in Equation 2.9 obtained for $H^1(\Omega)$ functions are equivalent to the ones shown for $L^2(\Omega)$ functions, in which the decrease of the error norm with the approximation rank is quite regular in CP-TT and behaves in a quite stable way also for higher order tensors.
Let us show in Figure A.1, a comparison between the compression computed by different rank-1 methods: CP-TT, ALS, ASVD and TTr1, for functions with regularity $H^1(\Omega)$ in 4 dimensions.

As a summary of the tests done we add Table A.1 as the counterpart of Table 2.1 for $\beta = \frac{d}{2} + 1.1$.

Conclusions on this second test case are similar to the ones obtained in Section 2.5.1. ALS seems to outperform all other rank-1 update methods in the case where $d = 4$, whereas CP-TT seems to outperform the other methods for higher values of d.

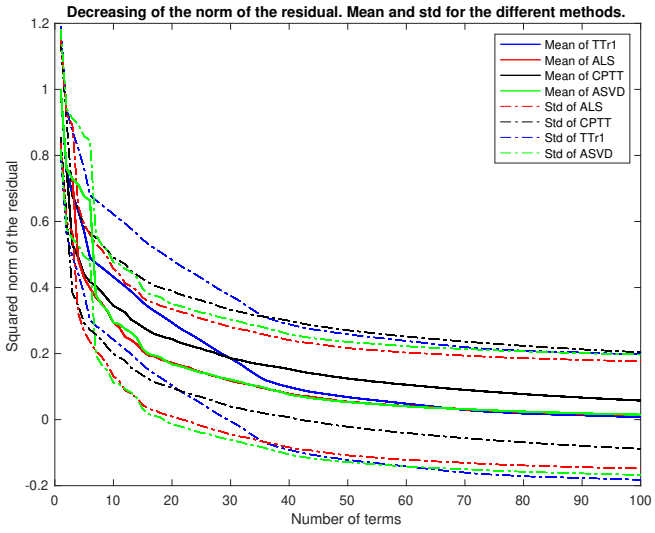


Figure A.1: Case $\beta = \frac{d}{2} + 1.1$. Mean and standard deviation of the L^2 norm of the difference between the exact function W and its approximation given by ALS (red), ASVD (blue) and CP-TT (black) as a function of the number of terms. See Table A.1 for more detailed information.

		Mean			Std		
Dimension (d)	Rank (r)	ALS	CPTT	ASVD	ALS	CPTT	ASVD
4	25	**0.1722**	0.2261	0.1759	**0.0643**	0.2261	0.1759
	50	**0.0572**	0.1382	0.0590	**0.0220**	0.1382	0.0232
	75	**0.0252**	0.0948	0.0262	**0.0103**	0.0948	0.0110
6	25	0.3741	**0.2938**	0.4171	0.1158	**0.0942**	0.1341
	50	0.2037	**0.1507**	0.2281	0.0655	**0.0523**	0.0791
	75	0.0851	**0.0579**	0.1045	0.0334	**0.0233**	0.0493
8	25	0.3676	**0.2560**	0.3977	0.1361	**0.0905**	0.1517
	50	0.2136	**0.1229**	0.2413	0.0807	**0.0451**	0.1023
	75	0.1046	**0.0455**	0.1145	0.0437	**0.0195**	0.0631
10	25	0.4574	**0.3737**	0.4753	**0.1235**	0.1548	0.1817
	50	**0.2613**	0.3483	0.3193	**0.0809**	0.1825	0.1648
	75	**0.1168**	0.3332	0.2352	**0.0628**	0.2034	0.1865
12	25	0.4634	**0.2505**	0.5182	0.1681	**0.0842**	0.2116
	50	0.2889	**0.1141**	0.3922	0.1421	**0.0384**	0.2170
	75	0.1278	**0.0382**	0.3144	0.0671	**0.0126**	0.2502
14	25	0.5943	**0.2169**	0.4386	0.2043	**0.1262**	0.2014
	50	0.2841	**0.0779**	0.3132	0.1277	**0.0686**	0.1915
	75	0.1422	**0.0244**	0.2021	0.0814	**0.0227**	0.2192
16	25	0.4598	**0.2460**	0.5543	0.1496	**0.0726**	0.1603
	50	0.2861	**0.1108**	0.3936	0.1268	**0.0348**	0.2022
	75	0.1395	**0.0438**	0.3181	0.0552	**0.0153**	0.2477

Table A.1: Mean and standard deviation of the norm of the residual for 32 random functions in the case where $\beta = \frac{d}{2} + 1.1$.

Appendix B

Appendix of Deep Learning-based schemes

B.1 l2 error plots

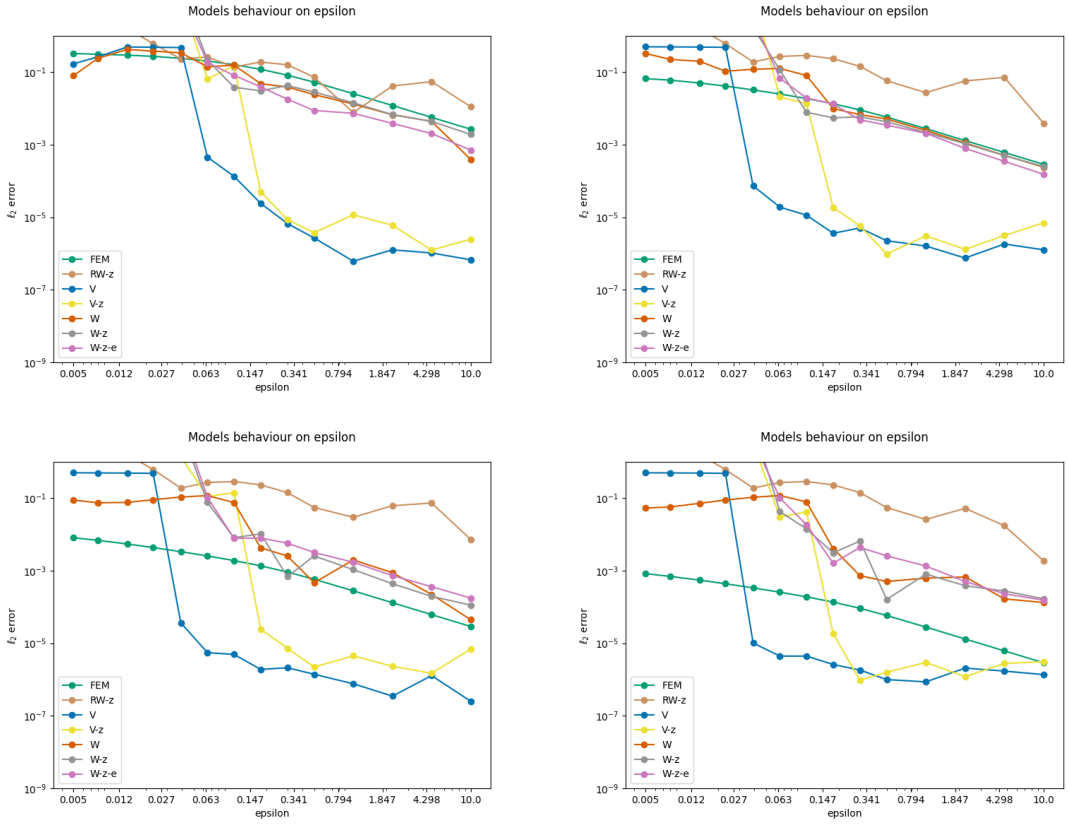


Figure B.1: The comparison of the behavior of the l^2 error for the different methods and different number of sampling points in training. From up to down and from left to right, the first figure is produced for $K = 10$, the second for $K = 100$, the third for $K = 1000$ and the last one for $K = 10000$. The set of points to train and test have been chosen with the *uniform* sampling method. The precision has been chosen as Float32 for all the tests.

