Numerical Study of Low Rank Approximation Methods for Mechanics Data and Its Analysis

Lucas Lestandi

Noname manuscript No.
(will be inserted by the editor)

Numerical Study of Low Rank Approximation Methods for Mechanics Data and its Analysis.

Lucas Lestandi

Received: date / Accepted: date

Abstract This paper proposes a comparison of the numerical aspect and efficiency of several low rank approximation techniques for multidimensional data, namely CPD, HOSVD, TT-SVD, RPOD, QTT-SVD and HT. This approach is different from the numerous papers that compare the theoretical aspects of these methods or propose efficient implementation of a single technique. Here, after a brief presentation of the studied methods, they are tested in practical conditions in order to draw hindsight at which one should be preferred. Synthetic data provides sufficient evidence for dismissing CPD, T-HOSVD and RPOD. Then, three examples from mechanics provide data for realistic application of TT-SVD and ST-HOSVD. The obtained low rank approximation provides different levels of compression and accuracy depending on how separable the data is. In all cases, the data layout has significant influence on the analysis of modes and computing time while remaining similarly efficient at compressing information. Both methods provide satisfactory compression, from 0.1% to 20% of the original size within a few percent error in L^2 norm. ST-HOSVD provides an orthonormal basis while TT-SVD doesn't. QTT is performing well only when one dimension is very large. A final experiment is applied to an order 7 tensor with $(4\times8\times8\times64\times64\times64\times64)$ entries (32GB) from complex multi-physics experiment. In that case, only HT provides actual compression (50%) due to the low separability of this data. However, it is better suited for higher order d. Finally, these numerical tests have been performed with `pydecomp` , an open source python library developed by the author.

Keywords Low rank approximation · tensor decomposition · HOSVD · ST-HOSVD · Tensor Train · QTT · HT · Hierarchical · Canonical Decomposition · RPOD · POD · SVD.

Mathematics Subject Classification (2010) 35Q35 · 15A69 · 15A21 · 78M34

1 Introduction and problem setting

In the past 50 years scientific computing has gained tremendous traction and is now ubiquitous to many fields of research and engineering especially in mechanics. It has been accompanied by the explosion of CPU power and the introduction of supercomputers and their massively parallel architectures since the 1990s. Still, the advent of exascale computing has only pushed forward the boundaries of computable problems slightly while raising a series of technical issues. Indeed, supercomputers are expensive infrastructures that require extensive amounts of energy. They produce data so large that storing and transferring data itself has become an issue. A famous simulation of the observable universe [1] 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.

The above mentioned figures are typical examples of the famous curse of dimensionality which remains the main obstacle to scientific computing development. Current programs are able to manipulate, with proven and often satisfying levels of accuracy, many direct space-time (3+1D) simulations. It is, however, not the case when one wants to perform parametric studies or optimization and control tasks such as shape optimization or active flow control. For these problems, the curse of dimensionality plays a major role and both the computing time and storage cost become out of reach. Extreme cases of the curse of dimensionality are offered by computational chemistry. A toy model may take the following form, let a small $n = 2$, be the number of discrete point per dimension, with $d = 50$ dimensions, then the storage cost $n^d = 2^{50}$ for a single state 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

Lucas Lestandi
Nanyang Technological University, School of Physical and Mathematical Science-MAS, Singapore
Engineering Mechanics, IHPC, A*STAR, Singapore
E-mail: Lucas_Lestandi@ihpc.a-star.edu.sg

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 for large d as shown above.

Another difficult problem is real-time simulation because high precision simulations require CPU times orders of magnitude larger than real time. These issues can be tackled in many ways. A prominent approach, since the turn of the century, lies in reduced order modeling (ROM) and the various techniques employed such as proper generalized decomposition (PGD) [2], POD-Garlekin ROM [3] or Petrov-Galerkin ROM [4,5]. The various methods presented in this paper aim primarily at enabling analysis and cheap storage of extensive datasets but can very well constitute the first layer of multidimensional solvers [6] or enable real-time [7] simulation or digital twin approaches.

Most of the methods studied in this paper rely on bidimensional separation methods such as singular value decomposition (SVD) and proper orthogonal decomposition (POD) that are essentially equivalent (see [8, Chapter 1]). They have been rediscovered many times in various fields e.g. as principal component analysis (PCA) in statistics [9,10], Karhunen-Loève expansion (KLE) in probability theory [11] or POD in fluid dynamics [12,13]. These methods provide a decomposition that can be truncated optimally [14] and reflect the physics of the problem studied. In fluid mechanics, POD bases have spurned the first wave of ROM. Indeed, this decomposition provides, among the many possible bases [15], an orthogonal basis of the functional space in which the solution lives. Consequently, many attempted at building Galerkin projection ROM on these reduced bases from the 1980s onward [13,16,17,18,19] but have registered modest success due to instability. Modern approaches are more successful [4, 3].

One extension of these work lies in the decomposition of multidimensional problems. Hitchcock is usually considered to have introduced tensor decomposition [20] in 1927. But, it is Tucker [21] that popularized the subject in the 1960s with the eponymous format, followed by Carroll and Chang [22] and Harshmann [23] in 1970 who introduced the canonical/parafac format and decomposition (CPD). The detailed review by Kolda and Bader [24] in 2009 is the basis of many of the modern developments in the field but it is devoid of numerical study. CPD has received dwindling interest due to poor numerical performance except in the context of ROM in which the PGD [2] has been extensively studied. For data analysis, the PGD can be interpreted a continuous form of CPD (see [8, Chapter 1] and [25]). Tucker format was at the center of attention since DeLathauwer paper in 2000 [26] which proposed an efficient approximation strategy, the Higher Order SVD (HOSVD) followed by HOOI [27]. More recently, he coauthored [28] which introduces ST-HOSVD that improves 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 and co-authors proposed the tensor train (TT) format [29,30], 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 as in QTT [31]. To do so, one may need to rely on partial evaluations of the interest data as performed by `TT-DMRG-cross` [32,33]. This approach is also known as blackbox algorithms [34] in the context of hierarchical tensors (HT) developed by Grasedyck, Kessner and Tobler [35,6]. HT actually incorporates all previously mentioned formats and approximations into a general d-linear format. These recent developments have been reviewed in [36] while an extensive mathematical analysis of tensors and their approximation is given in Hackbush's book [25]. Cichocki [37] gives a detailed presentation of many of these methods and their implementation variants in a visual and colorful way.

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 Oseledets [38] and continued by Bigoni and Gorodetsky [39,40,41] while many approaches now consider n-way array tensors and multivariate function as a single object [25,42,43]. Additionally, a Recursive-POD (RPOD) was proposed in [44]. Finally, in [45], the authors propose a practical presentation of these methods for actual decomposition.

The present paper proposes to apply the methods detailed in [45], complemented with HT and QTT to actual data and devise a context dependent hierarchy of methods for compression. The goal is to complement method specific numerical studies and high performance parallel implementations such as [46]. Also, this study tackles phenomena analysis, in the same spirit as POD has been used for decades to analyze modal behaviors in mechanics. This paper supports, with examples, that the same is possible for multidimensional decomposition.

In this paper, we focus on the practical aspect of using decomposition methods in the context of multidimensional data compression and analysis. For this reason, we do not enter the theory behind each of these methods, this work has already been performed in [45]. But, a short reminder of the methods is given in the next sections.

The paper is organized as follows. The next subsection presents the problem formulation. Section 2 briefly reviews bivariate decomposition methods. Higher dimension formats and methods are presented in section 3. Finally, section 4 focuses on numerical experimentation on various examples from the mechanics background. These results are analyzed and compared in this last section in order to provide relevant insight to the reader for practical use of these methods.

1.1 Problem formulation

The goal of this paper is to *present and compare low rank approximation techniques on actual numerical data with a dual objective of compression rate and accuracy for further use and analysis.* Of course, other computational costs are considered such as memory, CPU time and the possibility to parallelize. The general formulation of this problem is presented here.

It is assumed that the following field is known, at least in a discrete representation e.g. as a result of a finite elements method,

$$\begin{aligned} f : \Omega \subset \mathbb{R}^d &\to E \subset \mathbb{R} \\ \mathbf{x} &\longmapsto \quad f(\mathbf{x}) \end{aligned}$$

where $\Omega = \prod_{i=1}^{d} [a_i, b_i]$.

We are seeking $\tilde{f}$ the *"best"* separated approximation of f. A separated representation of a function consists in a combination (sum and product) of univariate functions. In this paper, for practical reasons, these functions are normalized and orthogonal (when possible) to the others so that it constitutes a basis. The definition of the *"best"* separated approximation of f depends strongly on the chosen norm to measure approximation error which defined as

$$\mathcal{E}(f, \tilde{f}, \mathcal{N}) = \frac{||f - \tilde{f}||_{\mathcal{N}}}{||f||_{\mathcal{N}}} \tag{1}$$

where $\mathcal{N}$ is commonly $L^2(\Omega)$ for functions or l^2 for discrete representations of fields (tensors) but any appropriate norm such as L^∞ can be used.

The other constrain on this optimization problem is the (reduced) space $\mathcal{V}$ to which $\tilde{f}$ belongs. We can then define the general optimization problem for multidimensional problem optimization.

Approximation problem Find $\tilde{f} \in \mathcal{V}$, such that

$$\tilde{f} = \underset{v \in \mathcal{V}}{\operatorname{argmin}}\, \mathcal{E}(f, v, N_{\mathcal{V}}) \tag{2}$$

where $N_{\mathcal{V}}$ is the chosen norm on $\mathcal{V}$. Some additional constraints may be needed to ensure existence and unicity of the solution as detailed in later sections.

One should note that this process is highly compatible with discrete representation. In this context, the multivariate function is replaced by an order d tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$. If a few modes containing the most relevant information is conserved, then a reduced approximation is obtained.

Each numerical implementation of the formats, is provided with meaningful comparison of complexity and compression rate for a given approximation error. The compression rate (CR) is defined by this formula

$$\mathrm{CR} = \frac{\texttt{Mem\_cost}(\tilde{f})}{\texttt{Mem\_cost}(f)}. \tag{3}$$

It enables, with a single indicator, to compare methods with different output formats. It can be interpreted as a "practical" CR, that compares the output memory cost versus the initial full tensor memory requirement.

2 Bivariate separation

A separated sum approximation for bivariate function reads as follows

$$f(\boldsymbol{x}, t) \approx \tilde{f}_r(\boldsymbol{x}, t) = \sum_{k=1}^{r} a_k(t)\phi_k(\boldsymbol{x}) \tag{4}$$

where we can impose a number of conditions on $\{a_k\}$ and $\{\phi_k\}$. Such approximation can be obtained through POD or PGD. A similar formulation can be written for algebraic spaces i.e. vectors and matrices in the 2D case. For any $A \in \mathbb{R}^{m \times n}$,

$$A \approx A_r = \sum_{i=1}^{r} \sigma_i \boldsymbol{u}_i \otimes \boldsymbol{v}_i \tag{5}$$

where $\otimes$ is the tensor product of $\boldsymbol{u}_i \in \mathbb{R}^m$ and $\boldsymbol{v}_i \in \mathbb{R}^n$ and $\sigma_i \in \mathbb{R}_+^*$. This decomposition can be obtained via SVD and is equivalent to the PCA.

2.1 Singular Value Decomposition (SVD)

This method can be viewed as a generalization of the eigenvalues decomposition (EVD) for rectangular matrices [47].

Theorem 1 *For any matrix $A \in \mathbb{R}^{m\times n}$, there are orthonormal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ so that*

$$A = U\Sigma V^\intercal \tag{6}$$

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 Eckart-Young theorem [14] ensures that the r-truncated SVD is the best rank-r approximation in the form of eq. 5 of a matrix in l^2 norm.

This is why SVD is the main tool for low rank approximation even when applied to higher dimensional problems.

Computing SVD is not as trivial as it seems. This question was extensively surveyed throughout the development of computer science. Standard algorithms range from QR or EVD and their improvements when it comes to dense matrices while Lanczos and other iterative methods may be more suitable for sparse matrices. Refer to [48] for complete discussion. Modern needs, e.g. machine learning, page rank algorithms or ROMs, have spurred a new wave of research to tackle problem of very large size. [49] provides an extensive discussion on computing SVD for these very large systems and describes PRIMME library. As for most of the state-of-the-art methods and programs, they address explicitly sparse matrices with MPI and high accuracy. However, this paper is focused on output data from physics simulation, hence meaning the data itself may be (much) larger than the memory available. Such problem can be addressed by so called out-of-core SVDs [50] and more generally linear algebra. This topic has been very active in recent years as shown in [51] bibliographic references. [51] describes a very efficient library to perform SVD without ever loading fully the matrix of interest. Similar approaches are becoming very common in the scientific computing community as well, as shown by recent paper [52] focusing on fine time step data from fluid flows. Including these approaches is beyond the scope of this article and is left for future development.

EVD solver to compute full matrix SVD In this work, the link between SVD and EVD is exploited and the actual implementation uses `dsyevd` from Lapack as it ensures high quality of the modes (mainly orthogonality) while significantly lowering the size of the singular value problem, see [45] section on bivariate decomposition for more details. As shown in Tab. 1, for any $A \in \mathbb{R}^{m\times n}$ with standard SVD decomposition $A = U\Sigma V^\intercal$ it is possible to solve an EVD problem of size $\min(m, n)$ to obtain the SVD at a lower computational cost, in particular for "tall" or "wide" matrices. However, the main drawback is that the eigen values are computed to machine precision ($\epsilon = 10^{-16}$) which limits the approximation error to 10^{-8}. It should be noted that EVD solver is very convenient for keeping a constant architecture for POD and SVD approaches. For completeness, `pydecomp` can also solve SVD by `primme_svds` for specific investigation e.g. for higher precision, but numerical tests have shown it is much slower than EVD for large tall skinny matrices. For instance, it is 10 times slower for a matrix of $10^5 \times 500$.

$m \geq n$	$A^\intercal A = V\Sigma^\intercal\Sigma V^\intercal = V\Sigma^2 V^\intercal$	is a $n \times n$	eigen value problem and $U = AV\Sigma^{-1}$
$n \geq m$	$AA^\intercal = U\Sigma\Sigma^\intercal U^\intercal = U\Sigma^2 U^\intercal$	is a $m \times m$	eigen value problem and $V = \Sigma^{-1}U^\intercal A$

Table 1: Solving an equivalent EVD to SVD can lead to substantial complexity gains for $n \gg m$ or $m \gg n$.

2.2 Proper Orthogonal Decomposition (POD)

It was discovered many times in different fields, however it is often attributed to Kosambi [53] who introduced it in 1943. It is a linear procedure that computes a basis of orthogonal proper modes. They are obtained by solving Fredholm's equation for data. Additionally, the POD offers an optimal linear subspace approximation which can be interpreted as optimal in terms of energy approximation in term of L^2 norm.

POD 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

$$\tilde{f}_r(\boldsymbol{x}, t) = \sum_{k=1}^{r} a_k(t)\phi_k(\boldsymbol{x}) \tag{7}$$

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.

The 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 $\{t_j\}_{j=1}^{n_t}$. The snapshot space is $\mathcal{F} = \text{span}\{f_1, f_2, ..., f_{n_t}\}$.

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:

$$\min_{\{\phi\}_{k=1}^{r}} \sum_{j=1}^{n_t} \|f_j - \tilde{f}_{r,j}\|^2, \text{ s.t. } (\phi_k, \phi_j) = \delta_{kj} \tag{8}$$

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 $\langle\cdot\rangle$. 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.

Optimality of the POD basis The POD basis generated by the POD algorithms is the optimal linear subspace approximation with respect to the chosen scalar product (usually L^2) i.e. the truncated approximation of f has the lowest error for a fixed rank r.

Subsequently, we admit that this problem can be solved numerically by an EVD method (see [45] and [8, Sec 1.2] for details and discussion on POD discrete implementation). POD can be seen as a continuous version of the SVD for bivariate functions $f(x,t)$. For this reason, in the next section, we only present one version of the methods while the continuous and discrete approach can be switched freely.

3 Multi-dimensional decomposition formats and methods

We are interested in the approximation of d dimensional data whether it is a tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ or a function of d variables $f(x_1, ..., x_d)$. Several decomposition formats and their methods are compared in this article: *Canonical (CPD), Tucker, Hierarchical (HT), Tensor Train (TT), Quantics-TT (QTT) decompositions and recursive POD*. They have been presented in details in many articles and books [45, 8, 25, 37] which is why they are briefly exposed in this section and only the most natural framework. The focus is set on tensors for which all the data is already available and is supposed to originate from mechanics or fluid dynamics. This implies that the approximation must account for all the data available since it is expensive to obtain and excludes of this study sub-sampling techniques such as cross approximation or greedy sampling.

First, a few definitions are introduced in order to describe accurately tensor decompositions and the associated algorithms. Let $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ be an *order* d tensor with the multi-index $\mathcal{N} = n_1 \times \cdots \times n_d$.

Products on tensors. The tensor product $\otimes$ is the generalization of outer product to tensors.

$$\begin{aligned} \otimes : \mathbb{R}^{\mathcal{N}} \times \mathbb{R}^{\mathcal{M}} &\to \mathbb{R}^{\mathcal{N} \times \mathcal{M}} \\ (\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}}) &\mapsto \quad \boldsymbol{\mathcal{X}} \otimes \boldsymbol{\mathcal{Y}} \end{aligned}$$

Entry-wise $\boldsymbol{\mathcal{T}} = \boldsymbol{\mathcal{X}} \otimes \boldsymbol{\mathcal{Y}}$ reads $T_{\boldsymbol{ij}} = x_{\boldsymbol{i}} y_{\boldsymbol{j}}$. It will mostly be used for vector to vector product to generate higher dimension objects. The **Kronecker** product (of matrices) is noted with the same sign $\otimes$ when no confusion is possible. The **Hadamar** (entrywise) product is noted $*$ while the **Kathri-Rao** product is noted $\odot$. Detailed definitions and discussion can be found in [45, 24].

Matricization. is the process of ordering the elements of a tensor into a matrix. The mode-μ matricization of a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{\mathcal{N}}$ is denoted by $\mathbf{X}_{(\mu)}$ and arranges the mode-μ fibers to be the columns of the resulting matrix. The map from tensor entries to matrix entries is uniquely defined and corresponds to `reshape` operation when implementing. Such matricization can be extended ([35, Definition 3.3]) to any binary split of the dimensions into two complementary sets of indices (t, t') for which is useful for HT decomposition. It is then similarly noted $\mathbf{X}_{(t)}$.

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

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

It is equivalent to say that each mode-μ fiber is multiplied by the matrix A, i.e.
$\boldsymbol{\mathcal{Y}} = \boldsymbol{\mathcal{X}} \times_\mu \mathbf{A} \Leftrightarrow \mathbf{Y}_{(\mu)} = \mathbf{A}\mathbf{X}_{(\mu)}$. This operation can be extended to **multilinear multiplication** (see [28]) for processing multiple μ-mode products in a single operation.

