Low Rank Approximation of Multidimensional Data

Mejdi Azaiez [*‡*] , Lucas Lestandi [†‡†] and Tomás Chacón Rebollo [♠‡]

[*] Bordeaux Institut National Polytechnique de Bordeaux, France

[†] Université de Bordeaux, Bordeaux, France

[‡] Institut de Mécanique et d'Ingénierie, UMR 5295, Bordeaux, France

[♠] Instituto de Matemáticas de la Universidad de Sevilla - IMUS, Sevilla, Spain

Abstract In the last decades, numerical simulation has experienced tremendous improvements driven by massive growth of computing power. Exascale computing has been achieved this year and will allow solving ever more complex problems. But such large systems produce colossal amounts of data which leads to its own difficulties. Moreover, many engineering problems such as multiphysics or optimisation and control, require far more power that any computer architecture could achieve within the current scientific computing paradigm. In this chapter, we propose to shift the paradigm in order to break the curse of dimensionality by introducing decomposition to reduced data. We present an extended review of data reduction techniques and intends to bridge between applied mathematics community and the computational mechanics one.

The chapter is organized into two parts. In the first one bivariate separation is studied, including discussions on the equivalence of proper orthogonal decomposition (POD, continuous framework) and singular value decomposition (SVD, discrete matrices). Then, in the second part, a wide review of tensor formats and their approximation is proposed. Such work has already been provided in the literature but either on separate papers or into a pure applied mathematics framework. Here, we offer to the data enthusiast scientist a description of Canonical, Tucker, Hierarchical and Tensor train formats including their approximation algorithms. When it is possible, a careful analysis of the link between continuous and discrete methods will be performed.

[*]The research of Mejdi Azaez was partially funded by the IV Research and Transfer Plan of the University of Sevilla.

[†]The research of Lucas Lestandi was partially funded by the Institut Carnot ARTZ.

[‡]The research of Toms Chacn was partially funded by Junta de Andalucia - Feder Fund Grant FQM 454, and by the IDEX program of the University of Bordeaux.

Key words: Data reduction, Model Reduction, Singular Values Decomposition, Data, MOR, POD, HOSVD, Low rank approximation, tensors, Tensor train.

1 Introduction

In the last 50 years, scientific computing has become a central tool in engineering design, especially in the mechanics field. A constant improvement in simulation techniques has accompanied the rocketing computing power embedded in Moore's law[1]. This explosion of CPU power was magnified by the introduction of supercomputers and their massively parallel architectures. Although some slowdown has been observed, this trend will continue, especially with the arrival of breakthrough technologies such as the much awaited quantum computer. Still, the advent of exascale computing has only pushed forward the boundaries of computable problems slightly while raising a series of technical issues. First, supercomputers are really expensive infrastructures that require huge amounts of energy[2]. Second, they produce data so large that storing and transferring data itself has become an issue. A famous simulation of the observable universe **?**] performed in 2012, exemplifies the dizzying proportions taken by numerical simulation. Approximately 5000 computing nodes used 300 TB of memory producing 50 PB of raw data in 10 million hours of computing time of which "only" 500 TB of useful data was finally kept. This kind of data is hard to manipulate and storage is usually performed on magnetic bands making it fairly slow to access. Also, any intent at handling such data, even in small slices, is vain on a personal computer, thus impairing the efficiency of analysis.

Actually, the framework of building numerical models has remained the same across the period of popularization of numerical simulation. This process has been finely tuned, improving gradually the quality and confidence in the simulations. This technology is now massively used in the industry, especially for designing new products that require precise knowledge in fields such as mechanics, thermodynamics, chemistry, electromagnetic fields, etc. In particular, computational fluid dynamics has become a central tool in designing new aircrafts, ranging from global flow around a plane to multiphysics-multiscale combustion inside the jet engine.

Building a direct model, also known as full order model (FOM), usually involves the following steps. First, one needs to select the adequate equations from basic physics laws and define carefully the limits of simulation.

[1]Gordon Moore predicted in 1965 that the density of transistors on chips would double every year. After being slightly downgraded to doubling every 18 month, it has been verified from 1975 to 2012. Current trend shows a slowing pace. Still, this exponential growth amounts to a 20 millions factor. Naturally, it corresponds to the computing power gain.

[2]As of June 2018, the largest supercomputer is the Summit at Oak Ridge, USA, with more than 2 million cores it requires 8MW for a peak performance of 122PFlop/s

Depending on the problem geometry, characteristic sizes and phenomena[3], one chooses the simplest equations set that captures the physics correctly. Then these equations are discretized in time and space while numerical schemes are used to solve the constructed discrete problems. Whether one uses finite differences, finite elements or finite volumes, the problem usually boils down to a linear algebra problem

$$Ax = b$$

where A is a $n \times n$ matrix, x is the unknown vector of size n and b the right hand side term of size n. Here, n is the number of discrete space points that typically range from millions for 2D to billion for high end 3D problems. Moreover, this linear problem has to be solved at each time step, often millions of times, in spite of typically costing $\mathcal{O}(n^2)$ floating point operations. More often than not, if one wants to simulate several interacting physical phenomena, they occur at different time and space scales, meaning that one needs to solve several concurrent problems of this kind. With the figures stated above, it becomes clear direct numerical simulation (DNS) is expensive. Consequently, problems involving to perform such simulations multiple times such as optimisation or control, remain out of reach.

It has spawned a vast body of literature on how to make these simulations more affordable. Among the typical solutions in fluid dynamics, Reynolds averaged Navier-Stokes methods (RANS) and Large eddy simulation (LES) have been very successful at capturing large structures and modeling (with more or less empirical terms) the smaller structures. These solutions however generate a great loss of information as it is impossible to know how the energy dissipation occurs in the small scale structures. To some, extent it prevents relevant simulations in which the interaction of small structures drive large scale behavior i.e. chaotic systems. Many models, in all areas of numerical simulation, have been proposed to reduce the computing cost with the same idea of modeling the most expensive terms of equation while retaining the same basic principles of discretization. We observe that, within this approach, the curse of dimensionality remains the main obstacle to scientific computing development. For instance, let the number of discrete points needed to capture a phenomenon on one dimension be $n = 1000$. Now, if the problem is 3D, the cube is discretized with $n^3 = 10^9$ points. If the phenomenon is actually a dynamic one, time has to be accounted for, which means an additional dimension. The discrete

[3] A typical example in fluid dynamics is Reynolds number $Re = UL/\mu$ which characterize the relative influence of inertia (U is a typical flow velocity and L a typical length) compared with viscosity (μ the kinematic viscosity.)

space time is now $n^4 = 10^{12}$ that amounts terabytes of data for double precision real numbers. Additionally, one might want to add a few parameters on which the simulation depends and both the computing time and storage cost become out of reach. Even with very small n, for instance n=2, this kind of difficulty emerges quickly. For example, with $d = 50$ (which is far below computational chemistry requirements), storage cost of $n^d = 2^{50}$ amounts to 9PB if all entries are stored. A tensor is a well suited object for such data representation, it is the discrete representation of multidimensional fields, i.e. an order d tensor of size $n_1 \times \cdots \times n_d$ is filled by sampling a field on a tensor product space $\Omega = [0,1]^d$ at discrete grid points. The necessity of storing low rank approximate tensors instead of keeping all the entries becomes essential in this context.

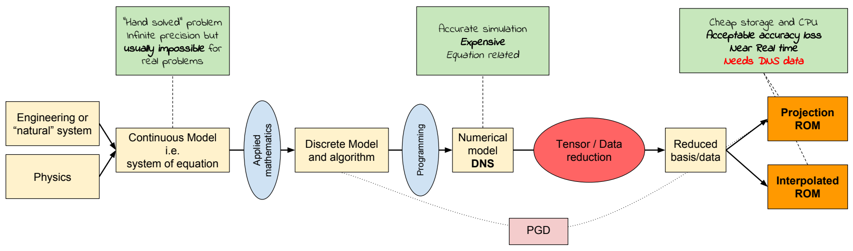


Figure 1: Scientific computing workflow enriched with tensor reduction and reduced order modeling

Finally, Fig. 1 summarizes the dominant work-flow in scientific computing i.e. physics modeling is followed by discretization techniques that can produce reliable simulation. The introduction of a new paradigm is represented here by tensor decomposition and the following steps of ROM.

In this chapter, we will explore branches that tackle the issue of how to reduce multidimensional data. It is divided into two parts, the first one provides a detailed presentation of bivariate decomposition techniques and points out to the fundamental equivalence of these methods. Next, the multivariate problem decomposition is treated in the second part. Our objective is to offer a comprehensive synthesis of decomposition methods from bivariate to multivariate data including both tensors and functions frameworks.

We start by presenting a selected state of the art in these fields.

State of the art

The need for order reduction is as old as numerical simulation, for instance matrix analysis techniques such as eigen value decomposition or singular value decomposition (SVD) have been used in the past centuries to capture structure in complex matrices. It turns out that the bivariate decomposition methods in principle equivalent to SVD but complies with their field formalism. Actually, they have been rediscovered many times in various fields: it is known as principal component analysis (PCA) in statistics [? ?], Karhunen-Love expansion (KLE) in probability theory [?] or proper orthogonal decomposition (POD) in fluid dynamics [? ?]. These methods, by themselves, provide a decomposition that can be truncated with optimality results [?] and reflect the physics of the problem studied. The first wave of reduced order models (ROM) in mechanics is a consequence of POD. Indeed this decomposition provides, among the many possible bases [?], an orthogonal basis of the functional space in which the solution problem lives. Consequently, many attempts at building Galerkin projection ROM on these reduced bases from the 1980s onward [? ? ? ? ?] followed with modest success. Indeed, in this approach, the weak form EDP is solved against test function in the selected basis. In order to decrease the size of the problem, one has to truncate the basis to a relatively small rank which means, in the case of fluid flows, that the small structures are lost.

Concerning the multidimensional case, Hitchcock [?] usually considered to have introduced tensor decomposition in 1927. But, it is Tucker [?] that popularized the subject in the 1960s, followed by Carroll and Chang [?] and Harshmann [?] in 1970. As for the bivariate decomposition, much of the research happened independently in several fields starting by psychometrics and chemometrics. A complete history is available in Kolda and Balder review paper [?]. This large overview of tensor formats includes canonical format ([? ?]) and Tucker format with the associated decomposition methods. The former has received dwindling interest due to poor numerical performance. Tucker format was at the center of attention since DeLathauwer paper in 2000 [?] which proposed an efficient approximation strategy, the Higher Order SVD (HOSVD) followed by HOOI [?]. More recently, he coauthored Vannieuwenhoven ST-HOSVD [?] that improved significantly the computing time. The early 2010s have seen the introduction of formats that overcome the exponential growth of the core tensor in Tucker format. Oseledets proposed the tensor train (TT) format [? ? ?] , also known as matrix product state (MPS), together with its decomposition algorithm. The storage cost of this format is linear in d allowing tensorization of data, i.e. the method is so efficient at handling large d that a new strategy consists in increasing artificially the number of dimensions.

To do so, one may need to rely on partial evaluations of the target field, `TT-DMRG-cross` performs this task[? ?]. This approach is also known as blackbox algorithms [?] in the context of hierarchical tensors (HT) developed by Grasedyck, Kessner and Tobler [? ?]. HT actually incorporates all previously mentioned formats and approximations into a general d-linear format. These recent developments have been reviewed in [?] while an extensive mathematical analysis of tensors and their approximation is given in Hackbush's book [?]. A selection of publicly available libraries will be discussed in detail in section 3.2.

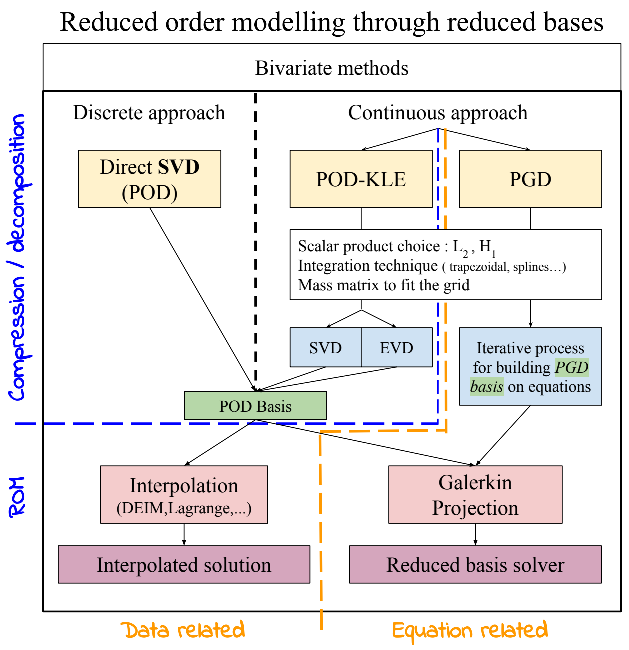
Finally, these formats have been extended to the continuous framework as they are often used to separate data representing functions. A functional TT was proposed by Bigoni and Gorodetsky [? ?] while many approaches now consider n-way array tensors and multivariate function as a single object [? ? ?]. Finally, a Recursive POD (RPOD) was proposed in [?].

2 Bivariate Decomposition

In order to give the full picture of data reduction technique, it is crucial to begin with bivariate problems. Indeed almost all multivariate techniques result from these 2D versions. Bivariate decomposition techniques were mainly theoretical at the time they were proposed in the first half of the 20th century Pearson [?] and Hotelling [?], manual computations limited the size of the studied problems. But the numerical analysis and properties have been studied in details with emerging spectral theory [? ?]. Actual implementations were carried on later in the second half of the 20th for fluid dynamics systems [? ? ?]. 2D data reduction techniques are well understood and have been applied to the widest variety of problems in the last 20 years either to compress data or build reduced order model [? ? ?].

In order to offer a broader view of the possible uses of bivariate decomposition, Fig. 2 proposes a schematic view of bivariate problem reduced order modeling methods. The decomposition techniques presented in this section form the base material of many ROMs. They are organized as follow. The dashed black line shows the dichotomy between the continuous approach[4] and the discrete one. Then the orange dashed line separates the techniques that only apply to data –namely SVD and POD– from the PGD which is usually used on the equation itself but can be degraded into a data

[4]These approaches are conceptually continuous but their implementations requires discrete description of the continuous space including grids, discrete operators,...



Reduced order modelling through reduced bases
Bivariate methods
Discrete approach
Continuous approach
Direct SVD (POD)
POD-KLE
PGD
Scalar product choice : L2 , H1
Integration technique (trapezoidal, splines…)
Mass matrix to fit the grid
SVD
EVD
Iterative process for building PGD basis on equations
POD Basis
Interpolation (DEIM,Lagrange,...)
Galerkin Projection
Interpolated solution
Reduced basis solver
Compression / decomposition
ROM
Data related
Equation related

Figure 2: Synthetic view of the procedures described in this section for model order reduction of bivariate PDEs. The vertical arrows describe the work flow of these techniques and the dotted lines highlight the conceptual differences between them.

decomposition method. Finally, the blue dashed line emphasizes the data compression nature of the POD and SVD while noting the possibility to obtain a ROM through the obtained basis as shown in the lower part of the diagram.

These two methods are presented in this section since they represent the foundation of higher order decomposition techniques.

2.1 Singular Value Decomposition

The Singular Value Decomposition (referred as **SVD**) is a generalization of the eigenvalues decomposition for rectangular matrices. Among its many applications it can be seen as a discrete version of the POD.

Theorem 2.1 (Singular Value Decomposition [**?**]). *For any matrix $A \in$*

$\mathbb{R}^{m\times n}$, there are orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ so that

$$A = U\Sigma V^{\mathsf{T}}$$

where Σ is a diagonal matrix of size $n \times m$ with diagonal elements $\sigma_{ii} \geq 0$.

Hereafter, it is assumed that the singular values are ordered decreasingly i.e. if $i < j$ then $\sigma_{ii} \geq \sigma_{jj}$. The SVD is not unique since the signs of U and V may vary.

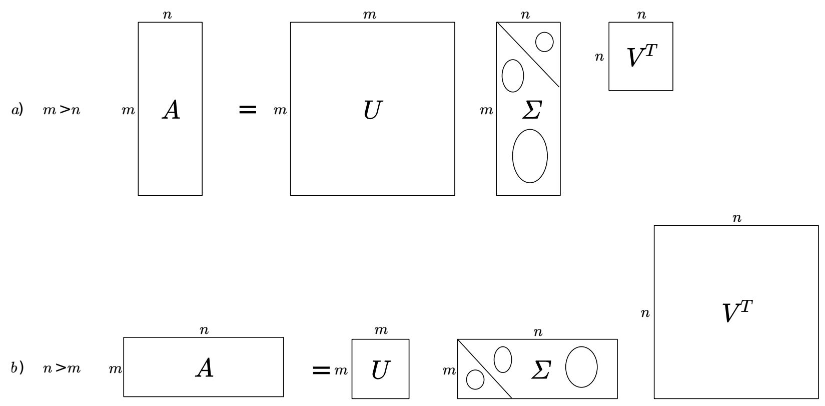


Figure 3: Singular Value Decomposition two configurations

One should note from figure 3 that a part of U in case a) and V in case b) only serves a dimension match without entering calculation of A, then the SVD reads for case a)

$$A = [U_1, U_2][\Sigma_1, 0]^{\mathsf{T}} V^{\mathsf{T}} = U_1\Sigma_1 V^{\mathsf{T}}$$

Let rank$(A) = r$ then for $k > r$, $\sigma_k = 0$. The SVD of A can be written as sum

$$A = \sum_{i=1}^{r} \sigma_i U_i V_i^{\mathsf{T}}$$

where σ_i are the diagonal entries of Σ and U_i and V_i refer to the columns of U and V respectively. Then $\|A\|_2 = \sqrt{\sum_{i=1}^{r}\sigma_i^2}$ leads to the optimality theorem proven by Eckart and Young in 1936 [**?**].

Theorem 2.2 (Eckart-Young)**.** *Let $k < r$ and $A_k = \sum_{i=1}^{k} \sigma_i U_i V_i^\intercal$ where the singular values are ordered decreasingly then*

$$\min_{\text{rank}(B)=k} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1} \tag{1}$$

Remark 2.3 (Link with the eigenvalue decomposition)**.** Singular and eigenvalues are closely linked. Let $A \in \mathbb{R}^{m\times n}$ with $m > n$. $A^\intercal A = V\Sigma^\intercal \Sigma V^\intercal$. Then the eigenvalue problem of $A^\intercal A$ is equivalent to the right singular value problem of A with $\lambda_i = \sigma_i^2$ and the eigenvectors of $A^\intercal A$ are collinear to A's right singular vectors $\boldsymbol{v}_i$. The same applies to $\boldsymbol{u}_i$ and the eigenvectors of $A^\intercal A$.

Remark 2.4 (Solving least square minimization problem with the SVD)**.** The classical least square minimization problem i.e. find x^n of minimum Euclidean norm that reaches the minimum of $\|b - Ax\|_2$ for $A \in \mathbb{R}^{m\times n}$, is solved by the SVD and the Monroe-Penrose pseudo inverse of A (see [**?**]).

The main information contained in the Eckart-Young theorem is that the truncated SVD (see Fig. 4) i.e. only keeping the k dominant modes gives an optimal approximation of rank-k of the matrix A which rank is $r \geq k$. It means that the k first singular vectors form the optimal projection basis of size k that reads as follow,

$$A \approx A_k = \sum_{i=1}^{k} \sigma_i \boldsymbol{u}_i \otimes \boldsymbol{v}_i \tag{2}$$

Numerics As for the eigenvalue decomposition, there are many algorithms to compute the SVD, among them, the QR algorithm is particularly well suited to slim matrices. In subsequent numerical experiment the LAPACK library is used either as direct SVD solver `dgesdd` or through eigenvalue decomposition `dsyev` if the matrix is slim (this strategy is also well suited for discrete POD as discussed in the next section). `dgesdd` relies on a divide and conquer approach which is one of the most efficient way to handle matrices of large size.

Other algorithm provide direct truncated SVD mainly based on iterative algorithm such as *Arnoldi procedure* based library `ARPACK`. Additionally, it should be noted that iterative algorithm are very efficient at finding eigen/singular values at both ends of the spectrum but face accuracy issues in other regions, especially for ill-conditioned matrices. This results in non orthonormal bases which may impair decomposition or ROM accuracy.

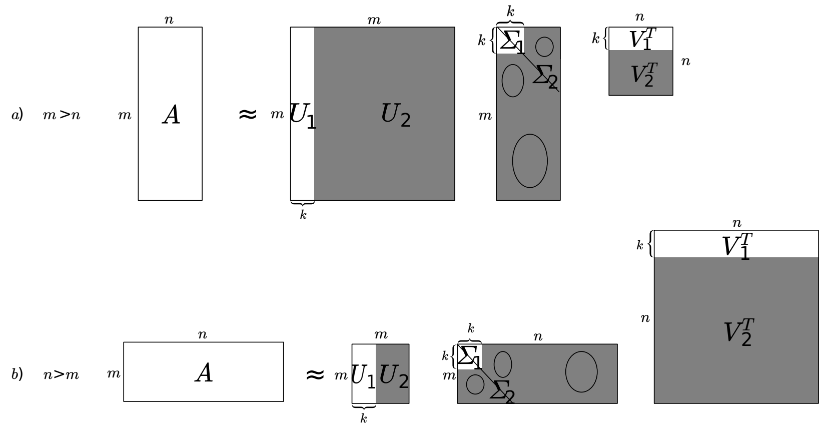


Figure 4: Rank k truncated-SVD for both configurations, the shadowed part is dropped upon truncation. $k \leq n,\ k \leq m$.

