Post-Processing of High-Dimensional Data*

Mike Espig[a], Wolfgang Hackbusch[b], Alexander Litvinenko[† c], Hermann G. Matthies[d], and Elmar Zander[d]

[a]Westsächsische Hochschule Zwickau, Zwickau, Germany
[b]Max-Planck-Institute for Mathematics in the Sciences (MIS), Leipzig, Germany
[c]Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen, Germany
[d]Technische Universität Braunschweig, Brunswick, Germany

13th June 2019

Abstract

Scientific computations or measurements may result in huge volumes of data. Often these can be thought of representing a real-valued function on a high-dimensional domain, and can be conceptually arranged in the format of a tensor of high degree in some truncated or lossy compressed format. We look at some common post-processing tasks which are not obvious in the compressed format, as such huge data sets can not be stored in their entirety, and the value of an element is not readily accessible through simple look-up. The tasks we consider are finding the location of maximum or minimum, or minimum and maximum of a function of the data, or finding the indices of all elements in some interval — i.e. level sets, the number of elements with a value in such a level set, the probability of an element being in a particular level set, and the mean and variance of the total collection.

The algorithms to be described are fixed point iterations of particular functions of the tensor, which will then exhibit the desired result. For this, the data is considered as an element of a high degree tensor space, although in an abstract sense, the algorithms are independent of the representation of the data as a tensor. All that we require is that the data can be considered as an element of an associative, commutative algebra with an inner product. Such an algebra is isomorphic to a commutative sub-algebra of the usual matrix algebra, allowing the use of matrix algorithms to accomplish the mentioned tasks. We allow the actual computational representation to be a lossy compression, and we allow the algebra operations to be performed in an approximate fashion, so as to maintain a high compression level. One such example which we address explicitly is the representation of data as a tensor with compression in the form of a low-rank representation.

Keywords: post-processing, high-dimensional data, compression, fixed point iteration, eigenvalue computation, low-rank tensor representation

Classification: 65J05, 65H17, 65F60, 65H99, 15A69

*Work partly supported by the Deutsche Forschungsgemeinschaft (DFG) and the Alexander von Humboldt Foundation (AvH).

†Corresponding author: RWTH Aachen, 52072 Aachen, Germany,
e-mail: Litvinenko@uq.rwth-aachen.de

Contents

1 Introduction

Many scientific and engineering computations or measurements, as well as economic or financial applications, produce large volumes of data. For simplicity, assume that we have computed or observed large quantities of one real-valued variate. Assume further that the amount of data is so large that it can not be held or stored in its entirety and has to be compressed in some way.

Frequently, the task arises for example to find the location with the maximum value of the data set, or to find the locations of all values which lie in a given interval — we call this a level set. Other, similar tasks we consider are finding the number of values in a given interval, the probability of being in a given interval, or finding the mean or the variance of the data set. These tasks are trivial for small data sets, typically performed by inspecting each datum. But if we consider truly huge amounts of data which can not be stored in full because this would exceed the capacity of any storage device, but only in a compressed manner, and where additionally it may not be possible due to time constraints to inspect each element, these tasks are not trivial any more. This is due to the fact that in the compressed representation the data values are normally not directly accessible and require additional processing. Such compression will in general be "lossy", so that not each value can be restored exactly, but only up to a certain reconstruction error.

Assuming the data as an element of some set $\mathcal{T}$, the algorithms are independent of the representation of the data, as well as from the compression and reconstruction technique used, subject only to the possibility of approximately performing the operations of an Euclidean associative commutative algebra. This means that we assume that $\mathcal{T}$ is a vector space with an inner product, and additionally an associative commutative bilinear multiplication, making it into an algebra. The simplest example of such a structure is to envisage the data points as a vector $\boldsymbol{w} \in \mathbb{R}^N$ with an appropriate $N \in \mathbb{N}$, reflecting the amount of data. The vector space operations are clear, as is the canonical Euclidean product, and for the commutative multiplication consider the point-wise or Hadamard product, i.e. the component-wise multiplication of two vectors. We shall allow that all these operations are performed only approximately, in order to maintain a high compression level.

Large volumes of data, especially when they can be thought of as samples or discrete values of some real-valued function on a high-dimensional space, can often be arranged in form of a tensor [40, 41, 8, 31]. This offers the possibility to use approximate or compressed tensor representations. Here we will especially show one possibility, namely low-rank representations, which generalise the truncated singular value decomposition for matrices. For the sake of completeness, we shall show the possible implementation of the above mentioned algebraic operations for some of the more common low-rank formats, and the approximated, compressed, or truncated representation of their results; see also [52].

The proposed algorithms are iterative in nature, and the convergence tolerance can be adapted to the reconstruction error. The basic idea for the algorithms, which operate only on the algebraic structure, is an iterative scheme which converges to a result which solves the desired problem in some way. Such iterations typically destroy the compressed representation, so they have to be combined with re-compression or truncation.

1.1 An example

Let us give an example which motivates much of the following formulation and development. Assume that we are interested in the time evolution of some system, described by

$$\frac{\mathrm{d}}{\mathrm{d}t}v(t) = A(\boldsymbol{\mu})(v(t)), \qquad \boldsymbol{\mu} \in \Omega,\, t \in [0,T] \tag{1}$$

where $v(t)$ is in some Hilbert space $\mathcal{V}$ and $A(\boldsymbol{\mu})$ is some parameter dependent operator; in particular $A(\boldsymbol{\mu})$ could be some parameter-dependent differential operator, for example

$$\frac{\partial}{\partial t}v(\boldsymbol{x},t) = \nabla\cdot(\boldsymbol{\kappa}(\boldsymbol{x},\boldsymbol{\mu})\nabla v(\boldsymbol{x},t)) + f(\boldsymbol{x},t), \quad \boldsymbol{\mu} \in \Omega,\, \boldsymbol{x} \in \mathcal{G},\, t \in [0,T] \tag{2}$$

where $\mathcal{G} \subset \mathbb{R}^\ell$ is a domain, $[0,T] \subset \mathbb{R}$ is the time window of interest, $\boldsymbol{\kappa}(\boldsymbol{x},\boldsymbol{\mu})$ is a parameterised random tensor field dependent on a random parameter in some probability space $\boldsymbol{\mu} \in \Omega \subset \mathbb{R}^d$ with probability measure $\mathbb{P}$, and one may take $\mathcal{U} = \mathrm{L}_2(\mathcal{G})$ [43, 46].

For each $\boldsymbol{\mu} \in \Omega$ one may thus seek for solutions in $\mathrm{L}_2([0,T],\mathcal{V}) \cong \mathcal{V}\otimes\mathrm{L}_2([0,T]) = \mathcal{V}\otimes\mathcal{Z}$, where we have set $\mathcal{Z} := \mathrm{L}_2([0,T])$. On the other hand, assume that for fixed $(\boldsymbol{x},t) \in \mathcal{G}\times[0,T]$ we are looking at the random variable $v(\boldsymbol{x},t,\cdot)$ which we for simplicity assume to have finite variance, i.e. $v(\boldsymbol{x},t,\cdot) \in \mathrm{L}_2(\Omega,\mathbb{P}) =: \mathcal{S}$. Taking into account the parametric dependence, we are hence looking for a function $v(\boldsymbol{x},t,\boldsymbol{\mu})$ which is defined on $\mathcal{G}\times[0,T]\times\Omega$; we are thus looking for a solution in $\mathcal{V}\otimes\mathcal{Z}\otimes\mathcal{S}$. This applies equally well to the abstract Eq. (1), as on the right hand side the operator depends on $\boldsymbol{\mu} \in \Omega$, so will the solution $v(t,\boldsymbol{\mu})$ to Eq. (1), and it will lie in a similar tensor product.