B.2 Architecture of the NN plots

The choice of the architecture (depth and width) and activation function (tanh, sigmoid, etc) of the underlying neural network have an effect. For the present work, we decided that this investigation was not going to be our main focus, and that we wanted to concentrate mainly in the different formulations of PINNs. Note that we originally choose the activation function to be *tanh* because it was used in the original paper of PINNs. Our architecture with two hidden layers of twenty neurons each was fixed to have a trade-off between expressivity and computation time. To illustrate that the choice of architecture and activation function is not obvious, we have drawn the four plots of figure B.2 below. They show that increasing the depth effectively increases in general the computation time to arrive to the optimum, but the choice of width and the activation function doesn't seem to shed clear tendencies as to which one should be chosen.

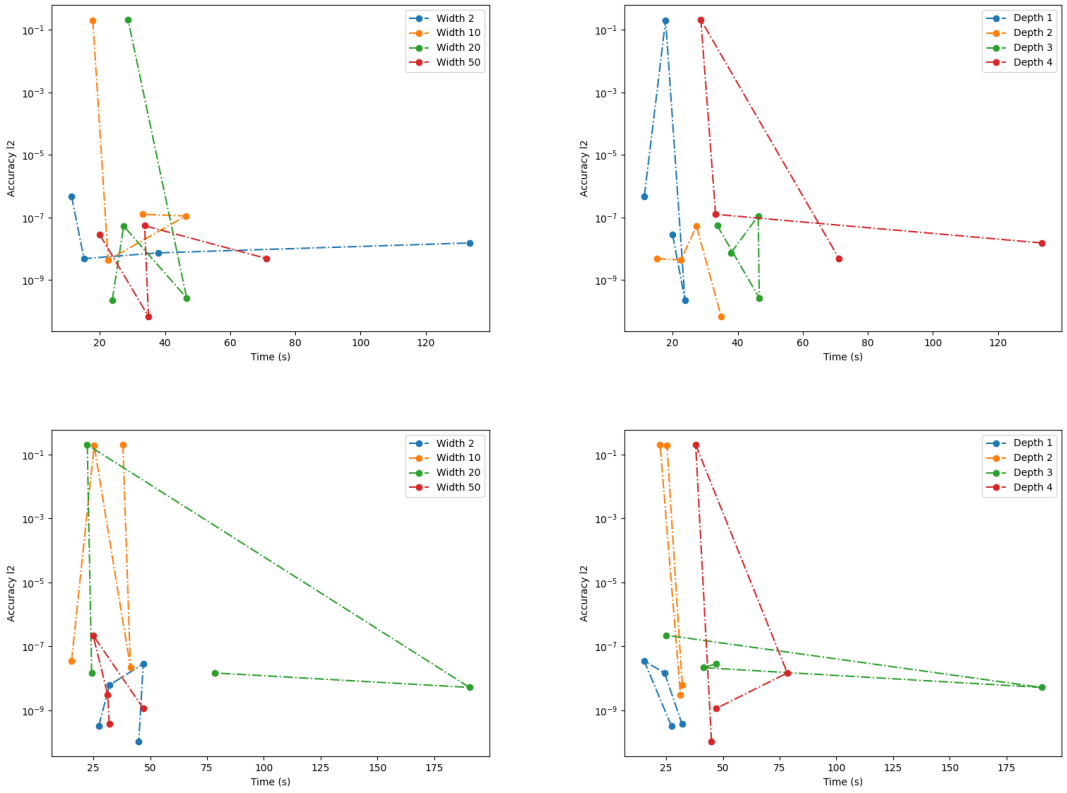


Figure B.2: Accuracy vs time to fit of networks depending on depth (left) or width (right) and activation function sigmoid (up), tanh (down). Each point is the best of two realizations of approximations to the solution of the test example for $\epsilon = 0.05$ and $K = 100$ using Vanilla PINN.

B.3 Training of the PINN

As the neural networks are neither very deep nor very wide, the training typically doesn't take too much time. In general we have observed that for initializations that finish by converging to a good approximation the time spent in training is bigger than those in which the optimization got stuck into a local minima B.3 (left). But it can be otherwise too as shown in figure B.4.

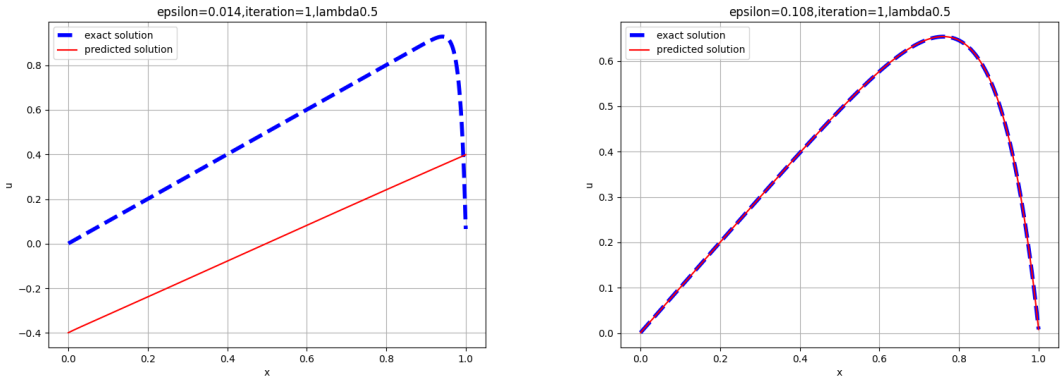


Figure B.3: For $\epsilon = 0.014$ (left) and $\epsilon = 0.108$ (right) we see the approximation obtained by Vanilla PINN with $K = 100$. In the left the local minimum that was found correspond to a straight line that plays a trade-off between minimizing the residuals (with $\frac{\partial u}{\partial^2 x} = 0$ and $\frac{\partial u}{\partial x} \approx 1 = f$) and the loss in the boundary conditions.

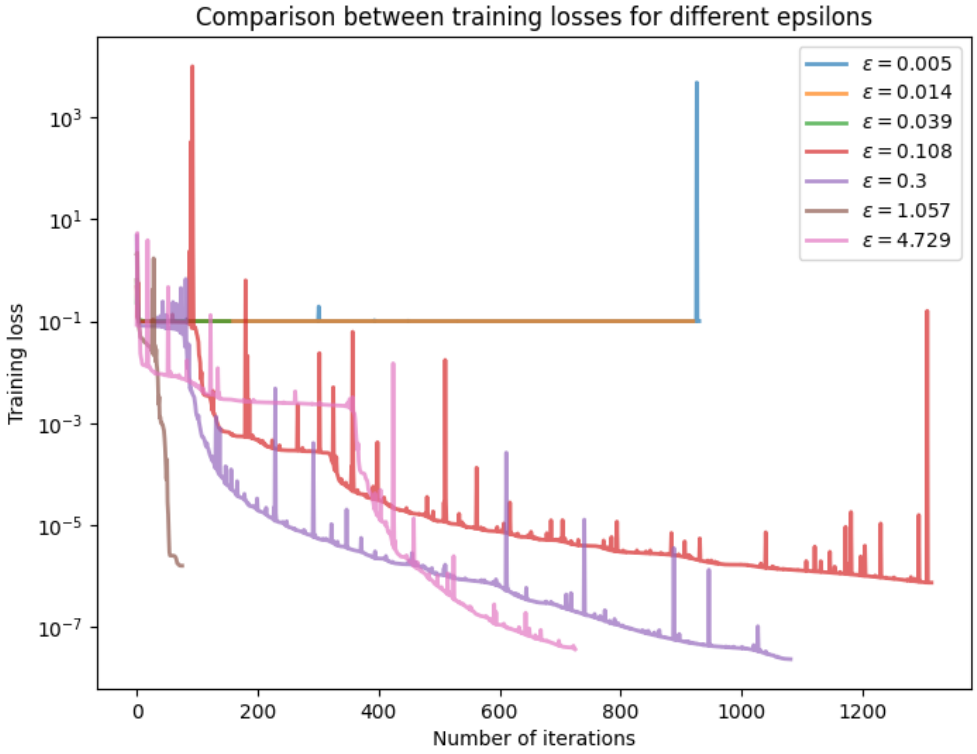


Figure B.4: Each curve correspond to the evolution of the train loss for a realization Vanilla PINN for different choices of ϵ and $K = 100$.

Bibliography

[1] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, and Danilo P Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Foundations and Trends® in Machine Learning, 9(4-5):249–429, 2016.