2.2 Proper Orthogonal Decomposition

The POD was discovered many times in many different fields, however it is often attributed to Kosambi [**?**] who introduced it in 1943. Also, the POD comes under many names depending on the field in which it is used or devised. For instance, it is rigorously equivalent to the Karhunen-Love expansion [**?**] or Principal Component Analysis (PCA) usually attributed to [**?**]. It is an elegant way to approximate a high dimensional[5] system into a low dimensional one. To do so, a linear procedure is devised to compute a basis of orthogonal proper modes that represent the energy repartition of the system. They are obtained by solving Fredholm's equation for data (usually) obtained through numerical simulations. Additionally the POD offers an optimal representation of the energy in term of L^2 norm.

It has been applied to extract dominant patterns and properties in wide variety of fields such as signal, data compression, neural activity, mechanics or fluid dynamics to name only a few. An enlightening description of the use of POD is given by Bergmann [**?**]: "*The POD defines uniquely and without ambiguity coherent structures*[6]*, as the realization of largest projection on*

[5]Here, high dimensionality is to be understood as rich phenomenon that require many degrees of freedom to be described properly as opposed to simpler system which are described by few degrees of freedom e.g. simple pendulum.

[6]The notion of coherent structures, introduced by [**? ?**] is central in the use of POD

the mean realization contained in the database".

Problem formulation (scalar case). Find the best approximation, in the sense of a given inner product $(\cdot,\cdot)$ and average operator $\langle\cdot,\cdot\rangle$, of $f : \mathcal{D} = \Omega_x \times \Omega_t \longrightarrow \mathbb{R}$ as a finite sum in the form

$$\tilde{f}_r(\boldsymbol{x},t) = \sum_{k=1}^{r} a_k(t)\phi_k(\boldsymbol{x}) \tag{3}$$

where $(\phi_k)_k$ are orthogonal for the chosen inner product. a_k is given by $a_k(t) = (f(\cdot,t), \phi_k(\cdot))$ then a_k only depends on ϕ_k.

Discrete POD problem is often found in the literature as follows. Let $\{f_1, ..., f_{n_t}\}$ the snapshots of f i.e. the representation of f at discrete time $\{t_j\}_{j=1}^{n_t}$. It is assumed that $\mathcal{F} = \operatorname{span}\{f_1, f_2, ..., f_{n_t}\}$.

POD generates an orthonormal basis of dimension $r \leq n_t$, which minimizes the error from approximating the snapshots space $\mathcal{F}$. The POD basis verifies the optimum of the following:

$$\min_{\{\phi\}_{k=1}^{r}} \sum_{j=1}^{n_t} \|f_j - \tilde{f}_{r,j}\|^2, \text{ s.t. } (\phi_k, \phi_j) = \delta_{kj} \tag{4}$$

where $\tilde{f}_{r,j} = \sum_{k=1}^{r} (f_j, \phi_k)\, \phi_k$ and δ_{kj} is the Kronecker symbol. One may observe that $\sum_{k=1}^{r} \cdot$ is the first order approximation of the time mean operator $\langle\cdot\rangle$. This problem can be solved with discrete Eigen Value Decomposition (EVD). Although it is the most common formulation of discrete POD in mechanics literature, it can be misleading regarding the construction and properties of the POD.

Building the POD This subsection aims at providing a rigorous, however mechanics oriented presentation of the POD. The present approach is based on Fahl's work [**?**] and also Bergmann and Cordier [**? ? ?**] as well as other work of the vast corpus available including [**? ?**]. Since POD is the cornerstone of several multivariate data reduction techniques, it is crucial to provide the mathematics underlying this method. Without loss of generality, the usual framework for POD where the two variables are space (possibly a position vector) and times. It makes mental representation easier for the reader and most of the POD jargon was introduced with time-space POD.

for mechanics.

Let $\boldsymbol{X} = (\boldsymbol{x}, t) \in \mathcal{D} = \Omega_x \times \Omega_t$ and $\boldsymbol{u} : \mathcal{D} \longrightarrow \mathbb{R}^d$ a vector valued function. Additionally we assume that a scalar product $(\cdot, \cdot)$ is defined on $\mathcal{D}$ and $||\cdot||$ its associated norm while an average operator $\langle \cdot \rangle$ is defined on $\mathcal{D}$[7]. We also need the following u to be of finite norm. The dominant modes of a set of realization $\{\boldsymbol{u}(\boldsymbol{X})\}$ are sought, i.e. the function $\boldsymbol{\phi}$ whith the largest projection on realizations $\{\boldsymbol{u}(\boldsymbol{X})\}$ in the least square sense. In other words, we seek $\boldsymbol{\phi}$ that maximizes $|(\boldsymbol{u}, \boldsymbol{\phi})|$ where $\boldsymbol{\phi}$ is normalized. Then the maximum of this expression is sought

$$\frac{\langle |(\boldsymbol{u}, \boldsymbol{\phi})|^2 \rangle}{\|\boldsymbol{\phi}\|^2} \tag{5}$$

This leads to the following constrained maximization problem

$$\max_{\boldsymbol{\psi} \in L^2(\mathcal{D})} \frac{\langle |(\boldsymbol{u}, \boldsymbol{\psi})|^2 \rangle}{\|\boldsymbol{\psi}\|^2} = \frac{\langle |(\boldsymbol{u}, \boldsymbol{\phi})|^2 \rangle}{\|\boldsymbol{\phi}\|^2} \tag{6}$$

with

$$(\boldsymbol{\phi}, \boldsymbol{\phi}) = 1$$

In order to rewrite problem (6), a linear operator $\mathcal{R} : L^2(\mathcal{D}) \longrightarrow L^2(\mathcal{D})$ is introduced, it is defined as

$$\mathcal{R}\boldsymbol{\phi}(\boldsymbol{X}) = \int_{\mathcal{D}} R(\boldsymbol{X}, \boldsymbol{X'}) \boldsymbol{\phi}(\boldsymbol{X'}) d\boldsymbol{X'} \tag{7}$$

where $R(\boldsymbol{X}, \boldsymbol{X'}) = \langle \boldsymbol{u}(\boldsymbol{X}) \otimes \boldsymbol{u}(\boldsymbol{X'}) \rangle$ is the tensor of spatio-temporal correlations. Now suppose that $\langle \cdot \rangle$ and $\int$ can be permuted then the following holds

$$\begin{aligned}
(\mathcal{R}\boldsymbol{\phi}, \boldsymbol{\phi}) &= \langle |(\boldsymbol{u}, \boldsymbol{\phi})|^2 \rangle && \geq 0 \\
(\mathcal{R}\boldsymbol{\phi}, \boldsymbol{\psi}) &= (\boldsymbol{\phi}, \mathcal{R}\boldsymbol{\psi}) && \forall (\boldsymbol{\phi}, \boldsymbol{\psi}) \in [L^2(\mathcal{D})]^2
\end{aligned}$$

Since $\mathcal{R}$ is a positive self-adjoint operator, the spectral theory applies and the solution of problem (6) is given by the largest eigen value of this new problem

$$\mathcal{R}\boldsymbol{\phi} = \lambda \boldsymbol{\phi} \tag{8}$$

It can be written as a Fredholm integral equation:

$$\sum_{j=1}^{d} \int_{\mathcal{D}} R_{ij}(\boldsymbol{X}, \boldsymbol{X'}) \phi^j(\boldsymbol{X'}) d\boldsymbol{X'} = \lambda \phi^i(\boldsymbol{X}) \quad \forall i \tag{9}$$

[7]The natural choice for fluid dynamics applications $L^2(\Omega_x)$ scalar product and a time average. The choice of the average operator $\langle \cdot \rangle$ kind (temporal, spatial,...) determines which kind of POD is used.

Some fundamental properties of the POD.

1. For $\mathcal{D}$ bounded, Hilbert-Schmidt theory applies and ensures the existence of countably infinitely many solutions to equation (9)
$$\sum_{j=1}^{d} \int_{\mathcal{D}} R_{ij}(\boldsymbol{X}, \boldsymbol{X'}) \phi_r^j(\boldsymbol{X'}) d\boldsymbol{X'} = \lambda_r \phi_r^i(\boldsymbol{X}) \tag{10}$$
where $\lambda_r, \boldsymbol{\phi}_r$ are respectively the POD eigenvalues and eigen functions of order $r = 1, 2, ..., +\infty$. Each new eigen function is defined as the solution of problem (8) adding a new constraint: orthogonality with the already known eigen functions.
$$\sum_{i=1}^{d} \int_{\mathcal{D}} \phi_r^i(\boldsymbol{X}) \phi_p^i(\boldsymbol{X}) d\boldsymbol{X} = \delta_{rp} \tag{11}$$
2. $\mathcal{R}$ is positive self-adjoint then $\lambda_i \geq 0$. Additionally, they are taken decreasing and they form a converging series i.e.
$$\sum_{r=1}^{\infty} \lambda_i \leq +\infty$$
.
3. The POD eigen functions form a complete basis, any realization $u(\boldsymbol{X})$ can be represented in that basis.
$$u^i(\boldsymbol{X}) = \sum_{r=1}^{\infty} a_r \phi_r^i(\boldsymbol{X}) \tag{12}$$
4. a_r is obtained by projecting $\boldsymbol{u}$ on $\boldsymbol{\phi}_r$
$$a_r = (\boldsymbol{u}, \boldsymbol{\phi}_r) = \sum_{i=1}^{d} \int_{\mathcal{D}} u_i(\boldsymbol{X}) \phi_r^i(\boldsymbol{X}) d\boldsymbol{X} \tag{13}$$
5. **Mercer's Theorem.** The spatio-temporal correlation matrix at two points R_{ij} is kernel based on $\mathcal{R}$ then Mercer's theorem provides a series representation,
$$R_{ij}(\boldsymbol{X}, \boldsymbol{X'}) = \sum_{r=1}^{\infty} \lambda_r \phi_r^i(\boldsymbol{X}) \phi_r^j(\boldsymbol{X'}) \tag{14}$$
6. Thanks to the previous property, it can be shown [**?**] that the coefficients a_r are uncorrelated and their quadratic average is equal to the POD eigenvalues
$$\langle a_r, a_p \rangle = \delta_{rp} \lambda_r \tag{15}$$

Remark 2.5. These properties ensure the uniqueness of the proper orthogonal decomposition (given that $||\boldsymbol{\Phi}|| = 1$).

Optimality of the POD basis. Let $\boldsymbol{u} : \mathcal{D} \longrightarrow \mathcal{E} \subset \mathbb{R}^d$ with $\boldsymbol{u} \in L^2(\mathcal{D})$ and $\bar{\boldsymbol{u}}$ an approximation of $\boldsymbol{u}$. On any orthogonal basis $(\boldsymbol{\psi}_r(\boldsymbol{X}))_{r=1}^{\infty}$ one can write

$$\bar{u}_i(\boldsymbol{X}) = \sum_{r=1}^{\infty} b_r \psi_r^i(\boldsymbol{X}) \tag{16}$$

Let $\{\boldsymbol{\phi}(\boldsymbol{X})\}_{r=1}^{\infty}$ a set of orthogonal POD eigen functions and $\{\lambda_r\}_{r=1}^{\infty}$ their associated eigenvalues. Then, $\boldsymbol{u}^{POD}$ the POD approximation of u is considered

$$u_i^{POD}(\boldsymbol{X}) = \sum_{r=1}^{\infty} a_r \phi_r^i(\boldsymbol{X}) \tag{17}$$

Properties 6 and 7 state that if $(\boldsymbol{\psi}_r(\boldsymbol{X}))_{r=1}^{\infty}$ are non dimensional, $\langle b_r, b_r \rangle$ represents the energy of mode n. Cordier and Bergmann [**?**] proved the optimality of the POD basis through the following lemma.

Lemma 2.6. *Optimality of POD basis For any rank $R \in \mathbb{N}^*$ the following inequality holds*

$$\sum_{r=1}^{R} \langle a_r, a_r \rangle = \sum_{r=1}^{R} \lambda_r \geq \sum_{r=1}^{R} \langle b_r, b_r \rangle \tag{18}$$

In other words, among all linear decomposition, POD is the most efficient, i.e. for a given number of POD modes R, the projection on the subset produced by the first R POD eigen-functions is the one that contains on average the most (kinetic) energy possible.

A POD algorithm. One of the many possible implementations of the POD is proposed in this section. Although it might not be the most computationally efficient version, it preserves all the functional approach framework. Indeed the user is free to implement any integration method so that the projector also apply to L^2, not to any matrix space.

Algorithm 1: POD (*Standard, Deflation Power Method*)

```
input  : $f$, target error $\epsilon$
output: $\tilde{f}_r = \sum_{k=1}^{r} \sigma_k X_k Y_k$
m=0
$\boldsymbol{R}(x,x') = \int_{\Omega_t} f(x,t) f(x',t) dt$;
while $\frac{\sigma_m}{\|f\|_{L^2}} \geq \epsilon$ do
    $k = k+1$
    $(\lambda_k, \phi_k) =$ Orthonormal_Power_method $[(\boldsymbol{\mathcal{R}} - \tilde{\boldsymbol{f}}_{\boldsymbol{k-1}})\phi_k = \lambda_k \phi_k]$
    $\sigma_k = \sqrt{\lambda_k}$
    $a_k = \int_{\Omega_x} f(x,t)\phi_k(x) dx / \sigma_k$
    $\tilde{f}^k = \tilde{f}_{k-1} + \sigma_k \phi_k a_k$
return $f_k$
```

Remark 2.7. From the previous sections, it clearly appears that POD and SVD share many of their properties. One can adopt two different angles to explore the link between POD and SVD.

a. Use the the optimality of the SVD to solve the discrete POD minimization problem. This is a straightforward application of the fact that eigenvectors can be computed either from eigenvalue decomposition or SVD. This approach has been described in detail by Bergmann and Fahl's work [**?** **?**].

b. The other way of looking at this link, was proposed among many others by Chaterjee [**?**]. It is a simpler presentation of the POD, only valid in the discrete framework. It relies on the fact that the SVD solves optimally a matrix problem that may be seen as the discrete equivalent of the infinite dimensional problem (4) using the Euclidian norm for vectors.

It shall be noted that these two interpretations leads to different algorithms which may not display the same properties of accuracy or efficiency especially when the basis is used for reduced order modeling as its orthogonality is a very important feature. The very illustration is the possibility to choose a problem adapted inner product in the POD algorithm while SVD is blind to data and will be performed in the same fashion for any problem, sometimes without preserving physical properties.

Remark 2.8. One of the many challenges in using POD efficiently is to chose a scalar product that suits the problem. This issue has been of great interest in the field of fluid mechanics since it provides fields of great complexity that are either scalar (pressure, bi-dimensional vorticity) or vectorial (velocity, velocity/pressure, etc.). Also these fields are the solution of the highly nonlinear Navier-Stokes equations, the induced properties can be taken into account when devising the decomposition method.

Originally, as implied in the above presentation, the introduction of POD in this field came through the analysis of the velocity field for which the natural idea is to rely on energy measure such as the $L^2(\Omega)$ (see [?]). However it has been shown that such an approach generates unstable ROM, which can be improved by using $H^1(\Omega)$ norms such as proposed by Iollo in [?]. The last fifteen years of research have proved that this approach is either impractical or requires too much effort as compared to the benefits since it has not been able to gain momentum among the community. Meanwhile, a new approach has been advocated by Sengupta [?] that relies on the use of enstrophy for the analysis of instability flows. This idea has been applied successfully to POD in a series of articles [? ? ?] and the analysis has been pushed even further in recent article [?]. Relying on enstrophy based POD has allowed the authors to provide in depth correlation between POD modes and instability i.e. Hopf bifurcation sequence in spite of the very high sensitivity of the studied lid driven cavity problem.

2.3 Numerical experiments

In this section a few numerical tests are conducted on all three methods. Although it has been shown that they are mathematically equivalent, the difference between these algorithm will inevitably produce different behavior, especially for ill-conditioned problems/matrices. This first numerical section provides a suggested technique over the others depending on the problems studied. First some synthetic data is used i.e. analytical functions, then an image is compressed with various levels of accuracy. Finally, data from numerical simulations is separated. The approximation error is measured as $||f - f_r||_{L^2}$ or $||f - f_r||_{\mathrm{F}}$ depending on the nature of the method[8].

[8]Actually the choice of the norm has little influence on the numerical results. This is especially true for trapezoidal rule on a Cartesian grid.

Synthetic data Let $\Omega = [0,1] \times [0,1]$ be the studied domain and four square integrable functions $f_1, f_2, f_3, f_4 : \Omega \to \mathbb{R}$ defined by

$$f_1(x,y) = xy \tag{19}$$

$$f_2(x,y) = \frac{1}{1+xy} \tag{20}$$

$$f_3(x,y) = \sin(\sqrt{x^2+y^2}) \tag{21}$$

$$f_4(x,y) = \sqrt{1-xy} \tag{22}$$

$$f_5(x,y) = \frac{1}{(1+xe^y)} \tag{23}$$

These functions range from already separated (f_1) to weakly separable, also known as *singular* functions in the literature. Thus these two expressions will be used indifferently in this manuscript. They are chosen to be easily extended to multiple variables.

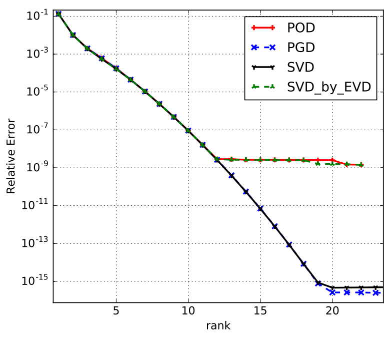
The four methods PGD, POD ($L^2(\Omega)$) SVD and `SVD_by_EVD` are applied on these functions for a 32×32 regular Cartesian grid.

Remark 2.9. The bivariate functions can be sorted in two groups with respect to these decomposition techniques:

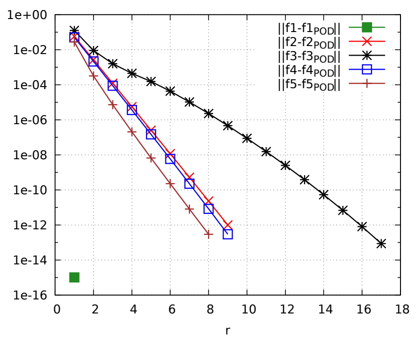
Definition 2.10 (Exponentially Separable function)**.** A function is called *exponentially separable* if the decrease in the singular values, thus in the approximation error, is exponential. In other words, a *semi-log* plot of the error is a straight line, regardless of its slope.

Definition 2.11 (Linearly separable function)**.** A function is called *linearly separable* or weakly separable if the decrease in the singular values, thus in the approximation error, is linear. In other words, a *log-log* plot of the error is a straight line, regardless of its slope.

These definition will be extended directly to multiparameter functions. Typically, weakly separable function are produced by highly non-linear processes or functions that display a sharp singularity. Thus singular function is often used to replace weakly separable in the literature as well as in this manuscript. Additionally, various levels of separability may be observed depending on the nature of the function. A moderate slope will often be referred to as less separable and an almost linear decay declared weakly separable. Finally, some peculiar function may show two different regions (relative to r) with distinct behavior such as first a sharp exponential decay



(a) 2D decompositions error of f_3.



(b) POD applied on a pool of functions.

Figure 5: Approximation error for bivariate methods

followed by a milder linear one. This generally fits the properties of the function such as length scale or turbulent behavior in fluid dynamics.

As mentioned in the theoretical paragraphs, a very efficient way to measure the separability of a field is to observe the decay of the singular values. It is also a reliable way to estimate the error decay. Fig. 5a presents a comparative view of the decay of the approximation error for f_3 which is a very common function for testing this property. The singular values are not displayed as their behavior is very similar to the error. All four methods are equivalent up to $r \simeq 12$ which is in agreement with the mathematical equivalence shown in the theoretical presentation. However for $r \geq 12$ it seems that the error is stuck in the 10^{-8} regions.