Observe further that if the probability measure $\mathbb{P}$ on Ω in Eq. (1) or Eq. (2) is a product measure $\mathbb{P} = \mathbb{P}_1\otimes\cdots\otimes\mathbb{P}_L$ on $\Omega = \Omega_1\times\cdots\times\Omega_L$, the space $\mathcal{S}$ can be split further $\mathcal{S} = \bigotimes_{\ell=1}^L \mathcal{S}_\ell$, with $\mathcal{S}_\ell := \mathrm{L}_2(\Omega_\ell,\mathbb{P}_\ell)$. Thus in total the solution to both Eq. (1) and Eq. (2) may be viewed as an element of a high order or degree tensor space [43, 46, 48]

$$\mathcal{V}\otimes\mathcal{Z}\otimes\mathcal{S} = \mathcal{V}\otimes\mathcal{Z}\otimes\bigotimes_{\ell=1}^{L}\mathcal{S}_\ell. \tag{3}$$

If we think of samples of $v(\boldsymbol{x},t,\boldsymbol{\mu})$ at space points $(\boldsymbol{x}_k)_{k=1}^K$, time instances $(t_j)_{j=1}^J$ and parameter values $(\boldsymbol{\mu}_{\boldsymbol{i}})_{\boldsymbol{i}} = (\mu_{i_1},\ldots,\mu_{i_L})$, with a multi-index $\boldsymbol{i} = (i_1,\ldots,i_L) \in \mathbb{N}^L$ with $1 \le i_\ell \le I_\ell$, the samples of the solution

$$\boldsymbol{\mathsf{v}} = (v(x_k,t_j,\mu_{i_1},\ldots,\mu_{i_L})) = (\mathsf{v}_{kji_1\ldots i_L}) \in \mathbb{R}^K\otimes\mathbb{R}^J\otimes\mathbb{R}^{I_1}\otimes\cdots\otimes\mathbb{R}^{I_L} \tag{4}$$

form a high order or degree *tensor* [31].

In many chemical applications one has $n = 100$, and d is few hundreds or even thousands [6]. Some other high-dimensional problems from chemistry and physics like Hartree-Fock-, Schrödinger-, or Master-equations, and low-rank tensor methods of their solution are for example considered in [32, 37, 39, 11], see also the references therein. Another example of large volumes of high-dimensional data are satellite data. Satellites collect data over a very large areas (e.g. the data collected by the National Center for Atmospheric Research (USA) [29]. Big data can also come from a computer simulator codes such as a solution of a multi-parametric equation, e.g. weather research and forecasting, and climate models [23]. Also Oil&Gas companies daily collect sensor data from multiple sources. Another source for huge volumes of data are high-energy particle accelerators like CERN [5].

For the sake of simplicity, we will treat all arguments of the function v in Eq. (4) equally, denoting them in that case by $\boldsymbol{p} = (x, t, \boldsymbol{\mu}) = (p_1, \ldots, p_d)$. So just consider a real-valued function $\boldsymbol{p} \mapsto w(\boldsymbol{p}) = w(p_1, \ldots, p_d)$ and its samples

$$\mathsf{W} = (w(\boldsymbol{p}_{\boldsymbol{m}}))_{\boldsymbol{m}\in\mathbb{N}^d} = (w(p_{m_1}, \ldots, p_{m_d}))_{\boldsymbol{m}} = (\mathsf{W}_{\boldsymbol{m}})_{\boldsymbol{m}} = (\mathsf{W}_{m_1\ldots m_d})_{m_1\ldots m_d} \in \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}, \quad (5)$$

which form a tensor of order or degree d representing a total of $N := \prod_{\ell=1}^{d} M_\ell$ values, and where for the sake of convenience multi-indices $\boldsymbol{m} \in \mathcal{M} := \times_{\ell=1}^{d}\{1, \ldots, M_\ell\} \subset \mathbb{N}^d$ have been introduced.

As one may see, a tensor can be simply defined as a high-order matrix or multi-index array, where multi-indices are used instead of indices. As an example, assume that we have $N = 10^{16}$ data points. Now a vector $\boldsymbol{w} \in \mathbb{R}^N$ is just a tensor of first order, but one may *reshape* the data, say into a matrix $\boldsymbol{W} \in \mathbb{R}^{10^8\times 10^8} \cong \mathbb{R}^{10^8} \otimes \mathbb{R}^{10^8}$ — a tensor of 2nd order. Reshaping further, one may take $\mathbb{R}^{10^4\times 10^4\times 10^4\times 10^4} \cong \bigotimes_{k=1}^{4} \mathbb{R}^{10^4}$ — a tensor of 4th order — as well as other combinations, such as $\mathsf{W} \in \bigotimes_{k=1}^{16} \mathbb{R}^{10}$ — an array of size $10 \times \cdots \times 10$ (16 times) — which is a tensor of 16th order; and finally all the way to $\mathsf{W} \in \bigotimes_{k=1}^{16}(\mathbb{R}^2 \otimes \mathbb{R}^5)$ — a tensor of order 32.

The tensors obtained in this way contain not only rows and columns, but also *slices* and *fibres* [40, 41, 8, 31]. These slices and fibres can be analysed for linear dependencies, super symmetry, or sparsity, and may result in a strong data compression. To have a first glimpse of possible compression techniques, assume that in the above example the data has been stored in the matrix $\boldsymbol{W} \in \mathbb{R}^{M\times M}$, where $M = 10^8 = \sqrt{N}$, and consider its singular value decomposition (SVD) $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\mathsf{T}} = \sum_{m=1}^{M} \varsigma_m \boldsymbol{u}_m \boldsymbol{v}_m^{\mathsf{T}}$, where $\boldsymbol{\Sigma} = \mathrm{diag}(\varsigma_1, \ldots, \varsigma_M)$ is the diagonal matrix of singular values $\varsigma_m \geq 0$, assumed arranged by decreasing value, and $\boldsymbol{U} = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_M]$, $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_M]$ collect the right and left singular vectors. Often there is a number $r \ll M$, such that for some small $\varepsilon > 0$ one has $\varsigma_m \leq \varepsilon$ for all $m > r$. Then one can formulate a compressed or truncated version

$$\boldsymbol{W} \approx \boldsymbol{W}_r = \boldsymbol{U}_r\boldsymbol{\Sigma}_r\boldsymbol{V}_r^{\mathsf{T}} = \sum_{m=1}^{r} \varsigma_m \boldsymbol{u}_m \boldsymbol{v}_m^{\mathsf{T}} = \sum_{m=1}^{r} \boldsymbol{w}_m^{(1)} \otimes \boldsymbol{w}_m^{(2)},$$

where $\boldsymbol{\Sigma}_r = \mathrm{diag}(\varsigma_1, \ldots, \varsigma_r)$, $\boldsymbol{U}_r = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r]$, $\boldsymbol{V}_r = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r]$, and $\boldsymbol{w}_m^{(1)} = \sqrt{\varsigma_m}\boldsymbol{u}_m$, $\boldsymbol{w}_m^{(2)} = \sqrt{\varsigma_m}\boldsymbol{v}_m$. This reduced SVD is a special case of Eq. (6). For the sake of a concrete example, assume that $r = 100$. One may observe that the required storage has been reduced from $N = M^2 = 10^{16}$ to $2 \times r \times M = 2 \times 10^{10}$ for the $2\,r$ vectors $\{\boldsymbol{w}_m^{(1)}, \boldsymbol{w}_m^{(2)}\}_{m=1}^{r}$.