[2] Tamara G Kolda. Tensor decomposition: A mathematical tool for data analysis. Technical report, Sandia National Lab.(SNL-CA), Livermore, CA (United States), 2018.

[3] Tamara G Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In 2008 Eighth IEEE international conference on data mining, pages 363–372. IEEE, 2008.

[4] Richard Bellman. Dynamic programming. Science, 153(3731):34–37, 1966.

[5] I.V. Oseledets and E.E. Tyrtyshnikov. Breaking the curse of dimensionality, or how to use svd in many dimensions. SIAM Journal on Matrix Analysis and Applications, 31(5):1084–1127, January 2009.

[6] Wilhelmus HA Schilders, Henk A Van der Vorst, and Joost Rommes. Model order reduction: theory, research aspects and applications, volume 13. Springer, 2008.

[7] Peter Benner, Mario Ohlberger, Albert Cohen, and Karen Willcox. Model reduction and approximation: theory and algorithms. SIAM, 2017.

[8] NC Nguyen, Gianluigi Rozza, DB Phuong Huynh, and Anthony T Patera. Reduced basis approximation and a posteriori error estimation for parametrized parabolic pdes: Application to real-time bayesian parameter estimation. Large-Scale Inverse Problems and Quantification of Uncertainty, pages 151–177, 2010.

[9] DV Rovas, L Machiels, and Yvon Maday. Reduced-basis output bound methods for parabolic problems. IMA journal of numerical analysis, 26(3):423–445, 2006.

[10] Alfio Quarteroni, Andrea Manzoni, and Federico Negri. Reduced basis methods for partial differential equations: an introduction, volume 92. Springer, 2015.

[11] Alfio Quarteroni, Gianluigi Rozza, and Andrea Manzoni. Certified reduced basis approximation for parametrized partial differential equations and applications. Journal of Mathematics in Industry, 1(1):1–49, 2011.

[12] Bernard Haasdonk and Mario Ohlberger. Reduced basis method for finite volume approximations of parametrized linear evolution equations. ESAIM: Mathematical Modelling and Numerical Analysis, 42(2):277–302, 2008.

[13] Martin A Grepl and Anthony T Patera. A posteriori error bounds for reduced-basis approximations of parametrized parabolic partial differential equations. ESAIM: Mathematical Modelling and Numerical Analysis, 39(1):157–181, 2005.

[14] Gal Berkooz, Philip Holmes, and John L Lumley. The proper orthogonal decomposition in the analysis of turbulent flows. Annual review of fluid mechanics, 25(1):539–575, 1993.

[15] Lawrence Sirovich. Turbulence and the dynamics of coherent structures. i. coherent structures. Quarterly of applied mathematics, 45(3):561–571, 1987.

[16] Kenneth S Breuer and Lawrence Sirovich. The use of the karhunen-loeve procedure for the calculation of linear eigenfunctions. Journal of Computational Physics, 96(2):277–296, 1991.

[17] Albert Cohen and Ronald DeVore. Approximation of high-dimensional parametric pdes. Acta Numerica, 24:1–159, 2015.

[18] Albert Cohen, Wolfgang Dahmen, Ronald A DeVore, and Angela Kunoth. Multiscale and high-dimensional problems. Oberwolfach Reports, 14(1):1001–1051, 2018.

[19] Tan Bui-Thanh, Karen Willcox, and Omar Ghattas. Model reduction for large-scale systems with high-dimensional parametric input space. SIAM Journal on Scientific Computing, 30(6):3270–3288, 2008.

[20] Peter Binev, Albert Cohen, Wolfgang Dahmen, Ronald DeVore, Guergana Petrova, and Przemyslaw Wojtaszczyk. Convergence rates for greedy algorithms in reduced basis methods. SIAM journal on mathematical analysis, 43(3):1457–1472, 2011.

[21] Ronald A DeVore and Vladimir N Temlyakov. Some remarks on greedy algorithms. Advances in computational Mathematics, 5(1):173–187, 1996.

[22] Vladimir N Temlyakov. The best m-term approximation and greedy algorithms. Advances in Computational Mathematics, 8(3):249–265, 1998.

[23] David L Donoho, Michael Elad, and Vladimir N Temlyakov. Stable recovery of sparse overcomplete representations in the presence of noise. IEEE Transactions on information theory, 52(1):6–18, 2005.

[24] Wolfgang Hackbusch. Tensor spaces and numerical tensor calculus, volume 42. Springer, 2012.

[25] Anthony Nouy. Low-rank tensor methods for model order reduction. arXiv preprint arXiv:1511.01555, 2015.

[26] Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. Acta Numerica, 30:327–444, 2021.

[27] Lars Grasedyck, Daniel Kressner, and Christine Tobler. A literature survey of low-rank tensor approximation techniques. GAMM-Mitteilungen, 36(1):53–78, 2013.

[28] Anthony Nouy. Low-rank methods for high-dimensional approximation and model order reduction. Model Reduction and Approximation: Theory and Algorithms, 15(171):3672148, 2017.

[29] T. Kolda and B. Bader. Tensor decompositions and applications. SIAM Review, 51:455–500, 2009.

[30] A. Cichocki, N. Lee, I. Oseledets, A. Phan amd Q. Zhaonand, and D. Mandic. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. Now Publishers Inc., 35, December 2016.

[31] I. Domanov and L. Lathauwer. On uniqueness and computation of the decomposition of a tensor into multilinear rank-(1, lr, lr) terms. SIAM J. Matrix Anal. Appl., 41:747–803, 2020.

[32] S. Friedland and G. Ottaviani. The number of singular vector tuples and uniqueness of best rank-one approximation of tensors. Found Comput Math, 14:1209–1242, March 2014.

[33] William A Adkins and Steven H Weintraub. Algebra: an approach via module theory, volume 136. Springer Science & Business Media, 2012.

[34] David Steven Dummit and Richard M Foote. Abstract algebra, volume 3. Wiley Hoboken, 2004.

[35] Serge Lang. Algebra, volume 211. Springer Science & Business Media, 2012.

[36] Wolfgang Hackbusch. Multi-grid methods and applications, volume 4. Springer Science & Business Media, 2013.

[37] Marvin Marcus. Finite dimensional multilinear algebra, volume 23. M. Dekker, 1973.

[38] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. The London, Edinburgh, and Dublin philosophical magazine and journal of science, 2(11):559–572, 1901.

[39] Bruce Moore. Principal component analysis in linear systems: Controllability, observability, and model reduction. IEEE transactions on automatic control, 26(1):17–32, 1981.

[40] Karen Willcox and Jaime Peraire. Balanced model reduction via the proper orthogonal decomposition. AIAA journal, 40(11):2323–2330, 2002.

[41] Kevin Carlberg, Charbel Bou-Mosleh, and Charbel Farhat. Efficient non-linear model reduction via a least-squares petrov–galerkin projection and compressive tensor approximations. International Journal for numerical methods in engineering, 86(2):155–181, 2011.

[42] Boris N Khoromskij. Tensor numerical methods in scientific computing, volume 19. Walter de Gruyter GmbH & Co KG, 2018.

[43] Venera Khoromskaia and Boris N Khoromskij. Tensor-based techniques for fast discretization and solution of 3d elliptic equations with random coefficients. arXiv preprint arXiv:2007.06524, 2020.

[44] Ivo Babuška, Fabio Nobile, and Raúl Tempone. A stochastic collocation method for elliptic partial differential equations with random input data. SIAM Journal on Numerical Analysis, 45(3):1005–1034, 2007.

[45] Hervé Abdi and Lynne J Williams. Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4):433–459, 2010.

[46] Harold Hotelling. Analysis of a complex of statistical variables into principal components. Journal of educational psychology, 24(6):417, 1933.

[47] Rene Vidal, Yi Ma, and Shankar Sastry. Generalized principal component analysis (gpca). IEEE transactions on pattern analysis and machine intelligence, 27(12):1945–1959, 2005.

[48] Nicholas J Higham. Computing the nearest correlation matrix—a problem from finance. IMA journal of Numerical Analysis, 22(3):329–343, 2002.