Image compression by decomposition As stated in the SVD section, these techniques can be used on any kind of data. An interesting example while presenting the data compression aspect of these methods is to apply it to images. Indeed it is efficient to compress large images. Indeed numerical images are stored in many formats but it always boils down to an array of integers representing colors. Let us consider the simpler case of grayscale images, usually stored in 1 byte per pixel. That is to say, the original 4000×3000 pixels grayscale image `"singapore.tiff"` used in Fig. 6 is a matrix of the same size whose coefficients are integers in $[\![0, 255]\!]$ which means 12×10^6bytes $\approx$ 12Mb without compression. Table 1 gives the compression rate for different number of SVD modes retained as displayed in Fig. 6. One can see easily that preserving very few modes yields high levels of

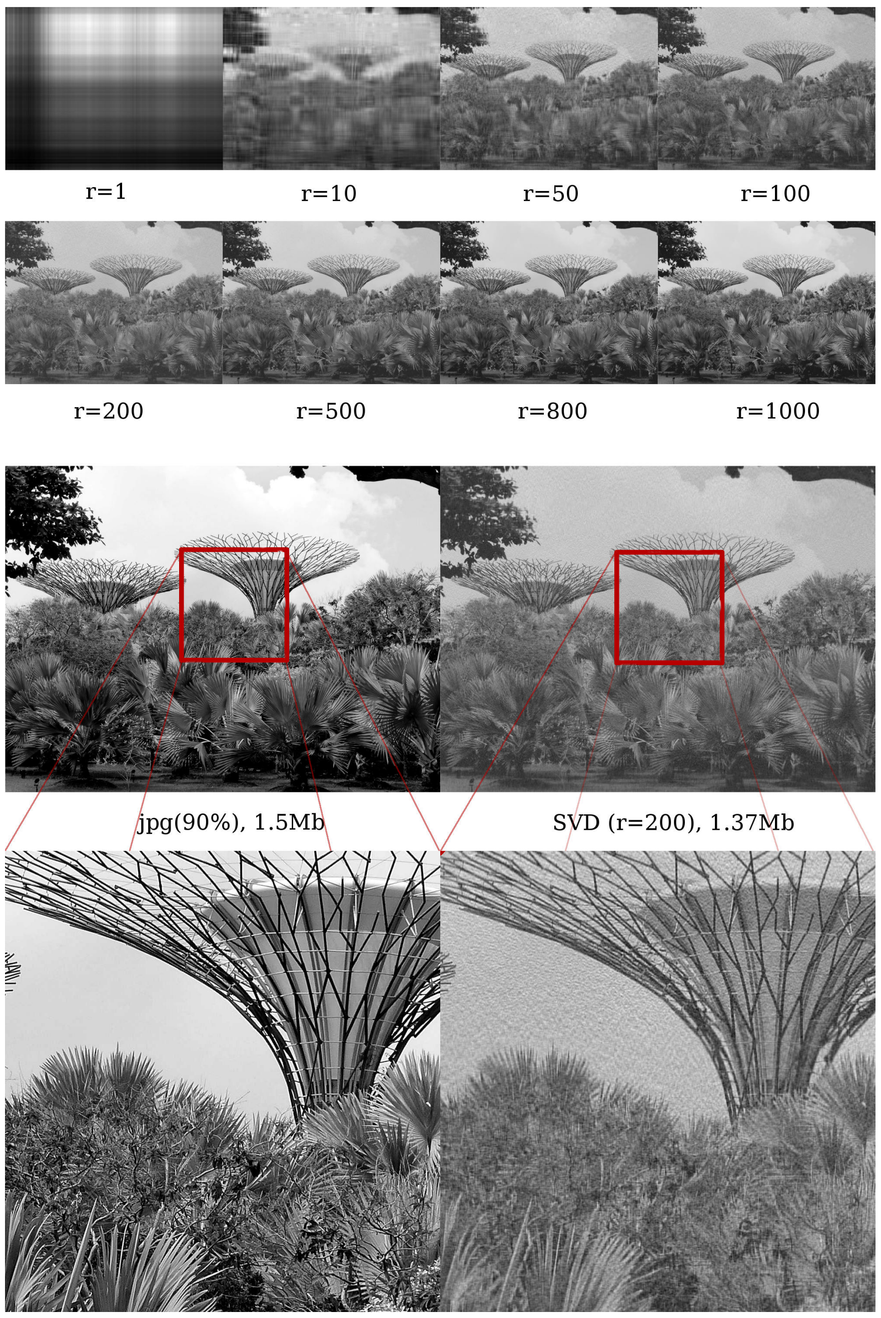


Figure 6: A 4000 $\times$ 3000 pixels picture of Singapore Gardens by the Bay compressed through SVD as compared with JPEG compression.

compression but the image features are not preserved. Indeed, on can see in Fig. 6 top two lines[9] that keeping only one mode gives a unrecognizable image. Increasing number of retained modes r leads to gradually better representation, 10 modes is sufficient to perceive the big structures of the image. The big leaves and sharp metallic structures are captured with 50 modes while 100 modes is enough to distinguish palm leaves. This behavior continues up to a few hundreds where all human-eye relevant structures are captured by the SVD compressed image. However at $r = 200$, the image is grainy (especially visible in the sky part) which is striking in the larger SVD image and close-up in the lower part of Fig. 6. Adding more and more modes reduces the noise of the image, at $r = 1000$ it is hard to tell that the image has been compressed without any reference point, while the size of the image is still halved as compared with the original uncompressed file. The only difference lies in the contrast level as one can see that the very dark and very bright regions of the image are not as deep as in the original image.

Table 1: Compression rate using SVD on 4000×3000 pixels grayscale image. Where CR is the compression rate and the error is computed with Frobenius norm.

r	SVD size (Mb)	CR (%)	Err. (%)
1	0,01	99.9	41.5
10	0,07	99	31.2
50	0,33	97	25.7
100	0,67	94	22.2
200	1,34	89	17.2
500	3,34	72	9.4
800	5,34	55	5.2
1000	6,68	44	3.2

A very interesting feature of this data lies in the very slow decay of the singular values, shown in Fig. 7. Indeed is was chosen on purpose so that no clear directional pattern appeared in the image and all length scales were present. Consequently, the first 50 singular values plummet then the slope become a lot milder with a decay of one order of magnitude per thousand modes. One can assert that the first exponential decay, associated with the large structures of the image, is followed by a linear one due to the profusion of small scales. This is the first example of this behavior shown

[9]The reader is advised to follow this description in the PDF version as it allows zooming of the row of small pictures.

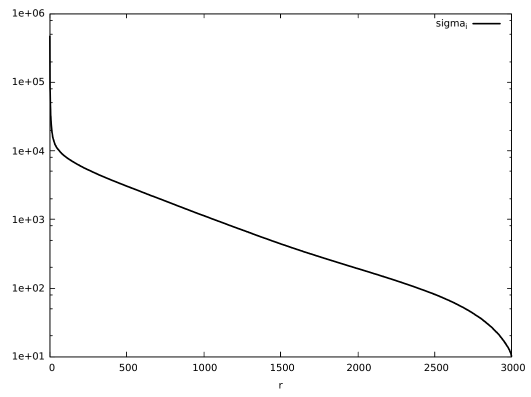


Figure 7: Singular values of `"singapore.tiff"`

in this chapter. It will appear again in complex flows and physics problem, either in 2D or 3D. As usual, if all modes are kept the image is exactly recovered. However, there is overhead in the storage space as U is of the same size as the original data and one still needs to store V and Σ.

To conclude on the image compression abilities of SVD, it is fairly efficient for large images as the ratio r/n_{pix} is very small but the method is not well suited for human-eye use. The Frobenius error presented in table 1 does not fit with the human experience of the image produced by SVD comparison. Indeed, SVD compares poorly with well established formats such as JPEG which was specifically designed to retain eye sensitivity such as contrast, color depth, etc.

3 Tensor Spaces and Formats for decomposition

This section lays the ground for building a general decomposition framework that works equally well for continuous and discrete multidimensional problems. The concept of a tensor space structure and its main properties are described. The the main features of tensors are presented on the particular case of multi-way arrays but are expendable to other kind of tensors. This dichotomy provides a general framework that will be needed in further development and eases the understanding of complex definition with the n-way array.

Tensors and tensor spaces. Tensors can be viewed as generalization of matrix to higher dimension i.e. an order d tensor is a d-way array or a function of d arguments. Such object rapidly become intractable, indeed for large $d > 3$, data size n^d is out of reach even for the most advanced computers and will remain that way for direct handling. A simple example of the *curse of dimensionality* is to take $n = 2$ and $d = 50$, although it appears to be of reasonable size, $2^d \approx 10^{15}$. This is of course far below the requirement of many scientific areas such as chemometrics, Boltzmann equation, multiparameter PDEs etc. This has led to the introduction of reduction techniques to overcome the *curse of dimensionality* starting with [**?**]. Lots of work have been separately performed in separate fields such as psychometrics ([**?**] and [**?**]) in the 1960s and 1980s or chemometrics from 1981 onwards ([**?**]). Since 2000, tensor decomposition has gained a lot of interest in many fields including solution of stochastic PDEs [**? ?**], solution of high dimensional Schrdinger equation, Boltzmann equation, computational finance, etc. Many more references are available in literature surveys by [**?**] and [**?**]. Actually, these surveys together with [**?**] book "Tensor spaces and numerical Tensor calculus" demonstrate the growing interest for decomposition among the applied mathematics community.

3.1 Tensor spaces

In order to build the approximation presented in the subsequent sections, a general framework is introduced. The mathematical framework we use in this section is based on [**?**] with addition from other authors. Further details can be found in the original manuscript while we only cover the necessary notions for tensor decomposition.

Definition 3.1 (Tensor Space)**.** Let V and W be vector spaces. The algebraic tensor space $\mathcal{V}$ is defined by

$$\mathcal{V} = V \otimes_a W = \operatorname{span}\{v \otimes w \ : \ v \in V, \ w \in W\} \tag{24}$$

where $\otimes_a$ connects vectors spaces and $v \otimes w$ is an element of $\mathcal{V}$.

Obviously, a tensor space is still a vector space however given a special structure.

Proposition 3.2. *Let V and W be vector spaces with respective bases B_V and B_W such that T be a tensor space over the field $\mathbb{R}$. A product $\otimes : V \times W \to T$ is a tensor product and T a tensor space, i.e., it is isomorphic to $V \otimes_a W$, if the following properties hold:*

i) span property : $T = \operatorname{span}\{v \otimes w \ : \ V \in V, w \in W\}$
ii) bilinearity
iii) linearly independent vectors $\{v_i \ : \ i \in B_V\} \subset V$ and $\{w_i \ : \ i \in B_W\} \subset W$ lead to independent vectors $\{v_i \otimes w_j \ : \ i \in B_V, j \in B_W\}$ in T

Note that the tensor product is associative and *universal*, i.e.

Proposition 3.3 (Universality of the tensor product)**.** *For any multilinear map $\varphi : V_1 \times \cdots \times V_d \to V$, there is a unique linear mapping $\Phi : \otimes_{j=1}^{d} V_j \to V$ so that $\varphi(v_1, ..., v_d) = \Phi(v_1 \otimes \cdots \otimes v_d)$.*

3.2 Overview of tensors of $\mathbb{R}^{n_1 \times \cdots \times n_d}$ i.e. multi-way arrays

In this section, a series of definitions and properties of the multi-way array tensors is provided. It should be noted that most of these definitions extend to other tensor spaces but most, if not all the work presented in this chapter uses discrete tensors. The properties and definitions presented here are limited to the one necessary for approximation of tensors.

First, we introduce some notations. Let $d \in \mathbb{N}$ be the number of dimensions and $n_1, ..., n_d \in \mathbb{N}$ the number of entries along each of these dimensions. Let $D = \{1, ..., d\}$ be a tuple and $\mathcal{I} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_d$ be a d-fold product index set with $\mathcal{I}_\mu = \{1, ..., n_\mu\}$

Definition 3.4 (Tensor)**.** A *tensor* is a multidimensional array i.e. a d-way or d^{th}-order tensor is an element of the tensor product of d vector spaces, each of which has its own coordinate system.

In terms of tensor space, here we have $\boldsymbol{\mathcal{X}} \in \mathbf{V} = {}_a\bigotimes_{i=1}^{d} V_i$ where $V_i = \mathbb{R}^{n_i}$. This notion of tensor is different from the many physical tensors which generally refer to a third order tensor that is defined in every points of the space. This forms a tensor field. Bold Euler script letters refer to order d tensors e.g. $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{\mathcal{I}}$.

Definition 3.5 (Order of a tensor)**.** The order of a tensor is defined as the number of dimensions, also known as ways or modes. $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{\mathcal{I}}$ where $\mathcal{I} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_d$, is an order d tensor[10].

Remark 3.6. A first-order tensor is a vector, a second-order tensor is a matrix and a third order tensor or more is called a higher order tensor. A visual representation of a third order tensors is proposed in figure 8.

The entries of a tensor are denoted in the same fashion as for vectors or matrices i.e.

- entry i of vector $\mathbf{a}$ is a_i
- entry (i, j) of matrix $\mathbf{A}$ is a_{ij}

[10]The order of a tensor is not to be confused with the rank of a tensor.

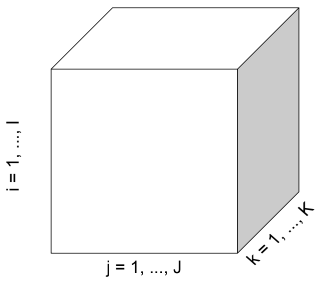


Figure 8: A third order tensor with $\mathfrak{T} \in \mathbb{R}^{I \times J \times K}$.

- entry $(i_1, i_2, \cdots, i_d)$ of order d tensor $\mathcal{A}$ is $a_{i_1 i_2 \cdots i_d}$

A subarray is formed when a subset of a tensor is taken e.g. subarrays of matrices are columns and rows. A colon is used to state that every element of a dimension is taken.

Definition 3.7 (Fibres)**.** Fibres are the higher order analogue of matrix rows and columns. A fibre is defined by fixing every indices but one. Mode-1 fibre of a matrix is a column mode-2 fibres are rows and mode-3 fibres are tube fibres.

Remark 3.8. *Slices* are two dimensional sections of a tensor defined by fixing every indices but two.

Definition 3.9 (Inner product and norm)**.** Given two same-sized tensors $\mathfrak{X}, \mathfrak{Y} \in \mathbb{R}^{\mathcal{I}}$, the *Inner Product* is defined as follow

$$\langle \mathfrak{X}, \mathfrak{Y} \rangle_F = \sum_{i_1=1}^{n_1} \cdots \sum_{i_d=1}^{n_d} x_{i_1 \cdots i_d} y_{i_1 \cdots i_d} \tag{25}$$

When there is no ambiguity on the nature of the inner product, the Frobenius inner product is simply noted $\langle \mathfrak{X}, \mathfrak{Y} \rangle$.

The norm associated with this inner product is the Frobenius norm defined by $\|\mathfrak{X}\|_F = \sqrt{\langle \mathfrak{X}, \mathfrak{X} \rangle}$ also

$$\|\mathfrak{X}\|_F = \sqrt{\sum_{i_1=1}^{n_1} \cdots \sum_{i_D=1}^{n_d} x_{i_1 \cdots i_d}^2} \tag{26}$$

Definition 3.10 (Rank-One tensor)**.** An N-way tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ is *rank-one* if it can be written as the outer product of d vectors $(\mathbf{a}^{(j)})_{j=1}^{d}$, i.e.

$$\mathcal{X} = \mathbf{a}^{(1)} \circ \cdots \circ \mathbf{a}^{(d)} \Leftrightarrow \forall\ 1 \leq i_j \leq n_j,\ x_{i_1 \cdots i_d} = \prod_{j=1}^{d} a_{i_j}^{(j)}$$

Definition 3.11 (Rank of a tensor)**.** The rank of a tensor, denoted rank($\mathcal{X}$), is the minimum number of rank-one tensors that generate $\mathcal{X}$ as their sum. In other words, this is the smallest number of components in an exact CP decomposition (see the definition 4.2). Further details are available in [**?**] concerning the link with the matrix rank.

Remark 3.12. There is no straightforward way to determine the rank of a higher order tensor even for small sizes (the problem is NP-hard).

Definition 3.13 (μ-rank or multilinear rank of a tensor)**.** The μ-rank of tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$, denoted $\mathrm{rank}_\mu(\mathcal{X})$ is the rank of $\boldsymbol{X}_{(\mu)}$. If we let $r_\mu = \mathrm{rank}_\mu(\mathcal{X})$ for $\mu = 1, ..., d$ then we can say that $\mathcal{X}$ is rank-$(r_1, \cdots, r_d)$ tensor. Beware not to confuse the μ-rank with the previous notion of rank of a tensor.

Remark 3.14. The notion of μ-rank was popularised by De Lathauwer [**?**].

Definition 3.15. matricization or *unfolding*
Matricization is the process of ordering the elements of a tensor into a matrix. The mode-n matricization of a tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ is denoted by $\mathbf{X}_{(\mu)}$ and arranges the mode-μ fibres to be the columns of the resulting matrix. We define the index set $\mathcal{I}^{(\mu)} = \mathcal{I}_1 \times \cdots \times \mathcal{I}_{\mu-1} \times \mathcal{I}_{\mu+1} \times \cdots \times \mathcal{I}_d$. The formal notation is more complex than the concept of unfolding, indeed the map from the tensor entries $(i_1, i_2, \cdots, i_d) \in \mathcal{I}$ to the matrix entries $(i_\mu, j) \in \mathcal{I}_\mu \times \mathcal{I}^{(\mu)}$ is

$$j = 1 + \sum_{\substack{k=1 \\ k \neq \mu}}^{d} (i_k - 1) J_k \quad \text{with} \quad J_k = \prod_{\substack{m=1 \\ m \neq \mu}}^{k-1} I_m \tag{27}$$

Only the special case of mode-n matricization is considered here, further details are available in [**?**].

Remark 3.16. The ordering in which the matricization does not matter as long as it is consistent through the computation.
One can also vectorize a tensor, the same goes concerning ordering

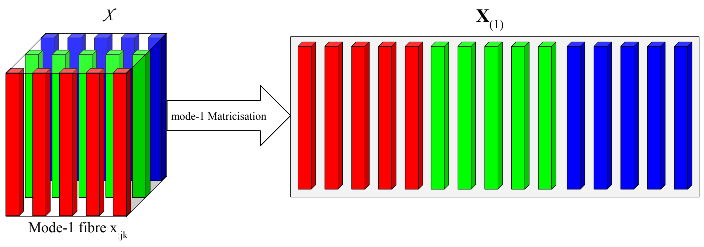


Figure 9: Mode one matricization of third order tensor with $\mathfrak{X} \in \mathbb{R}^{I \times J \times K}$.

Tensor multiplication It is possible to define product between tensors in a variety of ways. It does require more complex notations and symbols than for matrices. We restrict ourselves to the ones which are actively used to describe tensor reduction. Information about other tensor products is widely available in the literature [**? ?**].

Definition 3.17 (Tensor product)**.** The tensor product is a special case of the outer product that allows multiplication between tensors. It is denoted by $\otimes$ or $\circ$ if a confusion with the Kronecker product is possible. Let $\mathcal{I} = \mathcal{I}_1 \times \cdots \mathcal{I}_p$ and $\mathcal{J} = \mathcal{J}_1 \times \cdots \mathcal{J}_q$ be multi index series. The the tensor product is defined by

$$\begin{aligned} \otimes : \quad \mathbb{R}^{\mathcal{I}} \times \mathbb{R}^{\mathcal{J}} &\to \quad \mathbb{R}^{\mathcal{I} \times \mathcal{J}} \\ (\mathfrak{X}, \mathfrak{Y}) &\mapsto \quad \mathfrak{X} \otimes \mathfrak{Y} \end{aligned}$$

Enrty-wise $\mathfrak{T} = \mathfrak{X} \otimes \mathfrak{Y}$ writes

$$T_{\boldsymbol{ij}} = x_{\boldsymbol{i}} y_{\boldsymbol{j}}$$

where $\boldsymbol{i} = \{i_1, ..., i_p\}$ and $\boldsymbol{j} = \{j_1, ..., j_q\}$.

Definition 3.18 (Kronecker product)**.** Kronecker product of matrices $\mathbf{A} \in \mathbb{R}^{I \times J}$ and $\mathbf{B} \in \mathbb{R}^{K \times L}$ is denoted by $\mathbf{A} \otimes \mathbf{B}$. The result is a matrix of size $(IK) \times (JL)$ and defined by

$$\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}$$

Remark 3.19. It should be noted the outer product of vectors is a special case of the Kronecker product.

Definition 3.20 (Kathri-Rao product)**.** of matrices $\mathbf{A} \in \mathbb{R}^{I\times K}$ and $\mathbf{B} \in \mathbb{R}^{J\times K}$ is denoted by $\mathbf{A} \odot \mathbf{B}$. The result is a matrix of size $(IJ) \times (K)$ and defined by

$$\mathbf{A} \odot \mathbf{B} = [\boldsymbol{a}_1 \otimes \boldsymbol{b}_1 \quad \boldsymbol{a}_2 \otimes \boldsymbol{b}_2 \quad \cdots \boldsymbol{a}_K \otimes \boldsymbol{b}_K]$$

If $\boldsymbol{a}$ and $\boldsymbol{b}$ are vectors, then the Kathri-Rao product and Kronecker product are identical.

Definition 3.21 (Hadamard product)**.** It is the elementwise matrix product. Let $\boldsymbol{A}$ and $\boldsymbol{B} \in \mathbb{R}^{I\times J}$, their Hadamard product is denoted by $\boldsymbol{A} * \boldsymbol{B}$ and it is also of size $I \times J$.

$$\boldsymbol{A} * \boldsymbol{B} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1J}b_{1J} \\ a_{21}b_{21} & a_{22}b_{22} & \cdots & a_{2J}b_{2J} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}b_{I1} & a_{I2}b_{I2} & \cdots & a_{IJ}b_{IJ} \end{bmatrix} \tag{28}$$

These products have many properties[**?**] that are relied upon to devise decomposition algorithms.

Definition 3.22 (μ-mode product)**.** The *μ-mode (matrix) product*, for $1 \leq \mu \leq d$ of tensor $\mathcal{X} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ with matrix $\mathbf{A} \in \mathbb{R}^{m\times n_\mu}$ is denoted by $\mathcal{X} \times_d \mathbf{A}$ and is of size $n_1 \times \cdots \times n_{\mu-1} \times m \times n_{\mu+1} \times \cdots n_d$. Element-wise, we have

$$(\mathcal{X} \times_\mu \mathbf{A})_{i_1\cdots i_{\mu-1} j i_{\mu+1}\cdots i_d} = \sum_{i_\mu=1}^{n_\mu} x_{i_1 i_2 \cdots i_d} a_{j i_\mu}$$

It is equivalent to say that each mode-μ fiber is multiplied by the matrix A, i.e.
$\mathcal{Y} = \mathcal{X} \times_\mu \mathbf{A} \Leftrightarrow \mathbf{Y}_{(\mu)} = \mathbf{A}\mathbf{X}_{(\mu)}$.