The set of possible objects we want to work with will be denoted by $\mathcal{T}$. We assume that this set is equipped with the operations of an associative and commutative real algebra, and carries an inner product. It is a well known result [60] that such algebras — modulo some technical details — are isomorphic via the Gel'fand representation to a function algebra — to be more precise continuous real valued functions on a compact set. Under point-wise addition and multiplication by scalars, such functions clearly form a vector space, and if one includes point-wise multiplication of functions, they form an associative and commutative real algebra. In our case this is simply the function algebra $(\mathcal{M} = \times_{\ell=1}^{d}\{1, \ldots, M_\ell\} \to \mathbb{R})$, which is obviously the same as $\bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$. The advantage of this abstract algebraic formulation is not only that it applies to any kind of data representation on which one may define these algebraic operations, but that one may use algorithms [35] which have been developed for the algebra of real $N \times N$ matrices $\mathfrak{gl}(\mathbb{R}, N) = \mathbb{R}^{N\times N} \cong \mathbb{R}^N \otimes \mathbb{R}^N$.

In our concrete examples we work directly with the data represented as tensors $\mathcal{T} := \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$. Even without the identification with $(\mathcal{M} \to \mathbb{R})$, it is obvious that this is naturally a vector space. As also $\mathcal{T} \cong \mathbb{R}^N$, it is clear that $\mathcal{T}$ can be equipped with the canonical Euclidean inner product from $\mathbb{R}^N$. The associative and commutative algebra product comes from the identification with the function algebra $(\mathcal{M} \to \mathbb{R})$, i.e. the point-wise product, which is also known as the *Hadamard* product [31]. Hence it is clear that for data $\boldsymbol{w} \in \mathbb{R}^N$ it is not difficult at all to define the algebraic operations, but rather how to perform them when the data is in a compressed format.

To simplify later notation on the number of operations and amount of storage needed, we will often make the assumption that $M_1 = \cdots = M_d = n$, so that the tensor in Eq. (5) represents $N = n^d$ values. As already mentioned, our example of data compression is based on low-rank approximations to elements in $\mathcal{T}$. Although low-rank tensor data formats and techniques are almost unavoidable if working with large high-dimensional data sets, we would like to stress that the algorithms presented here are formulated purely in terms of the abstract algebraic structure, and are thus independent of the particular representation.

Whereas a general element $\boldsymbol{w} \in \mathcal{T}$ hence has $N = n^d$ terms, a compressed representation — some low-rank versions of which will be discussed in more detail in Section 3 — will have significantly fewer terms. For example, the *CP*-decomposition (*canonical polyadic*) representation, truncated to r terms,

$$\boldsymbol{w} \approx \boldsymbol{w}_r = \sum_{i=1}^{r} \bigotimes_{k=1}^{d} \boldsymbol{w}_i^{(k)};\ \boldsymbol{w}_i^{(k)} \in \mathbb{R}^n,\ \text{has}\ r \times n \times d\ \text{terms in}\ \boldsymbol{w}_r. \tag{6}$$

If the *rank* r is reasonably small compared to $N = n^d$ independent of n and d, then we have an approximation $\boldsymbol{w}_r \in \mathcal{T}$ with much less storage, which also only depends on the *product* of n and d and is not exponential in the dimension d. But now the maximum can not be easily looked up; to get a particular element, the expression Eq. (6) has to be evaluated for that index combination. In the following we shall assume that we work with approximations such as in Eq. (6) which need much less storage. One has to make sure then that the *algebraic* operations of the Hadamard algebra structure on $\mathcal{T}$ can be performed efficiently in an approximative manner, so that they have much lower complexity than the obvious $\mathcal{O}(n^d)$, although the single elements of $\boldsymbol{w}$ which are usually needed for point-wise operations — and post-processing, see Subsection 1.2 — are not directly available. Thus the motivating factors for applying compression, and in particular low-rank tensor techniques, include the following:

- The storage cost is reduced, depending on the tensor format, from $\mathcal{O}(n^d)$ to $\mathcal{O}(drn)$, or to $\mathcal{O}(drn + r^d)$, where $d > 1$.
- The low-rank tensor approximation is relatively new, but already a well-studied technique with free software libraries available. Various low-rank formats are available.
- The approximation accuracy is fully controlled by the tensor rank. The full rank gives an exact representation.
- Even more complicated operations like the Fourier transform can be performed efficiently in low-rank format. The basic fast Fourier transform on $\boldsymbol{w} \in \mathcal{T}$ would have complexity $\mathcal{O}(n^d \log(n^d))$. These low-rank techniques can either be combined with the fast Fourier transform giving a complexity of $\mathcal{O}(drn)$, or can even be further

accelerated at the price of an additional approximation error yielding a *superfast* Fourier transform [49, 10].

On the other hand, general limitations of a compression technique are that

- it could be time consuming to compute a compression, or in particular a low-rank tensor decomposition;
- with sampling, it requires an axes-parallel mesh;
- although many functions have a low-rank representation, in practice only some theoretical estimates exist.

But the fact still remains that there are situations where storage of all items is not feasible, and some kind of compression has to be employed.

1.2 Post-processing tasks

This work is about exploiting the compression together with the structure of a commutative algebra and the ability to perform the algebraic operations in the compressed format, at least approximately. The tensor product structure which appears in Eq. (3) is something which allows efficient calculations to be performed on a sample of the solution like Eq. (4) or Eq. (5) with the help of expressions such as Eq. (6) or other compression techniques.

This tensor product structure—in this case multiple tensor product structure—is typical for such parametric problems [47]. What is often desired, is a representation which allows for the approximate evaluation of the state of Eq. (1) or Eq. (2) as function of $\boldsymbol{\mu} \in \mathcal{M}$ without actually solving the system again. Furthermore, one would like this representation to be inexpensive to evaluate, and for it to be convenient for certain post-processing tasks, for example like finding the minimum or maximum value over some or all parameter values.

The tasks we consider here are

- finding the location and value of maxima or minima,
- finding the location and value of maxima or minima of a function of $\boldsymbol{w}$,
- finding the value in the tensor closest to some given number,
- finding level sets, i.e. the indices where the value is between two levels,
- finding the number of indices between two levels,
- computing the probability of being between two levels,
- computing the mean and the variance.

As an example, consider finding the maximum value. A naïve approach to compute the maximum would be to visit and inspect each element, but then the number of visits is $\mathcal{O}(n^d)$, exponential in d, which may be unacceptable for high d. Such phenomena when the total complexity/storage cost depends exponentially on the problem dimension have been called the curse of dimensionality [31].

The idea for such iterative post-processing algorithms, in the form of finding the maximum, was first presented in [14] for low-rank tensor approximations. Some further

post-processing tasks for such low-rank tensor representations were investigated in [20]. To give an example from [14] of the possible savings in space and time, assume that one looks at the solution of an example of Eq. (1) in Subsection 1.1, or more specifically Eq. (2) in the stationary case with $\kappa(\omega, x) \equiv 1$, a d-dimensional Poisson equation:

$$-\nabla^2 u = f \quad \text{on} \quad \mathcal{G} = [0,1]^d \quad \text{with} \quad u|_{\partial\mathcal{G}} = 1,$$

and right-hand-side

$$f(x_1, \ldots, x_d) \propto \sum_{k=1}^{d} \prod_{\ell=1, \ell\neq k}^{d} x_\ell (1 - x_\ell).$$

Assume further that this is solved numerically by a standard finite-difference method with $n = 100$ grid-points in each direction, so that the solution vector $\boldsymbol{u}$ has $N = n^d$ data points in $\mathbb{R}^N$, but can also naturally be viewed as a tensor $\boldsymbol{u}$ of degree d in $\bigotimes_{k=1}^{d} \mathbb{R}^{100}$. So for full storage, one needs $N = n^d$ storage locations, and if one is looking for the maximum by inspecting each element, one would have to inspect N elements.