[49] Anthony Nouy. A priori tensor approximations for the numerical solution of high dimensional problems: alternative definitions. In The Seventh International Conference on Engineering Computational Technology (ECT2010), pages Paper–44, 2010.

[50] P. Ladevèze, J.-C. Passieux, and D. Néron. The latin multiscale computational method and the proper generalized decomposition. Computer Methods in Applied Mechanics and Engineering, 199(21):1287–1296, 2010. Multiscale Models and Mathematical Aspects in Solid and Fluid Mechanics.

[51] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. SIAM review, 51(3):455–500, 2009.

[52] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. Journal of Mathematics and Physics, 6(1-4):164–189, 1927.

[53] Frank L Hitchcock. Multiple invariants and generalized rank of a p-way matrix or tensor. Journal of Mathematics and Physics, 7(1-4):39–79, 1928.

[54] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. Psychometrika, 35(3):283–319, 1970.

[55] Richard A Harshman et al. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[56] Henk AL Kiers. Towards a standardized notation and terminology in multiway analysis. Journal of Chemometrics: A Journal of the Chemometrics Society, 14(3):105–122, 2000.

[57] J Mocks. Topographic components model for event-related potentials and some biophysical considerations. IEEE transactions on biomedical engineering, 35(6):482–484, 1988.

[58] I. Domanov and L. De Lathauwer. Canonical polyadic decomposition of third-order tensors: reduction to generalized eigenvalue decomposition. SIAM Journal on Matrix Analysis and Applications, 35(2):636–660, December 2014.

[59] Lars Karlsson, Daniel Kressner, and André Uschmajew. Parallel algorithms for tensor completion in the cp format. Parallel Computing, 57:222–234, 2016.

[60] Canyi Lu, Jiashi Feng, Yudong Chen, Wei Liu, Zhouchen Lin, and Shuicheng Yan. Tensor robust principal component analysis: Exact recovery of corrupted low-rank tensors via convex optimization. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 5249–5257, 2016.

[61] Alwin Stegeman and Nicholas D Sidiropoulos. On kruskal's uniqueness condition for the candecomp/-parafac decomposition. Linear Algebra and its applications, 420(2-3):540–552, 2007.

[62] Johan Håstad. Tensor rank is np-complete. Journal of Algorithms, 11(4):644–654, 1990.

[63] Vin de Silva and Lek-Heng Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. SIAM Journal on Matrix Analysis and Applications, 30(3):1084–1127, June 2008.

[64] André Uschmajew. Local convergence of the alternating least squares algorithm for canonical tensor approximation. SIAM Journal on Matrix Analysis and Applications, 33(2):639–652, 2012.

[65] Lieven De Lathauwer and Dimitri Nion. Decompositions of a higher-order tensor in block terms—part iii: Alternating least squares algorithms. SIAM journal on Matrix Analysis and Applications, 30(3):1067–1083, 2008.

[66] Myriam Rajih, Pierre Comon, and Richard A Harshman. Enhanced line search: A novel method to accelerate parafac. SIAM journal on matrix analysis and applications, 30(3):1128–1147, 2008.

[67] Sebastian Holtz, Thorsten Rohwedder, and Reinhold Schneider. The alternating linear scheme for tensor optimization in the tensor train format. SIAM Journal on Scientific Computing, 34(2):A683–A713, 2012.

[68] David Ruppert and Matthew P Wand. Multivariate locally weighted least squares regression. The annals of statistics, pages 1346–1370, 1994.

[69] Cécile Haberstich, Anthony Nouy, and Guillaume Perrin. Boosted optimal weighted least-squares. Mathematics of Computation, 91(335):1281–1315, 2022.

[70] Tihomir Asparouhov and Bengt Muthén. Weighted least squares estimation with missing data. Mplus technical appendix, 2010(1-10):5, 2010.

[71] Albert Cohen and Giovanni Migliorati. Optimal weighted least-squares methods. The SMAI journal of computational mathematics, 3:181–203, 2017.

[72] Balázs Hidasi and Domonkos Tikk. Fast als-based tensor factorization for context-aware recommendation from implicit feedback. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pages 67–82. Springer, 2012.

[73] Evrim Acar, Daniel M Dunlavy, Tamara G Kolda, and Morten Mørup. Scalable tensor factorizations for incomplete data. Chemometrics and Intelligent Laboratory Systems, 106(1):41–56, 2011.

[74] Giorgio Tomasi and Rasmus Bro. Parafac and missing values. Chemometrics and Intelligent Laboratory Systems, 75(2):163–180, 2005.

[75] Wolfgang Hackbusch, Boris N Khoromskij, and Eugene E Tyrtyshnikov. Hierarchical kronecker tensor-product approximations. 2005.

[76] Brett W Bader, Michael W Berry, and Murray Browne. Discussion tracking in enron email using parafac. In Survey of text mining II, pages 147–163. Springer, 2008.

[77] Pierre Ladevèze, J-C Passieux, and David Néron. The latin multiscale computational method and the proper generalized decomposition. Computer Methods in Applied Mechanics and Engineering, 199(21-22):1287–1296, 2010.

[78] Pierre Ladevèze. Nonlinear computational structural mechanics: new approaches and non-incremental methods of calculation. Springer Science & Business Media, 2012.

[79] Anthony Nouy and Pierre Ladevèze. Multiscale computational strategy with time and space homogenization: a radial-type approximation technique for solving microproblems. International Journal for Multiscale Computational Engineering, 2(4), 2004.

[80] Jie Chen and Yousef Saad. On the tensor svd and the optimal low rank orthogonal approximation of tensors. SIAM journal on Matrix Analysis and Applications, 30(4):1709–1734, 2009.

[81] Tamara G Kolda. A counterexample to the possibility of an extension of the eckart–young low-rank approximation theorem for the orthogonal rank tensor decomposition. SIAM Journal on Matrix Analysis and Applications, 24(3):762–767, 2003.

[82] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. SIAM journal on Matrix Analysis and Applications, 21(4):1253–1278, 2000.

[83] Antonio Falco and Anthony Nouy. A proper generalized decomposition for the solution of elliptic problems in abstract form by using a functional eckart–young approach. Journal of Mathematical Analysis and Applications, 376(2):469–480, 2011.

[84] Francisco Chinesta, Roland Keunings, and Adrien Leygue. The proper generalized decomposition for advanced numerical simulations: a primer. Springer Science & Business Media, 2013.

[85] Amine Ammar, Béchir Mokdad, Francisco Chinesta, and Roland Keunings. A new family of solvers for some classes of multidimensional partial differential equations encountered in kinetic theory modeling of complex fluids. Journal of non-Newtonian fluid Mechanics, 139(3):153–176, 2006.

[86] Amine Ammar, Francisco Chinesta, and Antonio Falco. On the convergence of a greedy rank-one update algorithm for a class of linear systems. Archives of Computational Methods in Engineering, 17(4):473–486, 2010.

[87] André Uschmajew and Bart Vandereycken. Greedy rank updates combined with riemannian descent methods for low-rank optimization. In 2015 International Conference on Sampling Theory and Applications (SampTA), pages 420–424. IEEE, 2015.

[88] Marie Billaud-Friess, Anthony Nouy, and Olivier Zahm. A tensor approximation method based on ideal minimal residual formulations for the solution of high-dimensional problems. ESAIM: Mathematical Modelling and Numerical Analysis, 48(6):1777–1806, 2014.

[89] Tamara G Kolda. Orthogonal tensor decompositions. SIAM Journal on Matrix Analysis and Applications, 23(1):243–255, 2001.

[90] Eric Cances, Virginie Ehrlacher, and Tony Lelievre. Convergence of a greedy algorithm for high-dimensional convex nonlinear problems. Mathematical Models and Methods in Applied Sciences, 21(12):2433–2467, 2011.

[91] Eric Cances, Virginie Ehrlacher, and Tony Lelièvre. Greedy algorithms for high-dimensional non-symmetric linear problems. In ESAIM: Proceedings, volume 41, pages 95–131. EDP Sciences, 2013.

[92] Anthony Nouy. A generalized spectral decomposition technique to solve a class of linear stochastic partial differential equations. Computer Methods in Applied Mechanics and Engineering, 196(45-48):4521–4537, 2007.