Definition 3.23 (multilinear multiplication [**?**])**.** Multilinear multiplication in one mode is equivalent to n-mode multiplication but is usefull to introduce a new notation

$$[(\boldsymbol{I}, ...\boldsymbol{I}, \boldsymbol{M}, \boldsymbol{I}, ..., \boldsymbol{I}) \cdot \mathcal{X}]^{(n)} = \boldsymbol{M}\boldsymbol{X}^{(n)} \tag{29}$$

Then in general, the unfolding of a multilinear multiplication is given by

$$[(\boldsymbol{M}_1, \cdots, \boldsymbol{M}_d) \cdot \mathcal{X}]^{(n)} = M_n \boldsymbol{X}^{(n)} (\boldsymbol{M}_1 \otimes \cdots \otimes \boldsymbol{M}_{n-1} \otimes \boldsymbol{M}_{n+1} \otimes \cdots \otimes \boldsymbol{M}_d)^\mathsf{T}$$

and multilinear multiplications can be transformed into one, as follow

$$(\boldsymbol{L}_1, \boldsymbol{L}_2, \cdots, \boldsymbol{L}_d)\,[(\boldsymbol{M}_1, \ldots, \cdots, \boldsymbol{M}_d) \cdot \mathcal{X}] = (\boldsymbol{L}_1\boldsymbol{M}_1, \boldsymbol{L}_2\boldsymbol{M}_2, \cdots, \boldsymbol{L}_d\boldsymbol{M}_d) \cdot \mathcal{X}$$

Tensor Formats

In the following subsections, some of the most common tensor formats or representations are described. Indeed, in applications one needs to represent the properties of a tensor using a finite numbers of parameters. Not all tensors belong to spaces of finite dimension (e.g. tensor Hilbert spaces), then the question of finite approximation arises. The decomposition or approximation of a tensor in a certain format is addressed in the next section.

Before entering these descriptions, one should note the difference between *representation* and *decomposition* that are complementary notions. On the one hand, the *representation* of a tensor is any way used to describe a tensor using a set of parameters $(p_1, ..., p_n)$ e.g. representation of tensor $\mathcal{X}$ on a computer using full real array format : $(p_1, ..., p_n) \to \mathcal{X}$. On the other hand, the *decomposition* does the opposite way by analyzing a tensor to determine a set of properties : $\mathcal{X} \to (p_1, ..., p_n)$.

These operation can be used alternately, for example the CP decomposition of a tensor yields a representation of it with a given accuracy. This leads to the following statement by Hackbush : *"'tensor decomposition' is applied, when features of a concrete object should be characterized by parameters of a tensor-valued data about this object"*.

For the sake of simplicity, the following presentation uses d-way array formats but they have equivalent versions for arbitrary tensor spaces so long as a finite basis exists.

3.3 Full format

Let $\mathcal{I} = I_1 \times \cdots \times I_d$ a d-fold product index and a tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$. Then the full format consists in storing the values taken by $\mathcal{X}$ for all $(i_1, ..., i_d) \in \mathcal{I}$ with the standard basis $\boldsymbol{e}_{\mu,i_\mu} \in \mathbb{R}^{\mathcal{I}^\mu}$ is defined by $(\boldsymbol{e}_{\mu,i_\mu})_{j_m u} = \delta_{i_\mu,j_\mu}$. We have

$$\mathcal{X} = \sum_{\boldsymbol{i} \in \mathcal{I}} x_{\boldsymbol{i}} \, \boldsymbol{e}_{1,i_1} \otimes \cdots \otimes \boldsymbol{e}_{d,i_d} \tag{30}$$

Storage. Since the basis is trivial, it is not needed to store the basis and the storage cost is $\prod_{\mu=1}^{d} n_\mu$. Let $n = \max_{\mu \in D} n_\mu$ then the storage cost is in $\mathcal{O}(n^d)$ which is intractable for large d. A more general definition of full format for tensors is given by J. Ballani in his thesis dissertation [**?**].

Evaluation Cost. The evaluation cost in full format is null since one just need to recover the value at a given index in the computer memory.

3.4 Canonical format or r-term format $\mathcal{C}_r$

Definition 3.24 (Canonical Format)**.** In this format, any tensor $\mathbf{\mathcal{X}} \in V = \bigotimes_{\mu=1}^{d} V_\mu$ a tensor space, is written as the finite sum of rank-1 tensors. $\mathbf{\mathcal{X}} \in \mathcal{C}_r(\mathbb{R}^{\mathcal{I}})$ is said to be represented in the canonical format and it reads,

$$\mathbf{\mathcal{X}} = \sum_{i=1}^{r} \bigotimes_{\mu=1}^{d} \boldsymbol{u}_{\mu,i} \quad \text{where } \boldsymbol{u}_{\mu,i} \in V_\mu = \mathbb{R}^{\mathcal{I}_\mu} \tag{31}$$

where $\boldsymbol{U}_\mu = [\boldsymbol{u}_{\mu,1}\ \boldsymbol{u}_{\mu,2}\ \cdots\ \boldsymbol{u}_{\mu,r}]$ for $\mu \in D$. The μ-matricization of $\mathbf{\mathcal{X}}$ can be computed by

$$\boldsymbol{X}_{(\mu)} = \boldsymbol{U}_\mu (\boldsymbol{U}_1 \odot \cdots \odot \boldsymbol{U}_{\mu-1} \odot \boldsymbol{U}_{\mu-1} \odot \cdots \odot \boldsymbol{U}_d)^\intercal \tag{32}$$

Remark 3.25.
a. r, the length of the sum, is the tensor rank of $\mathbf{\mathcal{X}}$ as stated in definition 3.11. However, the reader is reminded that computing the rank of an arbitrary tensor is a NP-complex problem.
b. $\mathcal{C}_r$ is not a linear space since the sum of $\mathbf{\mathcal{X}}, \mathbf{\mathcal{Y}} \in \mathcal{C}_r$ belongs to $\mathcal{C}_{2r}$ and $\mathbf{\mathcal{X}} + \mathbf{\mathcal{Y}} \notin \mathcal{C}_r$ in general.

Storage. Accordingly to the previous remark, it is assumed that r is known since the tensor is already in $\mathcal{C}_r$. Then each parameter vector $(\boldsymbol{u}_{\mu,i})$ storage complexity is in $\mathcal{O}(\#\mathcal{I}_\mu)$ which leads to the following tensor storage complexity in $\mathcal{C}_r$ with $n = \max_{\mu \in D}(n_\mu)$.

$$N_{storage}(\mathcal{C}_r) = r \sum_{\mu=1}^{d} \mathcal{I}_\mu = \mathcal{O}(drn) \tag{33}$$

If r remains small then the storage complexity remains moderate even for a large number of dimensions.

Evaluation. The evaluation of a single entry $x_{\boldsymbol{i}}$, $\boldsymbol{i} = (i_1, ..., i_d) \in \mathcal{I}$ of $\mathbf{\mathcal{X}} \in \mathcal{C}_r$ requires the multiplication of the values $(u_{\mu,i})_{i_\mu}$ for $\mu \in D$. Indeed $x_{\boldsymbol{i}} = \sum_{j=1}^{r} \prod_{\mu=1}^{d} (u_{\mu,j})_{i_\mu}$ which means the complexity to evaluate a single entry is $N_{entry}(\mathcal{C}_r) = dr$ leading to the following complexity to evaluate the whole tensor

$$N_{full\ eval}(\mathcal{C}_r) = \mathcal{O}(n^d dr) \tag{34}$$

This cost is optimal in the sense of linear complexity, however the non-linearity of the space raises the question of truncation or approximation which is treated in section 4.1.

As for the full format, $\mathcal{C}_r$ is fully compatible with other underlying vector spaces. Further information is available in [**?** **?**].

3.5 Tucker format $\mathcal{T}_k$

This section focuses on the crucial Tucker format which consists for $\mathcal{X} \in V = \mathbb{R}^{\mathcal{I}}$ in finding smaller subspaces $U_\mu \subset V_\mu$ such that $\mathcal{X} \in \bigotimes_{\mu=1}^{d} U_\mu$. Indeed if $k_\mu = \dim(U_\mu) < \dim(V_\mu)$ then $\mathcal{X}$ can be represented more efficiently than in full representation. This leads to the following definition.

Definition 3.26 (Tucker format $\mathcal{T}_k$)**.** Let $\boldsymbol{k} = (k_1, ..., k_d) \in \mathbb{N}^d$ and a family of linearly independent vectors $(\boldsymbol{u}_{\mu,i})_{\mu,1\leq i\leq k_\mu}$ for $\mu \in D$ such that $(\boldsymbol{u}_{\mu,i})_{\mu,1\leq i\leq k_\mu}$ is a basis of U_μ. Then the tucker representation of $\mathcal{X} \in U$ is

$$\mathcal{X} = \sum_{i_1=1}^{k_1} \cdots \sum_{i_d=1}^{k_d} w_{i_1,\dots,i_d} \boldsymbol{u}_{1,i_1} \otimes \cdots \otimes \boldsymbol{u}_{d,i_d} \tag{35}$$

with the weights $w_{i_1,\dots,i_d} \in \mathbb{R}$. They form the core tensor $\mathcal{W} \in \mathbb{R}^{k_1\times\cdots\times k_d}$. $\boldsymbol{k}$ is the representation rank (or *Tucker rank*) of $\mathcal{X}$ in the tucker format $\mathcal{T}_{\boldsymbol{k}}$. One can also write $\mathcal{X}$ as a product of $\mathcal{W}$ and matrices $\boldsymbol{U}_\mu = [(\boldsymbol{u}_{\mu,i})]_{i=1}^{k_\mu}$ which reads

$$\mathcal{X} = \mathcal{W} \times_1 \boldsymbol{U}_1 \times_2 \boldsymbol{U}_2 \cdots \times_d \boldsymbol{U}_d. \tag{36}$$

Its μ-matricized version reads

$$\boldsymbol{X}_{(n)} = \boldsymbol{U}_\mu \boldsymbol{W}_{(\mu)} (\boldsymbol{U}_1 \otimes \cdots \otimes \boldsymbol{U}_{\mu-1} \otimes \boldsymbol{U}_{\mu+1} \otimes \cdots \otimes \boldsymbol{U}_d)^\intercal. \tag{37}$$

Remark 3.27. a. As stated by Ballani, for general tensors, $\mathcal{T}_{\boldsymbol{k}}$ the set of tensors whose Tucker representation rank is lower than $\boldsymbol{k}$ is not a linear space.

b. The tuple formed of all the μ-ranks is the lowest $\boldsymbol{k}$ for which $\mathcal{X} \in \mathcal{T}_{\boldsymbol{k}}$.

Storage complexity. In order to represent a tensor in $\mathcal{T}_{\boldsymbol{k}}$ format, one only needs to store the core tensor of size $\mathcal{O}(\prod_{\mu=1}^{d} k_\mu)$ and the basis vectors stored in matrices for each dimension of size $\mathcal{O}(k_\mu n_\mu)$. This yields a total storage complexity of

$$N_{storage}(\mathcal{T}_{\boldsymbol{k}}) = \prod_{\mu=1}^{d} k_\mu + \sum_{\mu=1}^{d} k_\mu n_\mu = \mathcal{O}(k^d + dkn) \tag{38}$$

One can clearly see that the term $\mathcal{O}(k^d)$ is very interesting if d is small since overhead cost compared with $\mathcal{C}_r$ is limited. However if d grows above 5, it will become impossible to use this format even if k remains small.

Evaluation complexity. In order to evaluate a single entry of a tensor in tucker format, one needs to compute the sum 35. Each term of the sum requires $(d+1)$ operations which leads to the entry evaluation complexity complexity of

$$N_{entry\ eval}(\mathcal{T}_{\boldsymbol{k}}) = (d+1) \prod_{\mu=1}^{d} k_\mu \tag{39}$$

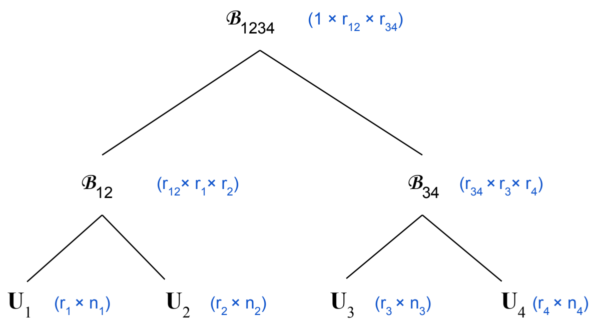
Then the overall complexity to evaluate the full tensor is in $\mathcal{O}((d+1)k^d n^d)$ which is very costly. However this representation remains interesting since the evaluation of the tucker rank only requires standard linear algebra tools and approximations of lower rank are easily accessible through HOSVD. See sections 4.2.

3.6 Hierarchical Tucker format $\mathcal{H}_k$

When dimension d gets above 5 to 10 Tucker format is not a relevant solution due to the core tensor exponential growth with d. Among the alternatives, the so-called Hierarchical Tucker (HT) format has gained momentum in the last decade since it was proposed by Grasedyck and collaborators [**?**]. It is based on the idea of recursively splitting the modes of the tensor. The process results in a binary tree $\mathcal{T}_D$ containing a subset $t \subset D := \{1, ..., d\}$ at each node e.g. figure 10 which leads to a linear growth of the storage cost with respect to the d. This approach introduces a new level of complexity that is beyond the scope of this chapter. Consequently, the reader is referred to the following papers [**? ? ? ?**] for theoretical development while [**?**] provide a MATLAB library. Among the numerous advantages of this format, it is shown that canonical, Tucker and TT (next section) formats can be represented exactly in HT format since they are subsets of the HT set. Consequently, the various algorithms to compute such decompositions can be transposed to this format with notorious efficiency gains regarding Tucker decomposition. Indeed, leaf to root truncation methods allow very efficient implementation for large numbers of dimensions. However, this versatility comes at the cost of complexity of the mathematical as well as digital setup. This is why we focus on the next section on the tensor train format introduced by Oseledets [**?**] which provides linear growth storage with a simple "train" structure.

3.7 Tensor Train format

The tensor train format (TT) is a special case of hierarchical tensor formats which displays some advantages. It was popularized by Oseledets et al. [**?**] followed by a substantial series of paper that is condensed in [**?**



$\mathcal{B}_{1234}$ $(1 \times r_{12} \times r_{34})$
$\mathcal{B}_{12}$ $(r_{12} \times r_1 \times r_2)$
$\mathcal{B}_{34}$ $(r_{34} \times r_3 \times r_4)$
$\mathbf{U}_1$ $(r_1 \times n_1)$
$\mathbf{U}_2$ $(r_2 \times n_2)$
$\mathbf{U}_3$ $(r_3 \times n_3)$
$\mathbf{U}_4$ $(r_4 \times n_4)$

Figure 10: Tree representation of the HT format of $\mathfrak{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}$. The size of the matrices and tensors are inside blue braces.

]. This format was first presented as a product of matrices that describes each element of the tensor which is why it is also known as matrix product state (MPS) in the literature. Entry-wise, $\mathfrak{X} \in \mathbb{R}^{\mathcal{N}}$ is given by the following product of matrices

$$x_{i_1,\dots,i_d} = \boldsymbol{G}_1(i_1)\boldsymbol{G}_2(i_2)\cdots\boldsymbol{G}_d(i_d), \quad \boldsymbol{G}_\mu \in \mathbb{R}^{k_{\mu-1}\times k_\mu} \tag{40}$$

where $k_0 = k_d = 1$. For every mode μ and every index i_μ the coefficients $\boldsymbol{G}_\mu(i_\mu)$ are matrices. There is no specific assumption on the orthogonality of the modes $G(:)_{i,j}$, only the construction of such representation may ensure it. The following definitions comes naturally.

Definition 3.28 (TT-decompostion)**.** Let $G_\mu \in \mathbb{R}^{k_{\mu-1}\times n_\mu \times k_\mu}$ for all $\mu \in [\![1, d]\!]$ a set of order 3 tensors called *TT-cores.* Then the order d tensor $\mathfrak{X} \in \mathcal{R}^{\mathcal{N}}$ with *TT-rank* $\boldsymbol{r} = \{k_i\}_{i=0}^d$ $(k_0 = k_d = 1)$ has the following TT decomposition

$$\mathfrak{X} = \sum_{\alpha_0,\dots,\alpha_d=1}^{\boldsymbol{k}} G_1(\alpha_0, i_1, \alpha_1)\cdots G_1(\alpha_{d-1}, i_d, \alpha_d) \tag{41}$$

Additionally, the TT format can be seen as a special case of the HT format with a linear structure. Here, all nodes have at least one son that is a leaf. One can see in Fig. 11 the link between HT and TT regarding the shape of the tree while Fig. 12 shows the dimension tree associated with TT format.

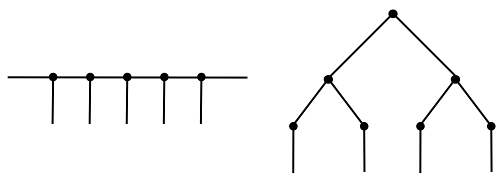


Figure 11: A graph representation of TT (left) and HT (right) format highlighting their similarities and differences.

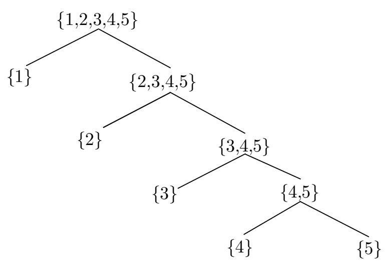


Figure 12: "Recursive" dimension tree associated with the extended tensor train of a 5th order tensor

Storage Complexity. It can be easily shown [?] that the storage cost is

$$\mathcal{O}(k^2dn) \tag{42}$$

where $k = \max_{t\in T_{\mathcal{I}}} k_t$ and $n = \max_{i\in D} n_i$.

Evaluation Complexity. In order to evaluate one entry of the tensor, one simply needs to apply (40), which yields with the usual assumption on the rank and dimension of $\mathfrak{X}$

$$N_{\text{entry eval}}(TT) = (d-1)k^3 \tag{43}$$

Remark 3.29. By construction, it is very easy (and cheap) to evaluate a single entry of a tensor. Same goes with very efficient algorithm for numerical integration/contraction as given by Oseledets [?].

Remark 3.30. Linear operations are straightforward to implement in TT format, including multiplications with matrices, vectors, tensor products, Hadamard product. See [? , Sec. 4] for details and algorithms as it is out of the scope of this manuscript.

TT format possesses many of the required properties for tensor reduction:

- simple structure,
- easier to handle than HT,
- any tensor can be represented exactly,
- memory complexity that scales linearly with d,
- straightforward multilinear algebra operations.

However, the bases associated with each space do not appear explicitly. Indeed the long fibers (middle dimension) of the cores span a vector space but do not form an orthonormal basis (naturally). This is a problematic feature for physics related applications where one usually wants to manipulate modes directly whether it is for analysis or processing.

Consequently, TT format needs to be improved for our applications. The reader might refer to the literature survey [?] for a bibliographic overview and a theoretical presentation of TT is given in [? , Chap. 12].

Remark 3.31. An extended tensor train format which displays the same recursive structures as TT while leaving direct access to the modes is also possible to use. Once can read details and description in [?]

4 Higher Order Decomposition methods

In this section, we finally tackle the approximation of tensors to reduced rank. This allows huge storage savings as each of the presented formats separates dimensions thus breaking the curse of dimensionality as long as the rank is kept small. In this section, three decomposition methods are studied starting with canonical decomposition. Then higher order SVD is used to compute truncated Tucker representations. Finally Tensor train decomposition through SVD is described. Hierarchical decompositions are obtained by reorganizing data in the other formats through algorithms that have been omitted in this document. Indeed, it does not improve the decomposition properties, only the storage cost is reduced. Thus due to the increased complexity, it was decided not to study Hierarchical Tucker decomposition, the reader is referred to [? ? ?] for additional information and implementations.

In order to describe decomposition techniques which are ways to approximate a tensor into a particular format, it is necessary to first define what is a best approximation.

Definition 4.1 (best approximation)**.** Let $(\mathcal{V}, \|\cdot\|)$ be a normed vector space and let $\emptyset \neq \mathcal{U} \subseteq \mathcal{V}$. An element $u_{\text{best}} \in \mathcal{U}$ is called a *best approximation* of $v \in \mathcal{V}$ (with respect to $\mathcal{U}$ if

$$\|v - u_{\text{best}}\| \leq \|v - u\| \quad \forall u \in \mathcal{U}$$

4.1 CP decomposition

The idea of decomposing a tensor as a finite sum of rank one tensors was first expressed by Hitchcock in 1927 [**?**] which he called polyadic form. It finally became popular when reintroduced by Caroll and Chang [**?**] in the form of CANDECOMP and Harshman [**?**] as PARAFAC (parallel factors). Then the method CANDECOMP/PARAFAC is referred as *CP Decomposition* but it can be found under other names such as polyadic decomposition of Topographic components models.

The CP decomposition yields a tensor in the canonical format $\mathcal{C}_r$.