We mention in passing that of course actually solving the discrete equation is a challenge for $d > 3$, but that is a different story, and for this we refer to [14] and the references therein.

d	# loc's.: $N = n^d$	$\approx$ years [a] inspect. N	actual [14] time [s] of Algorithm 2
25	10^{50}	1.6×10^{33}	0.16
50	10^{100}	1.6×10^{83}	0.42
75	10^{150}	1.6×10^{133}	1.16
100	10^{200}	1.6×10^{183}	2.58
125	10^{250}	1.6×10^{233}	4.97
150	10^{300}	1.6×10^{283}	8.56

Age of the universe: $\approx 14 \times 10^9$ years.

Number of hadrons (elementary particles) in the universe $\approx 10^{80}$.

Table 1: Computing times (4th column) on 2 GHz dual-core CPU to find maximum.

Now assume for the sake of simplicity that one can inspect 2×10^9 elements per second — on an ideal 2 GHz CPU with one inspection per cycle. Then for $\boldsymbol{u} \in (\mathbb{R}^n)^{\otimes d} \cong \mathbb{R}^{n^d}$ the times needed to find the maximum for the full data-set are shown in Table 1 in the third column—assuming that it were somehow possible to store all the values indicated in the second column—whereas the actual computation with a compressed format is shown in the last and fourth column.

It is obvious that for growing d for the full representation the computational complexity and the storage requirements quickly become not only unacceptable, but totally impossible to satisfy. The second and third column behave like $\mathcal{O}(n^d)$ and grow exponentially with d, whereas for a low-rank representation $\boldsymbol{u}_r$ of $\boldsymbol{u}$ not only can the data be stored on a modest laptop, but for the simple Algorithm 2 — to be explained later in Subsection 2.4 — the computing times in the fourth column are in terms of seconds and behave like $\mathcal{O}(n\, d^3)$.

1.3 State of the art

This is not a discussion of the state of the art regarding general tensor formats and their low-rank approximations, in quantum physics also known as *tensor networks*. For the

physical motivations and numerical developments from there see [64, 59, 22, 50, 4, 3]. For the mathematical and numerical view we refer to the review [41], the monographs [31, 37, 39], and to the literature survey on low-rank approximations [26]. In the following, we concentrate on the question of post-processing such data.

The idea of finding the largest element and the corresponding location of a tensor by solving an eigenvalue problem, where the matrix and the vectors are tensors given in the CP tensor format, was introduced in [14]. Additionally, the first numerical schemes to compute the point-wise inverse and the sign function were introduced, as well as the rank-truncation procedure for tensors given in the CP format. Later, in [20], these ideas were extended and applied to tensors which were obtained after discretisation and solution of elliptic PDEs with uncertain coefficients. Another group of authors, in [9], by combining the advances of the density matrix renormalisation group and the variational numerical renormalisation group methods, approximated several low-lying eigenpairs of large Hermitian matrices simultaneously in the block version of the TT format via the alternating minimisation of the block Rayleigh quotient sequentially for all TT cores.

In [13, 12], the authors suggested methods to compute the mean, the variance, and sensitivity indices in the TT train tensor format with applications to stochastic PDEs, whereas the diagonalisation of large Toeplitz or circulant matrices via combination of the fast Fourier and the CP tensor format was shown in [49].

An investigation of approximations to eigenfunctions of a certain class of elliptic operators in $\mathbb{R}^d$ by finite sums of products of functions with separated variables is the topic of [33]. Various tensor formats were used for a new class of rank-truncated iterative eigensolvers. The authors were able to reduce the computational cost from $\mathcal{O}(n^d)$ to $\mathcal{O}(n)$. They demonstrated the introduced algorithms by solving large-scale spectral problems from quantum chemistry: the Schrödinger, the Hartree–Fock, and the Kohn–Sham equations in electronic structure calculations.

1.4 Outline of the paper

In the following Section 2 the necessary material for abstract algebras is quickly reviewed. Then the algorithms and functions on the algebra $\mathcal{T}$ used to compute the post-processing tasks outlined in Subsection 1.2 are formulated in an abstract fashion, independent of the representation chosen for the data. Even as the formulation uses only the abstract algebra operations, we point out and motivate what this means in terms of the Hadamard algebra. But also for the Hadamard algebra, the presentation is independent of the tensor format to be chosen. Furthermore, for the iterative algorithm — a fixed point iteration — it is discussed how the possible truncation operation influences the convergence.

In Section 3 we present some concrete examples of compression of high-dimensional data $\boldsymbol{w} \in \mathbb{R}^N$, once it is identified with a tensor $\boldsymbol{w} \in \mathcal{T} := \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ with $N = \prod_{\ell=1}^{d} M_\ell$. For such tensors of degree d we discuss the *canonical polyadic* (CP) representation, the *Tucker* representation, and the *tensor-train* (TT) representation, and the compression in terms of low-rank approximations based on the different tensor formats. In particular, we show how the algebra operations can be carried out in the different tensor formats, the numerical effort involved, and the effect these algebraic operations have on the compression or low-rank representation. As the compression level may deteriorate, i.e. the rank of the approximation may grow, it is important that one is able to re-compress. Pointers to such methods in the literature are included as well.

The Section 4 contains a few numerical examples and a discussion of their results used to illustrate the algorithms of Section 2 and the representations from Section 3. These

examples are solutions of elliptic high-dimensional or parametric resp. stochastic partial differential equations, a domain where the data is naturally in a tensor format. But as already pointed out, for any data it is just a — often only conceptual — *reshape* to consider it as an element of a tensor space. The conclusion is contained in Section 5.

2 Algorithms for post-processing

Given a data-set $\boldsymbol{w}$ in some compressed tensor format, where looking up each element is not trivial, and where additionally it may not be computationally feasible to look at each element, we still want to be able to perform tasks which essentially require looking at each datum. The *tasks* outlined in Subsection 1.2 we want to consider are finding

1. the indices or values of *maxima, minima* of $\boldsymbol{w}$,
2. the indices or values of *maxima, minima* of some function f of $\boldsymbol{w}$,
3. the index or value of the datum of $\boldsymbol{w}$ *closest* to a given number,
4. level sets, i.e. (the indices of) all $w_{m_1 \dots m_d} \in [\omega_0, \omega_1]$,
5. the *number of indices* such that $w_{m_1 \dots m_d} \in [\omega_0, \omega_1]$,
6. the mean (sum of all items) and variance (sum of all items squared),
7. some auxiliary functions, such as the algebraic inverse $\boldsymbol{w}^{\odot -1}$ of $\boldsymbol{w}$ — in our case the Hadamard inverse — and others like the sign($\boldsymbol{w}$) function which will be seen to be needed for the above tasks in Subsection 2.3.

The above tasks and auxiliary functions will be computed by finding an *iteration* (mapping) Φ_P for each post-processing task P, such that the *fixed-point* $\boldsymbol{v}_*$ of Φ_P — i.e. $\Phi_P(\boldsymbol{v}_*) = \boldsymbol{v}_*$ — is either the sought solution for the task, or computes one of the auxiliary functions.

2.1 Iteration with Truncation

When performing computations using the operations of the algebra — like the Hadamard algebraic operations on tensors in some compressed format — after the operation the compression will typically be sub-optimal, which means that the result has to be compressed again. Thus, when performing the algebraic operations for a fixed-point iteration, the compression would either get worse and worse in each iteration, or one has to re-compress or *truncate* the result again to a good compression level. Therefore a re-compression after each or after a number of algebraic operations may be necessary, and we thus allow that the algebraic operations are possibly only executed approximately.

In our example case the low-rank representation of tensors explained in Section 3 acts as compression, and the rank may increase through the algebraic operations, and hence we will use *truncation* $\mathcal{T}_\epsilon$ to *low rank* r with error ϵ [31, 2] to avoid the problem of a deteriorating compression level.

In other words, with a general compressed representation $\boldsymbol{w}_r$ of $\boldsymbol{w}$ the computation will be a *truncated* or *perturbed iteration* [34, 48]. If we denote the general compression mapping by $\mathcal{T}_\epsilon$ — meaning compression with an accuracy ϵ — the iteration map is changed from Φ_P to $\mathcal{T}_\epsilon \circ \Phi_P$.