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

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

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

In the following subsections, a brief presentation of each format and most suitable decomposition is provided. Again, more details can be found in [45, 8]. The reader is referred to [25] for theoretical developments and [54] for `htucker` toolbox and multiple references.

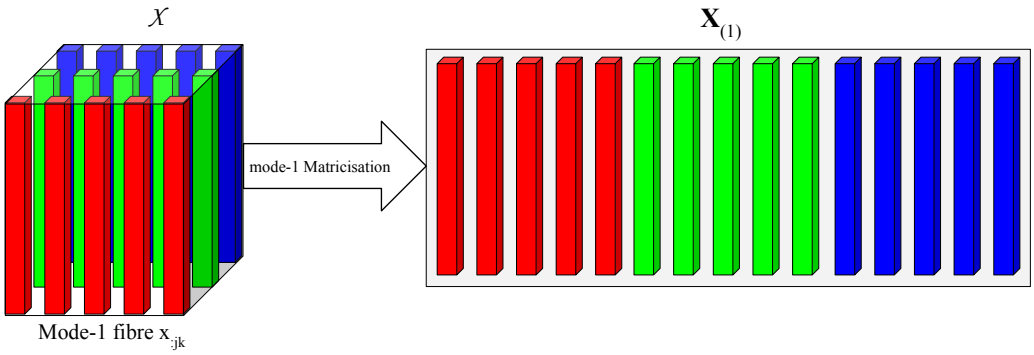


Fig. 1: Mode one matricization of third order tensor with $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I \times J \times K}$.

3.1 Canonical decomposition

The canonical or PARAFAC decomposition (CPD) consists in sum of rank-1 tensors. A rank-1 tensor of order d can be written by a single tensor product of d vectors. The algebraic CPD reads

$$\boldsymbol{\mathcal{F}} \approx \boldsymbol{\mathcal{F}}_r = \sum_{k=1}^{r} \bigotimes_{i=1}^{d} \tilde{\boldsymbol{x}}_i^k \tag{9}$$

where r is the rank of the approximation and $\{\{\tilde{\boldsymbol{x}}_i^k\}_{k=1}^r\}_{i=1}^d$ are r sets of vectors associated with each dimension, $\forall, i \leq d,\ k \leq n_i,\ \tilde{\boldsymbol{x}}_i^k \in \mathbb{R}_i^n$. A schematic view of such a decomposition is given in Fig. 2.

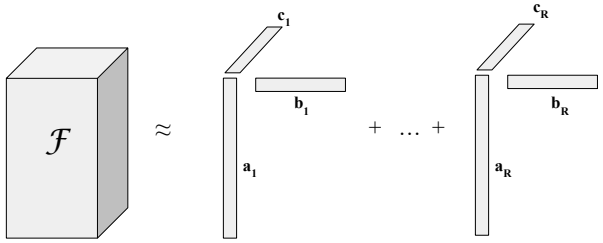


Fig. 2: CP decomposition of third order tensor $\boldsymbol{\mathcal{F}} \in \mathbb{R}^{I \times J \times K}$.

The continuous formulation of the canonical decomposition reads

$$f(x_1, ..., x_d) \approx f_r(x_1, ..., x_d) = \sum_{k=1}^{r} \prod_{i=1}^{d} X_i^k(x_i) \tag{10}$$

where $\{\{X_i^k\}_{k=1}^r\}_{i=1}^d$ can be viewed as basis functions in the functional space of f.

In practice, this decomposition is obtained through a successively enriching algorithm such as PGD-like algorithms (see [2,55,56]) or alternating least squares (ALS) [24]. The storage cost is linear in d ($\mathcal{O}(drn)$), but the convergence of these methods is not certain. The idea of the algorithm is to compute progressively the basis in all dimensions by enriching it of a new vector at each iteration so as to ensure closedness of the optimization problem. This process does not produce an optimal basis but it improves after each iteration although the improvement might become negligible for poorly separable functions. The tensor version of the algorithm is given in Algorithm 1.

3.2 Tucker decomposition

Tucker decomposition uses a similar format as to Canonical but takes advantage of all the combinations of modes of different dimensions storing the associated weight in a *core tensor* $\boldsymbol{\mathcal{W}}$ whose entries are $w_{\boldsymbol{k}}$ with $\boldsymbol{k} = k_1, ..., k_d$.

Algorithm 1: Alternating Least Square (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 $\forall 1 \leq i \leq d, \quad \boldsymbol{x}_i$;
while *Error* $\geq \varepsilon$ **do**
  for $i = 1, d$ **do**
1 $V = \boldsymbol{X}_1{}^{\intercal}\boldsymbol{X}_1 * \cdots * \boldsymbol{X}_{i-1}{}^{\intercal}\boldsymbol{X}_{i-1} * \boldsymbol{X}_{i+1}{}^{\intercal}\boldsymbol{X}_{i+1} * \cdots * \boldsymbol{X}_d{}^{\intercal}\boldsymbol{X}_d$; /* $V \in \boldsymbol{R}^{R \times R}$ */
2 $\boldsymbol{X}_i = \mathcal{F} \cdot (\boldsymbol{X}_d \odot \cdots \odot \boldsymbol{X}_{i+1} \odot \boldsymbol{X}_{i-1} \odot \cdots \odot \boldsymbol{X}_1)V^{\dagger}$; /* † denotes the Monroe-Penrose pseudo-inverse */
3 $w_i = \|\boldsymbol{X}_i\|_2$;
4 $\boldsymbol{X}_i = \frac{\boldsymbol{X}_i}{w_i}$
return $\mathcal{X} = [\![\boldsymbol{w}; \boldsymbol{X_1}, \cdots, \boldsymbol{X_d}]\!]$

A 3D visual is given in Fig. 3. The tucker format approximation reads

$$\mathcal{F} \approx \mathcal{F}_{\boldsymbol{k}} = \sum_{k_1=1}^{r_1} \cdots \sum_{k_d=1}^{r_d} w_{\boldsymbol{k}} \bigotimes_{i=1}^{d} \hat{\boldsymbol{x}}_i^{k_i} \tag{11}$$

where the rank $\boldsymbol{r} = (r_1, ..., r_d)$ of the core tensor is the Tucker rank of the decomposition. Each basis $\{\hat{\boldsymbol{x}}_i^k\}_{k=1}^{r_i}$ is orthogonal and contains r_i vectors of size n_i. For each dimension i, the overall basis can be concatenated into a matrix $\hat{X}_i$ of size $r_i \times n_i$. The functional Tucker decomposition reads

$$f(x_1, ..., x_d) \approx f_{\boldsymbol{k}}(x_1, ..., x_d) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_d=1}^{r_d} w_{\boldsymbol{k}} \prod_{i=1}^{d} X_i^k(x_i) \tag{12}$$

Any tensor can be represented exactly in Tucker format, however, we are interested in approximated with minimal

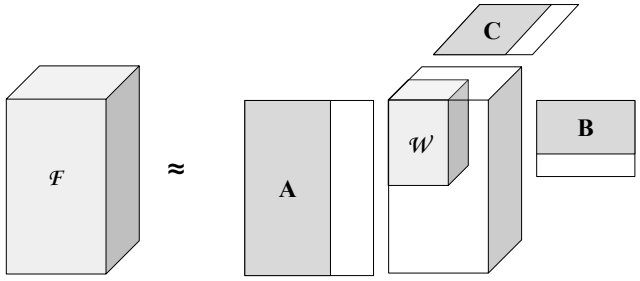


Fig. 3: *Truncated* Tucker Decomposition of a third order tensor $\mathcal{F} \approx (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}) \cdot \mathcal{W}$ with $r_i \leq n_i$ for $i = 1, 2, 3$.

(l^2 or L^2) error and minimal rank i.e. size of $\mathcal{W}$, thus minimum storage cost. Many algorithms are available to compute Tucker decomposition (HOOI, T-HOSVD, ST-HOSVD,...). In this paper we focus on ST-HOSVD (sequentially truncated) proposed by [28] that has superseded the commonly used T-HOSVD [26] (truncated). Indeed, it provides a similar accuracy (as shown in the numerics section) to T-HOSVD while significantly reducing the number of operations.

The storage cost of Tucker format is quasi-linear in d for small r ($\mathcal{O}(r^d + drn)$) but this cost grows exponentially with d which limits the use for $d \geq 5$. The convergence is certain with a quasi-optimal error estimate.

The idea of the ST-HOSVD is to perform successively SVDs on each dimension against all the others after flattening the tensor and truncate the decomposition to a prescribed level. Doing so loses the inherent independence of dimensions of T-HOSVD and thus the possibility to directly parallelize. The core tensor containing the associated weight corresponds to multiplying the singular values or by projecting of $\mathcal{F}$ onto the basis. This process is detailed in algorithm 2.

Algorithm 2: ST-HOSVD

input : $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, truncation rank $\boldsymbol{r}$, processing order $\boldsymbol{p} = (p_1, ..., p_d)$
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   $(\boldsymbol{U}_{tr}, \boldsymbol{\Sigma}_{tr}, \boldsymbol{V}_{tr}^\intercal) = truncate(\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V}^\intercal, r_i)$;
3   $\hat{\boldsymbol{X}}_i = \boldsymbol{U}_{tr}$;
4   $\hat{\mathcal{W}}_{(i)} = \boldsymbol{\Sigma}_{tr} \boldsymbol{V}_{tr}^\intercal$;
return $\mathcal{X} = [\![\hat{\mathcal{W}}; \hat{\boldsymbol{X}}_1, ..., \hat{\boldsymbol{X}}_d]\!]$

3.3 Hierarchical Tensor

The main flaw of the tucker format lies in the exponential growth of its core tensor. A very elegant way to address this issue is to introduce the Hierarchical Tensor (HT) format often referred to as H-Tucker.

Ample theoretical analysis and definitions are given in Grasedyck, Ballani and Hackbush work [35,57,58,59, 25] and the reader is strongly encouraged to refer to these for a better presentation of this method.

It is based on the idea of recursively splitting the core tensor under a tree structure. The process results in a binary tree $\mathcal{T}_D$ containing a subset of dimensions $t \subset D := \{1, ..., d\}$ at each node as shown figure 4. One can navigate the tree from the root (uppermost node) at level $l = 0$ to the maximum depth ($l = L$) which contains only leaves. A leaf is a node that represents only one dimension i.e. $t = \{i\}$ with $i \in D$. The leaves are equivalent to Tucker format basis matrices (X_i).

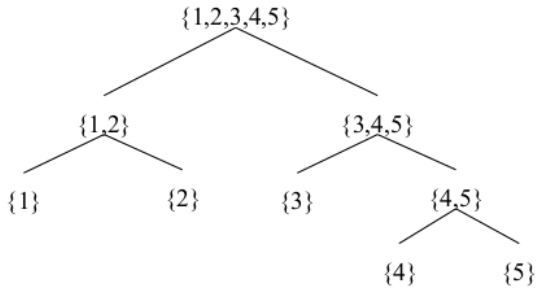


Fig. 4: Binary dimension partition tree of $D = \{1, ..., 5\}$ showing t at each node.

The Hierarchical rank $(r_t)_{t \in \mathcal{T}_D}$ of a tensor $\mathcal{X} \in \mathbb{R}^{\mathcal{I}}$ is defined by

$$\forall t \in \mathcal{T}_D,\ r_t = \mathrm{rank}(\boldsymbol{X}^{(t)}) \tag{13}$$

In other words, it corresponds to the rank of each t-matricization throughout the tree.

HT format relies on the following decomposition that holds for each node $t = t_1, t_2$ with rank r_t, r_{t_1}, r_{t_2}. For all $i \in \{1, ..., r_t\}$,

$$(\boldsymbol{U}_t)_i = \sum_{j=1}^{r_{t_1}} \sum_{l=1}^{r_{t_2}} (\mathcal{B}_t)_{i,j,l} (\boldsymbol{U}_{t_1})_j \otimes (\boldsymbol{U}_{t_2})_l, \tag{14}$$

where U_t is the standard HT notation for subspace of $\mathcal{F}$ for the subset of dimensions t. $\mathcal{B}_t$ is a third order transfer tensor of shape (r_t, r_{t_1}, r_{t_2}) and the leaves $\boldsymbol{U}_i$ are of dimension $r_i \times n_i$. Fig. 5 shows the nested structure and dimension of an order 4 HT tensor.

Here, we focus on the practical application of H-Tucker, but many other approaches are described in the above references. In this specific case, $\boldsymbol{U}_i = \boldsymbol{X}_i$ obtained from Tucker decomposition (HOSVD). Then the leaf-to-root truncation (see [54]) is applied to the core tensor in order to compute the transfer tensors at each level up to the root. This procedure, shown in Algorithm 3, involves several truncated SVDs that can further reduce the size of the HT approximation.

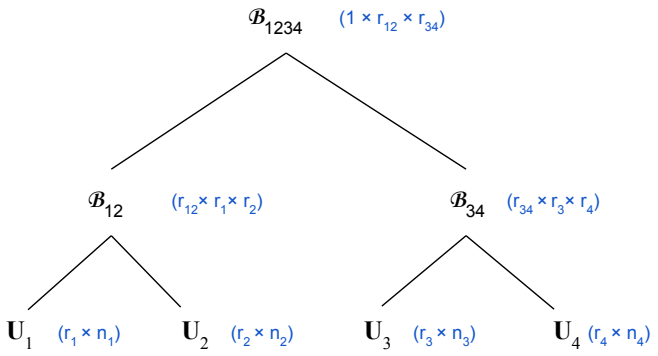


Fig. 5: Tree representation of the HT format of $\mathcal{X} \in \mathbb{R}^{n_1 \times n_2 \times n_3 \times n_4}$. The size of the matrices and tensors are inside blue braces.

Algorithm 3: Hierarchical Tucker Decomposition

input : $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, ε
output: HT root node: $\hat{\mathcal{X}} = \{(\mathcal{B}_t)_{t \in nodes}, (\boldsymbol{U}_t)_{t \in leaves}\}$
Compute HOSVD truncated to ε: $(\mathcal{C}^{\mathcal{L}}; \boldsymbol{U}_1, ..., \boldsymbol{U}_d) = \mathcal{F}$;
`/* Apply leaf to root procedure on core tensor */`
for *level l from L to 0* **do**
  Let $\mathcal{C}^l = \mathcal{C}^{l+1}$ the level core tensor. **for** *Each cluster node t at level l* **do**
    Compute SVD truncated to r_t corresponding to ε;
    $(\boldsymbol{S}_{tr}, \boldsymbol{\Sigma}_{tr}, \boldsymbol{V}_{tr}^{\intercal}) = \text{tSVD}(\mathcal{C}^l{}_{(t)})$;
    $\mathcal{B}_t = \text{reshape}(\boldsymbol{S}_{tr}, (r_t, r_{t1}, r_{t2})$;
    Update transfer tensor $\mathcal{C}^l = \boldsymbol{S}_{tr}{}^{\intercal} \circ_t \mathcal{C}^l$;

Consequently, one needs to store only the leaves matrices $\boldsymbol{U}_i$ and tranfer tensors $\mathcal{B}_t$ which amounts to storing $\mathcal{O}(kdn + (d-1)k^3)$ entries with standard rank r and size n across the tree. From an accuracy standpoint, HT decomposition performance is bounded by the initial T-HOSVD accuracy and may be lower if the transfer tensors are truncated too aggressively (i.e. with larger ε when aiming for hyper-reduction of the core tensor). The T-HOSVD, also constitutes the most expensive step as core tensor size diminishes when going up the tree.

All things considered, HT format and the associated decomposition are well suited to high dimensional tensor reduction since it enables both hyper-reduction (no tensor core) and physical modes decomposition. It is often overlooked by practitioners (see the vast number of packages covering Tucker, CPD and TT) as it is quite complex and is challenged for large d by the now ubiquitous Tensor Train decomposition that can be seen as a special case of HT imbalanced partitioning.

3.4 Tensor train (TT)

TT is a relatively recent method popularized by Oseledets team around 2010 [29,30] that allows easy implementation and is very efficient for $d \geq 5$. This format was first presented as a product of matrices that describe each entry of the studied tensor which is why it is also known as matrix product state (MPS) in the literature. It can be viewed as a specific case of hierarchical formats (see below) and boasts a d-linear storage cost $\mathcal{O}(dnr^2)$ which makes it a good candidate for large dimension decomposition.

The functional TT decompostion reads

$$f(x_1, ..., x_d) \approx \sum_{k_1, ..., k_{d-1}} G_1(x_1, k_1) G_2(k_1, x_2, k_2) \cdots G_d(k_{d-1}, x_d) \tag{15}$$

where $G_i(k_{i-1}, x_i, k_i)$ are the entries of a matrix of functions (for all $i \leq d$) and of size $r_{i-1} \times r_i$ (by convention $r_0 = r_d = 1$). $(r_1, ..., r_d)$ is the tensor-train rank of the decomposition. For any $\boldsymbol{x} = (x_1, ..., x_d)$, one can evaluate the value of $\boldsymbol{G_i}$ matrices and then compute the approximation of $f(\boldsymbol{x})$ at this specific point by evaluating matrix product $\boldsymbol{G_1}(x_1)\boldsymbol{G_2}(x_2) \cdots \boldsymbol{G_d}(x_d)$. Equivalently, the same can be applied to tensor $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times \cdots \times n_d}$, in which case each $\{x_j\}_{j=1}^d$ are replaced by and integer index $\{i_j\}_{j=1}^d$ and the decomposition reads $\forall\ 1 \leq i_1 \leq n_1, ..., 1 \leq i_d \leq n_d$,

$$\mathcal{F}(i_1, ..., i_d) \approx G_1(i_1) G_2(i_2) \cdots G_d(i_d). \tag{16}$$

Fig. 6 shows a visual of the decomposition of a 4th order tensor.

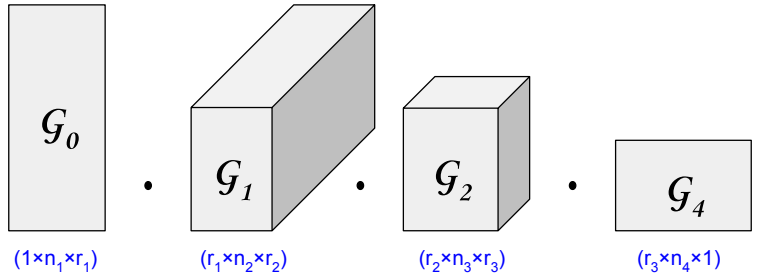


Fig. 6: Visual representation of the TT decomposition of an order 4 tensor of dimension $n_1 \times n_2 \times n_3 \times n_4$ and TT rank (r_1, r_2, r_3).

Among the many algorithms (TT-cross, TT-DMRG-cross,...) developed by Oseledets team to compute TT, TT-SVD has been chosen for this article. It is easy to implement and efficient for reasonable number of dimensions and full tensor data ($d \leq 6$) which is the setup of this numerical study.

The idea of this algorithm is to use a series of SVDs (or PODs) to separate the remaining tensor into a *transfer tensor* and a tensor of 1 order smaller until the last dimension is reached. The corresponding TT-SVD algorithm is given in algorithm 4. The main drawback of TT decomposition is the partial orthogonality of the transfer tensors (G_i) thus making it hard to use for ROM.