Definition 4.2. The CP decomposition of a tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ is to factorize it into a finite sum of rank-one tensors i.e. it is an approximation of a tensor of $\mathbb{R}^{\mathcal{I}}$ in $\mathcal{C}_r$. It means that either of these problems have to be solved

a. Given $\varepsilon > 0$, find $\tilde{\mathcal{X}} \in \mathcal{C}_r$ with minimal $r \in \mathbb{N}^*$ such that $\|\tilde{\mathcal{X}} - \mathcal{X}\| \leq \varepsilon$.

b. Given $r \in \mathbb{N}$, find $\tilde{\mathcal{X}} \in \mathcal{C}_r$ that minimizes the error $\varepsilon = \|\tilde{\mathcal{X}} - \mathcal{X}\|$

Given that either of these problem has a solution the following, approximated identity is obtained

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \sum_{i=1}^{r} \bigotimes_{\mu=1}^{d} \tilde{\boldsymbol{x}}_{\mu}^{i} \tag{44}$$

Remark 4.3. $\tilde{\mathcal{X}}$ can be seen as the optimal projection of $\mathcal{X}$ on $\mathcal{C}_r$.

Example 4.4 (3D case)**.** Then we want to write the CP decomposition of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ a rank 3 tensor with $R \in \mathbb{N}_+$ terms

$$\mathcal{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \tag{45}$$

where $\mathbf{a}_r \in \mathbb{R}^{n_1}$, $\mathbf{b}_r \in \mathbb{R}^{n_2}$ and $\mathbf{c}_r \in \mathbb{R}^{n_3}$. Alternatively, it can be written element-wise as

$$\forall (i,j,k) \in [\![1, n_1]\!] \times [\![1, n_2]\!] \times [\![1, n_3]\!], \quad x_{ijk} \approx \sum_{r=1}^{R} a_{ir} b_{jr} c_{kr}$$

Figure 13 displays a visual of the CP decomposition where the rank one tensors are represented directly as a product of vectors.

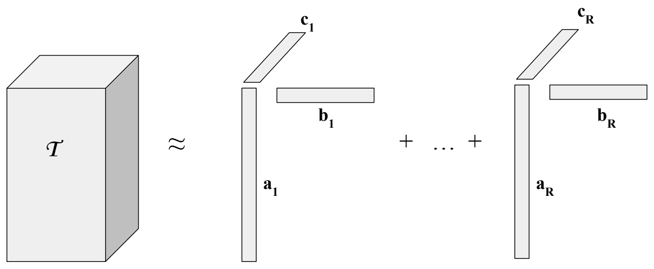


Figure 13: CP decomposition of third order tensor $\mathfrak{X} \in \mathbb{R}^{I\times J\times K}$.

The matrix $\mathbf{A}$ formed by the combination of vectors from the rank-one components (the *factor vectors*) i.e. $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2 \cdots \mathbf{a}_R]$ likewise for each dimension is referred as *factor matrices.* Then Kolda introduced the following concise notation for CP decomposition

$$\mathfrak{X} \approx [\![\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!] \equiv \sum_{r=1}^{R} \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r$$

It is of practical interest to assume that the factor vectors are normalized to one and their weights are stored into a vector $\boldsymbol{\lambda} \in \mathbb{R}^R$ so that

$$\mathfrak{X} \approx [\![\boldsymbol{\lambda}; \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!] \equiv \sum_{r=1}^{R} \lambda_r\, \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r \tag{46}$$

Remark 4.5. There is *no direct algorithm* to compute the optimal CP decomposition of a tensor, the problem is NP complex [**?**]. Although the uniqueness condition for rank decomposition is weaker for tensors than for matrices (permutation and scaling are allowed), it is often unique (e.g. [**?**]). Some criteria for uniqueness have been proposed in the literature.

Existence of a low rank approximation in $\mathcal{C}_r$

Lemma 4.6 ([**?** , Remark 9.1] and [**?** , Lemma 4.7])**.** *Problem (a) in definition 4.2 has a solution.*

For a matrix, the best rank-k approximation is given by the k first factors of the Singular Value Decomposition of that matrix (see 2.1). Then for $d = 2$

problem 4.2(b) has a solution however statement becomes false for tensors of higher order.

A tensor is called *degenerate* if several rank-k approximation give the same arbitrary approximation, in this case there is no best rank-k approximation. The best rank-k approximation may not be found *sequentially*, e.g. the best rank one approximation of $\mathcal{X}$ may not be found by minimizing the distance to the best rank 2 approximation of $\mathcal{X}$. Then all factors must be found *simultaneously* to ensure optimality.

Lemma 4.7 (Special case $\mathcal{C}_1$)**.** *The set $\mathcal{C}_1$ is closed for all $d \in \mathcal{N}^*$.*

Indeed $\mathcal{T}_{1,\ldots,1} = \mathcal{C}_1$ and $\mathcal{T}_{\boldsymbol{k}}$ is closed for any $\boldsymbol{k}$ [**?** , Lemma 4.20]. This means that problem 4.2(b) has a solution in $\mathcal{C}_1$. However this is not true for higher ranks if $d \geq 3$, indeed it has been shown repeatedly [**? ?**] that $\mathcal{C}_r$ is not closed in these conditions. Ballani provides a nice view of the issue [**?** , Lemma 4.15]. The literature provides abundant examples of series of rank r tensors converging toward a rank $r+1$ tensor. This is mainly due to severe cancellation effects.

Lemma 4.8. *Given $r \geq 2$ and $d \geq 3$, the set $\mathcal{C}_r$ is not closed.*

It means that in the general case, problem 4.2(b) does not necessarily have a solution. The occurrence of such tensors is not rare event, see [**?**]. The next set is introduced in order to overcome these difficulties.

Lemma 4.9 ([**?** , Lemma 4.16])**.** *Let $r \in \mathbb{N}^*$ and $c > 0$. The set*

$$\mathcal{C}_r^c = \left\{ \sum_{j=1}^{r} \mathcal{X}_j \ : \ \mathcal{X}_j \in \mathcal{C}_1(\mathbb{R}^{\mathcal{I}}), \|\mathcal{X}_j\| \leq c, \ j = 1, \ldots, r \right\} \subset \mathcal{C}_r(\mathbb{R}^{\mathcal{I}})$$

is closed.

Corollary 4.10. *Let $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$. The following problem has a solution : Given $r \in \mathbb{N}$ and $c > 0$, find a tensor $\tilde{\mathcal{X}} \in \mathcal{C}_r^c$ that minimizes the error $\varepsilon = \|\tilde{\mathcal{X}} - \mathcal{X}\|$.*

Several algorithms ensure the boundedness of the norms of the terms $\mathcal{X}_j$ but the drawback is the existence of local minima which are usually not a problem in practical applications. Next section introduces a classical CP decomposition algorithm.

Computing the CP decomposition : the ALS algorithm Although there are many approaches to compute a CP decomposition, in this section we focus on the classical Alternating Least Square (**ALS**) approach. This method was introduced by Carroll and Chang [**?**] and Harshman [**?**]. If not the most efficient it is highly reliable and quite simple. To ease the presentation we stick to a third order tensor although the algorithm can be easily extended to a d-way tensor.

Let $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ a third order tensor. $\tilde{\mathcal{X}}$, the best rank-R approximation of $\mathcal{X}$ is sought i.e.

$$\min_{\tilde{\mathcal{X}}} \|\mathcal{X} - \tilde{\mathcal{X}}\| \quad \text{with} \quad \tilde{\mathcal{X}} = [\![\boldsymbol{\lambda}; \boldsymbol{A}, \boldsymbol{B}, \mathbf{C}]\!] \equiv \sum_{r=1}^{R} \lambda_r \, \boldsymbol{a}_r \circ \boldsymbol{b}_r \circ \boldsymbol{c}_r \tag{47}$$

The ALS approach is to fix $\boldsymbol{B}$ and $\boldsymbol{C}$ to solve for $\boldsymbol{A}$ then fix $\boldsymbol{A}$ and $\boldsymbol{C}$ to solve for $\boldsymbol{B}$ etc. until the procedure converges. Having fixed all but one matrices, the problem reduces to a linear least-square problem which can be solved using the usual tools. Although this algorithm is quite simple to implement and understand, it does not necessarily converges to the global minimum of the objective function. Only a local minimum is ensured. Moreover, it can take a large number of iteration to converge. Finally, its result may depend on the arbitrary initial values (see Kolda [**?**] for a detailed algorithm).

Algorithm 2: ALS

```
  input  : $\mathcal{F} \in \mathbb{R}^{I_1\times\cdots\times I_d}$
  output: $\mathcal{X} = w \bigotimes_{i=1}^{d} \boldsymbol{x}_i$
  Initialize $\forall 1 \le i \le d, \quad \boldsymbol{x}_i$ ;
  while $Error \ge \varepsilon$ do
      for $i = 1, d$ do
1         $V = \boldsymbol{X}_1{}^\intercal \boldsymbol{X}_1 * \cdots * \boldsymbol{X}_{i-1}{}^\intercal \boldsymbol{X}_{i-1}{}^\intercal * \boldsymbol{X}_{i+1}{}^\intercal \boldsymbol{X}_{i+1} * \cdots * \boldsymbol{X}_d{}^\intercal \boldsymbol{X}_d$
          ;                                                  /* $V \in \boldsymbol{R}^{R\times R}$ */
2         $\boldsymbol{X}_i = \mathcal{F} \cdot (\boldsymbol{X}_d \odot \cdots \odot \boldsymbol{X}_{i+1} \odot \boldsymbol{X}_{i-1} \odot \cdots \odot \boldsymbol{X}_1) V^\dagger$ ;     /* $\dagger$
           refer to the Monroe-Penrose pseudo-inverse */
          $w_i = \|\boldsymbol{X}_i\|_2$;
          $\boldsymbol{X}_i = \frac{\boldsymbol{X}_i}{w_i}$
  return $\mathcal{X} = [\![\boldsymbol{w}; \boldsymbol{X_1}, \cdots, \boldsymbol{X_d}]\!]$
```

This algorithm led to many developments but they are generally outperformed in the production stage by several Tucker Decomposition methods such as the HOSVD which will be discussed in the next section.

It is possible to rewrite the CP format using vector spaces of unknown nature such as infinite spaces. Still one needs to define storage on a computer the continuous bases function for example. The case of function decomposition into CP format is studied in section 4.6.

4.2 Tucker decomposition

The Tucker decomposition was first introduced by Tucker during the 1960s [**?**] and further refined. As for the CP decomposition, the Tucker Decomposition has been "rediscovered" many times in several fields leading to several names (HOSVD [**? ?**], N-Modes PCA, etc.). It is an extension of the SVD to higher dimensions. A tensor is decomposed into a *core* tensor that is multiplied by a matrix along each mode.

Once again, the case of a third order tensor is proposed for introduction simplicity. But, the Tucker decomposition is well defined for dimensions higher than 3. Figure 14 shows a graphical interpretation of the following equation for $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$,

$$\boldsymbol{\mathcal{X}} \approx [\![\boldsymbol{\mathcal{W}}; \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}]\!] = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} w_{pqr}\, \boldsymbol{a}_p \circ \boldsymbol{b}_q \circ \boldsymbol{c}_r \tag{48}$$

Where $\boldsymbol{A} \in \mathbb{R}^{I \times P}$, $\boldsymbol{B} \in \mathbb{R}^{J \times Q}$ and $\boldsymbol{C} \in \mathbb{R}^{K \times R}$ are the factor matrices. There are usually set orthonormal and can be viewed as the principal components of each modes. $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{P \times Q \times R}$ is the *core tensor.* If $I < P$, $J < Q$ and $K < R$ then it can be seen as the compression of $\boldsymbol{\mathcal{X}}$ given the basis formed by $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$.

Element wise, the tucker decomposition in 48 is $\forall (i, j, k) \in [\![1, I]\!] \times [\![1, J]\!] \times [\![1, K]\!]$,

$$x_{ijk} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} w_{pqr}\, a_{ip}\, b_{jq}\, c_{kr}$$

It is easy to find the exact decomposition of a rank-$(R_1, ..., R_D)$ tensor (see def. 3.13) as presented in the next subsection. However, if one wants to compute a rank-$(R_1, ..., R_D)$ Tucker decomposition of a tensor where $\exists\, n \leq D \mid R_n < \mathrm{rank}_n(\boldsymbol{\mathcal{X}})$ then this decomposition is necessarily inexact which may raise some computational difficulties. Since such a decomposition excludes some eigen vectors, it is called a *truncated* Tucker decomposition, a visual example is shown in figure 15.

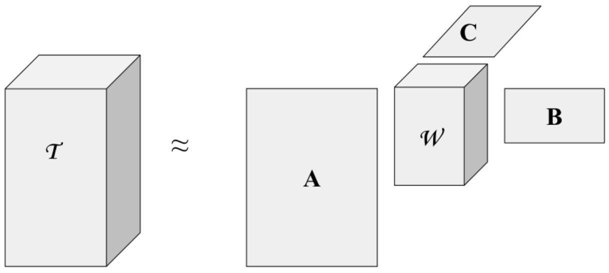


Figure 14: Tucker Decomposition of a third order array $\mathfrak{T}$

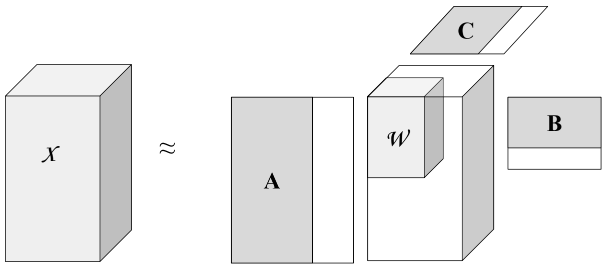


Figure 15: *Truncated* Tucker Decomposition of a third order array $\mathfrak{X}$

It should be noted that there are many ways to compute truncated tucker decompositions, among them various ALS based methods and the Higher Order Orthogonal Iteration (HOOI) proposed by [**?**] which yields some optimality properties. Finally, the most common method, because it is computationally the most efficient, is the Higher Order Singular Value Decomposition (**HOSVD**) which was introduced by [**?**].

In this paragraph, some mathematical properties of the Tucker decomposition are reviewed. They lead to the classical tensor Tucker format reduction technique Higher Order Singular Value Decomposition (HOSVD) which is presented in two forms. The first one was proposed by [**?**] and the second one is an improvement from [**?**], the Sequentially Truncated HOSVD.

Definition 4.11. The Tucker decomposition of a tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ is to find an approximation of a tensor of $\mathbb{R}^{\mathcal{I}}$ in $\mathcal{T}_{\boldsymbol{k}}$. It means that either of these problems have to be solved

a. Given $\varepsilon > 0$, find $\tilde{\mathcal{X}} \in \mathcal{T}_{\boldsymbol{k}}$ with minimal $N_{\text{storage}}(\mathcal{T}_{\boldsymbol{k}})$ such that $\|\tilde{\mathcal{X}} - \mathcal{X}\| \leq \varepsilon$.
b. Given $\boldsymbol{k} \in (\mathbb{N}*)^d$, find $\tilde{\mathcal{X}} \in \mathcal{T}_{\boldsymbol{k}}$ that minimises the error $\varepsilon = \|\tilde{\mathcal{X}} - \mathcal{X}\|$.

Given that either of these problem has a solution the following identity is obtained

$$\mathcal{X} \approx \tilde{\mathcal{X}} = \sum_{j_1=1}^{k_1} \cdots \sum_{j_d=1}^{k_d} w_{\boldsymbol{j}} \bigotimes_{\mu=1}^{d} \tilde{\boldsymbol{x}}_{\mu}^{i} \tag{49}$$

Lemma 4.12. *Problem (a) has a solution.*

Lemma 4.13. *Let $\boldsymbol{k} = (k_1, ..., k_d) \in (\mathbb{N}*)^d$. The set $\mathcal{T}_{\boldsymbol{k}} \subset \mathbb{R}^{\mathcal{I}}$ is closed. Consequently Problem (b) has a solution.*

Tucker format is closely related to the matricization of tensors. Then the idea of using the SVD (see 2.1) on matricizations of the investigated tensor has been used to devise algorithms to give an approximate solution to problems 4.11(a) and (b). For most applications, it is not necessary to find the best approximation, an *almost* best approximation is sufficient.

Theorem 4.14 (HOSVD as proved in [**?**])**.** *Every tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ admits a* higher-order singular value decomposition*:*

$$\mathcal{X} = (\boldsymbol{U}_1, \boldsymbol{U}_2, ..., \boldsymbol{U}_d) \cdot \mathcal{W}, \tag{50}$$

where the factor matrix $\boldsymbol{U}_\mu$ *is an orthogonal $n_\mu \times n_\mu$ matrix, obtained from the SVD of the mode-μ matricization of $\mathcal{X}$,*

$$\boldsymbol{X}^{(\mu)} = \boldsymbol{U}_\mu \boldsymbol{\Sigma}_\mu \boldsymbol{V}_\mu{}^{\intercal}, \tag{51}$$

and the core tensor $\mathcal{W} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ *can be obtained from*

$$\mathcal{W} = (\boldsymbol{U}_1{}^{\intercal}, \boldsymbol{U}_2{}^{\intercal}, ..., \boldsymbol{U}_d{}^{\intercal}) \cdot \mathcal{X}, \tag{52}$$

Remark 4.15 (Truncation)**.** Theorem 4.14 refers to full HOSVD which is an exact Tucker decomposition. However it gives a lot of information about a studied tensor such as the multilinear rank, it is rarely the pursued goal. This kind of decomposition is aimed at extracting the most relevant information, possibly by reducing data size. The optimality of the SVD truncation encourages to think of truncating $(\boldsymbol{U}_\mu)$. This is what is done is the Truncated-HOSVD (T-HOSVD) which is generally referred as HOSVD. However in this section the T-HOSVD notation will be used in order to prevent confusion.

Algorithm idea. The T-HOSVD algorithm relies on the simple truncation idea. First compute $(\boldsymbol{U}_\mu)$ defined in equation (51) in each direction, then truncate to a given rank/column (set prior to computing). Finally compute $\boldsymbol{\mathcal{W}}^t$, the truncated core tensor projecting $\boldsymbol{\mathcal{X}}$ on the reduced basis $(\boldsymbol{U}_\mu^t)$ as in equation (52).

Of course the truncation of the SVD does not mean that the 2D optimality is preserved. Optimality is not the goal of most applications and this algorithm is easy to use, consequently a quasi-optimality is sufficient. The quasi-optimality with respect to the optimal rank-$\boldsymbol{k}$ approximation is given by the following theorem.

Theorem 4.16 (Quasi-optimality of the T-HOSVD [? , Property 10])**.** *Let $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{\mathcal{I}}$ with a μ-rank $\boldsymbol{r} = (r_1, ..., r_d) \in \mathbb{N}^d$. Given $\boldsymbol{k} = (k_1, ..., k_n) \in \mathbb{N}^d$, let $\boldsymbol{\mathcal{X}}_{best}$ be the best approximation of $\boldsymbol{\mathcal{X}}$ in $\mathcal{T}_{\boldsymbol{k}}$ i.e. $\boldsymbol{\mathcal{X}}_{best} = \operatorname{argmin}_{\boldsymbol{\mathcal{Y}} \in \mathcal{T}_{\boldsymbol{k}}} \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{Y}}\|_2$. Then the error of HOSVD projection is bounded by*

$$\|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}_{\text{hosvd}}\|_2 \leq \sqrt{\sum_{\mu=1}^{d} \sum_{j=k_\mu+1}^{r_\mu} \sigma_{\mu,j}^2} \leq \sqrt{d}\, \|\boldsymbol{\mathcal{X}} - \boldsymbol{\mathcal{X}}_{best}\|_2 \tag{53}$$

where the $\sigma_{\mu,j}$ are the singular values defined in equation (51)*.*

The approximation error of HOSVD is bounded by the middle term in equation (53), namely $\sqrt{\sum_{\mu=1}^{d} \sum_{j=k_\mu+1}^{r_\mu} \sigma_{\mu,j}^2}$. Forcing this term to be lower than a given ε leads to an adaptively truncated HOSVD for which an error bound is chosen.

Algorithm 3 presents the truncated HOSVD algorithm that computes $\boldsymbol{\mathcal{X}} \in \mathcal{T}_{\boldsymbol{k}}$ of rank $\boldsymbol{k}$ the approximation of $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. It is a rather compact algorithm given that one has efficient methods to compute basic tensor operations. The implementation simplicity of the algorithm is one of the main reason of its success.

Algorithm 3: T-HOSVD

```
input  : $\mathcal{X} \in \mathbb{R}^{n_1\times\cdots\times n_d}$ , imposed rank : $\boldsymbol{k} = (k_1, ..., k_d)$
output: $\widetilde{\mathcal{X}} = (\boldsymbol{U}_1, \cdots, \boldsymbol{U}_d) \cdot \mathcal{W}$
for $i = 1, d$ do
1   $\boldsymbol{X}^{(\mu)}$ =matricize($\mathcal{F}, \mu$) ;
2   $(\boldsymbol{U}_\mu, \boldsymbol{\Sigma}_\mu, \boldsymbol{V}_\mu{}^\intercal)$=SVD($\boldsymbol{X}^{(\mu)}$) ;
$\mathcal{W} = ((\boldsymbol{U}_1^{k_1})^\intercal, (\boldsymbol{U}_2^{k_2})^\intercal, ..., (\boldsymbol{U}_d^{k_d})^\intercal) \cdot \mathcal{X}$;
/* $\boldsymbol{U}_i^{k_i}$ contains the first $k_i$ columns of $\boldsymbol{U}_i$ */
return $\widetilde{\mathcal{X}} = [\![\mathcal{W}; \boldsymbol{U}_1, \cdots, \boldsymbol{U}_d]\!]$
```

Remark 4.17. This algorithm is easily parallelized to the number of dimension (lines **1** and **2**), each processor computing an SVD. Additionally, it is possible to reach higher level of parallelization using parallel linear algebra routines.