The general structure of the iterative algorithms for a post-processing task P or for an auxiliary function is shown in Algorithm 1. Here we want to collect some results for this kind of iteration. For such a truncated or perturbed iteration as in Algorithm 1, it is

Algorithm 1 Iteration with truncation

Start with some initial *compressed* guess $\boldsymbol{v}_0$ depending on task P.
$i \leftarrow 0$
while *no convergence* **do**
    $\boldsymbol{z}_i \leftarrow \Phi_P(\boldsymbol{v}_i)$;
        ▷ the iterator Φ_P may *deteriorate* the compression level
    if compression level of $\boldsymbol{z}_i$ is too bad **then**
        $\boldsymbol{v}_{i+1} \leftarrow \mathcal{T}_\epsilon(\boldsymbol{z}_i)$;
            ▷ use *truncation* $\mathcal{T}_\epsilon$ to compress $\boldsymbol{z}_i$ with error ϵ
    else
        $\boldsymbol{v}_{i+1} \leftarrow \boldsymbol{z}_i$;
    end if
    $i \leftarrow i + 1$
end while

known that

1. if the iteration by Φ_P is *super-linearly* convergent, the *truncated iteration* $\mathcal{T}_\epsilon \circ \Phi_P$ will still *converge* super-linearly, but finally *stagnate* in an ϵ-neighbourhood of the fixed point $\boldsymbol{v}_*$ [34]. One could loosely say that the super-linear convergence of Φ_P is stronger than the truncation by $\mathcal{T}_\epsilon$.

2. if the iteration by Φ_P is *linearly* convergent with *contraction* factor q, the *truncated iteration* $\mathcal{T}_\epsilon \circ \Phi_P$ will still *converge* linearly, but finally *stagnate* in an $\epsilon/(1-q)$-neighbourhood of $\boldsymbol{v}_*$ [48]. Again, one could loosely say that iteration by Φ_P and truncation by $\mathcal{T}_\epsilon$ balance each other, thus resulting in a larger neighbourhood of stagnation.

We shall assume that the truncation level has thus been chosen according to the desired re-construction accuracy and taking into account the possible influence due to the convergence behaviour of the iterator Φ_P.

2.2 Preliminaries and basic algebraic operations

As already pointed out, we assume in general that the set $\mathcal{T}$ is a vector space. Our example of the space $\mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ of tensors of interest was already introduced in Subsection 1.1. This is clearly a vector space, so we are able to add two such tensors, and multiply each by some real number, in other words we may form linear combinations. The additive neutral element — the zero tensor — will be denoted as $\mathbf{0}$. It is important that the compressed storage format chosen allows to perform the vector space operations without going into the *full* representation. This will be shown for some of the familiar tensor formats in Section 3. Furthermore, as $\mathcal{T} := \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell} \cong \mathbb{R}^{M_1 \times \cdots \times M_d} \cong \mathbb{R}^N$ as vector spaces, we can carry the canonical Euclidean inner product on $\mathbb{R}^N$ to $\mathcal{T}$, which for $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{T}$ is denoted by

$$\langle \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}} := \sum_{m_1=1,\ldots,m_d=1}^{M_1,\ldots,M_d} u_{m_1,\ldots,m_d} \cdot v_{m_1,\ldots,m_d} = \sum_{\boldsymbol{m} \in \mathcal{M}} u_{\boldsymbol{m}} \cdot v_{\boldsymbol{m}}.$$

This makes $\mathcal{T}$ into a Euclidean or Hilbert space. We assume that the computation of this inner product is feasible when both $\boldsymbol{u}$ and $\boldsymbol{v}$ are in a compressed representation; again this will be demonstrated in Section 3 for the low-rank representations we shall consider as examples.

On this space $\mathcal{T}$ an additional structure is needed, namely a *multiplication* which will make it into a unital associative and commutative algebra. In our example this will be the *Hadamard* multiplication, which apparently goes back to *Schur*, which is the point- or component-wise multiplication on $\bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell} \cong \mathbb{R}^{\times_{\ell=1}^{d} M_\ell}$. It is the usual point-wise multiplication of functions, in this case the functions $\times_{\ell=1}^{d} M_\ell \to \mathbb{R}$. For $\boldsymbol{u}, \boldsymbol{v} \in \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$, define the Hadamard product as

$$\odot : \mathcal{T} \times \mathcal{T} \to \mathcal{T}, \quad \boldsymbol{u} \odot \boldsymbol{v} \mapsto \boldsymbol{w} = (w_{m_1 \dots m_d}) := (u_{m_1 \dots m_d} \cdot v_{m_1 \dots m_d}). \tag{7}$$

It is a bi-linear operation (linear in each entry), and it is commutative. Thus this product makes $\mathcal{T}$ into an *associative* and *commutative algebra*. The symbol $\odot$ will be used both for the abstract algebra product on $\mathcal{T}$, as well as for the Hadamard product on $\bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$. As is any such algebra, for any $\boldsymbol{w} \in \mathcal{T}$ it holds that $\boldsymbol{w} \odot \mathbf{0} = \mathbf{0}$.

Especially, as a function of $\boldsymbol{v}$, the product $\boldsymbol{u} \odot \boldsymbol{v}$ is a linear map $\boldsymbol{L}_{\boldsymbol{u}} \in \mathscr{L}(\mathcal{T})$ on $\mathcal{T}$. This is the familiar canonical representation $\mathcal{T} \ni \boldsymbol{u} \mapsto \boldsymbol{L}_{\boldsymbol{u}} \in \mathscr{L}(\mathcal{T})$ of an associative algebra as linear maps on itself, i.e. in this case in a commutative sub-algebra of the algebra of all linear maps $\mathscr{L}(\mathcal{T})$ with concatenation as product.

It easy to see that this Hadamard algebra has a unit element $\mathbf{1} \in \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ for the product,

$$\mathbf{1} := (1_{m_1 \dots m_d}), \quad \forall \boldsymbol{w} \in \mathcal{T} : \ \boldsymbol{w} = \mathbf{1} \odot \boldsymbol{w} = \boldsymbol{w} \odot \mathbf{1}, \tag{8}$$

a tensor with all entries equal to unity — this makes $\bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ into a *unital* algebra. Observe that by defining for all $1 \le k \le d$ the *all-ones* vectors $\mathbf{1}_{M_k} = [1, \dots, 1]^T \in \mathbb{R}^{M_k}$, the Hadamard unit has rank *one* according to Eq. (6): $\mathbf{1} = \bigotimes_{k=1}^{d} \mathbf{1}_{M_k}$. Obviously one has $\boldsymbol{L}_{\mathbf{1}} = \boldsymbol{I}_{\mathcal{T}}$, the identity in $\mathscr{L}(\mathcal{T})$.

Having defined a unit or neutral element for multiplication, we say that $\boldsymbol{w} \in \mathcal{T}$ has a *multiplicative inverse* iff there is an element, denoted by $\boldsymbol{w}^{\odot -1} \in \mathcal{T}$, such that

$$\mathbf{1} = \boldsymbol{w}^{\odot -1} \odot \boldsymbol{w} = \boldsymbol{w} \odot \boldsymbol{w}^{\odot -1}. \tag{9}$$

Obviously not all elements have an inverse, e.g. the zero element $\mathbf{0} \in \mathcal{T}$ is never invertible. In any case, not every $\boldsymbol{w} \in \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ has a Hadamard inverse, for this it is necessary that all entries $w_{m_1 \dots m_d} \neq 0$ are non-zero, and then $\boldsymbol{w}^{\odot -1} = (w^{-1}_{m_1 \dots m_d}) = ((1/w)_{m_1 \dots m_d})$. Note that $(\boldsymbol{w}^{\odot -1})^{\odot -1} = \boldsymbol{w}$ and $\mathbf{1}^{\odot -1} = \mathbf{1}$ as it should be, and it is easily seen that $\boldsymbol{L}_{\boldsymbol{w}^{\odot -1}} = \boldsymbol{L}_{\boldsymbol{w}}^{-1}$, the inverse of $\boldsymbol{L}_{\boldsymbol{w}}$ in $\mathscr{L}(\mathcal{T})$.