[93] Francisco Chinesta, Pierre Ladeveze, and Elias Cueto. A short review on model order reduction based on proper generalized decomposition. Archives of Computational Methods in Engineering, 18(4):395–404, 2011.

[94] Francisco Chinesta, Amine Ammar, and Elías Cueto. Recent advances and new challenges in the use of the proper generalized decomposition for solving multidimensional models. Archives of Computational methods in Engineering, 17(4):327–350, 2010.

[95] Ledyard R Tucker. Implications of factor analysis of three-way matrices for measurement of change. Problems in measuring change, 15(122-137):3, 1963.

[96] Ledyard R Tucker et al. The extension of factor analysis to three-dimensional matrices. Contributions to mathematical psychology, 110119, 1964.

[97] Ledyard R Tucker. Some mathematical notes on three-mode factor analysis. Psychometrika, 31(3):279–311, 1966.

[98] Pieter M Kroonenberg and Jan De Leeuw. Principal component analysis of three-mode data by means of alternating least squares algorithms. Psychometrika, 45(1):69–97, 1980.

[99] Arie Kapteyn, Heinz Neudecker, and Tom Wansbeek. An approach ton-mode components analysis. Psychometrika, 51(2):269–275, 1986.

[100] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. On the best rank-1 and rank-(r 1, r 2,..., rn) approximation of higher-order tensors. SIAM journal on Matrix Analysis and Applications, 21(4):1324–1342, 2000.

[101] Boris N Khoromskij and Venera Khoromskaia. Multigrid accelerated tensor approximation of function related multidimensional arrays. SIAM Journal on Scientific Computing, 31(4):3002–3026, 2009.

[102] Boris N Khoromskij, Venera Khoromskaia, and H-J Flad. Numerical solution of the hartree–fock equation in multilevel tensor-structured format. SIAM journal on scientific computing, 33(1):45–65, 2011.

[103] Ivan V Oseledets, DV Savostianov, and Eugene E Tyrtyshnikov. Tucker dimensionality reduction of three-dimensional arrays in linear time. SIAM Journal on Matrix Analysis and Applications, 30(3):939–956, 2008.

[104] Lars Grasedyck. Hierarchical singular value decomposition of tensors. SIAM journal on matrix analysis and applications, 31(4):2029–2054, 2010.

[105] Roland Badeau and Rémy Boyer. Fast multilinear singular value decomposition for structured tensors. SIAM Journal on Matrix Analysis and Applications, 30(3):1008–1021, 2008.

[106] Osman Asif Malik and Stephen Becker. Low-rank tucker decomposition of large tensors using tensorsketch. Advances in neural information processing systems, 31, 2018.

[107] Linjian Ma and Edgar Solomonik. Fast and accurate randomized algorithms for low-rank tensor decompositions. Advances in Neural Information Processing Systems, 34:24299–24312, 2021.

[108] Antonio Falcó, Wolfgang Hackbusch, and Anthony Nouy. Tree-based tensor formats. SeMA Journal, 78(2):159–173, 2021.

[109] Jonas Ballani and Lars Grasedyck. Tree adaptive approximation in the hierarchical tensor format. SIAM journal on scientific computing, 36(4):A1415–A1431, 2014.

[110] Ivan V Oseledets and Eugene E Tyrtyshnikov. Breaking the curse of dimensionality, or how to use svd in many dimensions. SIAM Journal on Scientific Computing, 31(5):3744–3759, 2009.

[111] René Henrion. N-way principal component analysis theory, algorithms and applications. Chemometrics and intelligent laboratory systems, 25(1):1–23, 1994.

[112] Lieven De Lathauwer and Joos Vandewalle. Dimensionality reduction in higher-order signal processing and rank-(r1,r2,. . . ,rn) reduction in multilinear algebra. Linear Algebra and its Applications, 391:31–55, 2004. Special Issue on Linear Algebra in Signal and Image Processing.

[113] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In European conference on computer vision, pages 447–460. Springer, 2002.

[114] Hongcheng Wang and N. Ahuja. Compact representation of multidimensional data using tensor rank-one decomposition. In Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., volume 1, pages 44–47 Vol.1, 2004.

[115] Wolfgang Hackbusch and Stefan Kühn. A new scheme for the tensor representation. Journal of Fourier analysis and applications, 15(5):706–722, 2009.

[116] Lars Grasedyck and Wolfgang Hackbusch. An introduction to hierarchical (h-) rank and tt-rank of tensors with examples. Computational methods in applied mathematics, 11(3):291–304, 2011.

[117] Anh-Huy Phan, Andrzej Cichocki, André Uschmajew, Petr Tichavskỳ, George Luta, and Danilo P Mandic. Tensor networks for latent variable analysis: Novel algorithms for tensor train approximation. IEEE transactions on neural networks and learning systems, 31(11):4622–4636, 2020.

[118] I. Oseledets. Tensor-train decomposition. SIAM J. Sci. Comput., 33:2295–2317, 2011.

[119] Cécile Haberstich. Adaptive approximation of high-dimensional functions with tree tensor networks for Uncertainty Quantification. PhD thesis, École centrale de Nantes, 2020.

[120] Markus Bachmayr, Reinhold Schneider, and André Uschmajew. Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. Foundations of Computational Mathematics, 16(6):1423–1472, 2016.

[121] Antonio Falcó and Wolfgang Hackbusch. On minimal subspaces in tensor representations. Foundations of computational mathematics, 12(6):765–803, 2012.

[122] Antonio Falcó, Wolfgang Hackbusch, and Anthony Nouy. On the dirac–frenkel variational principle on tensor banach spaces. Foundations of computational mathematics, 19(1):159–204, 2019.

[123] Antonio Falcó, Wolfgang Hackbusch, and Anthony Nouy. Geometric structures in tensor representations (final release). arXiv preprint arXiv:1505.03027, 2015.

[124] Mazen Ali and Anthony Nouy. Approximation with tensor networks. part i: Approximation spaces. arXiv preprint arXiv:2007.00118, 2020.

[125] Mazen Ali and Anthony Nouy. Approximation with tensor networks. part ii: Approximation rates for smoothness classes. arXiv preprint arXiv:2007.00128, 2020.

[126] Erwan Grelier, Anthony Nouy, and Mathilde Chevreuil. Learning with tree-based tensor formats. arXiv preprint arXiv:1811.04455, 2018.

[127] Cécile Haberstich, Anthony Nouy, and Guillaume Perrin. Active learning of tree tensor networks using optimal least-squares. arXiv preprint arXiv:2104.13436, 2021.

[128] Ivan Oseledets and Eugene Tyrtyshnikov. Tt-cross approximation for multidimensional arrays. Linear Algebra and its Applications, 432(1):70–88, 2010.

[129] Guifré Vidal. Efficient classical simulation of slightly entangled quantum computations. Physical review letters, 91(14):147902, 2003.

[130] Lars Grasedyck and Sebastian Krämer. Stable als approximation in the tt-format for rank-adaptive tensor completion. Numerische Mathematik, 143(4):855–904, 2019.

[131] Thorsten Rohwedder and André Uschmajew. On local convergence of alternating schemes for optimization of convex problems in the tensor train format. SIAM Journal on Numerical Analysis, 51(2):1134–1162, 2013.

[132] Boris N Khoromskij. O (dlog n)-quantics approximation of nd tensors in high-dimensional numerical modeling. Constructive Approximation, 34(2):257–280, 2011.

[133] Sergey V Dolgov, Boris N Khoromskij, Ivan V Oseledets, and Dmitry V Savostyanov. Computation of extreme eigenvalues in higher dimensions using block tensor train format. Computer Physics Communications, 185(4):1207–1216, 2014.

[134] Richard E Bellman and Stuart E Dreyfus. Applied dynamic programming, volume 2050. Princeton university press, 2015.

[135] Allan Pinkus. N-widths in Approximation Theory, volume 7. Springer Science & Business Media, 2012.

[136] Annalisa Buffa, Yvon Maday, Anthony T Patera, Christophe Prud'homme, and Gabriel Turinici. A priori convergence of the greedy algorithm for the parametrized reduced basis method. ESAIM: Mathematical modelling and numerical analysis, 46(3):595–603, 2012.