Algorithm 4: `TT-SVD`

input : $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, truncation rank $\boldsymbol{r}$ or prescribed error ε
output: $\mathcal{X}(i_1, ..., i_d) = \sum_{\alpha_0,...,\alpha_d=1}^{\boldsymbol{r}} G_1(\alpha_0, i_1, \alpha_1) \cdots G_1(\alpha_{d-1}, i_d, \alpha_d)$

Compute the truncation parameter $\delta = \frac{\varepsilon}{\sqrt{d-1}} ||\mathcal{F}||_F$;
Temporary tensor: $\mathcal{C} = \mathcal{A}$, $r_0 = 1$;
for $i = 1, ..., d$ **do**
  `/* reshape(`$\mathcal{C}, r_{i-1}n_i, \frac{\texttt{numel}(C)}{r_{i-1}n_i}$`)` `*/`
  $\boldsymbol{C} = \boldsymbol{C}^{(i*)}$;
  `/* truncated SVD at given rank` r_i `*/`
  $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal = \text{tSVD}(C, r_k, \delta)$;
  $\mathcal{G}_i = \text{reshape}(\boldsymbol{U}, [r_{i-1}, n_i, r_i])$;
  $\boldsymbol{C} = \boldsymbol{\Sigma}\boldsymbol{V}^\intercal$;
$\mathcal{G}_d = \boldsymbol{C}$;
return $\mathcal{X} = [\![\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_d]\!]$

3.5 Quantics TT

We have seen that TT is very efficient at handling large number of dimensions since it stores only order 3 tensors. Thus, it appears that the smaller n_i the more efficient the format becomes regardless how large d is. For this reason, the Quantics-TT (QTT) format was introduced in [32] and later [60] studied in a more general form. The author shows that this compression method complexity is in $O(d \log(n))$ for an order d tensor of size n^d.

The idea is to reshape the tensor into a very large order $D = d \log_q(n)$ where q is the base (or quanta) by splitting each dimension of $\mathcal{X}$ into a L q-size bits. For the best efficiency, q is usually a small integer like 2 or 3. This operation requires that $n = q^L$ which limits direct applications to datasets that can be expressed a power of q. In practice though, each n_i that doesn't fulfill this condition can be extended with zero valued elements until the new dimension $n_i' = q^L$. As there is no variance in the added data, it is usually captured easily during decomposition as shown in experiments.

All the algorithms discussed in the previous section apply to QTT including TT-SVD that is used in this paper. Sub-sampling approaches, like TT-DMRG, are particularly well suited to estimate very large tensors.

The main limitation of this approach is that by increasing tremendously the number of dimensions, the internal rank may grow to very large values (typically in the thousands) which leads to extremely tall and skinny matrices. These may be sensitive to bad conditioning and round-off error. Experimentally, it may be desirable to use an accuracy oriented SVD. In this work, an iterative solver is used when a problem is detected on the standard SVD method.

From a physical analysis point of view, most of the dimensional structure of the data is lost with QTT compression as discussed above. Direct interpretation of compression modes is not possible as well as building ROM. However, some approaches take advantage of QTT by mixing it with other methods such as HOSVD (see [60]).

Algorithm 5: `QTT-SVD`

input : $\mathcal{F} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$, truncation rank $\boldsymbol{r}$ or prescribed error ε
output: $\mathcal{X}(i_1, ..., i_d) = \sum_{\alpha_0,...,\alpha_D=1}^{\boldsymbol{r}} G_1(\alpha_0, i_1, \alpha_1) \cdots G_1(\alpha_{d-1}, i_D, \alpha_d)$

if *any* $n_i \neq q^{L_i}$ **then**
1 Extend dimension i with zeroes to $n_i' = \text{ciel}(log_q(n_i))$;
2 $\mathcal{F}^q$ =`reshape2quantic`$(\mathcal{F}, q)$;
3 Apply TTSVD, $\mathcal{X}$=`TTSVD`$(\mathcal{F}^q)$;
return $\mathcal{X} = [\![\mathcal{G}_1, \mathcal{G}_2, ...\mathcal{G}_D]\!]$

3.6 Recursive decomposition

The Recursive-POD (RPOD) is not properly a format but merely a convenient generalization of POD to higher dimensions. Note that as for the other methods, POD and SVD remain interchangeable. The idea is to perform PODs recursively on the first variable of the function or its partial decomposition until all the parameters have been separated. This creates a tree structure (shown in Fig. 7) that does not allow the bases $\{X_i^k\}_{k=1}^{r_i}$ to be orthogonal but enables efficient truncation. Because of the tree irregularity for a prescribed accuracy, it is not possible to give meaningful *a priori* storage cost. The general expression of RPOD reads

$$f(x_1, ..., x_d) \approx \sum_{k_1=1}^{R_1} \cdots \sum_{k_{d-1}=1}^{R_{d-1}(k_1,...,k_{d-2})} X_1^{k_1}(x_1)...X_d^{(k_1,...,r_{d-1})}(x_d) \tag{17}$$

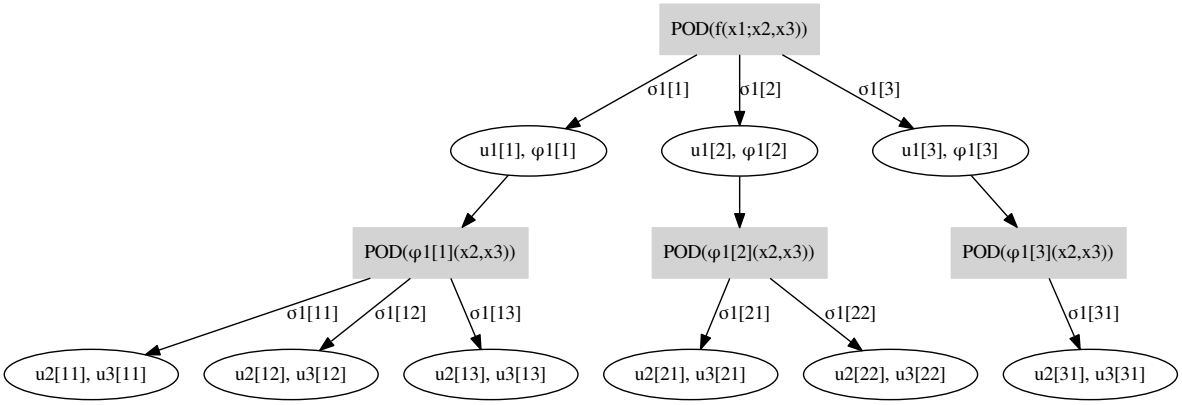


Fig. 7: Example of *a Recursive POD graph* of $f(x_1, x_2, x_3)$

To better understand the method, the trivariate function $f(x_1, x_2, x_3)$ RPOD, shown in Fig. 7, is presented in more details.

1. Separate x_1 from (x_2, x_3) with standard truncated POD at appropriate rank r_1:

$$f(x_1, (x_2, x_3)) \approx \sum_{k_1=1}^{R_1} X_1^{k_1}(x_1) \Phi^{k_1}(x_2, x_3)$$

2. Repeat the POD on each $\Phi^{k_1}(x_2, x_3)$ i.e. for all $1 \leq k_1 \leq r_1$:

$$\Phi^{k_1}(x_2, x_3) \approx \sum_{k_2=1}^{R_2(k_1)} X_2^{k_1,k_2}(x_2) X_3^{(k_1,k_2)}(x_3)$$

3. Then the full sum can be easily reconstructed:

$$f(x_1, (x_2, x_3)) \approx \sum_{k_1=1}^{R_1} \sum_{k_2=1}^{R_2(k_1)} X_2^{k_1,k_2}(x_2) X_1^{k_1}(x_1) X_3^{(k_1,k_2)}(x_3)$$

This algorithm can easily be extended to larger dimensions. Then notation can be quite clumsy (see eq. 17) and it has motivated the tree structure and implicit recursion programming in the actual implementation. Also, notice how the last two variables share the same decomposition and rank. Algorithm 6 produces RPOD tree decomposition as shown in Fig. 7.

Algorithm 6: RPOD

input : $f \in L^2(\mathcal{D})$, computing domain $\mathcal{D}$, target error ε
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 $f_{\mathcal{R}} = [[\mathcal{R}, \mathcal{S}, \mathcal{X}]]$

RPOD is not a continuous TT. The above algorithm and Fig. 7 show that the RPOD is intrinsically different from a continous TT [40] or TT-POD (see [8, Section 3.3]). Although it follows a similar recursive strategy, the structure of the branches varies widely. A simple way to visualize this, is that RPOD generates very wide trees since each mode leads to a new branch at the lower level leading to an exponential growth of the number of branches. On the contrary, TT-(SVD) achieves compactness by "aggregating" the lower level information when computing C at line 6 of algorithm 4. Numerically, it means each branch of RPOD is independent from its neighbors while TT computes all modes of $\mathcal{G}_i$ from a single SVD. These operations are not equivalent. Also, RPOD (or RSVD) is a Hierarchical decomposition but fairly distinct from HT. Despite its tree structure, the subspace approach is violated in the proposed description and rewriting as a TT-like structure would require adding many 0 entries to preserve shape across the tree since each POD is performed independently.

Conclusion on format

All these decomposition methods can be embedded in a general hierarchical format both numerically and theoretically ([25]) but their efficiency vary widely due to specific structures. In practice, computing time and memory use is a central issue that is explored in the next section. The main characteristics of the above formats are presented in Tab. 2 including common algorithms, storage cost and evaluation cost. The author does not attempt at giving an evaluation of the operation count since it is highly dependent on the outcome of intermediate evaluation as well as runtime parameters. Instead, a CPU time will be provided for realistic test cases. In the next section, a comprehensive numerical study is provided.

Table 2: Synoptic table of tensor formats for order d tensor $\mathcal{F} \in \mathbb{R}^{n \times \cdots \times n}$ with rank r or $\boldsymbol{r} = (r, ..., r)$.

Format	Algorithm	Storage	Evaluation
Full	Native array format	$\mathcal{O}(n^d)$	0
Canonical	ALS, PGD	$\mathcal{O}(drn)$	$\mathcal{O}(dr)$
Tucker	ST-HOSVD, HOSVD	$\mathcal{O}(k^d + dkn)$	$\mathcal{O}((d+1)k^d)$
Hierarchical	H-Tucker, root to leaf	$\mathcal{O}(kdn + dk^3)$	$\mathcal{O}((d-1)k^3)$
Tensor Train	TT-SVD, TT-POD,TT-DMRG-cross	$\mathcal{O}(dk^2n)$	$\mathcal{O}((d-1)k^3)$
Quantics TT	QTT-SVD, QTT-DMRG-cross	$\mathcal{O}(d\log(n))$	$\mathcal{O}((D-1)k^3)$
Recursive format	RPOD RSVD	Similar to TT	Similar to TT

4 Experimental results

First, we recall the criteria for measuring the error and compression. The approximation or decomposition error of $\mathcal{T}$ is defined by

$$E_{\mathcal{N}} = \frac{||\mathcal{T}_{\text{exact}} - \mathcal{T}_{\text{decomp}}||_{\mathcal{N}}}{||\mathcal{T}_{\text{exact}}||_{\mathcal{N}}}, \tag{18}$$

with $\mathcal{N}$ the norm defined by the user. Here, L^2 norm is used by default and, on some cases, L^∞ is selected. The compression rate (in %) which is the storage cost of an approximation in a given format for a specific rank divided by the storage cost of the full format. It reads

$$CR = \frac{\texttt{Mem\_cost}(\mathcal{T}_{\text{decomp}})}{\texttt{Mem\_cost}(\mathcal{T}_{\text{exact}})} (\times 100 \text{ for } \%). \tag{19}$$

Throughout this paper numerical experiments, CR is evaluated by arithmetic estimation. This choice excludes object oriented overheads such as list or dictionaries which are not studied here and can be ignored when compared to large size arrays they contain. Other considerations such as CPU time or online memory use are discussed but not studied in a comprehensive manner. Indeed, the data is supposed to come from large simulation which means proportional computing capabilities are available to the practitioner. All CPU times reported in this article have been obtained on a recent workstation equipped with an Intel Xeon Gold 6230 chip with 40 cores, 256G RAM. CPU times are measure with ipython magics `%time` which provides walltime and CPU time (multithread) while memory use is given by `%memit`. This hardware solution is extremely suitable for `pydecomp` (see below) as it relies on shared memory parallel algorithms shipped with `numpy`.

4.1 `pydecomp` software

In order to evaluate and compare these techniques, an open source (under CECILL license) software was developed. It is freely available at `https://git.notus-cfd.org/llestandi/python_decomposition_library`. This library relies heavily on `numpy` and `lapack` for computing efficiency of projections and SVD/POD. On top of that, a few classes are built, including the above formats, and the low rank approximation of tensors in these formats is automated. A few benchmark functions are available, they can serve both as tutorial (the functions are documented in the code itself) and as actual comparison tool. Fig. 8 provides a graph describing the structure of the library. Ample discussion on the structure and genesis of the software is provided in [8].

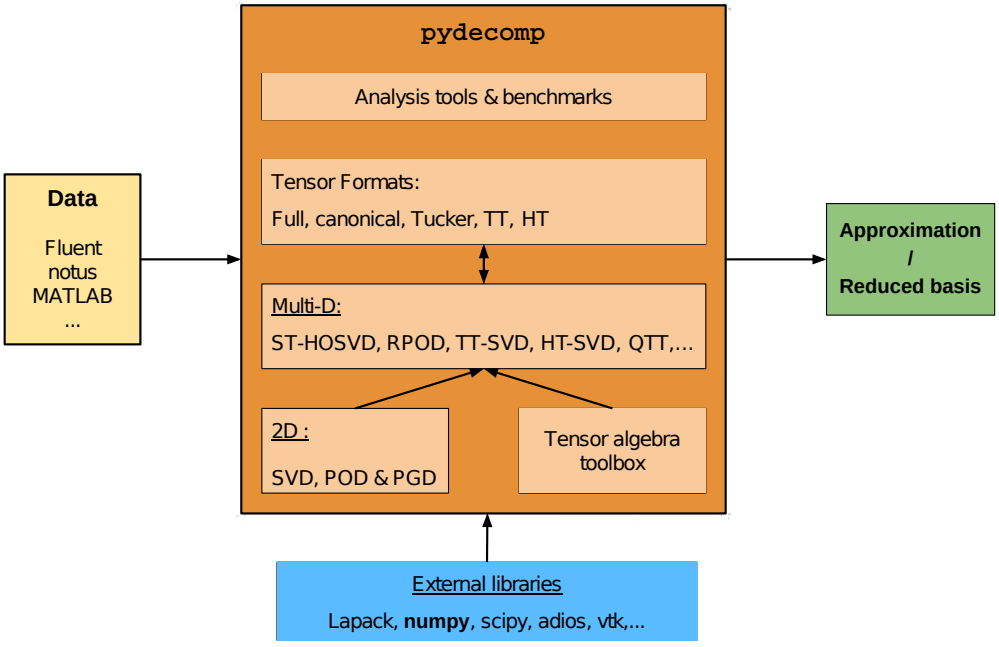
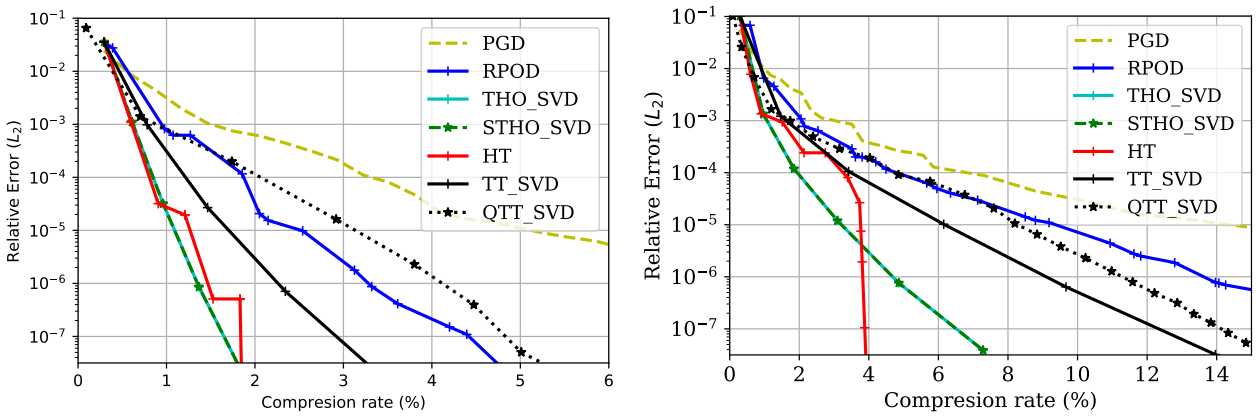


Fig. 8: `pydecomp` software architecture graph

For a standard order d tensor $\mathcal{F} \in \mathbb{R}^{n \times \cdots \times n}$ `pydecomp` complies with the general approximation characteristics of each format presented in table 2. For each of these formats, the storage cost is evaluated algebraically in `pydecomp` once the decomposition has been computed.



(a) f_1 (b) f_2

Fig. 9: Decomposition of 2 C^∞ test functions with $d = 3$ on a $32 \times 32 \times 32$ grid with 7 decomposition methods, using L^2 integration and norm.

Software design. The initial intent of this code is to provide a tool to evaluate low rank approximation in the context of fluid dynamics and mechanics simulation. The compression of data is an important feature but the long term goal is to provide bases for ROM and surrogate modeling as well as help analyze data in a similar fashion as POD does (see [61,62]). In the long run, when tackling several TB of data, the whole workflow will need to be adapted, e.g. by limiting communication and using local approximations. Fully parallel (MPI) and/or out of core data management is a computer science challenge in itself (see [46,51]). For these reasons the architecture of `pydecomp` has not been developed to handle very large data but can be seen as an easy tool to try several tensor decomposition methods on medium scale data. Finally, the user is referred to the many specialized (mostly open-source) tools for format specific approximation such as `htucker`[54] for HT format in matlab, `TensorToolbox` (Kolda & Bader,[63], C++/matlab) for Tucker, CP and others, another `TensorToolbox` by Bigoni [39] python or `ttpy` for TT format by Oseledets (many languages). Many other programs and problem specific solutions can be found on open repositories.

4.2 Synthetic data

In this subsection we briefly recall numerical results reported in [45] for synthetic data, by which we mean data directly computed from a function's expression. The data is generated on uniform grids of $n_1 \times \cdots \times n_d$ that discretizes $\Omega = [0, 1]^d$. The following real test functions are used

$$f_1(\boldsymbol{x}) = \frac{1}{1 + \sum_i x_i}$$
$$f_2(\boldsymbol{x}) = \sin(||\boldsymbol{x}||_2)$$

In order to evaluate the separability of these test functions with the studied methods, a relatively coarse grid of $32 \times 32 \times 32$ is used. This first experiment is performed with a functional approach for RPOD and PGD while the other methods rely on SVD since prior tests show very little influence when choosing POD or SVD. The results are presented for both functions in Fig. 9 in which one can compare the compression capacity of each method for these simple functions. The steeper the slope, the higher compression power is observed.