ST-HOSVD The Sequentially Truncated HOSVD (ST-HOSVD) was introduced by [**?**]. This method is a variation of the usual T-HOSVD. Basically, instead of throwing away most of the work performed by each SVD, it is chosen to keep that information and perform SVD sequentially -on a reduced tensor- along all dimensions. Since processing is sequential and the order in which the operations are performed has an influence on the approximation, the sequence order is stored in a vector $\boldsymbol{p}$. For the sake of simplicity, it is assumed that $\boldsymbol{p} = (1, 2, ..., d)$ even though many of the results depend on the permutations of $\boldsymbol{p}$.

The ST-HOSVD has been presented using successive projections. In this framework it is easy to both understand the idea of the method and to demonstrate its properties.

Definition 4.18 (Orthogonal multilinear projector)**.** An *orthogonal projector* is a linear transformation P that projects a vector $\boldsymbol{x} \in \mathbb{R}^n$ onto a vector space $E \subseteq \mathbb{R}^n$ such that the residual $\boldsymbol{x} - P\boldsymbol{x}$ is orthogonal to E. Such a projector can always be represented as in matrix form $P = \boldsymbol{U}\boldsymbol{U}^\intercal$ given that the columns of $\boldsymbol{U}$ form an orthonormal basis of E.

Then, De Silva [**?**] proposed the introduction of *orthogonal multilinear projectors* from tensor space $\boldsymbol{\mathcal{V}} = V_1 \otimes \cdots \otimes V_d$ onto $\boldsymbol{\mathcal{U}} = U_1 \otimes \cdots \otimes U_d \subset \boldsymbol{\mathcal{V}}$. It is given by

$$\pi_i \mathcal{X} := (I, ..., I, \boldsymbol{U}_i \boldsymbol{U}_i{}^\intercal, I, ..., I) \cdot \mathcal{X} \quad \text{with } \mathcal{X} \in \boldsymbol{\mathcal{V}} = \mathbb{R}^{\mathcal{I}} \tag{54}$$

Definition 4.19. ST-HOSVD [**?** , Def. 6.1.]
A rank-$(r_1, ..., r_d)$ *sequentially truncated higher-order singular value decomposition* (ST-HOSVD) of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{\mathcal{I}}$, corresponding to the processing order $\boldsymbol{p} = [1, 2, ..., d]$, is an approximation of the form

$$\hat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{p}} := (\hat{\boldsymbol{U}}_1, \hat{\boldsymbol{U}}_2, ..., \hat{\boldsymbol{U}}_d) \cdot \hat{\boldsymbol{\mathcal{W}}} \approx \boldsymbol{\mathcal{X}} \quad \in \mathbb{R}^{n_1 \times \cdots \times n_d} \tag{55}$$

whose *truncated core tensor* is defined as

$$\hat{\boldsymbol{\mathcal{W}}} := (\hat{\boldsymbol{U}}_1^{\mathsf{T}}, \hat{\boldsymbol{U}}_2^{\mathsf{T}}, ..., \hat{\boldsymbol{U}}_d^{\mathsf{T}}) \cdot \boldsymbol{\mathcal{X}} \quad \in \mathbb{R}^{r_1 \times \cdots \times r_d} \tag{56}$$

and every *factor matrix* $\hat{\boldsymbol{U}}_i^{\mathsf{T}} \in \mathbb{R}^{n_i \times r_i}$ has orthonormal columns. In terms of orthogonal multilinear projectors, one writes

$$\hat{\boldsymbol{\mathcal{X}}}_{\boldsymbol{p}} := \hat{\pi}_1 \hat{\pi}_2 \cdots \hat{\pi}_d \boldsymbol{\mathcal{X}} = (\hat{\boldsymbol{U}}_1 \hat{\boldsymbol{U}}_1^{\mathsf{T}}, \hat{\boldsymbol{U}}_2 \hat{\boldsymbol{U}}_2^{\mathsf{T}}, ..., \hat{\boldsymbol{U}}_d \hat{\boldsymbol{U}}_d^{\mathsf{T}}) \cdot \boldsymbol{\mathcal{X}}$$

The i-th *partially truncated core tensor* is defined as

$$\hat{\boldsymbol{\mathcal{W}}}^i := (\hat{\boldsymbol{U}}_1^{\mathsf{T}}, \hat{\boldsymbol{U}}_2^{\mathsf{T}}, ..., \hat{\boldsymbol{U}}_i^{\mathsf{T}}, \boldsymbol{I}, ..., \boldsymbol{I}) \cdot \boldsymbol{\mathcal{X}} \quad \in \mathbb{R}^{r_1 \times \cdots \times r_i \times n_{i+1} \times \cdots \times n_d} \tag{57}$$

with $\hat{\boldsymbol{\mathcal{W}}}^0 := \boldsymbol{\mathcal{X}}$ and $\hat{\boldsymbol{\mathcal{W}}}_d = \hat{\boldsymbol{\mathcal{W}}}$. The rank-$(r_1, ..., r_i, n_{i+1}, ..., n_d)$ *partial approximation to* to $\boldsymbol{\mathcal{X}}$ is defined as

$$\hat{\boldsymbol{\mathcal{X}}}^i = (\hat{\boldsymbol{U}}_1, \hat{\boldsymbol{U}}_2, ..., \hat{\boldsymbol{U}}_i, \boldsymbol{I}, ..., \boldsymbol{I}) \cdot \hat{\boldsymbol{\mathcal{W}}}^i \quad \in \mathbb{R}^{n_1 \times \cdots \times n_d}$$

with $\hat{\boldsymbol{\mathcal{X}}}^0 = \boldsymbol{\mathcal{X}}$ and $\hat{\boldsymbol{\mathcal{X}}}^0 = \hat{\boldsymbol{\mathcal{X}}}$.

The factor matrix $\hat{\boldsymbol{U}}_i$, $1 \leq i \leq d$, is the matrix of the r_i dominant left singular vectors of the mode-i vector space of $\hat{\boldsymbol{\mathcal{W}}}^{i-1}$. It is obtained from the rank r_i truncated singular value decomposition of the $(i-1)$th partially truncated core tensor, as follows:

$$\hat{\boldsymbol{\mathcal{W}}}^{i-1}_{(i)} = \boldsymbol{U}_i \boldsymbol{\Sigma}_i \boldsymbol{V}_i^{\mathsf{T}}$$

where $\boldsymbol{U}_i = [\hat{\boldsymbol{U}}_i \; \tilde{\boldsymbol{U}}_i]$.

The hat projector $\hat{\pi}_i$ is defined recursively contrary to T-HOSVD. Indeed, the definition of the $i+1$ projector is optimal for the partially approximated tensor $\hat{\boldsymbol{\mathcal{X}}}^i$. This leads to strongly improved performance if r_i is small. However, as stated earlier, the processing order is very important since it changes both the approximation and projectors. The ST-HOSVD algorithm is given next.

Algorithm 4: ST-HOSVD

```
input  : $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, truncation rank $\boldsymbol{r}$, processing order $\boldsymbol{p}$
output: $\hat{\mathcal{X}} = (\hat{\boldsymbol{X}}_1, ..., \hat{\boldsymbol{X}}_d) \cdot \hat{\mathcal{W}}$
$\hat{\mathcal{W}} = \mathcal{F}$ ;
for $i = p_1, ..., p_d$ do
    /* Compute SVD of $\hat{\mathcal{W}}_{(i)}$ then truncate to $r_i$ */
1   $(\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\intercal) = \mathrm{SVD}(\hat{\mathcal{W}}_{(i)})$ ;
2   $(\boldsymbol{U}_{tr}, \boldsymbol{\Sigma}_{tr}, \boldsymbol{V}_{tr}^\intercal) = \mathit{truncate}(\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\intercal, r_i)$;
3   $\hat{\boldsymbol{X}}_i = \boldsymbol{U}_{tr}$ ;
4   $\hat{\mathcal{W}}_{(i)} = \boldsymbol{\Sigma}_{tr} \boldsymbol{V}_{tr}^\intercal$ ;
return $\mathcal{X} = [\![\hat{\mathcal{W}}; \hat{\boldsymbol{X}}_1, ..., \hat{\boldsymbol{X}}_d]\!]$
```

It is possible to use a compact SVD which only yields the truncated SVD. This improves memory efficiency as well as computing speed, especially if the multilinear rank is small. One can see that the approximated tensor reduces after each truncated SVD finally reaching its final shape after the last dimension has been reduced. It is interesting to note that if the gray area is large, the next tensor size can be much smaller than the original tensor. Thus the SVD will be much faster than its T-HOSVD counterpart.

Remark 4.20. The processing order has been reported to influence greatly the computing time in addition to the obvious influence on the approximation itself. [**?**] proposed a heuristic that attempts to minimize the number of operations required to compute the dominant subspace. Then one should first process the dimension with lowest size and so on. This may even reduced the rank of the remaining terms, i.e. "*forcing more energy into fewer modes*". However choosing a processing order that minimizes the error is still an open question.

Error estimate. For a given multilinear rank, both ST and T-HOSVD approximations satisfy the same error bounds. However, usually, ST-HOSVD performs better in term of actual approximation error (see [**?**] section 7).

Theorem 4.21 (error bound ST-HOSVD, [**?** , Theorem 6.5])**.** *Let $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ a tensor and $\hat{\mathcal{X}}$ be the rank- $(r_1, ..., r_d)$ ST-HOSVD of $\mathcal{X}$. Let the SVD of $\mathcal{X}_{(i)}$ be given as in* (51)*. Then the bounds of the ST-HOSVD are*

$$\min_i \|\tilde{\boldsymbol{\Sigma}}_i\|_F^2 \leq \|\mathcal{X} - \hat{\mathcal{X}}\|_F^2 \leq \sum_{k=1}^{d} \|\tilde{\boldsymbol{\Sigma}}_i\|_F^2 \qquad (58)$$

where $\tilde{\mathbf{\Sigma}}$ is the discarded part of $\mathbf{\Sigma}$ obtained from the SVD.

In this section on computing the Tucker decomposition of a tensor, two methods were investigated. Both satisfy the same error bounds. On the one hand, the T-HOSVD is straightforward to implement and allows easy parallelized implementation for low number of CPU. Analysis is also relatively easy and the processing order has no influence on the approximation. On the other hand the ST-HOSVD is inherently sequential which means that processing order changes both the operation count and the approximation. This leads to analysis complexity and rises the question of an optimal processing order. However, the operation count and approximation error are overwhelmingly lower compared to T-HOSVD according to Vannieuwenhoven et al. This should be confirmed in the numerical experiments section.

As a conclusion, if the problem is large and the tensor has large differences in the directions length, the ST-HOSVD should be preferred to compute truncated Tucker decomposition. Indeed the advantages overcome by large margin the implementation increased complexity.

4.3 Tensor Train decomposition

Tensor Train format has been discussed in section 3.7, it is specially recommanded for larger dimensions as it scales linearly with d. Moreover, numerous theorems and algorithms have been proposed in the literature, most importantly one may rely on the following set:

- existence of the full-rank approximation (3.28, [? , Th. 2.1]),
- existence of the low-rank best approximation (4.22),
- `TT-SVD` algorithm for quasi optimal TT approximation (Algorithm 5),
- sampling algorithms (`TT-cross` [?], `TT-DMRG-cross` [?], `maxvol` [?],...).

In this section, we go through the decomposition properties and briefly outline the sampling algorithms.

TT-SVD As we have seen in the previous sections, SVD is a very efficient tool to decompose tensors, it turns out that TT decomposition is well suited, relying on SVD too with the help of the generalized matricization (see [? , Def. 2.2.4]). Using the reduced notation $\boldsymbol{X}^{(\mu*)} = \mathfrak{X}(i_1...i_\mu; i_{\mu+1}...i_d)$, from [?] we have the following property that enables the decomposition.

Theorem 4.22. *For any tensor $\mathcal{X} \in \mathcal{R}^{\mathcal{I}}$ there exists a TT approximation $\mathcal{T} \in \mathcal{R}^{\mathcal{I}}$ with compression rank $r_\mu = \operatorname{rank}(\boldsymbol{X}^{(\mu *)})$ such that*

$$||\mathcal{X} - \mathcal{T}||_F \leq \sqrt{\sum_{\mu=1}^{d-1} \varepsilon_\mu^2} \tag{59}$$

where ε_μ^2 is the distance (in Frobenius norm) from $\boldsymbol{X}^{(\mu *)}$ to its best rank-r_μ approximation:

$$\varepsilon_\mu^2 = \min_{\text{rankB} \leq r_\mu} ||\boldsymbol{X}^{(\mu *)} - \boldsymbol{B}||_F \tag{60}$$

Proof. The detailed proof is available in [**?**], here an adapted version is provided as it is constructive of the `TT-SVD` algorithm.

First, consider the case $d = 2$. The TT decomposition of $\mathcal{Z}$ reads

$$Z(i_1, i_2) = \sum_{\alpha_1=1}^{r_1} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2) \tag{61}$$

and coincides with the dyadic decomposition of matrix $\boldsymbol{Z}$. As shown in section 2.1, such an expression can be obtained optimally using truncated SVD at rank r_1 which is associated with truncation error ε_1.

By induction, the same is true for $\boldsymbol{X}^{(1)}$ the 1-matricization of $\mathcal{X}$ an order d tensor.

$$\boldsymbol{X}^{(1)} = [X(i_1; i_2 ... i_d)] = U \Sigma V^\intercal \tag{62}$$

Let $\boldsymbol{Y}_1 = \boldsymbol{U}_1 \tilde{\boldsymbol{\Sigma}} \tilde{V^\intercal}$ be the (best) r_1-rank approximation of $\boldsymbol{X}^{(1)}$ by truncated SVD i.e.

$$\boldsymbol{X}^{(1)} = \boldsymbol{Y}_1 + \boldsymbol{E}_1 \tag{63}$$

where $||E_1||_F = \varepsilon_1$. Of course, $\boldsymbol{Y}_1$ can be considered as a tensor $\mathcal{Y} = [Y(i_1, ..., i_d)]$. Then the approximation problem of $\mathcal{X}$ reduces to the one for $\mathcal{Y}$. $\mathcal{Y}$ being the best r_1-rank approximation any tensor $\mathcal{T}$ with $\boldsymbol{T}^{(1)} = \boldsymbol{U}_1 \boldsymbol{W}$ has a null projection on E_1. It implies the following equality

$$||(\mathcal{X} - \mathcal{Y}) + (\mathcal{Y} - \mathcal{T})||_F = ||\mathcal{X} - \mathcal{Y}||_F + ||\mathcal{Y} - \mathcal{T}||_F \tag{64}$$

So far the dimensionality of $\mathcal{Y}$ has not been reduced, to do so one can rewrite $\boldsymbol{Y}^{(1)}$ such that element-wise it reads

$$Y(i_1; i_2, ..., i_d) = \sum_{\alpha_1=1}^{r_1} U_1(i_1; \alpha_1) \tilde{X}(\alpha_1; i_2 ..., i_d)$$

where $\tilde{\boldsymbol{X}} = \tilde{\Sigma} V_1$. Then, the concatenation of indices α_1 and i_2 into one long index leads to the following order $(d-1)$ tensor

$$\tilde{\mathfrak{X}} = [\tilde{X}(\alpha_1 i_2, i_3, ..., i_d)]$$

By induction, $\tilde{\mathfrak{X}}$ admits a TT approximation $\tilde{\mathfrak{T}} = [\tilde{T}(\alpha_1 i_2, i_3, ..., i_d)]$ of the form

$$\tilde{T}(\alpha_1 i_2, i_3, ..., i_d) = \sum_{\alpha_2,...,\alpha_{d-1}} G_2(\alpha_1 i_2, \alpha_2) G_3(\alpha_2, i_3, \alpha_3) \cdots G_d(\alpha_{d-1}, i_d)$$

Such that

$$||\tilde{\mathfrak{X}} - \tilde{\mathfrak{T}}||_F \leq \sqrt{\sum_{k=2}^{d} \tilde{\varepsilon}_k^2}$$

with $\tilde{\varepsilon}_k^2 = \min_{\mathrm{rank}(C) \leq r_\mu} ||\tilde{\boldsymbol{X}}^{(\mu*)} - C||_F$.

Now let us set $G_1(i_1, \alpha_1) = U_1(i_1, \alpha_1)$, separate indices α_1 and i_2 from the long index $\alpha_1 i_2$ and define $\mathfrak{T}$ by the following tensor train:

$$T(i_1, i_2, ..., i_d) = \sum_{\alpha_1,...,\alpha_{d-1}} G_1(i_1, \alpha_1) G_2(\alpha_1, i_2, \alpha_2) \cdots G_d(\alpha_{d-1}, i_d)$$

*The rest of the demonstration consists in estimating $||\mathfrak{X} - \mathfrak{T}||_F$ through evaluations of the approximation error between $||\tilde{\mathfrak{X}} - \tilde{\mathfrak{T}}||_F$ which bounds the former. Details in [**?**].* □

Corollary 4.23. *If a tensor $\mathfrak{X}$ admits a canonical approximation rank R and accuracy ε, then there exists a tensor train approximation with compression ranks $r_k \leq R$ and accuracy $\sqrt{d-1}\varepsilon$.*

Corollary 4.24. *Given a tensor $\mathfrak{X}$, denote by $\varepsilon = \inf_{\mathfrak{Y}} ||\mathfrak{X} - \mathfrak{Y}||_F$ the infimum of distances between $\mathfrak{X}$ and tensor train $\mathfrak{Y}$ with prescribed upper bounds r_μ on the ranks of unfoldings matrices (compression ranks), i.e.* $\mathrm{rank}\boldsymbol{Y}^{(\mu)} \leq r_\mu$. *Then the optimal $\mathfrak{Y}$ exists (in a fact a minimum) and the TT approximation $\mathfrak{T}$ constructed in the proof of Theorem 4.22 is quasi-optimal in the sense that*

$$||\mathfrak{X} - \mathfrak{T}||_F \leq \sqrt{d-1}\varepsilon. \tag{65}$$

It is then natural to propose the `TT-SVD` [**?**] algorithm for the approximation of a full format tensor into TT format.

Algorithm 5: `TT-SVD`

```
input  : $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, truncation rank $\boldsymbol{r}$ or prescribed error $\varepsilon$
output: $\mathcal{X}(i_1, ..., i_d) = \sum_{\alpha_0,...,\alpha_d=1}^{r} G_1(\alpha_0, i_1, \alpha_1) \cdots G_1(\alpha_{d-1}, i_d, \alpha_d)$
1 Compute the truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}} ||\mathcal{F}||_F$ ;
2 Temporary tensor: $\mathcal{C} = \boldsymbol{\mathcal{A}}$, $r_0 = 1$ ;
  for $i = 1, ..., d$ do
      /* reshape($\mathcal{C}, r_{i-1}n_i, \frac{\texttt{numel}(C)}{r_{i-1}n_i}$)                    */
3     $\boldsymbol{C} = \boldsymbol{C}^{(i*)}$;
      /* truncated SVD at given rank $r_i$                             */
4     $\boldsymbol{U\Sigma V}^\intercal = \text{tSVD}(C, r_k, \delta)$ ;
5     $\mathcal{G}_i = \text{reshape}(\boldsymbol{U}, [r_{i-1}, n_i, r_i])$ ;
6     $\boldsymbol{C} = \boldsymbol{\Sigma V}^\intercal$ ;
7 $\mathcal{G}_d = \boldsymbol{C}$;
  return $\mathcal{X} = [\![\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_d]\!]$
```

Remark 4.25. In addition to the linear algebra algorithms mentioned in section 3.7, many algorithms have been developed to convert from canonical [**?**], Tucker or HT to TT, for instance, one can refer to [**?** , Chap. 12 & 13]. Also, one may need to recompress an existing TT tensor (for example after summing two TT tensors), to do so Oseledets proposes the `TT-rounding` algorithm [**?**] based on a combination of QR decompositions and SVD.

Actually this algorithm relies on the same methodology as the `ST-HOSVD` but stores the results in the cores thus leading to TT format. As stated earlier, this leads to a linear storage cost in d which is much more efficient than Tucker format. In addition to that, the weights of the entries are stored in the last mode/core $\boldsymbol{G}_d = \mathcal{G}_d$ and modes relations are stored within the cores themselves without requiring a single core tensor.

Sampling algorithms for high dimensional TT This kind of algorithm is very well suited to analyze data from existing simulations in the context of fluid dynamics. However, if the dimensions of the studied problem grows above 5 it becomes intractable to either store the data or solve the SVD problem. In order to circumvent this difficulty, one might rely on a family of methods that will be referred as *sampling* algorithms. They come under many names including `maxvol` for matrices skeleton decomposition or `TT-cross, TT-DMRG-cross,...`. Obviously this can be done in many formats, included the also well suited HT format (see `BlackBox algorithm` [**?**

]). A short overview is proposed, many more can be found in the literature, including in [? ? ?].