[137] Seaar Al-Dabooni and Donald Wunsch. Model order reduction based on agglomerative hierarchical clustering. IEEE transactions on neural networks and learning systems, 30(6):1881–1895, 2018.

[138] Thomas Daniel, Fabien Casenave, Nissrine Akkari, and David Ryckelynck. Model order reduction assisted by deep neural networks (rom-net). Advanced Modeling and Simulation in Engineering Sciences, 7(1):1–27, 2020.

[139] Olivier Zahm, Marie Billaud-Friess, and Anthony Nouy. Projection-based model order reduction methods for the estimation of vector-valued variables of interest. SIAM Journal on Scientific Computing, 39(4):A1647–A1674, 2017.

[140] Clarence W Rowley, Tim Colonius, and Richard M Murray. Model reduction for compressible flows using pod and galerkin projection. Physica D: Nonlinear Phenomena, 189(1-2):115–129, 2004.

[141] Christophe Prud'Homme, Dimitrios V Rovas, Karen Veroy, Luc Machiels, Yvon Maday, Anthony T Patera, and Gabriel Turinici. Reliable real-time solution of parametrized partial differential equations: Reduced-basis output bound methods. J. Fluids Eng., 124(1):70–80, 2002.

[142] Gianluigi Rozza, Dinh Bao Phuong Huynh, and Anthony T Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. Archives of Computational Methods in Engineering, 15(3):229–275, 2008.

[143] Guillaume Perrin, Christian Soize, Denis Duhamel, and Christine Funfschilling. A posteriori error and optimal reduced basis for stochastic processes defined by a finite set of realizations. SIAM/ASA Journal on Uncertainty Quantification, 2(1):745–762, 2014.

[144] Antonio Falcó, Lucia Hilario, Nicolás Montés, and Marta C Mora. Numerical strategies for the galerkin–proper generalized decomposition method. Mathematical and Computer Modelling, 57(7-8):1694–1702, 2013.

[145] Vladimir N Temlyakov. Nonlinear methods of approximation. Foundations of Computational Mathematics, 3(1), 2003.

[146] John A Lee and Michel Verleysen. Nonlinear dimensionality reduction, volume 1. Springer, 2007.

[147] Ronald A DeVore. Nonlinear approximation. Acta numerica, 7:51–150, 1998.

[148] Virginie Ehrlacher, Damiano Lombardi, Olga Mula, and François-Xavier Vialard. Nonlinear model reduction on metric spaces. application to one-dimensional conservative pdes in wasserstein spaces. ESAIM: Mathematical Modelling and Numerical Analysis, 54(6):2159–2197, 2020.

[149] Andrea Bonito, Albert Cohen, Ronald DeVore, Diane Guignard, Peter Jantsch, and Guergana Petrova. Nonlinear methods for model reduction. ESAIM: Mathematical Modelling and Numerical Analysis, 55(2):507–531, 2021.

[150] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In International conference on artificial neural networks, pages 583–588. Springer, 1997.

[151] Heiko Hoffmann. Kernel pca for novelty detection. Pattern recognition, 40(3):863–874, 2007.

[152] Sebastian Mika, Bernhard Schölkopf, Alex Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel pca and de-noising in feature spaces. Advances in neural information processing systems, 11, 1998.

[153] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. science, 313(5786):504–507, 2006.

[154] Matthias Scholz and Ricardo Vigário. Nonlinear pca: a new hierarchical approach. In Esann, pages 439–444, 2002.

[155] David Amsallem, Matthew J Zahr, and Kyle Washabaugh. Fast local reduced basis updates for the efficient reduction of nonlinear systems with hyper-reduction. Advances in Computational Mathematics, 41(5):1187–1230, 2015.

[156] David Amsallem, Matthew J Zahr, and Charbel Farhat. Nonlinear model order reduction based on local reduced-order bases. International Journal for Numerical Methods in Engineering, 92(10):891–916, 2012.

[157] Kyle Washabaugh, David Amsallem, Matthew Zahr, and Charbel Farhat. Nonlinear model reduction for cfd problems using local reduced-order bases. In 42nd AIAA Fluid Dynamics Conference and Exhibit, page 2686, 2012.

[158] Jeffrey Ho, Ming-Husang Yang, Jongwoo Lim, Kuang-Chih Lee, and David Kriegman. Clustering appearances of objects under varying illumination conditions. In 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings., volume 1, pages I–I. IEEE, 2003.

[159] Aleš Leonardis, Horst Bischof, and Jasna Maver. Multiple eigenspaces. Pattern recognition, 35(11):2613–2627, 2002.

[160] Todd K Moon. The expectation-maximization algorithm. IEEE Signal processing magazine, 13(6):47–60, 1996.

[161] Michael E Tipping and Christopher M Bishop. Mixtures of probabilistic principal component analysers. 1998.

[162] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. Chemometrics and intelligent laboratory systems, 50(1):1–18, 2000.

[163] Gunilla Borgefors. Distance transformations in digital images. Computer vision, graphics, and image processing, 34(3):344–371, 1986.

[164] Allan D Gordon. A review of hierarchical classification. Journal of the Royal Statistical Society: Series A (General), 150(2):119–137, 1987.

[165] Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. The computer journal, 16(1):30–34, 1973.

[166] Stuart Lloyd. Least squares quantization in pcm. IEEE transactions on information theory, 28(2):129–137, 1982.

[167] J MacQueen. Classification and analysis of multivariate observations. In 5th Berkeley Symp. Math. Statist. Probability, pages 281–297, 1967.

[168] Charu C Aggarwal et al. Data mining: the textbook, volume 1. Springer, 2015.

[169] LKPJ Rdusseeun and P Kaufman. Clustering by means of medoids. In Proceedings of the statistical data analysis based on the L1 norm conference, neuchatel, switzerland, volume 31, 1987.

[170] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. Wiley interdisciplinary reviews: data mining and knowledge discovery, 1(3):231–240, 2011.

[171] Petr Novák, Pavel Neumann, and Jiří Macas. Graph-based clustering and characterization of repetitive sequences in next-generation sequencing data. BMC bioinformatics, 11(1):1–12, 2010.

[172] Amit Saxena, Mukesh Prasad, Akshansh Gupta, Neha Bharill, Om Prakash Patel, Aruna Tiwari, Meng Joo Er, Weiping Ding, and Chin-Teng Lin. A review of clustering techniques and developments. Neurocomputing, 267:664–681, 2017.

[173] Anil K Jain, M Narasimha Murty, and Patrick J Flynn. Data clustering: a review. ACM computing surveys (CSUR), 31(3):264–323, 1999.

[174] Mark K Goldberg, Mykola Hayvanovych, and Malik Magdon-Ismail. Measuring similarity between sets of overlapping clusters. In 2010 IEEE Second International Conference on Social Computing, pages 303–308. IEEE, 2010.

[175] Thomas Daniel. Machine learning for nonlinear model order reduction. PhD thesis, Université Paris sciences et lettres, 2021.

[176] Thomas Daniel, Fabien Casenave, Nissrine Akkari, Ali Ketata, and David Ryckelynck. Physics-informed cluster analysis and a priori efficiency criterion for the construction of local reduced-order bases. Journal of Computational Physics, 458:111120, 2022.

[177] Thomas Daniel, Fabien Casenave, Nissrine Akkari, and David Ryckelynck. Optimal piecewise linear data compression for solutions of parametrized partial differential equations. arXiv preprint arXiv:2108.12291, 2021.

[178] David Amsallem and Bernard Haasdonk. Pebl-rom: Projection-error based local reduced-order models. Advanced Modeling and Simulation in Engineering Sciences, 3(1):1–25, 2016.

[179] Benjamin Peherstorfer, Daniel Butnaru, Karen Willcox, and Hans-Joachim Bungartz. Localized discrete empirical interpolation method. SIAM Journal on Scientific Computing, 36(1):A168–A192, 2014.

[180] Harry Zhang. The optimality of naive bayes. Aa, 1(2):3, 2004.

[181] Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. Machine learning, 73(2):185–214, 2008.