On these figures, one can see a group of very efficient methods (TT-SVD and *-HOSVD) i.e. the error reduces in a quasi-exponential fashion. On the other end of the spectrum PGD (the functional CP decomposition algorithm) is the least efficient in both cases. This can be attributed to the non-optimality of this method that is too important to be compensated by the storage d-linear efficiency. Additionally, this approach requires CPU times that are *orders of magnitude* bigger than the other methods and *grow exponentially* with the size of the problem. For this reason, we dismiss canonical decomposition (in both tensor and function form) from the following study as it is simply not able to compute a decomposition in a "reasonable" time.

On these simple examples, it is unclear whether RPOD is a good candidate or not, the decay of the error is not exactly exponential although the error is orders of magnitude smaller than PGD. Still, further analysis, as shown in [45], confirms that for all tested configurations, in terms of CR, recursive methods consistently perform below Tucker and tensor train methods in spite of a *much higher CPU time* (more than *5 times* more, see [45, Relative CPU time paragraph]). In addition to that, the complex tree data structure and the non-orthogonality of the collection of vectors leads to dismissing RPOD for further study of the numerical efficiency in this article.

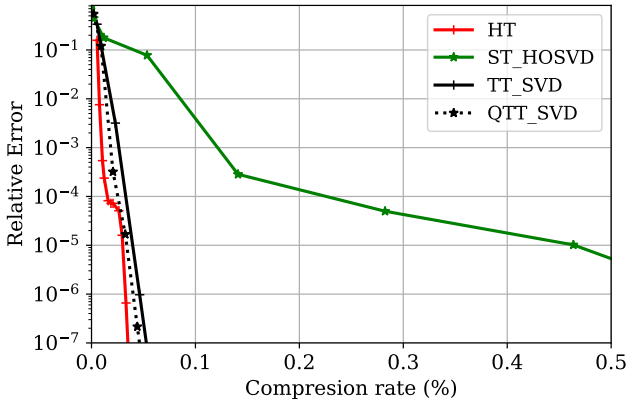


Fig. 10: Decomposition of f_3 on a 32^d cube by various methods.

One might be tempted to draw a similar conclusion for QTT, but this example is by design unfavorable to such approach. Indeed, round-off error may become important due to the multiplication of SVDs compared with other methods while the total size of the problem remains quite small.

Two decomposition methods (ST-HOSVD and T-HOSVD) have been tested for the Tucker format. As expected, (see [28]), the results obtained are almost identical, in particular in terms of compression rate. Consequently, the only criterion left as to which one to use is computing time. As shown in [8], sequentially truncated decomposition is, on average, 4 times faster than truncated decomposition. Thus, in the following sections, the study focus on ST-HOSVD only for Tucker decomposition.

Meanwhile, HT decomposition accuracy is bounded by T-HOSVD by construction, but on this highly separable examples it can be seen that the core reduction follows an irregular pattern and may even increase CR for a given relative error as the core tensors remain small in these 3D cases.

Another, more challenging, synthetic data set is introduced for the 32^5 tensor case which is equivalent to 256MB. This test is more adequate to compare CPU times. We introduce a 5d function f_3 designed to be less separable.

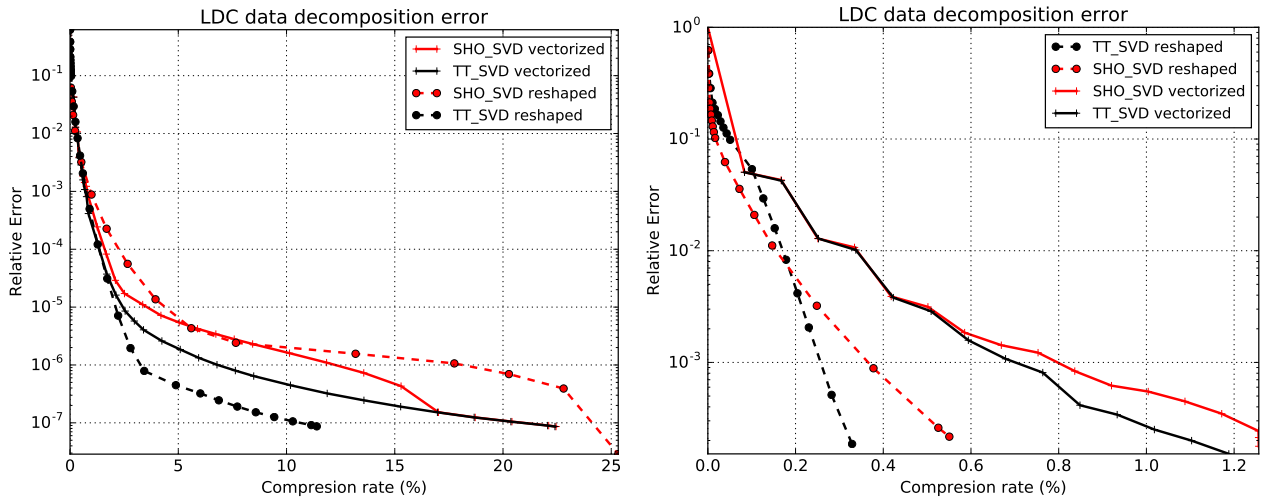
$$\begin{aligned} f_3(x_1,x_2,x_3,x_4,x_5) = {} & x_1^2\{\sin[5x_2\pi + 3\log(x_1^3 + x_2^2 + x_4^3 + x_3 + \pi^2)] - 1\}^2 \\ & + (x_1 + x_3 - 1)(2x_2 - x_3)(4x_5 - x_4)\cos[30(x_1 + x_3 + x_4 + x_5)] \\ & \log(6 + x_1^2x_2^2 + x_3^3) - 4x_1^2x_2x_5^3(-x_3 + 1)^{3/2} \end{aligned}$$

The lower separability of this case is not clearly apparent in Fig. 10. However, a different behavior is observed with ST-HOSVD achieving a much higher CR due to the large core tensor. The other three methods performs very similarly in terms of CR although a short-lived plateau is observed due to the slow convergence of the Tucker modes. On these simple examples all norms tested performed very similarly with little to no difference. Table 3 provides an overall view of the methods computational performances. It can be seen that the ranks to achieve this accuracy are high compared to the data size (but it would not increase for finer grids). A striking feature is the pyramidal shape of the QTT rank, increasing monotically till a maximum of 42 and then decreasing monotically (not shown). CPU times show large differences, TT and ST-HOSVD being by far the fastest while HT CPU time is essentially driven by T-HOSVD in the current implementation. As for the first example, it is not very appropriate for QTT use since n is small for a small d. The best comparative performance is achieved when one of the dimensions is very large.

	QTT	TT	ST-HOSVD	T-HOSVD	HT
CPU (s)	6.11	0.85	0.51	5.18	5.5
Memory (MB)	1237	882	807	1020	1220
rank	[2,4,8,16,18,30,42,42,33...]	[18,20,9,7]	[18,13,19,10,7]	[18,13,19,10,7]	(0,1):19, (2,3,4):19, (3,4):9, [18,13,19,10,7]
Compressed size	128K	157K	2.2M	2.2M	96K

Table 3: Metrics of decomposition methods for f_3 on a $32 \times 32 \times 32 \times 32 \times 32$ cube with a tolerance of 10^{-7} for each of these methods. Compressed data uses between 100 and 1000 times less space than the original tensor. Computed on Intel Xeon Gold 6230 chip with 40 cores, 256G RAM.

The goal of the following sections is to provide insight for efficient **data reduction and qualitative analysis** of their use. The first three experiments are restricted to relatively small dataset i.e. in the order of 1GB so that decomposition as well as postprocessing is reproducible on a laptop. For the sake of simplicity, we focus on the



(a) Cutoff tolerance of $\varepsilon = 10^{-8}$

(b) Cutoff tolerance of $\varepsilon = 10^{-4}$

Fig. 11: Lid Driven Cavity Simulation within the stable limit cycle time range, see [64,61], input tensor is of shape $6 \times 201 \times 66049$. $t = 1900$ to 1940 with a stepping of 0.2, space is a 257×257 grid that can be **vectorized** (solid lines) i.e. taken as a long vector of size 66049. Space, treated as 2 dimensions, is referred as **reshaped** (dashed lines). Reynolds is a parameter dimension with $Re \in [10000, 10100]$ and a stepping of 20.

pair ST-HOSVD/TT-SVD which is representative of general behavior of tensor reduction methods. Two dataset are scalar data of only one variable, that is to say $f : \mathbb{R}^d \longrightarrow \mathbb{R}$ as it is the simpler case, one from numerical and one from actual experiment. The third example addresses the multiple variables of the vectorial case i.e. some of the many ways to approximate the discretization of $f : \mathbb{R}^d \longrightarrow \mathbb{R}^p$.

Finally, a much larger (32GB) example embedding very complex physics is proposed and provides better insight for actual use case with data handling as a main focus. Due to the difference in scale, the four methods (HT, QTT, TTSVD and ST-HOSVD) are compared.

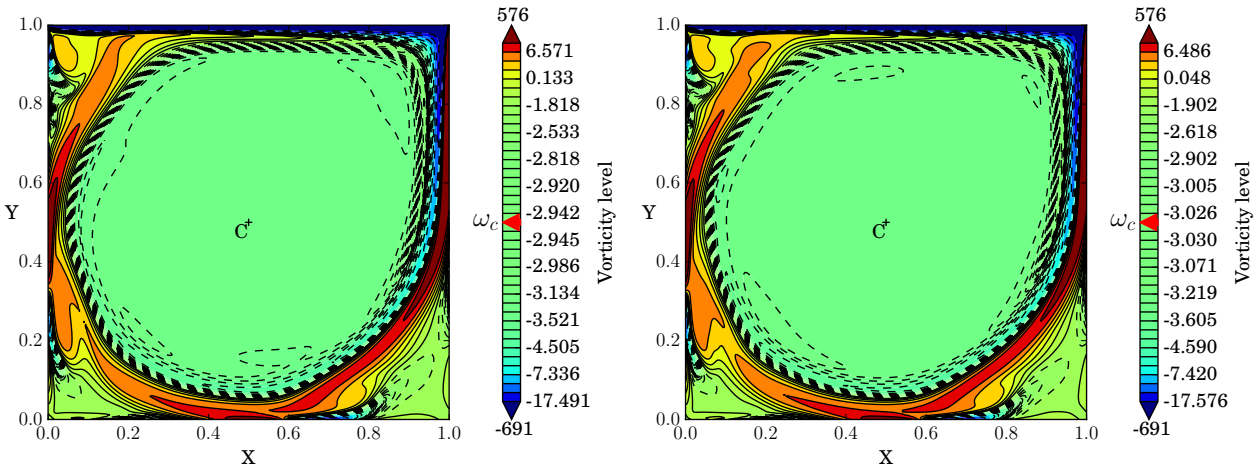
4.3 A scalar simulation: 2D lid driven cavity at high Reynolds number

A direct numerical simulation (DNS) of the 2D singular lid driven cavity problem in streamfunction-vorticity formulation with high accuracy (NCCD 6th order scheme, see [64] for implementation details of T.K. Sengupta code and analysis of the flow). High Reynolds numbers are studied, here we focus on range $Re \in [10000, 10100]$ with a spacing of 20. Time steps are very small, $dt = 10^{-3}$ thus snapshots sampling is coarser: $\delta t = 0.2$ in order to capture longer time series and especially limit cycles. To capture the flow behavior from initial quiescent state to the limit cycle, simulation must run from $t = 0$ to a few thousands which represent too many snapshots, indeed, each snapshots requires 0.5MB of memory thus leading to 5GB minimum per simulation). Consequently, for this analysis, a narrow range of the limit cycle is sampled from $t = 1900.2$ to 1940, leading to 200 snapshots per Re. Finally, a relatively coarse space grid of 257×257 is chosen for easier handling as we have shown that the number of modes is only weakly affected by grid density. In conclusion, after interfacing `pydecomp` with LDC code, an order 3 tensor $\mathcal{T}$ of shape $66049 \times 201 \times 6$ is obtained. In this case, space is given as a single dimension and is referred to as *vectorized*. As it is not clear whether it is preferable to decompose using this layout, the "fully" separated decomposition is also studied. In this case, space is seen as two separate dimensions leading to an order 4 tensor of shape $257 \times 257 \times 201 \times 6$ which is called *reshaped*. Both approaches are compared in Fig. 11. Note that the data is not preprocessed, i.e. no centering of trajectories is performed before applying the decomposition algorithms.

This data is highly separable, all four configurations reach machine precision with relatively low compression rates 10% to 25%. Indeed, both decomposition methods and both data layouts display exponential decay of the error as function of the compression rate. This is particularly visible when the error $E > 10^{-5}$ in the top graph Fig. 11a. For lower truncation one can see an abrupt change of slope which can be attributed to reaching "noisy" data. Indeed, this phenomenon is observed on most actual datasets.

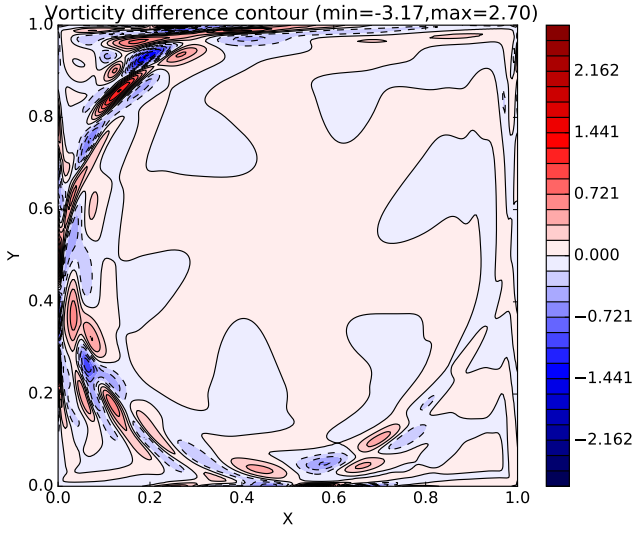
Next, one can observe that all four methods display comparable accuracy for moderate accuracy ($E \gtrapprox 10^{-3}$), which means that the choice must be driven by the goal of the decomposition. For optimal storage, one is advised to prefer TT-SVD for both layout although vectorized layout allows the user to reduce the truncation error by almost a decade. Finally, the latter offers, by far, the best CR=10% for maximum accuracy as compared to the roughly 20% of concurrent methods. Regarding ST-HOSVD, the observation regarding layout is the opposite of TT-SVD as compression efficiency is (slightly) reduced with reshaping.

Remark 1 (Handling of the space dimension) As shown for this example, the compression rate is weakly influenced by the space layout. This confirms the intuition that the amount of information contained in space does not depend



(a) Original data

(b) Reconstructed from ST-HOSVD, 1% error.



(c) Difference i.e. 12a- 12b

Fig. 12: Vorticity field of the lid driven cavity at Re=10000, t=1900s decomposition is reconstructed compared with 1% relative error in Frobenius norm i.e. rank=(10,10,3) and compared to original dataset. Isolines are plotted as well as colormap, they are exponentially spaced from the center of the square value, solid is superior to $\omega(C)$ while dashed lines are inferior. This is to make comparison with centered data.

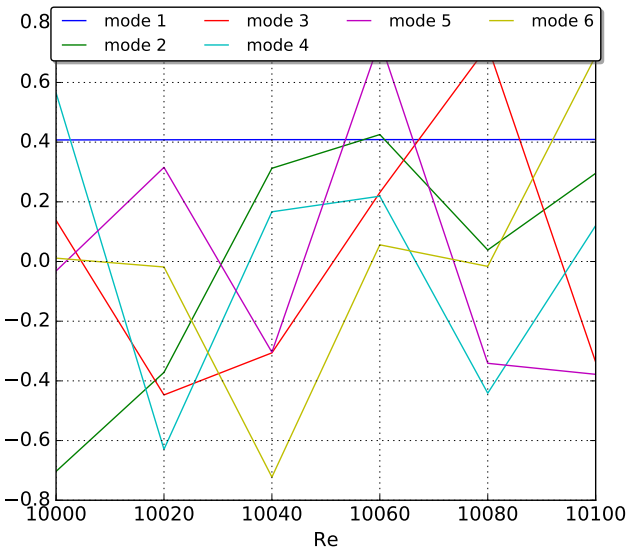
Table 4: LDC decomposition ranks with the same prescribed cutoff value $\epsilon = 10^{-4}$ (last point in Fig. 11b).

Data layout	Vectorized	Reshaped
ST-HOSVD	[15,18,6]	[59,63,18,6]
TT-SVD	[15,6]	[59,15,6,]

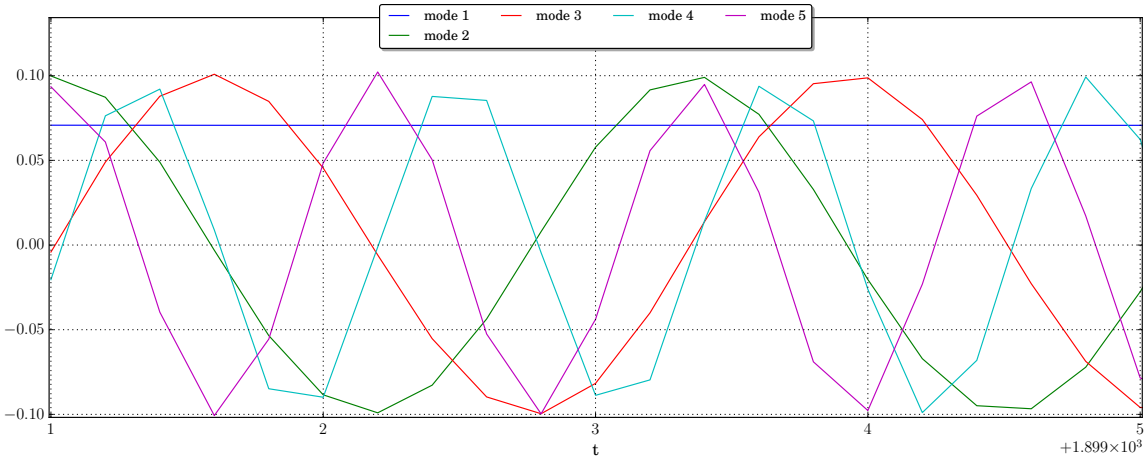
on its layout. However, we can see that it is not entirely true since differences appear early on, one can merely affirm that the qualitative separability of the field does not depend on the layout. In specific cases such as quasi 2D problems (not shown here, e.g. thin plate simulation), the third dimension *must* be separated as it represents a huge gain to treat it separately. Indeed, it can be seen as an identity function.