Elements of the algebra which can be written as a square $\boldsymbol{w} = \boldsymbol{u} \odot \boldsymbol{u} = \boldsymbol{u}^{\odot 2}$ are called positive, they form a convex cone $\mathcal{T}_+ \subset \mathcal{T}$; note that obviously the zero tensor $\mathbf{0}$ and the multiplicative unit $\mathbf{1}$ are positive, and that if an invertible $\boldsymbol{w}$ is positive, so is its inverse $\boldsymbol{w}^{\odot -1}$. As usual, this defines an order relation $\boldsymbol{u} \le \boldsymbol{v} \Leftrightarrow \boldsymbol{v} - \boldsymbol{u} \in \mathcal{T}_+$, which implies that for any positive $\boldsymbol{w} \in \mathcal{T}_+$ one has $\mathbf{0} \le \boldsymbol{w}$. Further one may observe that $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{T}_+ \Rightarrow \boldsymbol{u} \odot \boldsymbol{v} \in \mathcal{T}_+$.

A certain compatibility of the inner product and the algebra product is needed, as we require that

$$\langle \boldsymbol{w} \odot \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}} = \langle \boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}} = \langle \boldsymbol{u} | \boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{v} \rangle_{\mathcal{T}} = \langle \boldsymbol{u} | \boldsymbol{w} \odot \boldsymbol{v} \rangle_{\mathcal{T}}, \tag{10}$$

i.e. that the action of the algebra product is *self-adjoint*, or, in other words, that the maps $\boldsymbol{L}_{\boldsymbol{w}}$ are self adjoint. This condition is satisfied for the Hadamard algebra.

The inner product with the unit element $\phi(\boldsymbol{w}) := \langle \boldsymbol{w} | \mathbf{1} \rangle_{\mathcal{T}}$ defines, as function of $\boldsymbol{w}$, a positive linear functional ϕ, as for $\boldsymbol{w} \in \mathcal{T}_+$ one has

$$\phi(\boldsymbol{w}) = \phi(\boldsymbol{u} \odot \boldsymbol{u}) = \langle \boldsymbol{u} \odot \boldsymbol{u} | \mathbf{1} \rangle_{\mathcal{T}} = \langle \boldsymbol{u} | \boldsymbol{u} \rangle_{\mathcal{T}} \geq 0.$$

It is a kind of "trace" or un-normalised state functional on the algebra, as $\phi(\mathbf{1}) = N = \dim \mathcal{T}$ in the Hadamard algebra. In particular

$$\phi(\boldsymbol{u} \odot \boldsymbol{v}) = \langle \boldsymbol{u} \odot \boldsymbol{v} | \mathbf{1} \rangle_{\mathcal{T}} = \langle \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}}. \tag{11}$$

Conversely, if the algebra comes equipped with such an un-normalised positive state functional, the Eq. (11) may be taken as the definition of an inner product, which automatically satisfies Eq. (10).

We shall assume that we can compute the algebraic operations — at least approximately — in compressed representation (this will be shown for the Hadamard algebra for the low-rank formats in Section 3), and that we can compute the multiplicative inverse also in compressed representation — again at least approximately.

As in any unital algebra, if for $\lambda \in \mathbb{C}$ the element $\boldsymbol{w} - \lambda \mathbf{1}$ fails to be invertible — this means that $\boldsymbol{L}_{\boldsymbol{w}} - \lambda \boldsymbol{I}_{\mathcal{T}}$ is not invertible in $\mathscr{L}(\mathcal{T})$ — we shall say that λ is in the *spectrum* of $\boldsymbol{w}$; here — as we are in finite dimensional spaces — it is an *eigenvalue*, i.e. there is an *eigenvector* $\boldsymbol{v}_\lambda \in \mathcal{T}$ such that $\boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{v}_\lambda = \boldsymbol{w} \odot \boldsymbol{v}_\lambda = \lambda \, \boldsymbol{v}_\lambda = \lambda \, \boldsymbol{I}_{\mathcal{T}} \boldsymbol{v}_\lambda$. From this discussion it is immediate that in the Hadamard algebra case, if λ is an eigenvalue of $\boldsymbol{w} = (w_{\boldsymbol{m}}) \in \mathcal{T}$ or $\boldsymbol{L}_{\boldsymbol{w}} \in \mathscr{L}(\mathcal{T})$, there must be a multi-index $\boldsymbol{m}_\lambda \in \mathcal{M}$ such that $w_{\boldsymbol{m}_\lambda} = \lambda$, and hence in the real Hadamard algebra $\mathcal{T}$ one has that always $\lambda \in \mathbb{R}$. Thus the spectrum of $\boldsymbol{w}$ resp. $\boldsymbol{L}_{\boldsymbol{w}}$ is $\sigma(\boldsymbol{w}) = \sigma(\boldsymbol{L}_{\boldsymbol{w}}) = \{w_{\boldsymbol{m}} \mid \boldsymbol{m} \in \mathcal{M}\}$. Observe that according to Eq. (10), each representing map $\boldsymbol{L}_{\boldsymbol{w}}$ from above is self-adjoint, which means that also in general all the spectra are real — $\sigma(\boldsymbol{w}) \subset \mathbb{R}$.

Specifically, the Euclidean Hadamard algebra $\mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ which we have constructed is obviously isomorphic — as a Euclidean algebra — to $\mathbb{R}^N$ with the canonical Euclidean inner product when equipped with the point- or element-wise Hadamard product $\odot_N$. Let us denote this isomorphism by $V : \mathcal{T} \to \mathbb{R}^N$; it implies some ordering of the terms of each $\boldsymbol{w} \in \mathcal{T}$.

More enlightening and less obvious may be the unital algebra isomorphism with the commutative sub-algebra of *diagonal* matrices $\mathfrak{diag}(\mathbb{R}, N)$ in the full matrix algebra $\mathfrak{gl}(\mathbb{R}, N) = \mathbb{R}^{N \times N} \cong \mathbb{R}^N \otimes \mathbb{R}^N \cong \mathscr{L}(\mathbb{R}^N)$ with the usual matrix multiplication. Let us denote this isomorphism by $M : \mathcal{T} \to \mathfrak{diag}(\mathbb{R}, N)$. If $\boldsymbol{w} = V(\boldsymbol{w}) \in \mathbb{R}^N$ is the vector containing all the elements of the tensor $\boldsymbol{w}$, then $M(\boldsymbol{w}) := \boldsymbol{W} = \mathrm{diag}(\boldsymbol{w}) = \mathrm{diag}(V(\boldsymbol{w})) \in \mathfrak{diag}(\mathbb{R}, N) \subset \mathfrak{gl}(\mathbb{R}, N)$ is the corresponding diagonal matrix, i.e. $M = \mathrm{diag} \circ V$. Hence, by choosing the canonical Euclidean basis in $\mathbb{R}^N$ and its image by V^{-1} as a basis in $\mathcal{T}$, the canonical representation $\boldsymbol{L}_{\boldsymbol{w}} \in \mathscr{L}(\mathcal{T})$ is itself represented by the matrix $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{w}) = M(\boldsymbol{w}) \in \mathfrak{diag}(\mathbb{R}, N) \subset \mathfrak{gl}(\mathbb{R}, N) \cong \mathscr{L}(\mathbb{R}^N)$.