[182] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. IEEE transactions on information theory, 13(1):21–27, 1967.

[183] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. The elements of statistical learning: data mining, inference, and prediction, volume 2. Springer, 2009.

[184] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. Supervised machine learning: A review of classification techniques. Emerging artificial intelligence applications in computer engineering, 160(1):3–24, 2007.

[185] Damiano Lombardi and Fabien Raphel. A greedy dimension reduction method for classification problems. 2019.

[186] Evelyn Fix and Joseph Lawson Hodges. Discriminatory analysis. nonparametric discrimination: Consistency properties. International Statistical Review/Revue Internationale de Statistique, 57(3):238–247, 1989.

[187] Peter Hart. The condensed nearest neighbor rule (corresp.). IEEE transactions on information theory, 14(3):515–516, 1968.

[188] Leif E Peterson. K-nearest neighbor. Scholarpedia, 4(2):1883, 2009.

[189] Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving the kolmogorov pde by means of deep learning. Journal of Scientific Computing, 88(3):1–28, 2021.

[190] Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6(1):1–12, 2018.

[191] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. Proceedings of the National Academy of Sciences, 115(34):8505–8510, 2018.

[192] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of computational physics, 375:1339–1364, 2018.

[193] Stefania Fresca and Andrea Manzoni. Pod-dl-rom: enhancing deep learning-based reduced order models for nonlinear parametrized pdes by proper orthogonal decomposition. Computer Methods in Applied Mechanics and Engineering, 388:114181, 2022.

[194] Gitta Kutyniok, Philipp Petersen, Mones Raslan, and Reinhold Schneider. A theoretical analysis of deep neural networks and parametric pdes. Constructive Approximation, 55(1):73–125, 2022.

[195] Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. ACS central science, 4(1):120–131, 2018.

[196] Dennis Elbrächter, Philipp Grohs, Arnulf Jentzen, and Christoph Schwab. Dnn expression rate analysis of high-dimensional pdes: Application to option pricing. Constructive Approximation, 55(1):3–71, 2022.

[197] E Weinan. A proposal on machine learning via dynamical systems. Communications in Mathematics and Statistics, 1(5):1–11, 2017.

[198] Jiequn Han, Arnulf Jentzen, et al. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Communications in mathematics and statistics, 5(4):349–380, 2017.

[199] G. Cybenko. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems, 2(4):303–314, 1989.

[200] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. Neural networks, 2(5):359–366, 1989.

[201] Helmut Bolcskei, Philipp Grohs, Gitta Kutyniok, and Philipp Petersen. Optimal approximation with sparsely connected deep neural networks. SIAM Journal on Mathematics of Data Science, 1(1):8–45, 2019.

[202] C. Ma, S. Wojtowytsch, L. Wu, et al. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. arXiv preprint arXiv:2009.10713, 2020.

[203] Ingrid Daubechies, Ronald DeVore, Simon Foucart, Boris Hanin, and Guergana Petrova. Nonlinear approximation and (deep) relu networks. Constructive Approximation, 55(1):127–172, 2022.

[204] Lukas Gonon and Christoph Schwab. Deep relu neural networks overcome the curse of dimensionality for partial integrodifferential equations. arXiv preprint arXiv:2102.11707, 2021.

[205] Johannes Schmidt-Hieber. Nonparametric regression using deep neural networks with relu activation function. The Annals of Statistics, 48(4):1875–1897, 2020.

[206] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378:686–707, 2019.

[207] Ameya D Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. Computer Methods in Applied Mechanics and Engineering, 365:113028, 2020.

[208] Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. Variational physics-informed neural networks for solving partial differential equations. arXiv preprint arXiv:1912.00873, 2019.

[209] E. Kharazmi, Z. Zhang, and G. E. Karniadakis. Variational physics-informed neural networks for solving partial differential equations. arXiv preprint arXiv:1912.00873, 2019.

[210] E. Kharazmi, Z. Zhang, and G.E. Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering, 374:113547, 2021.

[211] Dongkun Zhang, Lu Lu, Ling Guo, and George Em Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. Journal of Computational Physics, 397:108850, 2019.

[212] Lawrence C Evans. Partial differential equations, volume 19. American Mathematical Soc., 2010.

[213] Walter Ritz. Über eine neue methode zur lösung gewisser variationsprobleme der mathematischen physik. 1909.

[214] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. Communications in Mathematics and Statistics, 5(4):349–380, 2017.

[215] W. E and B. Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. Communications in Mathematics and Statistics, 6(1):1–12, 2018.

[216] G. M. Rotskoff, A. R. Mitchell, and E. Vanden-Eijnden. Active importance sampling for variational objectives dominated by rare events: Consequences for optimization and generalization. arXiv preprint arXiv:2008.06334, 2020.

[217] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378:686–707, 2019.

[218] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34:26548–26560, 2021.

[219] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data. arXiv preprint arXiv:1808.04327, 2018.

[220] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Proceedings of COMPSTAT'2010, pages 177–186. Springer, 2010.

[221] Justin Sirignano and Konstantinos Spiliopoulos. Stochastic gradient descent in continuous time. SIAM Journal on Financial Mathematics, 8(1):933–961, 2017.

[222] J. Sirignano and K. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. Journal of computational physics, 375:1339–1364, 2018.

[223] Jingrun Chen, Rui Du, and Keke Wu. A comparison study of deep galerkin method and deep ritz method for elliptic problems with different boundary conditions. arXiv preprint arXiv:2005.04554, 2020.

[224] Virginie Ehrlacher, Maria Fuente Ruiz, and Damiano Lombardi. Sott: greedy approximation of a tensor as a sum of tensor trains. SIAM Journal on Scientific Computing, 44(2):A664–A688, 2022.

[225] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. Journal of the American statistical association, 58(301):236–244, 1963.

[226] Adrien Beguinet, Virginie Ehrlacher, Roberta Flenghi, O Mula, A Somacal, et al. Deep learning-based schemes for singularly perturbed convection-diffusion problems. arXiv preprint arXiv:2205.04779, 2022.

[227] H. Risken. Fokker-planck equation. In The Fokker-Planck Equation, pages 63–95. Springer, 1996.

[228] R. Jordan, D. Kinderlehrer, and F. Otto. The variational formulation of the fokker–planck equation. SIAM journal on mathematical analysis, 29(1):1–17, 1998.

[229] V.I. Bogachev, N.V. Krylov, M. Röckner, and S.V. Shaposhnikov. Fokker-Planck-Kolmogorov Equations, volume 207. American Mathematical Soc., 2015.

[230] Dongbin Xiu and Jan S Hesthaven. High-order collocation methods for differential equations with random inputs. SIAM Journal on Scientific Computing, 27(3):1118–1139, 2005.

[231] Jinchao Xu and Ludmil Zikatanov. A monotone finite element scheme for convection-diffusion equations. Mathematics of Computation, 68(228):1429–1446, 1999.

[232] BR Baliga and SV Patankar. A new finite-element formulation for convection-diffusion problems. Numerical Heat Transfer, 3(4):393–409, 1980.

[233] Kenneth Eriksson and Claes Johnson. Adaptive streamline diffusion finite element methods for stationary convection-diffusion problems. mathematics of computation, 60(201):167–188, 1993.

[234] Juncai He, Lin Li, Jinchao Xu, and Chunyue Zheng. Relu deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973, 2018.

[235] Raman Arora, Amitabh Basu, Poorya Mianjy, and Anirbit Mukherjee. Understanding deep neural networks with rectified linear units. arXiv preprint arXiv:1611.01491, 2016.

[236] Gregory Beylkin and Martin J. Mohlenkamp. Algorithms for numerical analysis in high dimensions. SIAM J. SCI. COMPUT, 26:2133–2159, 2005.

[237] M. Vandecappelle, N. Vervliet, and L. De Lathauwer. Nonlinear least squares updating of the canonical polyadic decomposition. In 2017 25th European Signal Processing Conference (EUSIPCO), pages 663–667, 2017.

[238] M. Rajih, P. Comon, and R. Harsman. Enhanced line search: A novel method to accelerate parafac. SIAM Journal on Matrix Analysis and Applications, 30, September 2008.