The rank, however, is drastically influenced by this choice as one can see in table 4. The same cutoff value of $\varepsilon = 10^{-4}$ has been used with each method and the truncation error is virtually the same. It is important to notice that in spite of the sequential nature of these methods the ranks of time and Re are unmoved by the layout choice. Yet, spatial decomposition rank is drastically changed, being multiplied 4 times for each one in ST-HOSVD. It is interesting to notice that only the first rank in TT-SVD is big, the second one remains the same as for the vectorized layout. It can be interpreted that the space spanned by space dimensions 1 and 2 (embedded at the second stage of the algorithm) remains the same no matter the layout thus leading the same value of 15.

Now, we shift our attention to Fig. 11b. In this case, we are interested in the ability of these methods to compress the data with moderate accuracy. The superiority of the reshaped representation of space is blatant as it proposes a much finer range of compression since many approximation levels are to be found for $CR \leq 0.1$ while



(a) First 6 Reynolds modes for $Re \in [10000, 10100]$



(b) First 5 modes closeup on $t \in [1900, 1905]$

Fig. 13: Time and Reynolds modes of lid driven cavity during limit cycle ($t \in [1900, 1905]$) in for $Re \in [10000, 10100]$.

the storage requirement is halved compared to *vectorized*. This phenomenon in enforced by relatively low spatial ranks as compared with n_x and n_y (see Table 4). In terms of compression power, this largely overcomes the highly intertwined nature of both space axes i.e. the rich flow behavior lies in complex 2D structure that in spite of not being represented well in the reshaped layout is overcome by rank truncation.

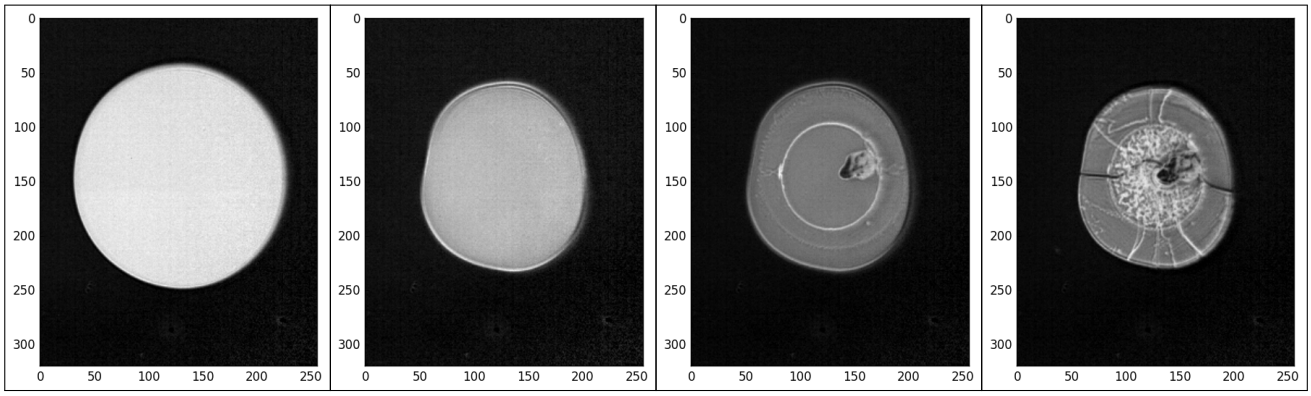
Fig. 12 shows that even with moderate accuracy of 1% error the reconstructed data is largely usable for qualitative analysis. This is very interesting for long term storage as the required space for this dataset is reduced to 0.2% of the original 634MB i.e. 1.2MB. Fig. 12a shows the original vorticity field of $Re = 10000$ at $t = 1900$ while Fig. 12b proposes the same field from reconstructed ST-HOSVD in the vectorized layout and Fig. 12c is the difference between these two fields. One can see that the structures are well captured as well as the minimum and maximum value. The central region vorticity level is off by a few percent. However, the lower amplitude structures are captured with less accuracy. Finally, the difference map shows that locally, the relative error remains small. As one would expect, most of the error is contained in large gradient regions near the boundaries of the domain.

Very limited physical hindsight can be drawn by observing "reshaped" space modes along X and Y which is why they are not shown here. The central region is mostly flat with varying mean values while the extremities of the domain show large spikes and modes Y present small scale oscillations in addition to larger structures near $y = 1$. The "vectorized" modes (not shown) are similar to the one given in space time decompositions in [61] and can be used for physical hindsight.

Finally, in order to acquire a better grasp of the decomposition obtained, Fig. 13 shows the first modes associated with Re and time. In both cases, the first mode plays a special role of virtually applying a constant offset, it can be referred as a *mean mode*. Indeed, this kind of mode is observed whenever the data has not been centered beforehand, the decomposition "naturally" separate the mean field from the fluctuations. A simple averaging of the data suppresses it and it is often advocated to do so in the literature as it should improve the decomposition. Next, Fig. 13b displays well organized modes, these pairs of modes (2-3, 4-5) are separated by a phase shift of

$\pi/4$ and the frequency of pair 2 is double the frequency of pair 1. This pattern is studied in greater details in [61], yet it is interesting to note that the same pattern is observed for multivariate decomposition involving Re as a parameter as well as usual bivariate POD. It is then possible to infer that the time behavior is the same for each Re in the chosen range. At the other end of the regularity spectrum, one finds Re associated modes in Fig. 13a. These modes appear to be a mean to exclude each other from combinations, no clear pattern emerges. This observation indicates low feasibility for Re based interpolated ROM.

4.4 Experimental data: droplets evaporation



(a) t= 1/29 (b) t= 23/29 (c) t= 24/29 (d) t= 29/29

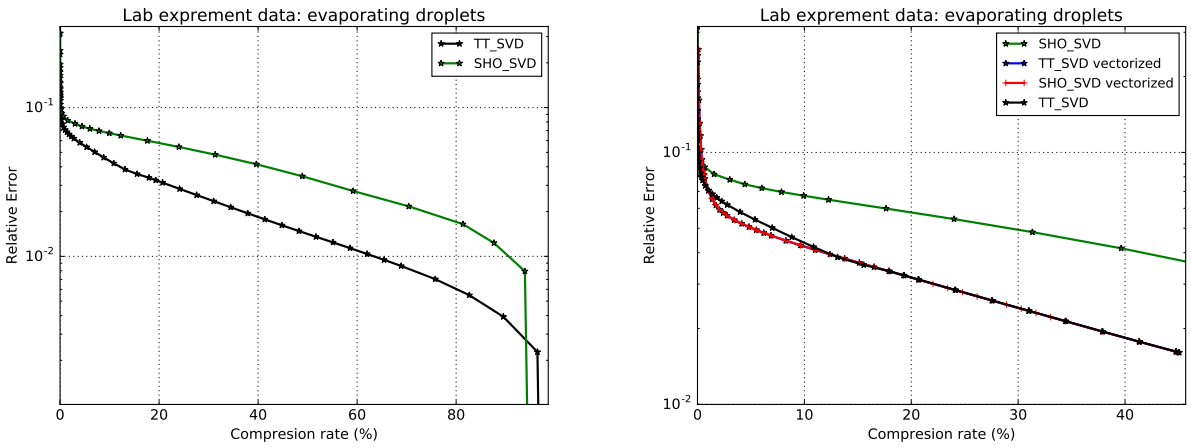
Fig. 14: Visualization of 4 snapshots of the density field at the 21$^{\text{st}}$ tabulated wavelength. Data kindly provided by C. Pradère (I2M Bordeaux).

In this second example, a scalar field obtained by a lab experiment is studied. The goal here is to emphasize that very little prior knowledge of the data is necessary to perform decomposition contrary to analysis. This dataset was kindly provided by C. Pradère, from I2M-TREFLE laboratory. It is a study of droplets evaporation during 29 timesteps with recording at 51 different wavelengths to evaluate the density field. The camera resolution is 320×356, no further detail on the technology used is required. Finally, a matlab "`.mat`" binary file off 800MB is given. `pydecomp` provides a simple interface for inputting such files that yields a $29 \times 51 \times 320 \times 356$ tensor.

Phenomenon Fig. 14 provides insight on the phenomenon studied, the circular drop at initial time evaporates and shrinks gradually up to frame 23. Cracks appear at t=24 (different wavelength may not show these cracks) and the droplet is completely shattered at t=29. One may infer that the droplet has solidified but this information (not given by C. Pradère) is not necessary for data decomposition.

The data is obtained experimentally and it is likely that many physical phenomena are happening simultaneously during the experiment. Then, one has to assess the separability of the array. In the absence of any information about the parameter spaces at stake, the Frobenius norm based decomposition is used. Fig. 15 shows that with both ST-HOSVD and TT-SVD, very little compression is achieved. Indeed, Fig. 15a shows that more than 60% compression rate is needed to reach a relative error of 1%. Yet, one can see that the error drops (actually down to machine error) with a compression rate slightly below 100% which means that the density field is represented "*exactly*" with a slight datasize reduction. Fig. 15b shows that attempts at vectorizing data provide no improvement in the error decay rates. This zoomed in view, informs us that a reduction to a few percent of error is attained within a few modes. Thus, some of the behavior is separable. But the complexity of this phenomenon lies in nonlinear physics, such as transport or phase change, that are known to cause poor error decay of the SVD/POD.

Finally, Fig. 16 provides a synoptic view of the ST-HOSVD decompositions with a prescribed error of 10^{-2} in both vectorized and reshaped layout. This means an actual error of 6% for the *vectorized* layout with a compression rate of 1.5%. The *separated* space global error is 9% for a compression rate of 0.3%. This partial choice of low accuracy high compression is aimed at showing that this kind of representation is sufficient for qualitative analysis. First, despite high global error level, the sequence of droplet evaporation is well captured by both methods, the crack appears at the expected frame in each decomposition. The main difference between the two layout lies in the sharpness of the spatial representation, indeed the vectorized approach produces a sharp edged representation while the separated space dimension lead to a "blurry" phenomenon. This is confirmed by the bottom frames, in which one clearly sees that the error is located at high density gradient regions. In conclusion, vectorized layout produces less efficient decomposition but allows for a sharp and easy to interpret reconstructed field while the separated space dimensions yields a blurry image, yet with lower global error.



(a) Space as 2 dimensions, tensor shape: $29 \times 51 \times 320 \times 356$.

(b) Space is vectorized, tensor shape: $29 \times 51 \times 81920$.

Fig. 15: Decomposition of experimental data kindly provided by C. Pradère (I2M Bordeaux). The density is given as a function of time, wavelength and space

4.5 A vectorial simulation: breaking wave

Here, we study a 2D simulation of a breaking wave which provides 5 output variables: density, pressure, vorticity, velocity along each dimension. It is not intended to be a state-of-the-art breaking wave physics simulation, the goal here is to provide a complex physics two phases flow computed with a validated HPC code: `notus-cfd`.

The Navier Stokes equation with two fluids is solved thanks to a level set methods with a velocity-pressure scheme. The spatial domain $\Omega = [0, 0.6] \times [0, 0.6]$ is discretized on a 256×256 Cartesian grid, while the time is solved with small times steps which are sampled in 201 equispaced snapshots. The third parameter is the ratio wave height over wave length, the latter being fixed for the whole set of simulation to 10cm, 3 heights are given: 9, 10 and 11 cm. In each case, the boundary conditions are periodic and the velocity field is initiated with an adapted velocity. Finally, the density field is equal to 1000 in the liquid phase and 1 in the gas phase. For stability reasons, the transition is smoothed on a few cells. Simulation with wave height of 9 cm is provided in Fig. 17 where one can see four typical snapshots of the breaking wave.

Data layout The previous examples have shown that in spite of providing sharper spatial description, a vectorized space is not the most efficient configuration in terms of storage cost. Additionally, the physics of the studied problem clearly has two separate domains, air and water which remain in the same region with respect to coordinate Y. Only a small portion of the Y range is affected by phase change. For these reasons, a space separated layout is used. Furthermore, this dataset provides 5 different output fields which are correlated since they solve the same Navier-Stokes equation system. But, they possess very different mathematical properties, for instance, density field is representing as sharply as possible an inherently discontinuous field whereas the pressure field is naturally smooth and continuous despite following the same interface. The velocity field is represented by two scalar values but has been solved at the same time. Finally, the vorticity field is post-processed from velocity but the field itself is much sharper due to the rotational operator, thus making decomposition less efficient. In conclusion, two data layouts are studied, both with separated X and Y axes.

a. Output data for each variable are processed sequentially. Five order 4 tensor of shape $3 \times 201 \times 256 \times 256$ are decomposed.
b. Output data for each variable is assembled into a new dimension that intends to account for embedded correlation among variables. One order 5 tensor of shape $5 \times 3 \times 201 \times 256 \times 256$ is decomposed.

Scalar product As for any decomposition problem, choosing the base scalar product and associated norm is thought carefully. In this case, two parameters, output vector in case b. and wave height, impose the use of l^2 scalar product. Thus, SVD based decomposition is preferred with TT-SVD and ST-HOSVD.

Low rank approximation analysis Fig. 18 provides the error versus compression rate graphs for layouts a. and b., we focus first on the top frames. Separability of the dataset with layout b. sits in the separable range. Indeed, a sharp decay of the error is observed for large scale evolution i.e. for error levels down to $E = \mathcal{O}(10^{-2})$. Then, a clear inflection is observer around 0.5% compression rate for both methods. Yet, it still appears that the error decay follow an exponential trail. Note that ST-HOSVD yields the best approximation at low compression levels (see Fig. 18d) and represents to machine error the data with a compression rate of 60% as seen in Fig. 18c. No such convergence is observed for TT-SVD.

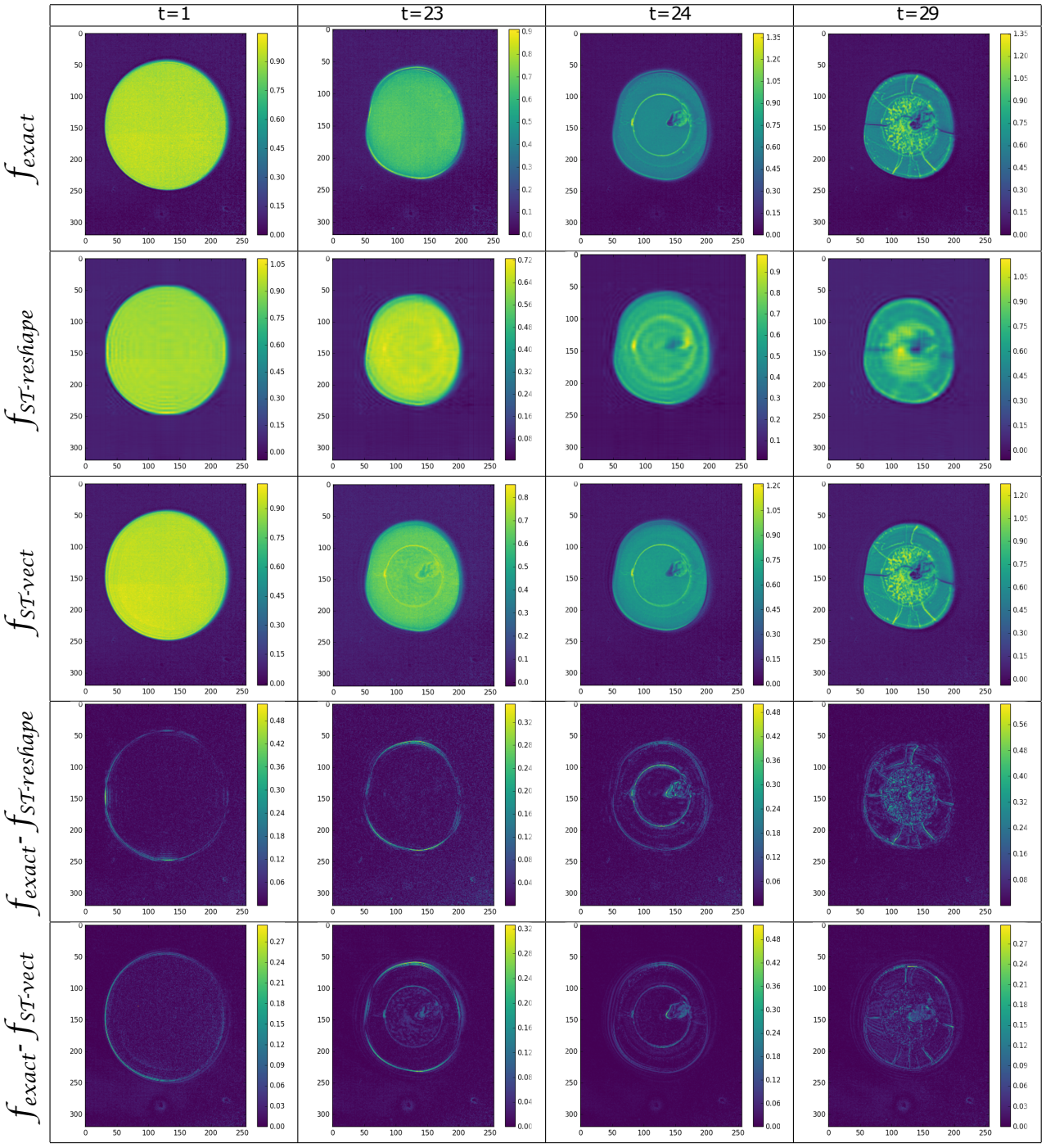


Fig. 16: Synoptic view of reconstructed ST-HOSVD decompositions of the density field, tolerance, $\varepsilon = 10^{-2}$, wavelength 21. Each line represent a different dataset, from top to bottom: f_{exact}, $f_{\text{ST,reshape}}$, $f_{\text{ST,vectorized}}$, difference ($f_{\text{exact}} - f_{\text{ST,reshape}}$) and difference ($f_{\text{exact}} - f_{\text{ST,vectorized}}$).

Regarding separate decompositions of variables through ST-HOSVD –lower frames of Fig. 18– it is observed in Fig. 18c that every single variable is represented to machine error within 50% of the original data size (per variable). It is uncommon for complex simulation data to present an "exact" tucker rank. Still, here, for each variable, machine error is reached for a tucker rank of $r = (3, 201, \approx 130, 256)$. Next, for small truncation error levels, all variable decrease at the same slope, only the extent of the initial drop varies. Fig. 18d provides a bigger truncation criterion in order to better grasp the moderate accuracy decomposition. Large differences between variables are observed, with pressure field being extremely separable while the vorticity field occupies the other end of the spectrum. Table 5 emphasizes the great variation of ranks among variable for an identical tolerance.

In conclusion, if one is interested specifically in an "easily" separable field, then the best choice is to treat variables separately. On the other hand, when interested in several variables, it is a better option to compress all the data together.