4.4 Hierarchical Tucker decomposition

Hierarchical Tucker decomposition is a growing topic in the tensor decomposition community [? ? ? ?]. It has been shown to be very efficient to tackle large datasets [? ? ?] since it can be viewed as a "specialization of Tucker format" for large number of dimensions. As seen in the tensor format section (3.2), efficient strategies have been developed to convert other formats into $\mathcal{HT}$ (see [?]) as well as truncation (leaf to root and root to leaf) and orthonormalization strategies proposed by Grasedyck [?]. These algorithms have already been implemented in publicly available libraries including D. Kressner and C. Tobler `htucker` MATLAB library [?]. It has also been shown that $\mathcal{HT}$ decomposition is very well suited for sampling algorithms, one such example is the Black Box algorithm proposed by Ballany, Grasedyck and Kluge in [?].

4.5 The Recursive-POD (R-POD)

The Recursive POD [?] is an extension of the usual bivariate POD, it fulfills quasi-optimality in higher dimension. The essence of this method is to perform successive (recursively) POD on the field that is to be tensorized. A field function $f : \mathcal{D} \subset \mathbb{R}^d \longrightarrow \mathbb{R}^q$ i.e. a function of d variables is first processed as a $(1, d-1)$ field that can be separated thanks to POD. Once the first POD has been performed, one obtains the POD modes $\boldsymbol{X}_1^r : \Omega_1 \longrightarrow \mathbb{R}^q$ basis functions of Ω_1 and $\boldsymbol{\phi}_1^r : \mathcal{D}/\Omega_1 \longrightarrow \mathbb{R}^q$, a set of functions of $d-1$ variables. The the same POD process is performed again on each POD mode recursively until the POD modes are univariate functions.

Remark 4.26. It should be noted that the RPOD is the extension to multiple variables that overcomes the bivariate nature of POD. Consequently every conclusion concerning POD remains true except optimality properties. For short, it means that any algorithm available to compute a POD i.e. POD, PGD and to some extent direct SVD may be used to compute the recursive POD. All algorithmic properties are preserved and method choices should align with 2D experiment conclusion.

Introductory example : R-POD on a 3D field Let $f : \mathcal{D} = \Omega_1 \times \Omega_2 \times \Omega_3 \subset \mathbb{R}^3 \longrightarrow \mathbb{R}$ a Lebesgue square integrable function and $w = (y, z) \in \mathbb{R}^2$. Since $L^2(\mathcal{D})$ and $L^2(\Omega_1, L^2(\mathcal{D}/\Omega_1))$ are isometric, the POD of $f(x, w)$ is well defined and reads

$$f(x,y,z) = f(x,w) \approx f^M_{POD}(x,w) = \sum_{m=1}^{M} X_m(x)\phi_m(w) \tag{66}$$

As for the 2D POD it is handy to normalize all modes and store their relative weight into $(\boldsymbol{\sigma}) \in \mathbb{R}^M$. Then the POD of f reads

$$f(x,y,z) = f(x,w) \approx f^M_{POD}(x,w) = \sum_{m=1}^{M} \sigma_m X_m(x)\phi_m(w) \tag{67}$$

with $X_m = X_m/\|X_m\|$, $\phi_m = \phi_m/\|\phi_m\|$ and $\sigma_m = \langle f, X_m\phi_m\rangle$.

It is now necessary to separate each 2D field ϕ_m obtained during the first step i.e.

$$\forall 1 \leq m \leq M, \quad \phi_m(w) = \phi_m(y,z) \approx \phi_{m,K(m)}(y,z) = \sum_{k=1}^{K(m)} \tilde{\sigma}^m_k \; Y^m_k(y) Z^m_k(z) \tag{68}$$

Then, these two results are combined into one tensorisation of field f,

$$f(x,y,z) \approx f_M(x,y,z) = \sum_{m=1}^{M} \sum_{k=1}^{K(m)} \sigma_m \tilde{\sigma}^m_k X_m(x) Y^m_k(y) Z^m_k(z) \tag{69}$$

Remark 4.27 ($K(m)$)**.** As each POD on level $\phi_m(y,z)$ is performed independently, if the number of dominant POD modes is dependent of an error estimator, then $K(m)$ may change with m. Then a *R-POD rank* is defined as the number of modes at each recursion level. An illustration of the spread of $K(m)$ is provided by figure 16. It can be seen that in this matrix representation that some $\tilde{\sigma}^m_k$ are missing. They correspond to the unneeded/uncomputed modes. For practical reason, this representation of $\tilde{\Sigma} = (\tilde{\sigma}^m_k)_{km}$ may be useful to compare RPOD with ST-HOSVD and also facilitates the implementation setting discarded modes as constant functions with a nil weight.

Obviously, this sum of sums can be reordered and written as one single sum. In this work, the following bijective numbering function is used

$$\begin{aligned} h: \quad & \mathbb{N}^2 && \longmapsto \mathbb{N} \\ & (m,k) && \longmapsto l = k + \sum_{i=1}^{m-1} K(i) \end{aligned}$$

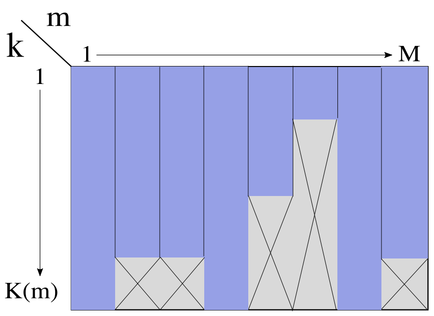


Figure 16: Visual RPOD rank for 3 parameter function. Blue columns correspond to the coordinates (m, k) where $\tilde{\sigma}_k^m$ is defined while gray crossed areas correspond to coordinates where $\tilde{\sigma}_k^m$ is not defined (not computed).

Then a new weight list is defined as $\sigma_{l=h(m,k)} = \sigma_m \tilde{\sigma}_k^m$. Finally the R-POD approximation of f reads

$$f_L(x, y, z) = \sum_{l=1}^{L} \sigma_l \; X_l(x) Y_l(y) Z_l(z) \tag{70}$$

Generalization of this example is straightforward, however, notations quickly become cumbersome for higher dimension.

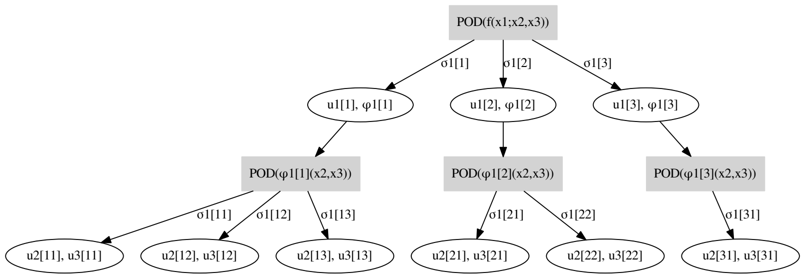


Figure 17: Example of *a Recursive POD graph* of $f(x_1, x_2, x_3)$

Another approach is to represent the *recursion graph* or *decomposition graph* as shown in Fig. 17. In this case, there is no need to introduce a renumbering, all the information is contained in the graph. Notations and programming remain simple as each decomposition (as well as reconstruction) is performed independently, each node of the tree only "knows" its children. This approach is very natural from a mathematical point of view however it is uncommon in computational mechanics.

The extension of the R-POD to any n-dimension is presented and analysed in [**?**] We limit here its presentation by the following algorithm

Algorithm 6: RPOD

```
   input  : $f \in L^2(\mathcal{D})$, computing domain $\mathcal{D}$, target error $\varepsilon$
   output: rpod_tree=$[[\mathcal{R}, \mathcal{S}, \mathcal{X}]]$
1  $\mathcal{R} = [\,]$ ;            /* List containing the exact RPOD rank */
   $\mathcal{S} = [\,]$ ;     /* List containing the local singular values */
   $\mathcal{X} = [\,]$ ;     /* List containing the local eigen functions */
2  $\phi(x, \boldsymbol{w}) = f(x_1, (x_2, ..., x_d))$ ;
3  $[R, \boldsymbol{\sigma}_R, \boldsymbol{U}_R(x), \boldsymbol{V}_R(\boldsymbol{w})] = \text{trunc\_POD}(\phi, \varepsilon)$ ;
4  $\mathcal{R}$.append($R$) ;
   $\mathcal{S}$.append($\boldsymbol{\sigma}_R$) ;
   $\mathcal{X}$.append($\boldsymbol{U}_R$), ;
   if $dim(w) > 2$ then
       for $m \leq R$ do
5          $\phi(x, \boldsymbol{s}) = V_r(\boldsymbol{w})$ ;
6          $(\mathcal{R}_{loc}, \mathcal{S}_{loc}, \mathcal{X}_{loc})$.append(RPOD($\phi, \mathcal{D}/\Omega_1, \varepsilon$)) ;
7      $(\mathcal{R}, \mathcal{S}, \mathcal{X})$.append($\mathcal{R}_{loc}, \mathcal{S}_{loc}, \mathcal{X}_{loc}$) ;
   else
8      $\mathcal{X}$.append($\boldsymbol{V}_R$) ; /* Last dimension, then keep $\boldsymbol{V}_R$ as RPOD
        modes */
   return $f_{\mathcal{R}} = [[\mathcal{R}, \mathcal{S}, \mathcal{X}]]$
```

Operation count In order to ease comparison with its most similar method, we suppose that each local POD is solved through the same truncate SVD algorithm that is used in the ST-HOSVD though it might not be the best choice for accuracy of computing efficiency. Then the SVD of a $n \times m$ matrix operation count is $\mathcal{O}(m^2 n)$. The sum of the last column of table yields the following estimate if the samples number is identical for all

Level	Operations	Count	Hypercube Cost
1	$1\times$ POD$[n_1 \times (n_2...n_d)]$	$\mathcal{O}(n_1^2(n_2...n_d))$	$\mathcal{O}(n^d+1)$
2	$M_1\times$ POD$[(n_2 \times (n_3...n_d)]$	$M_1\mathcal{O}(n_2^2(n_3...n_d))$	$\mathcal{O}(Mn^d)$
3	$\sum_{m_1\leq M_1} M_2(m_1)\times$ POD$[(n_3 \times (n_4...n_d)]$	$M_1M_2\mathcal{O}(n_3^2(n_4...n_d))$	$\mathcal{O}(M^2n^{d-1})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
d-1	$\sum\cdots\sum M_{d-2}(m_1,...,m_d)\times$ POD$[(n_{d-1} \times (n_d)]$	$M_1\cdots M_d - 2\mathcal{O}(n_{d-1}^2(n_d))$	$\mathcal{O}(M^{d-2}n^3)$

Table 2: Operation count at each step of the RPOD algorithm.

variables as sigma map tensor is a full hypercube.

$$\mathcal{O}\left(\sum_{i=1}^{d-1} R^{i-1}n^{d-i+2}\right) \tag{71}$$

One can see that this is exactly the same term as the first term in ST-HOSVD operation count evaluation. The second one is not necessary since the RPOD algorithm does not requires to compute an intermediate function/tensor. Additionally, this results does not account sum length number of modes variation within each dimension.

4.6 Proper Generalized Decomposition

In this section we present the PGD for d parameter functions both for *a priori* and *a posteriori* model reduction. The first subsection presents the theoretical justification of this class of methods. The second section focuses on the algorithm proposed by Chinesta which is the only PGD implemented so far. Finally, in subsection 4.6, a brief overview of the link with the CP decomposition is proposed and some conclusions about this kind of methods are drawn.

Theoretical background of the PGD The general setting of weak formulation in an Hilbert space is used in this presentation of the PGD.

On V a Hilbert space, we define the following abstract formulation

$$u \in V,\ \mathcal{A}(u,v) = \mathcal{L}(v) \quad \forall v \in V \tag{72}$$

Where $\mathcal{A}$ is a bilinear form on V and $\mathcal{L}$ is a linear form on V. $V = V_1 \otimes \cdots \otimes V_d$ is a tensor product of Hilbert spaces provided with a scalar product and its associated norm.

$\mathcal{S}_1$ the set of rank-one tensors is introduced

$$\mathcal{S}_1 = \{z = w^1 \otimes \cdots \otimes w^d \ ; \ w^k \in V_k, \ k \in \{1, ..., d\}\} \tag{73}$$

as well as $\mathcal{S}_m$ the set of rank-m tensors

$$\mathcal{S}_m = \{v = \sum_{i=1}^{m} z_i \ ; \ z_i \in \mathcal{S}_1, \ i \in \{1, ..., m\}\} \tag{74}$$

The naive problem of finding an optimal representation $u_m \in \mathcal{S}_m$ of a given element $u \in V$ is not trivial and has been extensively studied. As stated in section 4.1, the problem is even ill posed for $d \geq 3$. Then one must add suitable constraints like orthogonality or boundedness to define a suitable optimization problem on $\mathcal{S}_m$. In the context of PGD, modes along one dimension are orthogonal to one another, for unicity purpose these modes are normalized with respect to the chosen norm except for the last dimension (here for $k = d$) which accounts for the norm of z_i.

For *a posteriori* processing, we have

$$\mathcal{A}(u, v) = \int_\Omega uv \ d\mu \tag{75}$$

$$\mathcal{L}(v) = \int_\Omega fv \ d\mu \tag{76}$$

Introducing these notations might seem cumbersome, however it eases a lot the use of more complex functionals as long as they verify the same properties. Now a short version of the rigorous analysis of the progressive PGD proposed by Falcó in [**? ?**]. In the following all the assumed properties are easily verified for $\mathcal{A}$ the scalar product operator and $\mathcal{L}$ a scalar product against f as defined in equations (75) and (76).

It is assumed that $\mathcal{A}$ is bounded and coercive. Then equation 72 is project on V_N an N-dimensional subspace of V which the classical way of Galerkin methods.

$$u \in V_N, \ \mathcal{A}(u, v) = \mathcal{L}(v) \quad \forall v \in V_N \tag{77}$$

Thanks to Riesz representation theorem, $A : V \longrightarrow V$ the operator associated to $\mathcal{A}$ is introduced

$$\mathcal{A}(u, v) = \langle Au, v \rangle \tag{78}$$

and $f \in V$ associated with $\mathcal{L}$

$$\mathcal{L}(v) = \langle f, v \rangle \tag{79}$$

Then the problem (77) can be rewritten in an operator form

$$Au = f \tag{80}$$

It is further assumed that $\forall v \in V, \exists c > 0$ such that $\|Av\| \geq C\|v\|$. From the properties of A and its adjoint $A^\star$, $AA^\star$ is a self adjoint continuous and V-elliptic operator. Consequently it defines an inner product on V denoted $\langle \cdot, \cdot \rangle_{AA^\star} = \langle A\cdot, A\cdot \rangle$ whose associated norm is equivalent to the $\|\cdot\|$ norm. Then formulation 77 is equivalent to the following minimal residual formulation

$$u_n = \arg\min_{v \in V_n} \|f - Av\| = \arg\min_{v \in V_n} \|A^{-1}f - v\| \tag{81}$$

If one chooses $V_N = \mathcal{S}_N$ then for $A = I$ the PGD solves the same problem as the truncated CP decomposition provided by an ALS algorithm. Now, a convergent PGD algorithm is provided. Moreover it coincides with the PGD definition given by Chinesta et al. [**?**].

Remark 4.28. The Galerkin problem can be solved on several basis which means that PGD is available on Hilbert tensor spaces in format that mimic any tensor reduction technique. For example Falcó demonstrates the convergence of PGD on a basis similar to the HOSVD in [**?**]. Thus one can conclude that PGD for *a posteriori* processing is the continuous version of the well established tensor low rank approximation. Even though a wide variety of integration technique and PGD algorithms are available, it seems that the vast literature investigating tensor reduction proves to be much more efficient at post-processing.
Additionally, the inverse observation can be made for solving PDEs on reduced basis using reduced tensor representation. These algorithms might benefit for the preexisting knowledge in *a priori* PGD.

On the convergence of the progressive PGD. In order to show the converge of this algorithm, a generalization of the Eckart-Young theorem has been provided by Falcó and Nouy in [**?**]. Since the general problem of a rank-k separated representation is ill posed [**?**], they proposed a progressive algorithm that converges. It is based on successive rank-1 approximations which are known to be optimal thus the link with singular values.

Lemma 4.29. *Given that $\mathcal{S}_1$ is weakly closed for $\|\cdot\|$ then for each $z \in V, \exists v^* \in \mathcal{S}_1$ such that*

$$\|z - v^*\|^2 = \min_{v \in \mathcal{S}_1} \|z - v\|^2$$

Finding v^* in the previous equation is a map defined by

$$\begin{aligned} \Pi : \quad & z \in V \longrightarrow \Pi(z) \in \mathcal{S}_1 \\ & z \longmapsto \arg\min\nolimits_{v \in \mathcal{S}_1} \|z - v\|^2 \end{aligned}$$

Definition 4.30 (Progressive separated representation of an element in V)**.** For a Given $z \in V$, the sequence $\{z_n\}_{n \geq 0}$ with $z_n \in \mathcal{S}_n$ is defined as follow: $z_0 = 0$ and for $n \geq 1$,

$$z_n = \sum_{i=1}^{n} z^{(i)} = \sum_{i=1}^{n} \sigma_i w^{(i)}, \quad z^{(i)} \in \Pi(z - z_{i-1}) \tag{82}$$

z_n is the rank-n progressive separated representation of z with respect to the norm $\| \cdot \|$.

Theorem 4.31 (Generalized Eckart-Young theorem according to Falcó and Nouy)**.** *For $z \in V$, the sequence $\{z_n\}_{n \geq 0}$ from definition 4.30 verifies*

$$z = \lim_{n \to \infty} z_n = \sum_{i=1}^{\infty} \sigma_i w^{(i)}$$

This proves the convergence of the PGD algorithm which is a succession of optimal progressive separated representation as defined in (4.30) with the projector associated to A.

Remark 4.32. As stated by Falcó and Nouy, this is the simplest definition of PGD, other definitions where provided in the literature which may display better convergence properties. One of them is the direct equivalent of the ALS algorithm [**?**].

A Galerkin PGD algorithm for d parameter functions according to Chinesta In order to determine each element of the sequence an enrichment process is devised. Let $\Omega = \Omega_1 \times \cdots \times \Omega_d$ where each $\Omega_i \in \mathbb{R}$ and $f \in L^2(\Omega)$[11]. Then, the goal is to compute univariate basis functions $(X_i^k)_{k=1}^r$, $\forall\, 1 \leq i \leq d$ using a fixed point algorithm in alternating directions. The weak formulation of our problem reads

$$\forall u^\star \in H^1(\Omega), \quad \int_\Omega u^\star (u - f) = 0 \tag{83}$$

[11]Here we assume without loss of generality that Ω_i is a subset of $\mathbb{R}$ but it could be any domain on which an integral can be defined. e.g. 2D or 3D domains.

It is assumed that $u^{r-1} = \sum_{k=1}^{r-1} \prod_{i=1}^{D} X_i^k(x_i)$ is known thus u^r is sought under the form

$$u^r = u^{r-1} + \prod_{i=1}^{d} X_i^r(x_i) \tag{84}$$

The process of adding terms to the sum, i.e. computing the sequence (u^r) is called the *enrichment process*. This process ends when a stopping criterion is fulfilled. Since in the general case, one does not knows the exact solution, it is chosen to stop the process when the weight of the last term compared to the rest of the series becomes negligible. This reads

$$\mathcal{E}(r) = \frac{||\prod_{i=1}^{d} X_i^r||_{L^2(\Omega)}}{||\prod_{i=1}^{d} X_i^1||_{L^2(\Omega)}} = \frac{||X_d^r||_{L^2(\Omega)}}{||X_d^1||_{L^2(\Omega)}} \leq \varepsilon_{enrichment} \tag{85}$$

Indeed the terms are of decreasing norm, then there is no need to compare the whole series, the first term is sufficient. In addition to that, we define X_i such as $\forall i < D, \quad ||X_i||_{L^2(\Omega)} = 1$ all the information about the norm is enclosed in X_D.

Fixed point algorithm. This is an iterative algorithms that, in practice, usually converges in a few iterations. It is an alternated direction algorithm, i.e. each direction is computed one at a time.

Remark 4.33. From now on, r in X_i^r is omitted to simplify the writing at enrichment step r.

It is assumed that the fixed point series $\{\widehat{X}_i^k\}_k, \forall i < d$ is known after step k. Thus $u = u^{r-1} + \prod_{i=1}^{D} \widehat{X}_i^k$. Moreover, it is assumed that direction s is to be updated which means $\widehat{X}_i^{k+1}$ is already known $\forall i < s$.