Let $\boldsymbol{v} \in \mathcal{T}$ be another tensor, with associated $V(\boldsymbol{v}) = \boldsymbol{v} \in \mathbb{R}^N$ and $M(\boldsymbol{v}) = \boldsymbol{V} = \mathrm{diag}(V(\boldsymbol{v})) = \mathrm{diag}(\boldsymbol{v}) \in \mathfrak{diag}(\mathbb{R}, N)$, then from the definition of the isomorphism M:

$$\begin{aligned} M(\boldsymbol{w} \odot \boldsymbol{v}) &= M(\boldsymbol{w}) M(\boldsymbol{v}) = \boldsymbol{W}\boldsymbol{V} = \mathrm{diag}(V(\boldsymbol{w})) \, \mathrm{diag}(V(\boldsymbol{v})) \\ &= \mathrm{diag}(\boldsymbol{w}) \, \mathrm{diag}(\boldsymbol{v}) = \mathrm{diag}\left(M(\boldsymbol{w}) V(\boldsymbol{v})\right) = \mathrm{diag}\left(\boldsymbol{W}\boldsymbol{v}\right) = \mathrm{diag}\left(\mathrm{diag}(\boldsymbol{w})\boldsymbol{v}\right), \end{aligned} \tag{12}$$

which contains the important equality

$$\begin{aligned} \boldsymbol{w} \odot \boldsymbol{v} = M^{-1}(\boldsymbol{W}\boldsymbol{V}) = M^{-1}(\mathrm{diag}\left(\boldsymbol{W}\boldsymbol{v}\right)) = M^{-1}(\mathrm{diag}(\boldsymbol{w}) \, \mathrm{diag}(\boldsymbol{v})) = \\ M^{-1}(\mathrm{diag}\left(M(\boldsymbol{w}) V(\boldsymbol{v})\right)) = V^{-1}(\boldsymbol{W}\boldsymbol{v}) = \boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{v}, \end{aligned} \tag{13}$$

from where one sees that

$$V(\boldsymbol{w} \odot \boldsymbol{v}) = \boldsymbol{W}\boldsymbol{v} = \mathrm{diag}(\boldsymbol{w})\boldsymbol{v} = M(\boldsymbol{w})V(\boldsymbol{v}), \tag{14}$$

which helps in understanding the following algorithms. Note that this shows that the algebra $\bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ with the Hadamard product is already jointly diagonalised and is essentially in its *Gel'fand* representation [60], which is an abstract way of saying what was already stated above, namely that the spectrum $\sigma(\boldsymbol{w})$ of an element $\boldsymbol{w} \in \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ are exactly the individual terms or data stored in the tensor. If in some abstract unital associative and commutative algebra the multiplication is *not* the point-wise multiplication — e.g. think of convolution — then it is known that modulo some technicalities [60] it is isomorphic via the Gel'fand "diagonalisation" morphism to a function algebra, and all the algorithms to follow would deal with the spectrum $\sigma(\boldsymbol{w})$ of $\boldsymbol{w}$ instead of with the "values of $\boldsymbol{w}$". These two notions only coincide for a function algebra.

2.3 Post-processing Tasks

Here the different post-processing tasks are explained together with how they will be computed. This may involve a number of auxiliary functions. The computation of these — again through truncated iteration Algorithm 1 — and the way in which some of the computations can be enhanced or accelerated is shown in Subsection 2.4.

Finding the maximum or minimum of $\boldsymbol{w} \in \mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ is the first task we consider. Observe, that the element of maximum modulus of the tensor $\boldsymbol{w}$ is also equal to the ∞-norm $\|\boldsymbol{w}\|_\infty$.

Finding the maximum means finding the index $\hat{\boldsymbol{m}} = (\hat{m}_1, \ldots, \hat{m}_d) \in \mathcal{M}$ where the maximum $\hat{w}$ of the elements in $\boldsymbol{w}$ occurs:

$$\hat{w} := w_{\hat{\boldsymbol{m}}} := w_{\hat{m}_1,\ldots,\hat{m}_d} := \max\{\, w_{\boldsymbol{m}} \;:\; \boldsymbol{m} = (m_1, \ldots, m_d) \in \mathcal{M}\}. \tag{15}$$

It was already established that each value of $\boldsymbol{w}$ is an eigenvalue. Defining for each $1 \leq m \leq M_\ell$ the canonical unit basis vectors $\boldsymbol{e}_{M_\ell}^{(m)}$ in each $\mathbb{R}^{M_\ell}$ in the tensor product $\mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$ as $\boldsymbol{e}_{M_\ell}^{(m)} := (\delta_n^{(m)})_{n=1,\ldots,M_\ell} \in \mathbb{R}^{M_\ell}$ via the Kronecker-δ-symbol, and similarly for each $\boldsymbol{m} \in \mathcal{M}$ the canonical unit basis vectors $\boldsymbol{e}^{(\boldsymbol{m})} = (\delta_{(\boldsymbol{n})}^{(\boldsymbol{m})})_{\boldsymbol{n}\in\mathcal{M}} \in \mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$, then for *any* $\boldsymbol{m} = (m_1, \ldots, m_d) \in \mathcal{M}$ the element $w_{\boldsymbol{m}}$ satisfies an *eigenvalue* equation:

$$\boldsymbol{L}_{\boldsymbol{w}}\boldsymbol{e}^{(\boldsymbol{m})} := V^{-1}\left(M(\boldsymbol{w})V(\boldsymbol{e}^{(\boldsymbol{m})})\right) = \boldsymbol{w} \odot \boldsymbol{e}^{(\boldsymbol{m})} = w_{\boldsymbol{m}}\boldsymbol{e}^{(\boldsymbol{m})}, \tag{16}$$

and the eigenvectors $\boldsymbol{e}^{(\boldsymbol{m})} = \bigotimes_{\ell=1}^{d} \boldsymbol{e}_{M_\ell}^{(m_\ell)}$ are evidently of rank one. This shows that each datum $w_{\boldsymbol{m}}$ of $\boldsymbol{w}$ may be found through *eigenvalue* computation. The eigenvalue is the data entry $w_{\boldsymbol{m}}$ with the corresponding eigenvector $\boldsymbol{e}^{(\boldsymbol{m})}$ indicating the location resp. the index $\boldsymbol{m} \in \mathcal{M}$.

Although quite obvious, it may be worthwhile pointing out that the linear span of any nonempty collection of eigen- resp. unit vectors $\mathcal{T}_{\mathcal{N}} := \mathrm{span}\{\boldsymbol{e}^{(\boldsymbol{n})} \mid \boldsymbol{n} \in \mathcal{N}\}$ ($\mathcal{N} \subseteq \mathcal{M}$) is an invariant subspace, and the orthogonal projector on $\mathcal{T}_{\mathcal{N}}$ is $\boldsymbol{L}_{\boldsymbol{p}^{(\mathcal{N})}} := \sum_{\boldsymbol{n}\in\mathcal{N}} \boldsymbol{L}_{\boldsymbol{e}^{(\boldsymbol{n})}}$ with $\boldsymbol{p}^{(\mathcal{N})} = \sum_{\boldsymbol{n}\in\mathcal{N}} \boldsymbol{e}^{(\boldsymbol{n})}$. With this, the spectral resolution of the identity for any and all $\boldsymbol{L}_{\boldsymbol{w}} \in \mathscr{L}(\mathcal{T})$ for $\boldsymbol{w} \in \mathcal{T}$ is $\boldsymbol{I}_{\mathcal{T}} = \sum_{\boldsymbol{m}\in\mathcal{M}} \boldsymbol{L}_{\boldsymbol{e}^{(\boldsymbol{m})}}$ corresponding to $\mathbb{1} = \sum_{\boldsymbol{m}\in\mathcal{M}} \boldsymbol{e}^{(\boldsymbol{m})}$, giving the spectral resolution

$$\boldsymbol{L}_{\boldsymbol{w}} = \sum_{\boldsymbol{m}\in\mathcal{M}} w_{\boldsymbol{m}}\boldsymbol{L}_{\boldsymbol{e}^{(\boldsymbol{m})}}, \quad \text{corresponding to} \quad \boldsymbol{w} = \sum_{\boldsymbol{m}\in\mathcal{M}} w_{\boldsymbol{m}}\boldsymbol{e}^{(\boldsymbol{m})} = \sum_{\boldsymbol{m}\in\mathcal{M}} w_{\boldsymbol{m}} \bigotimes_{\ell=1}^{d} \boldsymbol{e}_{M_\ell}^{(m_\ell)}, \tag{17}$$

which may be seen as the trivial basis representation of $\boldsymbol{w}$.