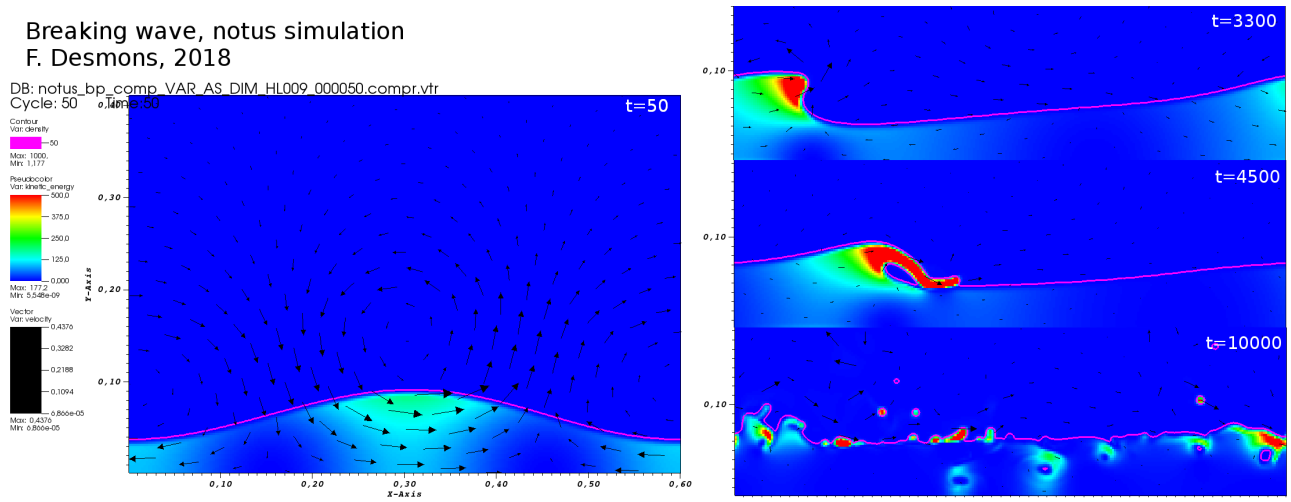
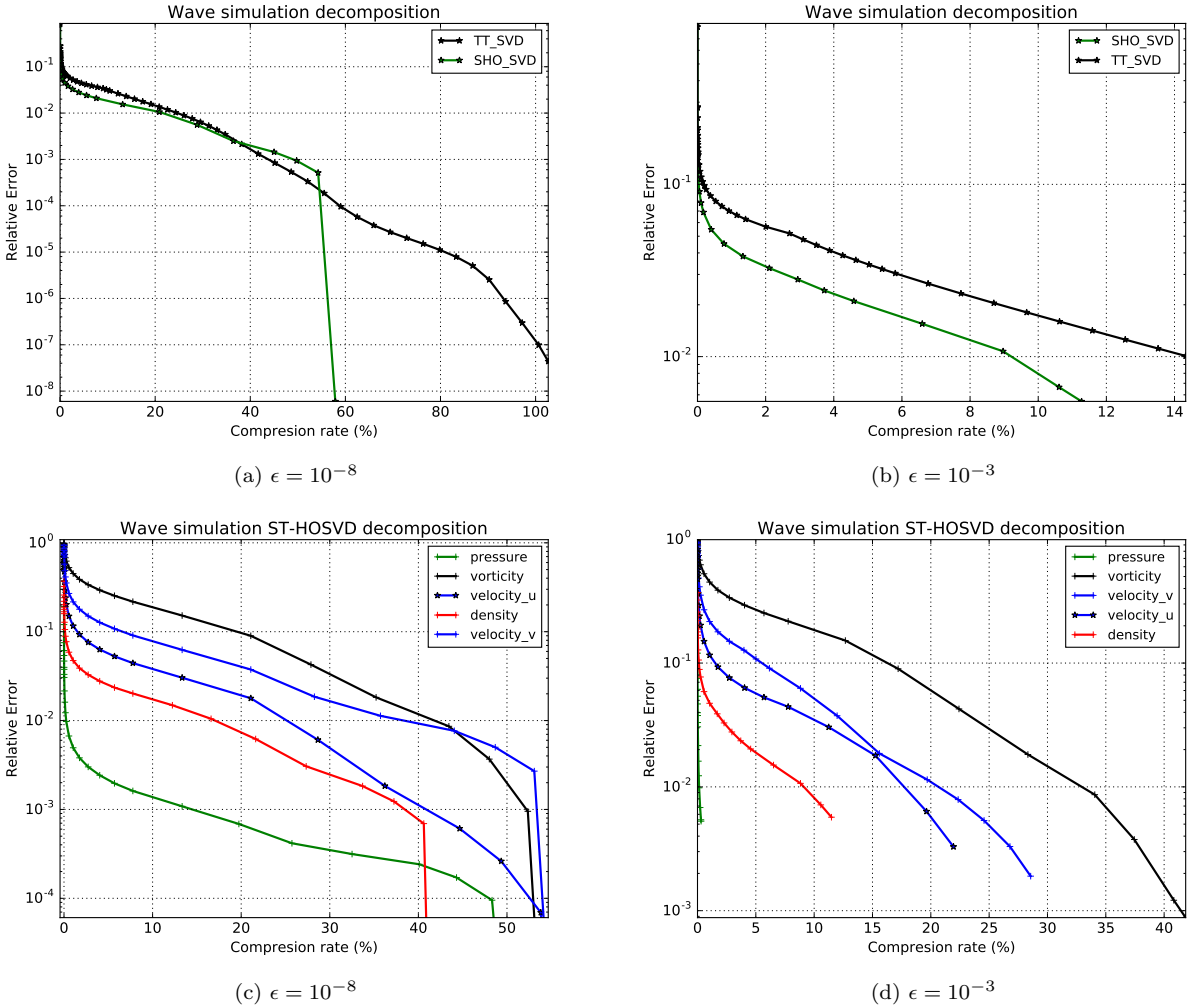


Fig. 17: Breaking wave simulation (by F. Desmons) computed with `notus CFD` code developed at I2M Bordeaux, wave height of 9cm and length of 10cm. The wave is going rightward from the initial state (left frame), crosses the periodic boundary (top right), breaks at $t \approx 4500$ follows to an unphysical chaotic state. Pink lines represent the water/air interface, arrows size are proportional to the velocity amplitude and the colormap accounts for kinetic energy.



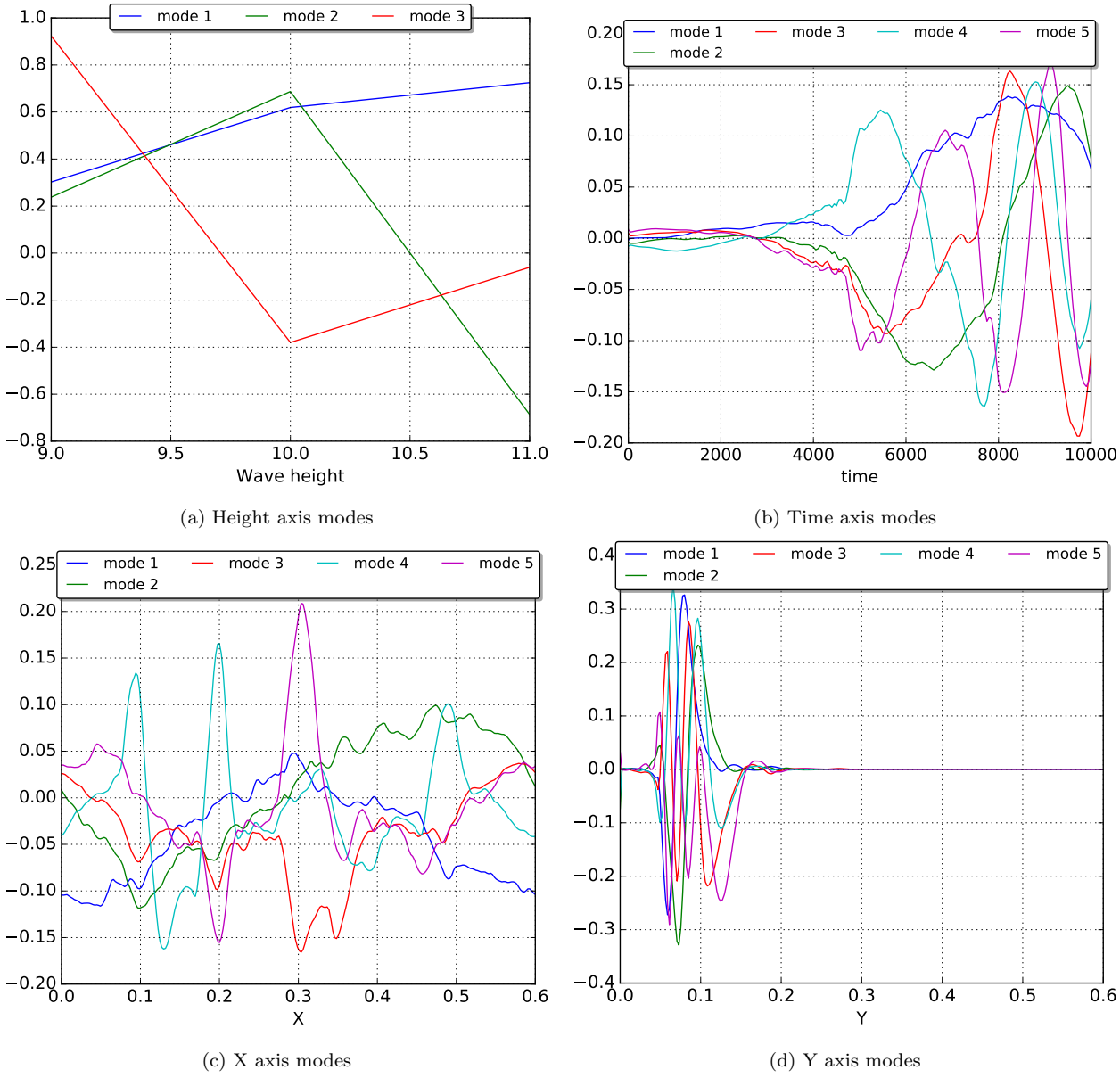
(a) $\epsilon = 10^{-8}$ (b) $\epsilon = 10^{-3}$

(c) $\epsilon = 10^{-8}$ (d) $\epsilon = 10^{-3}$

Fig. 18: Compression of breaking wave simulation data from `notus`. Parameters: 5 output variables, 3 wave heights, $n_t = 201$, $n_x = 256$, $n_x = 256$. Top frames are decomposition with output variable taken as an additional dimension with $n = 5$ (case b.), bottom frames is the same dataset but each variable is seen as a separate scalar decomposition problem (case a.).

Field	Rank
density	[3,174,58,147]
pressure	[3,52,14,44]
velocity_u	[3,184,79,256]
velocity_v	[3,179,101,158]
vorticity	[3,195,114,246]

Table 5: Breaking wave approximation ST-HOSVD ranks with prescribed cutoff value $\varepsilon = 10^{-3}$ (last point in Fig. 18d).



(a) Height axis modes

(b) Time axis modes

(c) X axis modes

(d) Y axis modes

Fig. 19: The first vorticity modes for separated variables layout.

Remark on graphs discrepancies One may notice that these graphs are not identical, this is because the truncation value ε is applied to the ST-HOSVD itself i.e. to each SVD. This leads to some mode combination to disappear from the larger ε although the actual projection norm is of the same order as ε. For instance, pretend that $\varepsilon = 10^{-3}$ yields a rank (3,7,27,35), there is no warranty that modes (3,**8**,27,32) from the full rank decomposition is associated with a weight $\omega_{3,8,27,32} < \varepsilon$.

Breaking wave vorticity modes The physics at stake in this example is different from the previous ones, then we look at the first five modes of each dimension (see Fig. 19) for the vorticity field. The top left frame, Fig. 19a, shows the modes associated with the initial height of the wave. No clear pattern is distinguishable and the sharp variation mostly indicates that they would be better considered as discrimination function rather than modes in the usual sense. Consequently, there is very little prospect for interpolated ROM on this parameter when the user

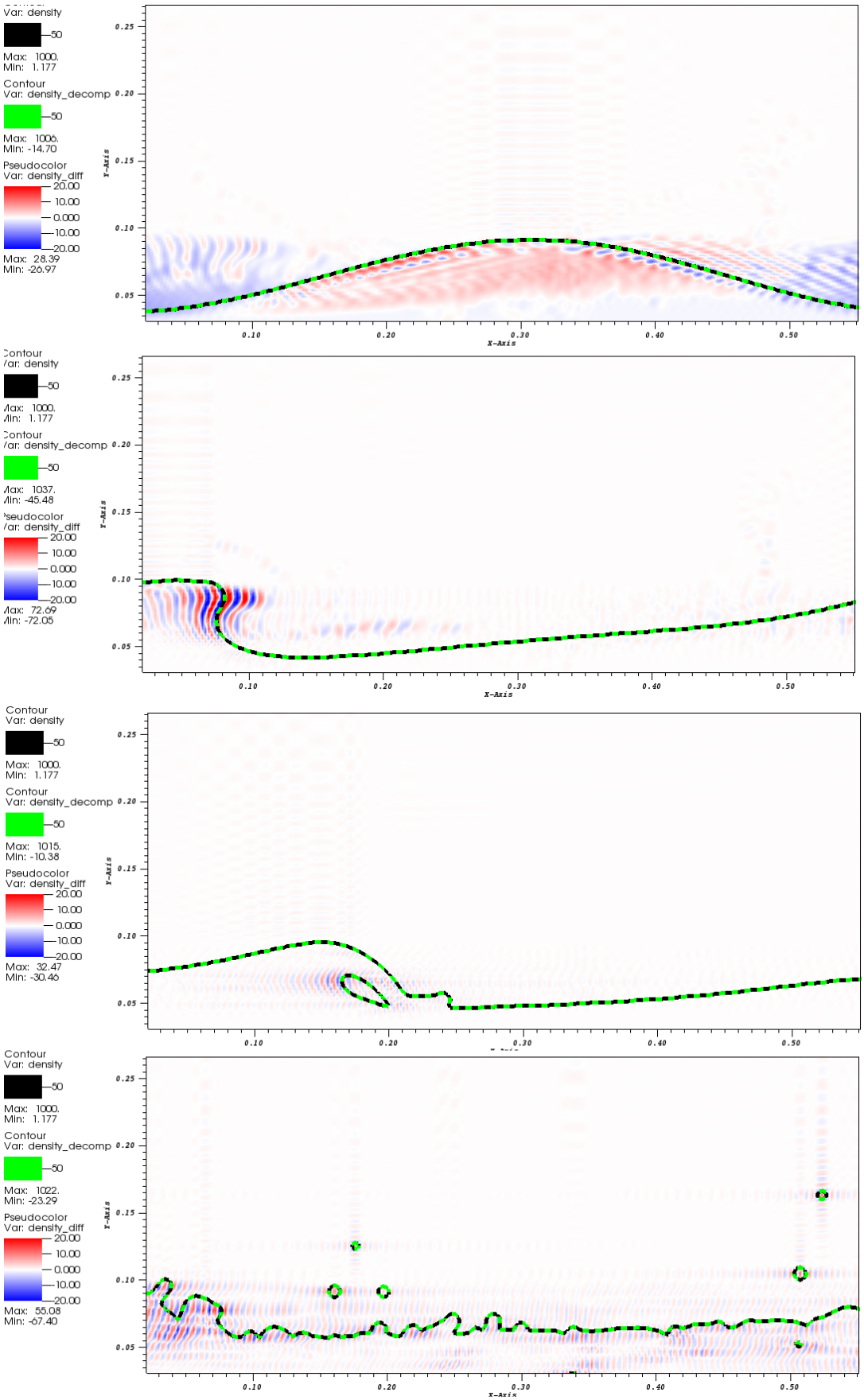


Fig. 20: Levelset 50 of the reconstructed density field at 4 time steps (same as Fig. 17) with the difference field between the original and reconstructed data.

Field	Size	Physical domain
Laser scan speed	4	[0.7:1.1:0.1] (m/s)
Laser Power	8	[255:325:10] (W)
Fields	8	3 phases, T, grain num. & orientation
time (snapshots)	64	every 1000 steps
nx	64	5μm
ny	64	5μm
nz	64 subsampled from 128	5μm

Table 6: Summary of input tensor from powder bed fusion simulation batch. Total binary input size is $4 \times 8 \times 8 \times 64 \times 64 \times 64 \times 64 \approx 32$GB.

has only 3 instances available. Times modes (Fig. 19b) can be interpreted as being activated by the breaking of the wave (time range is approximately [3300,4500]) and further agitation. As expected, space modes along dimensions X and Y produce remarkably contrasted patterns. On one hand, X modes describe global agitation with distinct patterns at impact ($x = 0.2$) and splash region ($x = 0.3$). On the other hand, Y modes show an intense activity near the interface and close to 0 value elsewhere. The same pattern is observed for other variables (not shown) but vorticity provides the most readable graphs.

Reconstructed fields Finally, a quick overview of the reconstruction is given by means of the density field and levelset reconstruction. Indeed, this is a very sensitive variable and it is required to correctly capture the interface for any interpretation of the stored results. Fig. 20 shows the same snapshots as Fig. 17 where the black solid line is the original isoline 50 of the density field and the green dotted line is its reconstructed counterpart from ST-HOSVD($\varepsilon = 10^{-3}$). The background color maps the difference between both density fields. In spite of marked error field, the reconstructed levelset fits perfectly with the original one, no bubble is omitted and the shapes are well captured. Still, some parts of the density field are negative (intense blue color in the air corresponds to $\rho < -20$). This is obviously non-physical and this issue should be addressed in order to prevent misinterpretation for cases in which the analysis is more complicated.

It should be noted that with this precision of $\varepsilon = 10^{-3}$, it is almost *impossible to distinguish* the reconstructed field from the original mode. Some slight oscillations may be spotted but are easily discarded by the observer as their amplitude is a few percent of the maximum field value.

4.6 Decomposition of a large simulation dataset: powder bed fusion additive manufacturing

In this last numerical experiment, we study the performance of the four most promising methods (HT, QTT, ST-HOSVD and TTSVD) on a larger data set. The data is generated thanks to the digital twin for the Additive Manufacturing Platform developed at A*STAR Institute of High Performace Computing, Singapore (see [65]). This tool provides end to end simulation of metals 3d printing. It generates high amounts of data, in particular the powder bed fusion (PBF) module that computes the micro-scale phenomena of a laser melting a metal powder. This process happens extremely fast at very small scales. Typical lengthscales are tens of microns and time scales range from 10ns to 1ms for the whole simulation. Consequently, the simulation requires millions of time steps in order to complete several layers of printing. A regular Cartesian grid is used to discretize space which incurs large number of degrees of freedom ($O(10^6)$). The simulation outputs many variables of interest including temperature, phase fields for solid, liquid and gas, the fluid dynamics fields and the grain microstructure information (index and 3d orientation). Altogether it can be seen as 11 scalar fields that are more or less correlated with a few millions of degrees of freedom for each snapshot. A few snapshots of the simulation outputs are presented in Fig. 21 where one can see the powder (colored by grain orientation) melts creating a meltpool (green surface) on the laser path and then solidifying until it forms a solid bloc. The complete process involves adding a new layer of powder and scan with the laser until the block reaches a sufficient size for analysis.