[239] Daniele Bigoni, Allan P Engsig-Karup, and Youssef M Marzouk. Spectral tensor-train decomposition. SIAM Journal on Scientific Computing, 38(4):A2405–A2439, 2016.

[240] Xiaokang Wang, Laurence T Yang, Yihao Wang, Xingang Liu, Qingxia Zhang, and M Jamal Deen. A distributed tensor-train decomposition method for cyber-physical-social services. ACM Transactions on Cyber-Physical Systems, 3(4):1–15, 2019.

[241] Maxim Rakhuba and Ivan Oseledets. Calculating vibrational spectra of molecules using tensor train decomposition. The Journal of Chemical Physics, 145(12):124101, 2016.

[242] Anh-Huy Phan, Konstantin Sobolev, Konstantin Sozykin, Dmitry Ermilov, Julia Gusak, Petr Tichavsky, Valeriy Glukhov, Ivan Oseledets, and Andrzej Cichocki. Stable low-rank tensor decomposition for compression of convolutional neural network. ECCV2020, August 2020.

[243] Kirandeep Kour, Sergey Dolgov, Martin Stoll, and Peter Benner. Efficient structure-preserving support tensor train machine. arXiv preprint arXiv:2002.05079, 2020.

[244] Laurent Sorber, Marc Van Barel, and Lieven De Lathauwer. Optimization-based algorithms for tensor decompositions: Canonical polyadic decomposition, decomposition in rank-(l_r,l_r,1) terms, and a new generalization. SIAM Journal on Optimization, 23(2):695–720, 2013.

[245] Vladimir N Temlyakov. Greedy algorithms andm-term approximation with regard to redundant dictionaries. Journal of Approximation Theory, 98(1):117–145, 1999.

[246] Werner H Greub. Linear algebra, volume 23. Springer Science & Business Media, 2012.

[247] Vin De Silva and Lek-Heng Lim. Tensor rank and the ill-posedness of the best low-rank approximation problem. SIAM Journal on Matrix Analysis and Applications, 30(3):1084–1127, 2008.

[248] Gene H Golub and Charles F Van Loan. Matrix computations. JHU press, 2013.

[249] Christopher J Hillar and Lek-Heng Lim. Most tensor problems are np-hard. Journal of the ACM (JACM), 60(6):1–39, 2013.

[250] Dario Bini. Border rank of m× n×(mn- q) tensors. Linear Algebra and Its Applications, 79:45–51, 1986.

[251] Dario Bini. Border rank of ap× q× 2 tensor and the optimal approximation of a pair of bilinear forms. In International Colloquium on Automata, Languages, and Programming, pages 98–108. Springer, 1980.

[252] J Landsberg. The border rank of the multiplication of 2× 2 matrices is seven. Journal of the American Mathematical Society, 19(2):447–459, 2006.

[253] Wim P Krijnen, Theo K Dijkstra, and Alwin Stegeman. On the non-existence of optimal solutions and the occurrence of "degeneracy" in the candecomp/parafac model. Psychometrika, 73(3):431–439, 2008.

[254] Mike Espig, Wolfgang Hackbusch, and Aram Khachatryan. On the convergence of alternating least squares optimisation in tensor format representations. arXiv preprint arXiv:1506.00062, 2015.

[255] Xiaofei Wang, Carmeliza Navasca, and Stefan Kindermann. On accelerating the regularized alternating least square algorithm for tensors. arXiv preprint arXiv:1507.04721, 2015.

[256] Ivan V Oseledets, Maxim V Rakhuba, and André Uschmajew. Alternating least squares as moving subspace correction. SIAM Journal on Numerical Analysis, 56(6):3459–3479, 2018.

[257] Shmuel Friedland, Volker Mehrmann, Renato Pajarola, and Susanne K Suter. On best rank one approximation of tensors. Numerical Linear Algebra with Applications, 20(6):942–955, 2013.

[258] Kim Batselier, Haotian Liu, and Ngai Wong. A constructive algorithm for decomposing a tensor into a finite sum of orthonormal rank-1 terms. SIAM Journal on Matrix Analysis and Applications, 36(3):1315–1337, 2015.

[259] Tong Zhang and Gene H Golub. Rank-one approximation to high order tensors. SIAM Journal on Matrix Analysis and Applications, 23(2):534–550, 2001.

[260] B Khoromskij and Venera Khoromskaia. Low rank tucker-type tensor approximation to classical potentials. Open Mathematics, 5(3):523–550, 2007.

[261] Steen Krenk and Jan Høgsberg. Statics and mechanics of structures. Springer Science & Business Media, 2013.

[262] Lance Parsons, Ehtesham Haque, and Huan Liu. Subspace clustering for high dimensional data: a review. Acm sigkdd explorations newsletter, 6(1):90–105, 2004.

[263] Alexander Hinneburg and Daniel A Keim. Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. 1999.

[264] Virginie Ehrlacher, Laura Grigori, Damiano Lombardi, and Hao Song. Adaptive hierarchical subtensor partitioning for tensor compression. SIAM Journal on Scientific Computing, 43(1):A139–A163, 2021.

[265] Matthew Brand. Fast low-rank modifications of the thin singular value decomposition. Linear algebra and its applications, 415(1):20–30, 2006.

[266] Matthew Brand. Fast online svd revisions for lightweight recommender systems. In Proceedings of the 2003 SIAM international conference on data mining, pages 37–46. SIAM, 2003.

[267] Ke Ye and Lek-Heng Lim. Distance between subspaces of different dimensions. arXiv preprint arXiv:1407.0900, 4:4, 2014.

[268] Ke Ye and Lek-Heng Lim. Schubert varieties and distances between subspaces of different dimensions. SIAM Journal on Matrix Analysis and Applications, 37(3):1176–1197, 2016.

[269] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2(1):86–97, 2012.

[270] Min-Ling Zhang and Zhi-Hua Zhou. Ml-knn: A lazy learning approach to multi-label learning. Pattern recognition, 40(7):2038–2048, 2007.

[271] Qianqian Zheng and Jianwei Shen. Pattern formation in the fitzhugh–nagumo model. Computers & Mathematics with Applications, 70(5):1082–1097, 2015.

[272] H. Lee and I. S. Kang. Neural algorithm for solving differential equations. Journal of Computational Physics, 91(1):110–131, 1990.

[273] L. Wang and J. M. Mendel. Structured trainable networks for matrix algebra. In 1990 IJCNN International Joint Conference on Neural Networks, pages 125–132. IEEE, 1990.

[274] B. Després and H. Jourdren. Machine learning design of volume of fluid schemes for compressible flows. Journal of Computational Physics, 408:109275, 2020.

[275] Sumeet Trehan, Kevin T Carlberg, and Louis J Durlofsky. Error modeling for surrogates of dynamical systems using machine learning. International Journal for Numerical Methods in Engineering, 112(12):1801–1827, 2017.

[276] Brian A Freno and Kevin T Carlberg. Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations. Computer Methods in Applied Mechanics and Engineering, 348:250–296, 2019.

[277] H.-G. Roos, M. Stynes, and L. Tobiska. Robust numerical methods for singularly perturbed differential equations: convection-diffusion-reaction and flow problems, volume 24. Springer Science & Business Media, 2008.

[278] A. N Brooks and T. JR. Hughes. Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. Computer methods in applied mechanics and engineering, 32(1-3):199–259, 1982.

[279] Leszek Demkowicz and Norbert Heuer. Robust DPG method for convection-dominated diffusion problems. SIAM Journal on Numerical Analysis, 51(5):2514–2537, 2013.

[280] J. Chan, N. Heuer, T. Bui-Thanh, and L. Demkowicz. A robust dpg method for convection-dominated diffusion problems ii: Adjoint boundary conditions and mesh-dependent test norms. Computers & Mathematics with Applications, 67(4):771–795, 2014.

[281] N. Kopteva and E. O'Riordan. Shishkin meshes in the numerical solution of singularly perturbed differential equations. 2010.

[282] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, and et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv, 2016.

[283] J. Nocedal D. C. Liu. On the limited memory bfgs method for large scale optimization. Mathematical programming, 45:503–528, 1989.