The test function $u^\star$ is set to

$$u^\star = \prod_{i=1}^{s-1} \widehat{X}_i^{k+1}(x_i) X^\star(x_s) \prod_{i=s+1}^{d} \widehat{X}_i^k(x_i) \tag{86}$$

Given all previous equation, the following weak formulation stands

$$\int_\Omega \left[\prod_{i=1}^{s-1} \widehat{X}_i^{k+1} X^\star \prod_{i=s+1}^{d} \widehat{X}_i^k \left(u^{r-1} + \prod_{i=1}^{s} \widehat{X}_i^{k+1} \prod_{i=s+1}^{d} \widehat{X}_i^k - f \right) \right] = 0 \tag{87}$$

This equation writes : find $\widehat{X}_s^{k+1}$ such that for each $X^\star$

$$\alpha^s \int_{\Omega_s} X^\star \widehat{X}_s^{k+1} dx_s = - \int_{\Omega_s} X^\star \sum_{j=1}^{p-1} \left(\beta^s(j) \widehat{X}_s^j \right) dx_s + \int_{\Omega_s} X^\star \gamma^s(x_s) dx_s$$

where

$$\alpha^s \qquad = \prod_{i=1}^{s-1} \int_{\Omega_i} (\widehat{X}_i^{k+1})^2 \prod_{i=s+1}^{d} \int_{\Omega_i} (\widehat{X}_i^{k})^2 \tag{88}$$

$$\beta^s(j) \quad = \prod_{i=1}^{s-1} \int_{\Omega_i} \widehat{X}_i^{k+1} X_i^j \prod_{i=s+1}^{d} \int_{\Omega_i} \widehat{X}_i^{k} X_i^j \quad \forall j < p \tag{89}$$

$$\gamma^s(x_s) \qquad = \int_{\Omega/\Omega_s} \prod_{i=1}^{s-1} \widehat{X}_i^{k+1} \prod_{i=s+1}^{D} \widehat{X}_i^{k} f \tag{90}$$

Finally the strong formulation stands

$$\widehat{X}_s^{k+1}(x_s) = \frac{-\sum_{j=1}^{p-1} \left(\beta^s(j) X_s^j(x_s)\right) + \gamma^s(x_s)}{\alpha^s} \quad, \forall x_s \in \Omega_s \tag{91}$$

All the $\widehat{X}_i$ are normalized i.e. $||\widehat{X}_i^{k+1}||_{L^2(\Omega_i)} = 1$ so that all the information relative to the norm is transfered to the last element X_d. This algorithm is performed for $s = 1, d$ and each time a family $(\widehat{X}_i^{k+1})_{1 \le i \le d}$ is complete the convergence stopping criterion is tested. It reads

$$\mathcal{E}_{fixed\,point}(k) = \frac{||X_d^{k+1} - X_d^k||_{L^2(\Omega_i)}}{||X_d^k||_{L^2(\Omega_i)}} < \varepsilon_{fixed\,point} \tag{92}$$

PGD and CPD It clearly appears that the PGD falls in the domain of canonical representation format $\mathcal{C}_r$ for functional spaces. The same statement can be made for CP decomposition where the underlying space is $\mathbb{R}^d$. Consequently, for *a posteriori* processing of tensor data, these two decomposition techniques are freely interchangeable. Then any favorable property of one is applicable to the other. Unfortunately, this remains true for the downsides like the ill-posedness of a general best rank-r approximation. This approach has been shown to be poorly efficient compared to Tucker format methods but may represent a first step in an attempt to compute low rank approximations of tensors.

However, the main strength of the recursive techniques is that they are mostly cheap[12], easy to program and produces *a priori* reduced order bases. Indeed in many situations where high precision is not a goal or simply unrealistic but many parameters are used, PGD (or CP alternatives) in some of its formulation is a very interesting process that enables calculations that are simply out of reach for direct simulations.

Remark 4.34. There is a vast literature concerning PGD algorithm applied to (mainly elliptic) problems [**?** **?**]. It turns out that different kind of PGD

[12]As long a one only requires a small number of modes as compared to the full representation, PGD can be efficient since it computes only the required information.

algorithm [?] work best on different kind of problems (Galerkin PGD, minimum residual PGD, Krylov PGD, Greedy Completely Orthogonal PGD ,etc.) . Then, there is no general PGD algorihtm however the one that was presented in the previous section seems to be robust though may require many iteration to converge.

5 Numerics

In this section, we propose a comprehensive numerical study of the decomposition methods that have been presented. Indeed, very limited comparison between these methods is available in the literature, the goal, here, is to provide a general view of decomposition methods at work and draw conclusions on their use in the context of scientific computing and in particular as a first stage to develop ROM.

These numerical experiments have been performed using a python decomposition library `pydecomp`[13] which is presented in greater detail in [?]. As we have seen, POD and SVD are essentially equivalent methods which is why the implementation allows the user to use either one for higher order decomposition. The same is true for canonical decomposition and PGD as shown in the previous section. Consequently these experiments explore a wide variety of setups for synthetic data. In [?], the reader can find many more numerical experiments and in particular a large section on actual data.

Synthetic data comparison

Using synthetic data is very useful to test the methods and a variety of parameters that might influence the convergence and compression rates. Our data is generated on uniform grids[14] of $n_1 \times \cdots \times n_d$ that discretizes $\Omega = [0,1]^d$. The following real test functions are used

$$\begin{aligned} f_1(\boldsymbol{x}) &= \frac{1}{1 + \sum_i x_i} \\ f_2(\boldsymbol{x}) &= \sin(||\boldsymbol{x}||_2) \\ f_3(\boldsymbol{x}) &= \sqrt{1 - \prod_i x_i} \end{aligned}$$

[13]`https://git.notus-cfd.org/llestandi/python_decomposition_library`

[14] Using a non uniform grid would have little influence on the accuracy given that one uses accurate integration schemes. However it may help to increase the computing speed by using a sparser grid.

A special function was used to reproduce singularity for $d = 5$,

$$\begin{aligned} f_s(\boldsymbol{x}) \quad = \quad & x_1^2\{\sin[5x_2\pi + 3\log(x_1^3 + x_2^2 + x_4^3 + x_3 + \pi^2)] - 1\}^2 \\ & +(x_1 + x_3 - 1)(2x_2 - x_3)(4x_5 - x_4)\cos[30(x_1 + x_3 + x_4 + x_5)] \\ & \log(6 + x_1^2x_2^2 + x_3^3) - 4x_1^2x_2x_5^3(-x_3 + 1)^{3/2} \end{aligned}$$

A typical case $d = 3$. In order to evaluate the separability of these three test functions, we chose a relatively coarse grid of $32 \times 32 \times 32$. The results are presented for all three functions in Fig. 18. These graphs present the relative decomposition error[15] defined by

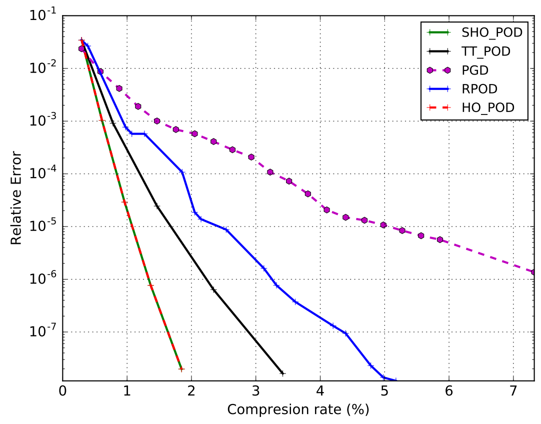
$$\epsilon = \frac{||\mathfrak{T}_{\text{exact}} - \mathfrak{T}_{\text{decomp}}||}{||\mathfrak{T}_{\text{exact}}||}, \tag{93}$$

as a function of the compression rate (in %) which is the storage cost of a decomposition at a given rank divided by the storage cost of the full format tensor i.e.

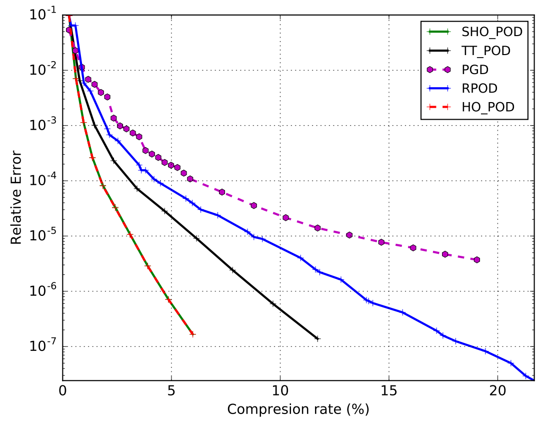
$$CR = \frac{\texttt{Mem\_cost}(\mathfrak{T}_{\text{decomp}})}{\texttt{Mem\_cost}(\mathfrak{T}_{\text{exact}})}(\times 100 \text{ for } \%). \tag{94}$$

First, all 5 methods are tested with L^2-norm and scalar product i.e. POD is applied as a bivariate decomposition method. A distinct pattern can be observed in these 3 figures. The least efficient compression method is PGD, which was expected in terms of CPU time due to the iterative algorithms at the center of the method. However as the format is very efficient by definition one could hope that the sub-optimality of the algorithm (see sections 4.6 and 4.1 for PGD and ALS NP hard problem) would not impact too much the decomposition. Actually, in spite of acceptable convergence of the fixed point algorithm, the compression rate of PGD grows much quicker than any other methods. Still it should be noted that in all three cases, it provides the best rank-1 decomposition as one should expect for a method based on successive rank-1 decompositions.Then it is clear for all 3 functions that TT-POD and ST-HOPOD are the most efficient methods, both showing exponential decay, although with a slope change for f_2 (Fig. 18b) as it is the least separable of all three functions. One should note that the ST-HOPOD and T-HOPOD are superposed, this behavior was already observed in [**?**] (for SVD based decompositions) in the case of easily separable functions. As we will see in the next paragraph the main difference lies in the computing time of the methods. Additionally, one can see that

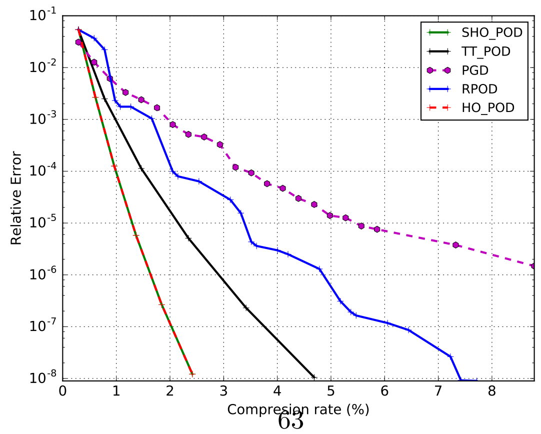
[15]The norm is not specified here as it can be either a Frobenius norm of tensors or the$L^2(\Omega)$ norm.



(a) f_1



(b) f_2



(c) f_3

Figure 18: Decomposition of 3 test functions with $d = 3$ on a 32^3 grid with 5 discretization methods, using L^2 integration and norm. Remark: in these graphs, SHO$_PODandHO_PODstandforST - HOPODandT - HOPODrespectively$.

TT-POD is less efficient for these small 3D problem as the core does not require much memory in Tucker format. Finally the RPOD is close to TT for the lowest truncation rank i.e. as long as they are virtually equivalent[16] but the nature of this recursive decomposition creates decomposition error jump when one enters a new branch with important weight. This phenomenon of steps is most prominent in Fig. 18c. As said in 2.3 it is useless to show different grid resolution as these functions are smooth and decomposition behavior is thus uncorrelated with grid density, only the compression rate would vary since it depends directly on the number of discrete points.

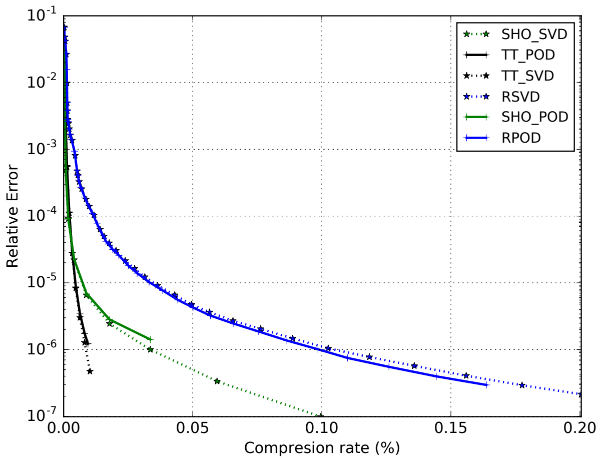


Figure 19: Decomposition of f_s on a 40^5 grid with L^2 and l^2 scalar products, decomposition error in their relative norm

In order to assess the influence of the scalar product for higher dimensions decomposition, f_s the least separable of the synthetic data functions is used, with $d = 5$ and 40 equispaced grid points in each dimension. Indeed,

[16] Actually, for TT rank of 1 and and RPOD rank of 1 i.e. 1 mode only for each dimension, then both algorithms are strictly equivalent, only the data structure is different. Then when the rank grows, the association of modes by explicit summation in Recursive format is less efficient than the implicit summation to the TT format. Finally the truncation strategy used in the software requires that any branch with a weight above truncation limit has at least one leaf kept in the evaluation and all other leaves below the truncation limit are ignored. This results in cumulative loss in precision which means that the rank/epsilon truncation in recursive format is less sharp than in TT format.

for easier decompositions on Cartesian grids, no difference can be seen in the relative error graphs. Fig. 19 shows recursive, TT and sequentially truncated tucker decompositions for both L^2 (POD) and l^2 (SVD) scalar products. One can see that for each method, the error and compression rate are almost the same for both scalar products. The trend being overwhelmingly driven by the method itself. Results might differ for different grid types and functions with sharp variations in which an actual integration would capture better these phenomena. Also one should notice that in this case where $d = 5$, TT decomposition is now more efficient than ST-HOSVD when high accuracy is required ($\epsilon < 10^{-4}$) and does not show any sign of linear decay contrary the other methods. In particular RPOD clearly shows a linear decay from 10^{-3} onward in spite of being competitive for accuracy up to 1%.

In conclusion as long as one uses a Cartesian grid, using SVD or POD does not influence the compression behavior and other factors should be used to decide which one to use depending on the use of this decomposition. For ROM building, one should use a EDP adapted scalar product i.e. POD to obtain orthonormal modes. It should also be preserved for physical analysis of a problem. Another criterion is CPU time, especially if one only intends to reduce storage cost of large datasets.

Relative CPU time On the same problem, let us focus on the CPU times for each methods as well as the reconstruction time needed to obtain a full tensor from the reduced representation. Results are shown in table 3. PGD has been voluntarily excluded from this table as it requires several hours, T-HO**D is not shown either as it requires roughly 4 times the ST-HO**D as expected from the number of dimensions. One can see that for all these methods, SVD based decomposition is faster. This is due to the implementation of POD that requires an additional diagonal (possibly multiple diagonals) matrix multiplication each time a scalar product is needed as compared to the SVD by EVD. In the end, for TT and ST decompositions for which the cost of the bivariate decomposition is controlling CPU time, this results in doubling the time for POD. For recursive decomposition, there are numerous overheads that makes the difference much smaller. Regarding the evaluation time, it was not particularly optimized as it is not a central task to reconstruct full tensors, more likely for higher order tensors, one might need only to reconstruct a slice of the tensor. The third column of table 3 is the evaluation of the last data point in Fig. 19. First one can see that both recursive methods takes roughly the same time which is 30 times more than the other two methods. This observation definitively disqualifies recursive methods for data reduction purpose. The the Tucker

method	computing (s)	evaluation (s)
RPOD	9.535	31.80
RSVD	7.964	32.20
ST-HOSVD	1.096	1.23
ST-HOPOD	2.378	0.98
TT-SVD	1.205	1.19
TT-POD	2.206	1.13

Table 3: CPU times on f_v for $n = 40$, $d = 5$ with a tolerance of $\epsilon = 10^{-12}$

and TT are in the same range of reconstruction time, the slight differences present here translate the slight variation in their number of modes due to different truncation criteria implementation.

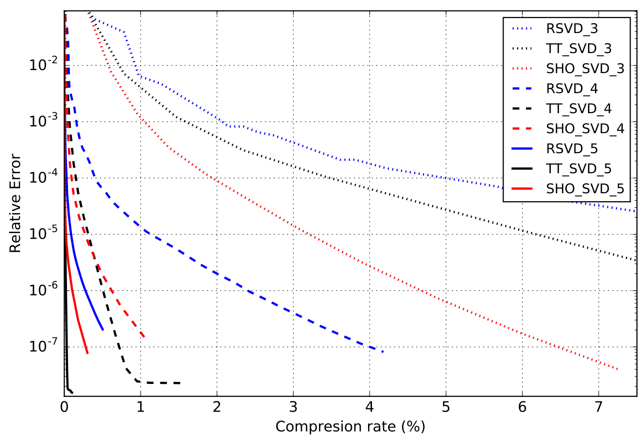


Figure 20: f_2 decomposition with $d = 3$ to 5 on a 32^d grid with three decomposition methods, using L^2 integration and norm.

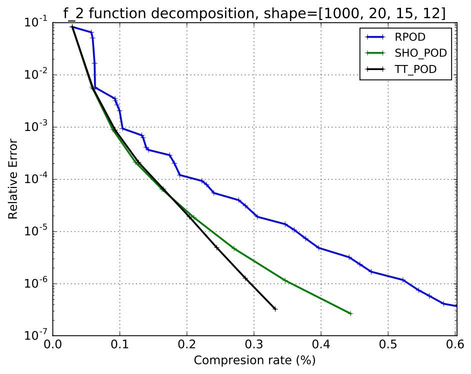
Number of dimensions and shift in the adequate methods. Now, we investigate the influence of the number of dimensions in order to decide upon which method to use. To do so, in Fig. 20 we compare the same 3 methods with SVD solvers and show on the same graphs relative error as a function of compression rate for $d = 3$ to 5 in the decomposition is function f_2. One can see, once again that RSVD is the worst in all cases but its distance to the other methods tends to diminish as d grows. Indeed, the recursive structures prevents the storage cost to explode with the number of dimensions d. This is also the main difference between TT and Tucker format. While the latter is more efficient for $d = 3$ and remains competitive[17] up to $d = 5$ thanks to efficient decomposition, it is outclassed for storage purpose by TT. This is particularly visible for $d = 5$ (sold lines). Thus, one can conclude that TT decomposition should be preferred as soon as $d \geq 5$ if the orhtonormality of the modes is not a criterion. For lower order problems, it is probably preferable to choose ST-HOSVD method as it ensures orhtonormality of the basis while being the most efficient method at the same time.

Unbalanced grid. Another interesting experiment is variating the grid resolution among dimensions. As mentioned in [**?**], a good heuristic for CPU time is to treat the largest dimension first in ST-HOSVD, this is also true for RPOD and TT-SVD. It is also quite important for compression rate in recursive format as few modes of the first dimension will be stored. As one can see in Fig. 21, the large imbalance in favor of the first dimension makes recursive decompositions comparable (although less efficient) with ST-HOSVD. Fig. 21a shows an exponential decay of the error with respect to the compression rate, just as observed for equal grid refinements in Fig. 20. The main difference lies in the comparatively higher efficiency of RPOD together with much clearer "stepping" phenomenon. In Fig. 21b, one can see that n_1 is 5 times bigger than the other n_i, this leads to a far greater efficiency of TT-SVD as compared with Fig. 19. Once again RPOD displays the same behavior as ST-HOSVD although the error is almost ten times greater. As expected, methods that treat dimensions sequentially are comparatively improving when the number of points in each dimension is imbalanced. This should be taken into account when dealing with experimental data.

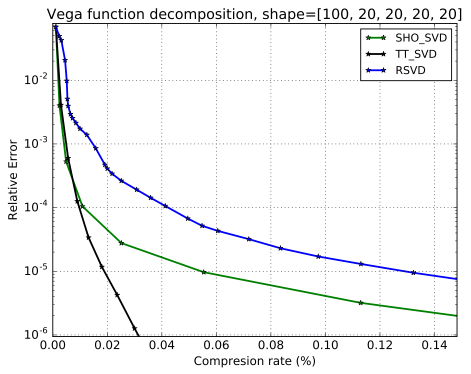
6 Conclusion

In the era of super computers, scientific computing is confronted more than ever to the curse of dimensionality. In this chapter, we have presented and explained a new paradigm that aims at solving this paradox. The general approach is to break the dimensionality with methods that turn exponential

[17]most efficient methods depends on required accuracy for $d = 4$.



(a) f_2 decomposition on a $1000 \times 20 \times 15 \times 12$ grid with POD based methods



(b) f_s decomposition on a $100 \times 20 \times 20 \times 20 \times 20$ grid with SVD based methods.

Figure 21: Decomposition of synthetic functions f_2 and f_s for unbalanced grid refinements.

growth with respect to the number of dimension into linear growth. This approach is two fold. First, data decomposition techniques aim at reducing existing data in order to facilitate storage and manipulation. Second step is to build reduced order models that solve slightly different problems with acceptable loss of accuracy but for considerable decrease of computing time (at least in the on-line phase). Often, low rank bases obtained with data decomposition are used which is why it is often referred to as off-line phase.

Obviously, complex problems require extended analysis prior to building such ROMs.

It was shown that bivariate decompositions are equivalent mathematically, they include matrix decomposition through SVD and function decomposition through POD or PGD. By equivalent, we mean that they perform the same operation on different spaces or norms. Their usual definitions involve different algorithms that can be tweaked into one another. This is supported by numerical implementation as long as convergence is reached. It was highlighted that some fields are more separable than others. Consequently, they have been deemed weakly separable and strongly or exponentially separable. Extensive insight on numerics has been provided.

A broad review of tensor formats and decompositions was provided. To do so, a complete description of these objects, their comparative advantages and algorithms have been provided. The theoretical aspect indicates that canonical decomposition, in spite of its d-linear storage cost, will produce poor approximation since the problem is NP-complex. Tucker decomposition is composed of modes and a correlation tensor core of the same order but of much smaller size than the original tensor. This structure makes it particularly suitable for decomposition of low order tensors by successive SVDs but larger dimension will cause exponential growth of the core tensor. Finally, TT and Hierarchical formats introduce formats that grow linearly with d while presenting SVD based decomposition. That makes them good candidates for decomposition of high to very high number of dimension. $d = \mathcal{O}(1000)$ is perfectly accessible, which leads to the new practice of tensorization. Also, the distinction between formats and their associated decomposition has been highlighted to prevent prejudicial confusion.