Obviously, such a complex process can be modified by a lot of parameters, ranging from material properties to temperature and laser properties. Since laser properties is the most influential parameter on the grain structure (and ultimately the mechanical properties of the printed object), the laser power (LP) and the laser scan speed (LS) are varied in this experiment respectively with 8 and 4 samples equally spaced. Altogether the studied dataset is summarized in Tab. 6. All numerical examples performed in this last section are done on standardized data, which means that for every simulation each field is standardized (centered and normalized) i.e. $\tilde{u} = \frac{u - \text{mean}(u)}{\text{stddev}(u)}$. This is often cited as a way to improve decomposition and has proved efficient at improving the decompositions for this dataset.

Before tackling a large dataset, it is important to have a good idea of how well the potential methods perform. Consequently, three tests, similar to situations studied in the previous sections are shown for a single set of parameters LP=255W and LS=0.7m/s.

- Case 1 tests the easiest configuration where only the temperature field is studied, leading to an order 4 tensor of size $64 \times 64 \times 64 \times 64$.
- Case 2 tests the influence of separating the fields as well, i.e. and order 5 tensor of size $8 \times 64 \times 64 \times 64 \times 64$.

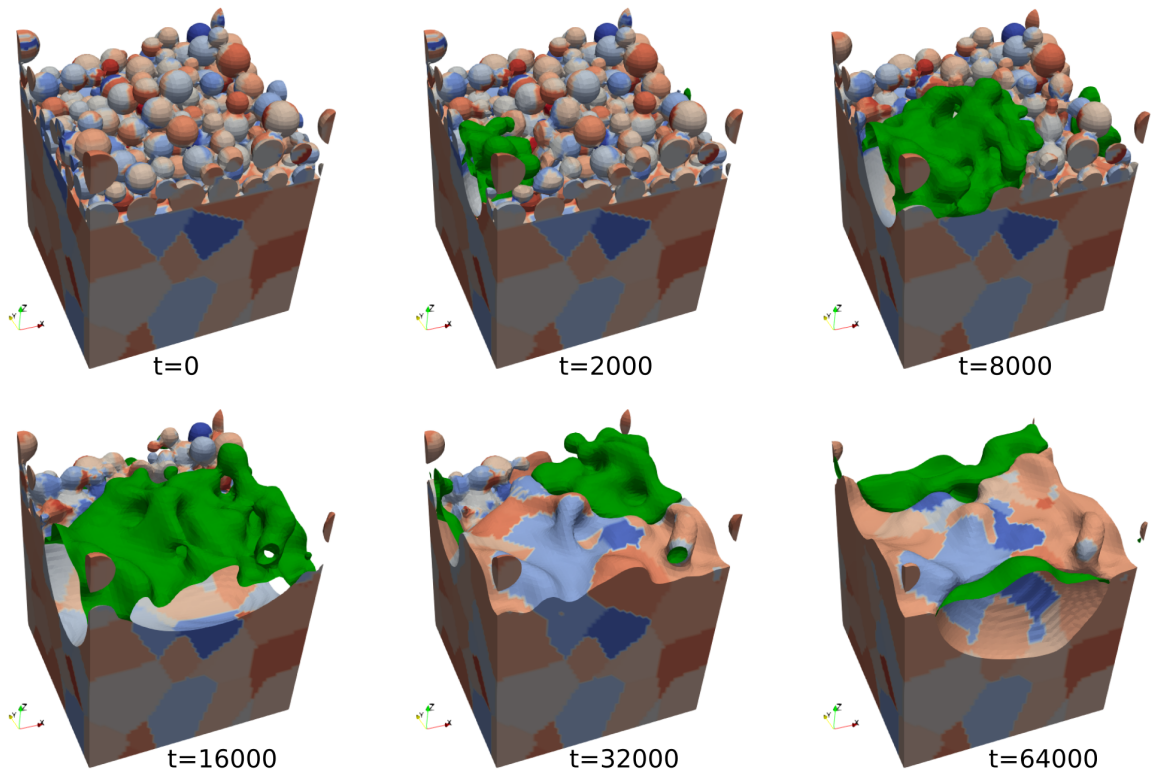
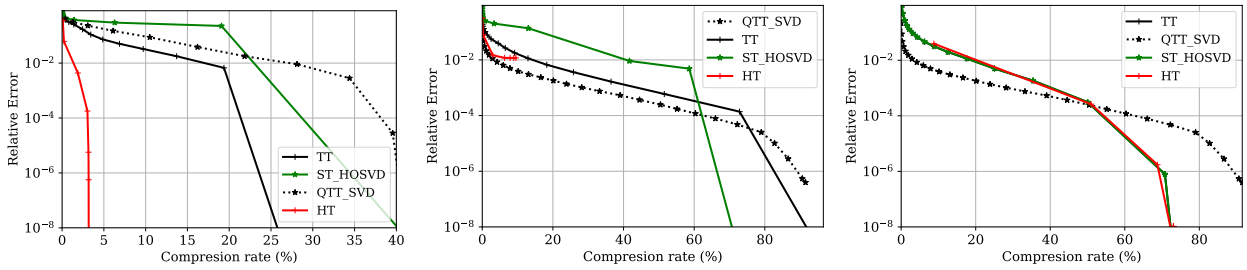


Fig. 21: First layer of the powder bed selective laser melting (SLM) simulation using A*star Additive manufacturing simulation on a $64 \times 64 \times 128$ domain. The solid part is colored using the grain orientation magnitude and the green surface is the boundary of the fluid phase. The other variables are not shown. On these snapshots, 3 different scans are seen, the first one till t=16000 and one two different scans for t=32000 and t=64000.

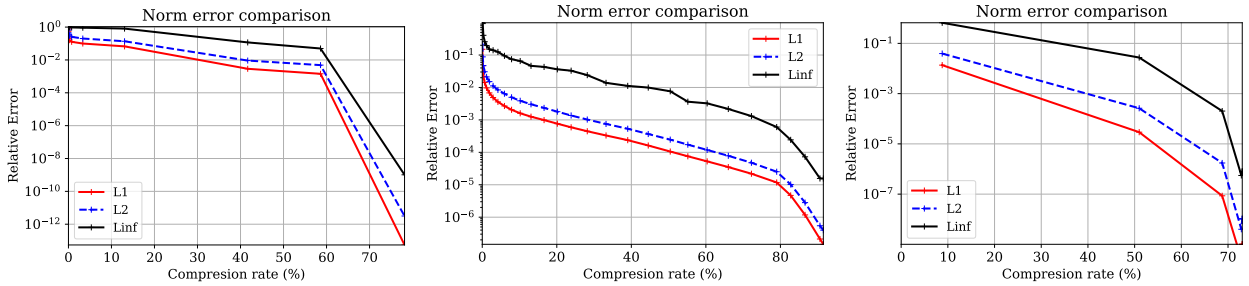


(a) Case 1, $64 \times 64 \times 64 \times 64$ (b) Case 2, $8 \times 64 \times 64 \times 64 \times 64$ (c) Case 3, $8 \times 64 \times 262144$

Fig. 22: Preliminary compression rate and associated L^2 error test showing 3 different data layout imply very different CR for the same underlying data.

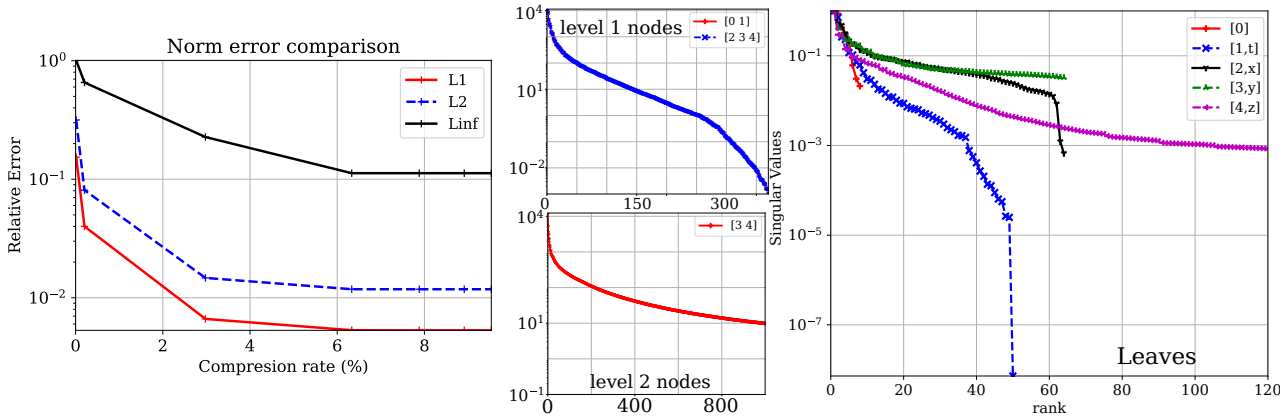
– Case 3 shows that representing space as a single variable is favorable for good spatial capture of the phenomenon, in particular with Tucker format.

Fig. 22 shows the decomposition error and CR for the three cases. Due to the larger size of the core in case 1 and 2, ST-HOSVD performs poorly until it reaches a threshold where all the variance is captured. Typically, that means that the core tensor i.e. the rank, is only slightly smaller than the initial dataset. In this case it's the time rank or the first space dimension that is slightly smaller than 64. This explains why the error drops sharply to machine error. The situation is even more pronounced when it comes to L^∞ norm as shown in Fig. 23. Specifically, one can see in Fig. 23a that the error remains above 0.1, barely affected by the number of modes kept until the rank threshold is met. A Similar pattern is observed for all methods in cases 2 as shown in Fig. 23b for QTT despite a more regular decrease of the error. Last, Fig. 23c shows that for ST-HOSVD, in case 3, the L^∞ error can be as much as two orders of magnitude larger than L^2. This can be particularly troublesome when trying to reconstruct grain structure or melted regions in this kind of simulation. The large amplitude oscillations are similar to the level set issue described in the previous section for Fig. 20. Visualization of these 3D fields is delicate, especially for the moderate resolutions studied here which usually means no visual difference between the fields if the L^2 error is below 1%. For this reason, no reconstruction is shown in this section.



(a) Case 2, ST-HOSVD (b) Case 2, QTT-SVD (c) Case 3, ST-HOSVD

Fig. 23: Comparing error measured with norms L^1, L^2 and L^∞ at the same truncation rank for poorly separable cases. Large values of norm L^∞ show that local approximation error may be very large even if the average error is low.



(a) Error vs CR for different norms (b) Nodewise Singular values

Fig. 24: HT compression analysis for case 2. Tree shape is identical to Fig. 4 but indices are numbered from 0.

Remarks on QTT. As expected QTT compares rather poorly in case 1 since the shape of the dimensions of the tensor are relatively small and even. However, it offers the best performance for both case 2 and 3 for low CR. It is only superseded by the "drop" to machine error of the other methods. As expected, it is not influenced by the initial shape of the tensor since the data is folded to quanta $q = 2$. Interestingly, QTT decomposition achieves low L^∞ error together with a possibility for controlling finely the reduction error. This makes the method very suited for complex data compression. It should be noted that for cases that require resizing data, QTT overall performance is similar albeit slightly poorer.

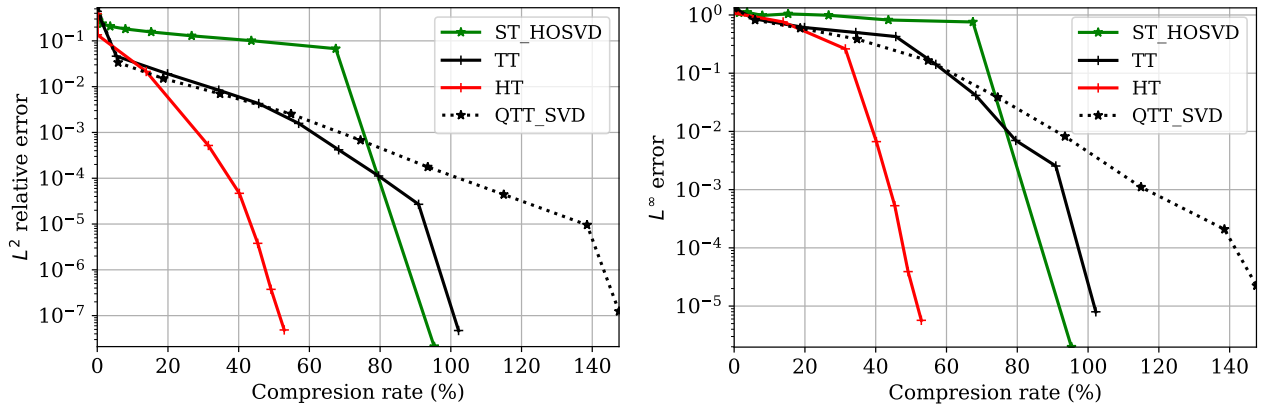
Focus on HT decomposition. This more demanding dataset offers all the behaviors encountered during the extended testing of this method. First, the overall accuracy of HT is bounded by the initial Tucker decomposition, hence prerequisite for HT success is satisfying accuracy of the HOSVD. It is fulfilled for these examples but requires mostly full rank, with at best, one dimension actually showing compression. Then HT compresses the core tensor, and may, as in case 1, achieve high compression compared with Tucker. In Fig. 22a, one can observe the almost vertical drop of the error for virtually constant CR. It means that most of the hierarchical tensor size is made of the leaves modes while the core is compressed efficiently. The flatter decay part ($E = 10^{-2}$ to $E = 10^{-4}$) is attributed to the poor separability of the data as observed for ST-HOSVD. Case 3 shows that for small size core, HT doesn't noticeably improve compression compared with ST-HOSVD. Finally, case 2 exposes a common flow of HT as shown in Fig. 24. Due to poor conditioning, it is difficult to obtain good singular values. Their decay is usually slow (see Fig. 24b) as shown in and both iterative and direct solvers may return negative singular values (small norm). This, despite requesting machine error tolerance, prevents the HT decomposition from properly approximating the data. Hence, one should expect this kind of difficulty when working on high complexity data as we do on this final experiment.

These preliminary tests confirm that the data provided here is separable but challenging. Thus, it is expected that an increased dimensionality and consequently of the size of the tensor may reduce the separability, in particular for methods sensitive to bad conditioning such as QTT.

Results. Table 7 provides a synoptic view of the four decomposition methods applied to the large 32GB tensor. One can see that again, the main limiting factor is memory, with all methods requiring a peak memory above 100GB (QTT), all the way up to 212GB for HT which is the most memory intensive approach. This whopping factor of

Method	Walltime(speedup)	Peak Mem.	rank	L2 error	CR	CR^2	CR^∞
QTT	3h15min (0.56)	100G	$[\{2^n\}_{n=1}^{12}, 8169, 15695, 28945, 29591, \{2^n\}_{n=15}^{1}]$	1.24e-07	148%	34%	93%
HT	22min15s(0.38)	212G	{012}:256, {3456}:256 {12}:63, {23}:2166, {34}:4096 leaves:[4, 8, 8, 64, 62, 64, 64]	4.86e-08	53%	≈20%	34%
TT-SVD	12min27s (0.64)	140G	[4, 32, 256, 15698, 4096, 64]	4.72e-8	102%	34%	80%
ST-HOSVD	4min11s (0.35)	162G	[4, 8, 8, 64, 61, 64, 64]	2.10e-8	95%	95%	95%

Table 7: Decomposition methods metrics for large data, Total binary input size is $4\times8\times8\times64\times64\times64\times64 \approx 32$GB. Tolerance for singular values is $\varepsilon = 10^{-8}$, corresponding to machine error for the intermediate EVD. Computed on a Intel Xeon Gold 6230 chip with 40 cores, 256G RAM. CR^2 and CR^∞ are the CR corresponding to a 1% error in L^2 and L^∞ norms respectively.



(a) L^2 (b) L^∞

Fig. 25: Comparing error in two norms versus CR for 4 decomposition methods of large tensor data ($4 \times 8 \times 8 \times 64 \times 64 \times 64 \times 64$). This standardized data is described in Tab. 6.

6 compared with the input data can be improved by better implementation in python (and its clumsy memory control). Nonetheless, the algorithm most expensive step is a matrix-matrix product that requires memory twice the size of the input data if performed in-core. One also has to store a very large tucker tensor at the first step. These considerations are addressed in specific packages. CPU walltime varies wildly as well from a few minutes for HOSVD, TT and HT to a few hours for QTT. This is easily explainable as, in QTT, rank has many extremely large values since almost no singular vector is cutoff for small tolerances (lower than $\varepsilon = 10^{-6}$). This leads to a series of large SVD problems with the smallest dimension in $O(2^{14})$.

The speedup, defined as $\frac{\text{CPUtime}}{\text{Walltime}\times n_{proc}}$ is relatively high for an interpreted language, hovering around 0.5. This is attributed to the high efficiency of the `lapack` embedding by `numpy`. Most of the variation between methods is due to the single thread operations such as reshaping or copies. Efficient reshaping of large order tensor remains an open problem with ongoing works such a `xtensor`.

Fig. 25 shows the L^2 and L^∞ errors for all of these methods. Although the behavior of the two error measures follows a similar pattern, L^∞ should be of great concern for the user as it remains extremely close to 1 for all the methods until the sharp decline onset. As expected, CR is very high, reaching 100% for QTT L^∞ error of 10^{-2}. This error value constitutes a good target for many storage applications, hence Table 7 compiles the CR associated with L^2 and L^∞ errors of 1%. Thus, one can conclude that ST-HOSVD alone does not constitute a good compression option for large number of dimension in mechanics simulation, providing no compression but can extract modes for further processing by HT. HT is the only method providing good compression even for L^∞ target whereas TT and QTT offer similar behavior. It should not come as a surprise, since in this example, no dimension is larger than 64, reducing the interest of QTT. Last, the reader is reminded that, these results are computed with a minimum tolerance of 10^{-8} on singular values computing which corresponds to machine error on intermediate operations. This explains why all methods are able to recover 10^{-8} error which corresponds to virtually no truncation. Still, HT, thanks to its particular structure, manages to halve the storage intensity of the studied data.

5 Discussion and Conclusion

This paper presents as briefly as possible the main techniques for multidimensional data decomposition and approximation. Then, they are applied to various problem engineering mechanics and fluid dynamics. Analysis for both numerical and physical experiments is provided. In short, this paper is an attempt at answering the following question.

"Which one of the numerous decomposition methods should be used for low rank approximation of mechanics data?"

To do so, a decomposition library, `pydecomp`, has been developed. It takes advantage of python numerous libraries for scientific computing, visualization, and data I/O handling. Benchmark cases are proposed in the library to test and compare each of the available methods: PGD, RPOD, T-HOSVD, ST-HOSVD, TT-SVD, QTT and HT in both discrete and continuous formulation (SVD and POD can be exchanged freely as they are essentially equivalent).

In section 4.2, these benchmarks have been put to use. The conclusion is that PGD cannot be used as a multidimensional decomposition method for it is extremely slow and compression power is insufficient. Yet, it should not be dismissed completely as it makes a relatively efficient 2D iterative methods. Indeed, it allows the user to compute only the required modes contrary to standard SVD/POD algorithm. The T-HOSVD has been dismissed as its results are indistinguishable from ST-HOSVD while being several times slower for large datasets. RPOD has also been classified as not suitable for three reasons. First, its decomposition performance is far poorer than TT-SVD and ST-HOSVD. Second, the computing time is much higher than its contenders (about 5 time more). HT and QTT are methods best suited for larger data for which their involved algorithms are intended. Third, the recursive nature of the methods does not translate naturally in the same way as other scientific computing algorithms and leads to shallow and wide trees that are slow to scan. Discussion (supported by [45,8]) on scalar product selection has led to the conclusion that default should be "blind" Eulerian scalar product, especially for Cartesian grids. However, some cases could benefit from L^2 scalar product such as Gaussian quadrature points, or contexts in which the physics is well-known and requires a special scalar product such as integrating a vector field using H^1 norm.

In sections 4.3 to 4.5, a close attention was given to the two most efficient methods for tackling actual data. Two scalar fields were used, one from an experiment, the other from a DNS simulation. The experimental data, taken from a droplet drying with 4 parameters (time, wavelength, 2D space) was found weakly separable with both methods. This is attributed to the nonlinear nature of the studied physics. Yet the reconstructed field for a tolerance of $\varepsilon = 10^{-2}$ seems sufficient for qualitative interpretation. The high error levels seem to lie in the cracks representation as shown in the brief analysis of the modes. The DNS data (LDC) was chosen to present regularities that were accurately captured. In each of these methods two data layouts for space decomposition were studied, one separates X and Y dimension while the other vectorize so that space is viewed as a single dimension. For both datasets, the separate dimension produces lower compression rate for a fixed error level but the inherent bidimensionality of the structure is captured with less accuracy (to the human eye) despite lower error. Indeed, mode combination leads to oscillations that reduce when the rank is increased. The last example was a breaking wave simulation computed using `notus CFD`. It proved the versatility of `pydecomp` implementation in handling data from several sources with different characteristics. It was shown that different variables from a single simulation present remarkably diverse separability levels. As expected, the smoother the field, the more separable it is. Another layout question was raised for this example, wondering whether these output variables should be treated as a distinct problem or as a single variable. Once again, there is no definitive answer and the user must adapt the layout to their need. The global decomposition allows easy handling, but the compression rate is dominated by the least separable variable. Thus, for this case, a distinct processing of each variable is preferable. Moreover, it was observed that for this (complex) dataset tensor has a finite Tucker rank at machine error.

Finally, in section 4.6 a much larger case is studied as a proof of concept for handling of parametric studies of multiphysics simulation code (microstructure, fluid dynamics, grain solidification, heat, etc.). This relatively modest experiment can be seen as a proxy for massive decomposition on supercomputers as pipelined post-processing. Unsurprisingly the strong non-linearity and heterogeneity of this kind of data makes low error decomposition difficult to obtain but HT seems to provide good approach for post processing. Indeed, *it halves the storage requirement (close to machine error) while providing useful insight in modes or slices for further analysis or surrogate modeling.*

As a rule of thumb, HT is the go-to candidate for any data decomposition related with physics since it provides the Tucker decomposition modes while tackling the ballooning core. This enables at least some storage saving and in-depth physical analysis, e.g. modes or slices can be manipulated from a simple laptop once the processing is complete. Still, its structure is more complex and makes it more difficult to optimize both accuracy (conditioning) and CPU-wise. Parallel processing of dimensions is possible and may be an interesting approach for $d \leq 10$. TT (or QTT for long dimensions) is a very reliable approach and can be used on any problem but does not provide orthogonal basis. Parallelization can be achieved at SVD/matrix level only since the algorithm is purely sequential.

In conclusion the low ranks approximation methods presented in this article constitute a tool that should not be overlooked in modern day scientific computing as it allows both cheaper storage (potentially orders of magnitudes) and enables modal analysis.

Future work

Among the many extensions of the present work, three promising items deserve further work.

Hierarchical format (HT) is a very promising method for data mining. It can be improved in many ways, both from an algorithm point of view (current implementation is quite naive) and the computer science one. This

includes a possible switch in programming language to improve index slicing or MPI approaches. As a format HT, embeds all the other formats described here which makes it a very good candidate for low level improvement. Also, it allows very efficient maxvol/blackbox algorithms for handling large datasets (see [34,32]) which is the necessary next step for handling very large data (and SVDs). Another improvement strategy will be to enable out-of-core computing and single pass algorithms. Then, HT would be very efficient method for handling cases in which $d \gg 1$.

This kind of sampling can be related to the way deep learning (DL) algorithms are trained. In fact, there are many ways to link DL and tensor reduction, both methods can be seen as a way to produce approximation of high dimensional spaces through some training phase (optimization process). The main divide between these two approaches is that tensor reduction as presented here is always a linear process while DL is built for non-linear problems. In many cases DL relies on tensors and tensor calculus which makes it a perfect candidate to take advantage of the reduction techniques. For instance, Daulbaev et al. [66] recently proposed a Deep Neural network training method that uses maxvol algorithm (from TT). Many others have proposed mixed formulation using both tensors and DL. Consequently, a `tensorflow` API within `pydecomp` will provide a new range of application of pydecomp, similar to Novikov et al. work [67].

The last axis of extension of this work is to apply these techniques to PDE in more ways. This includes studying the effect of scalar product to improve convergence and select specific properties such as H^1 norm for transport problems [68] but in the context of multidimensional problems. This is also intended at building multiparameter ROM (as proposed by the authors in [62]) and improving its stability.

Declaration

Funding

Financial support was provided by grant MOE2018-T2-1-05 in the context of the author's research fellowship at NTU in the team of Pr. Wang Li-Lian.

Financial support was provided by the Science and Engineering Research Council, A*STAR, Singapore (Grant no. A19E1a0097) in the context of the author's new position as a Scientist at A*STAR, IHPC, Engineering Mechanics Department, Singapore.

Data availability

The data that support the findings of this study are not available due to change of affiliation of the author. The data used in the last section is obtained through a software that has not been made public at the moment of writing this article and consequently cannot be shared. Synthetic data used in sec. 4.2 can be directly reproduced using the ipython notebook provided with `pydecomp` library. For other information, please contact the author.

Code availability

The main code used for this data is publicly available under CECILL license. `https://git.notus-cfd.org/llestandi/python_decomposition_library`

Conflict of Interest

The author has no relevant financial or non-financial interests to disclose.

Acknowledgments

The author gratefully acknowledge using IHPC's *simulation platform for additive manufacturing* to generate some data used in this paper. For further information on the simulation platform, readers can contact the author.

The author would like to thank Pr. Mejdi Azaïez for the support he provided in the writing of this paper and frequent discussions on the subject displayed here, in particular in the valorization of the numerical results. Many thanks to Diego Britez as well, who coded many of the functions in `pydecomp` as part of his masters' internship at I2M. The author would like to thank former colleagues at I2M Bordeaux and in particular `notus CFD` dev team for providing data as well as Pr. T.K. Sengupta (IIT Kanpur) for providing high precision LDC simulation code and the many discussions we had.

References

1. J. M. Alimi, V. Bouillot, Y. Rasera, V. Reverdy, P. Corasaniti, I. Balmès, S. Requena, X. Delaruelle, and J.-N. Richet, "First-ever full observable universe simulation," in *Int. Conf. for HPC, Networking, Storage and Analysis, SC*, 2012.
2. F. Chinesta, R. Keunings, and A. Leygue, *The Proper Generalized Decomposition for Advanced Numerical Simulations*. Springer, 2013.
3. G. Stabile and G. Rozza, "Finite volume POD-Galerkin stabilised reduced order methods for the parametrised incompressible Navier-Stokes equations," *Computers and Fluids*, vol. 173, pp. 273–284, 2018.
4. K. Carlberg, C. Farhat, J. Cortial, and D. Amsallem, "The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows," *Journal of Computational Physics*, vol. 242, pp. 623–647, 2013.
5. K. Lee and K. Carlberg, "Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders," 2018.
6. D. Kressner and C. Tobler, "Low-Rank Tensor Krylov Subspace Methods for Parametrized Linear Systems," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 4, pp. 1288–1316, 2011.
7. C. Quesada, D. González, I. Alfaro, E. Cueto, and F. Chinesta, "Computational vademecums for real-time simulation of surgical cutting in haptic environments," *International Journal for Numerical Methods in Engineering*, 2016.
8. L. Lestandi, *Low rank approximation techniques and reduced order modeling applied to some fluid dynamics problems*. Phd. thesis, Université de Bordeaux, 2018.
9. K. Pearson, "LIII. On lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.
10. H. Hotelling, "Analysis of a complex of statistical variables into principal components.," *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933.
11. M. Loève, *Probability Theory*, vol. 9. 1977.
12. J. L. Lumley, "Coherent Structures in Turbulence," in *Transition and Turbulence* (R. E. MEYER, ed.), pp. 215–242, Academic Press, 1981.
13. L. Sirovich, "Turbulence and the dynamics of coherent structures. I - Coherent structures. II - Symmetries and transformations. III - Dynamics and scaling," *Quarterly of Applied Mathematics*, vol. 45, no. July, p. 561, 1987.
14. C. Eckart and G. Young, "The approximation of one matrix by another of lower rank," *Psychometrika*, vol. 1, no. 3, pp. 211–218, 1936.
15. K. Ito and S. Ravindran, "A Reduced-Order Method for Simulation and Control of Fluid Flows," *Journal of Computational Physics*, 1998.
16. a. E. Deane, I. G. Kevrekidis, G. E. Karniadakis, and S. a. Orszag, "Low-dimensional models for complex geometry flows: Application to grooved channels and circular cylinders," *Physics of Fluids A: Fluid Dynamics*, vol. 3, no. 10, p. 2337, 1991.
17. W. Cazemier, R. W. C. P. Verstappen, a. E. P. Veldman, and I. Introduction, "Proper orthogonal decomposition and low-dimensional models for driven cavity flows," *Physics of Fluids*, vol. 10, no. 7, pp. 1685–1699, 1998.
18. M. Fahl, *Trust-region Methods for Flow Control based on Reduced Order Modelling*. PhD thesis, 2001.
19. M. Bergmann, *Optimisation aérodynamique par réduction de modèle POD et contrôle optimal. Application au sillage laminaire d'un cylindre circulaire*. PhD thesis, Institut National Polytechnique de Lorraine / LEMTA, 2004.
20. F. Hitchcock, "Multiple invariants and generalized rank of a p-way matrix or tensor," *J. Math. Phys*, vol. 7, pp. 39–79, 1927.
21. L. R. Tucker, "Some mathematical notes on three-mode factor analysis," *Psychometrika*, vol. 31, no. 3, pp. 279–311, 1966.
22. J. D. Carroll and J. J. Chang, "Analysis of individual differences in multidimensional scaling via an n-way generalization of "Eckart-Young" decomposition," *Psychometrika*, vol. 35, no. 3, pp. 283–319, 1970.
23. R. a. Harshman, "Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis," *UCLA Working Papers in Phonetics*, vol. 16, no. 10, pp. 1– 84, 1970.
24. T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, 2009.
25. W. M.-P.-I. f. M. i. t. S. Hackbush, *Tensor spaces and numerical Tensor calculus*. No. 1, Leipzig, Germany: Springer Heidelberg Dordrecht London New York, 2014.
26. L. De Lathauwer, B. De Moor, and J. Vandewalle, "A multilinear singular value decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1253–1278, 2000.
27. L. de Lathauwer, B. de Moor, and J. Vandewalle, "On the best rank-1 and rank-(R1,R2,...,RN) approximation of higher order tensors," *SIAM Journal on Matrix Analysis and Applications*, vol. 21, no. 4, pp. 1324–1342, 2000.
28. N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, "A New Truncation Strategy for the Higher-Order Singular Value Decomposition," *SIAM Journal on Scientific Computing*, vol. 34, no. 2, pp. A1027—-A1052, 2012.
29. I. Oseledets and E. E. Tyrtyshnikov, "Tensor tree decomposition does not need a tree," *Preprint*, 2009.
30. I. V. Oseledets, "Tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 33, no. 5, pp. 2295–2317, 2011.
31. B. N. Khoromskij, "O(logN)-Quantics Approximation of N-d Tensors in High-Dimensional Numerical Modeling," *Constructive Approximation*, vol. 34, pp. 257–280, oct 2011.
32. I. V. Oseledets, "Approximation of 2dx2d Matrices Using Tensor Decomposition," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 2130–2145, 2010.
33. I. V. Oseledets, S. Dolgov, and D. Savostyanov, "ttpy," 2018.
34. J. Ballani, L. Grasedyck, and M. Kluge, "Black Box Approximation of Tensors in Hierarchical Tucker Format," *Linear algebra and its applications*, vol. 438, no. 2, pp. 639–657, 2010.
35. L. Grasedyck, "Hierarchical Singular Value Decomposition of Tensors," *SIAM Journal on Matrix Analysis and Applications*, vol. 31, no. 4, pp. 2029–2054, 2010.
36. L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.
37. A. Cichocki, "Tensor Networks for Big Data Analytics and Large-Scale Optimization Problems," *arXiv preprint arXiv:1407.3124*, pp. 1–36, 2014.
38. I. V. Oseledets, "Constructive Representation of Functions in Low-Rank Tensor Formats," *Constructive Approximation*, no. 37.1, pp. 1–18, 2013.
39. D. Bigoni, A. P. Engsig-karup, and Y. M. Marzouk, "Spectral tensor-train decomposition," *SIAM Journal on Scientific Computing*, vol. 38, pp. 1–32, 2016.
40. A. Gorodetsky, *Continuous low-rank tensor decompositions, with applications to stochastic optimal control and data assimilation*. PhD thesis, MIT, 2016.
41. A. Gorodetsky, S. Karaman, and Y. Marzouk, "A continuous analogue of the tensor-train decomposition," *Computer Methods in Applied Mechanics and Engineering*, 2019.
42. A. Nouy, "Low-rank tensor methods for model order reduction," pp. 1–73, 2015.
43. A. Falco, W. Hackbusch, and A. Nouy, "Geometric Structures in Tensor Representations (Final Release)," pp. 1–50, 2015.

44. M. Azaïez, F. B. Belgacem, and T. C. Rebollo, "Recursive POD expansion for reaction-diffusion equation," *Advanced Modeling and Simulation in Engineering Sciences*, no. December, 2016.
45. M. Azaïez, L. Lestandi, and T. C. Rebollo, "Low Rank Approximation of Multidimensional Data In: Pirozzoli S., Sengupta T. (eds) High-Performance Computing of Big Data for Turbulence and Combustion.," vol. 592, Springer International Publishing, cism inter ed., 2019.
46. W. Austin, G. Ballard, and T. G. Kolda, "Parallel Tensor Compression for Large-Scale Scientific Data," in *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016*, 2016.
47. B. Philippe and Y. Saad, "Calcul des valeurs propres," in *Techniques de l'ingénieur. Sciences fondamentales, (AF1224)*, 2014.
48. Y. Saad, "Numerical methods for large eigenvalue problems," *Algorithms and architectures for advanced scientific computing*, p. 346 p., 1992.
49. L. Wu, E. Romero, and A. Stathopoulos, "PRIMME_SVDS: A High-Performance Preconditioned SVD Solver for Accurate Large-Scale Computations," *SIAM Journal on Scientific Computing*, vol. 39, no. 5, pp. S248–S271, 2017.
50. E. Rabani and S. Toledo, "Out-of-core SVD and QR decompositions," *Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, VA, CD-ROM, SIAM, Philadelphia*, no. 572, pp. 1–9, 2001.
51. V. Demchik, M. Bačák, and S. Bordag, "Out-of-core singular value decomposition," pp. 1–11, 2019.
52. A. M. Dunton, L. Jofre, G. Iaccarino, and A. Doostan, "Pass-efficient methods for compression of high-dimensional turbulent flow data," *Journal of Computational Physics*, vol. 423, may 2020.
53. D. D. Kosambi, "Statistics in function spaces," *Journal of the Indian Mathematical Society*, 1943.
54. D. Kressner and C. Tobler, "htucker – A Matlab toolbox for tensors in hierarchical Tucker format," pp. 1–28, 2013.
55. F. Chinesta and P. Ladavèze, *Separated Representations and PGD-Based Model Reduction*, vol. 554. 2014.
56. P.-E. Allier, L. Chamoin, and P. Ladevèze, "Proper Generalized Decomposition computational methods on a benchmark problem: introducing a new strategy based on Constitutive Relation Error minimization," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 2, no. 1, p. 17, 2015.
57. L. Grasedyck, W. Hackbusch, and B. Nr, "An Introduction to Hierachical (H) Rank and TT – Rank of Tensors with Examples," *Comput. Methods Appl. Math*, vol. 11, no. 3, pp. 291–304, 2011.
58. J. Ballani, *Fast evaluation of near-field boundary integrals using tensor approximations*. Phd, University of Leipzig, 2012.
59. J. Ballani and L. Grasedyck, "Hierarchical tensor approximation of output quantities of parameter-dependent PDEs," vol. 3, pp. 1–19, 2014.
60. B. N. Khoromskij, "O(d logN)-Quantics Approximation of N-d Tensors in High-Dimensional Numerical Modeling," *Constructive Approximation*, vol. 34, no. August, pp. 257–280, 2011.
61. L. Lestandi, S. Bhaumik, T. K. Sengupta, G. R. Krishna Chand Avatar, and M. Azaïez, "POD Applied to Numerical Study of Unsteady Flow Inside Lid-driven Cavity," *Journal of Mathematical Study*, vol. 51, no. 2, pp. 150–176, 2018.
62. T. K. Sengupta, L. Lestandi, S. I. Haider, A. Gullapalli, and M. Azaïez, "Reduced order model of flows by time-scaling interpolation of DNS data," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 5, p. 26, oct 2018.
63. B. W. Bader, T. G. Kolda, and Others, "MATLAB Tensor Toolbox Version 3.0-dev." Available online, 2017.
64. L. Lestandi, S. Bhaumik, G. R. K. C. Avatar, M. Azaiez, and T. K. Sengupta, "Multiple Hopf bifurcations and flow dynamics inside a 2D singular lid driven cavity," *Computers and Fluids*, vol. 166, pp. 86–103, 2018.
65. L.-X. Lu, S. Narayanaswami, and Y. W. Zhang, "Phase field simulation of powder bed-based additive manufacturing," *Acta Materialia*, vol. 144, pp. 801–809, 2018.
66. T. Daulbaev, J. Gusak, E. Ponomarev, A. Cichocki, and I. Oseledets, "Reduced-Order Modeling of Deep Neural Networks," oct 2019.
67. A. Novikov, P. Izmailov, V. Khrulkov, M. Figurnov, and I. Oseledets, "Tensor Train decomposition on TensorFlow (T3F)," jan 2018.
68. A. Iollo, S. Lanteri, and J.-A. Désidéri, "Stability Properties of POD – Galerkin Approximations for the Compressible Navier – Stokes Equations," *Theoret. Comput. Fluid Dynamics*, vol. 13, pp. 377–396, 2000.