As the basic algorithms for eigenvalue computation like power iteration for self-adjoint linear maps converge to the eigenvalue of maximum absolute value or modulus, it is now clear that if we assume that the maximum is also the element of maximum absolute value, we only have to perform power iteration. This gives us the value $\hat{w} = w_{\hat{m}}$, and the location through its corresponding eigenvector $\boldsymbol{e}^{(\hat{m})}$. How to find the minimum in this case, or what to do when the element of maximum modulus is the minimum will be explained later in Subsection 2.4; it will all be accomplished with spectral shifts familiar from eigenvalue computations.

Finding the maximum or minimum of some function f of $\boldsymbol{w}$, where it is assumed that $f : \sigma(\boldsymbol{w}) \to \mathbb{R}$, is not very difficult if one can compute $f(\boldsymbol{w})$, which will be simply understood as $f(\boldsymbol{w}) := (f(w_{\boldsymbol{m}}))_{\boldsymbol{m}\in\mathcal{M}}$. This definition of $f(\boldsymbol{w})$ is also the one which comes via the isomorphy with the algebra of diagonal matrices $\mathfrak{diag}(\mathbb{R}, N) \subset \mathfrak{gl}(\mathbb{R}, N)$, where $f(\boldsymbol{w})$ is defined [35] through the corresponding matrix function $f(M(\boldsymbol{w})) = f(\boldsymbol{W})$, and is also the same as the one which comes from the abstract functional calculus on the Banach algebra $\mathcal{T}$ [60].

Observe that one may compute $f(\boldsymbol{w})$ for any real or complex function f defined on the spectrum $\sigma(\boldsymbol{w})$ — which is a finite set — at least in principle, through an interpolating polynomial $p(w) = \sum_k \alpha_k w^k$. As usual, the number $w \in \sigma(\boldsymbol{w})$ is replaced by $\boldsymbol{w}$ (or $\boldsymbol{L}_{\boldsymbol{w}}$) in the polynomial, resulting in $f(\boldsymbol{w}) = p(\boldsymbol{w}) = \sum_k \alpha_k \boldsymbol{w}^{\odot k}$. This may be highly inefficient, as in general the degree of the polynomial would be equal to the number of data, in this case the huge number $N - 1$. For the functions which will be needed for the post-processing tasks, the computation of the function will be done differently, namely iteratively via Algorithm 1.

Assume now that $f(\boldsymbol{w})$ has been computed, at least approximately. To find maxima or minima of $f(\boldsymbol{w})$, one now has to simply apply the considerations of the preceding paragraph to the tensor $f(\boldsymbol{w})$. Obviously, $f(\boldsymbol{w})$ has the same spectral resolution as $\boldsymbol{w}$ in Eq. (17):

$$f(\boldsymbol{L}_{\boldsymbol{w}}) = \sum_{\boldsymbol{m}\in\mathcal{M}} f(w_{\boldsymbol{m}}) \boldsymbol{L}_{\boldsymbol{e}^{(\boldsymbol{m})}}, \quad \text{corresponding to}$$

$$f(\boldsymbol{w}) = \sum_{\boldsymbol{m}\in\mathcal{M}} f(w_{\boldsymbol{m}}) \boldsymbol{e}^{(\boldsymbol{m})} = \sum_{\boldsymbol{m}\in\mathcal{M}} f(w_{\boldsymbol{m}}) \bigotimes_{\ell=1}^{d} \boldsymbol{e}_{M_\ell}^{(m_\ell)}. \quad (18)$$

Finding the index and value closest to a given number $\rho \in \mathbb{R}$ is now simply finding the eigenvector and eigenvalue of

$$(\boldsymbol{L}_{\boldsymbol{w}} - \rho\, \boldsymbol{I}_{\mathcal{T}})^{-1} = \boldsymbol{L}_{\boldsymbol{y}} \quad \text{with} \quad \boldsymbol{y} = (\boldsymbol{w} - \rho\, \boldsymbol{1})^{\odot -1}. \quad (19)$$

This is a special case of the preceding paragraph for the function $f : t \mapsto (t - \rho)^{-1}$. Therefore, if $\lambda_{\boldsymbol{w}}$ is any eigenvalue of $\boldsymbol{L}_{\boldsymbol{w}}$, then the corresponding eigenvalue of $\boldsymbol{L}_{\boldsymbol{y}}$ is the transformed one $\lambda_{\boldsymbol{y}} = (\lambda_{\boldsymbol{w}} - \rho)^{-1}$. Hence the value of $\boldsymbol{w}$ closest to ρ is the eigenvalue of maximum modulus of $\boldsymbol{L}_{\boldsymbol{y}}$, and the element of largest magnitude of $(\boldsymbol{w} - \rho\, \boldsymbol{1})^{\odot -1}$. To find the element of smallest magnitude, or even the vanishing elements of $\boldsymbol{w}$, one would use $\rho = 0$. Thus one may use the same algorithms as in the maximum search above. Observe that this operation here requires the Hadamard inverse, which will be one of the auxiliary functions.

Finding the indices in a level set requires the so-called *sign* function, where sign : $\mathcal{T} \to \mathcal{T}$ is defined component-wise as

$$\mathcal{T} \ni (\text{sign}(\boldsymbol{w})_{m_1,\ldots,m_d}) := \begin{cases} 1, & \text{if } w_{m_1,\ldots,m_d} > 0; \\ 0, & \text{if } w_{m_1,\ldots,m_d} = 0; \\ -1, & \text{if } w_{m_1,\ldots,m_d} < 0. \end{cases} \tag{20}$$

Again this is a special case of the previous one with a general function f.

The *characteristic* function for a subset $S \subset \mathbb{R}$ — we shall only look at intervals — $\chi_S : \mathcal{T} \to \mathcal{T}$, is again defined component-wise:

$$\mathcal{T} \ni (\chi_S(\boldsymbol{w})_{m_1,\ldots,m_d}) := \begin{cases} 1, & \text{if } w_{m_1,\ldots,m_d} \in S; \\ 0, & \text{if } w_{m_1,\ldots,m_d} \notin S. \end{cases} \tag{21}$$

With these two auxiliary functions, it is possible to define the characteristic function of a level set, i.e. all values between $\omega_1, \omega_2 \in \mathbb{R}$. We then have with $-\infty < \omega_1 < \omega_2 < \infty$:

$$(\chi_S(\boldsymbol{w}))_{m_1,\ldots,m_d}) := \begin{cases} \frac{1}{2}\left(\mathbf{1} + \text{sign}(\omega_2\,\mathbf{1} - \boldsymbol{w})\right), & \text{if } S =]-\infty, \omega_2[; \\ \frac{1}{2}\left(\mathbf{1} - \text{sign}(\omega_1\,\mathbf{1} - \boldsymbol{w})\right), & \text{if } S =]\omega_1, +\infty[; \\ \frac{1}{2}\left(\text{sign}(\omega_2\,\mathbf{1} - \boldsymbol{w}) - \text{sign}(\omega_1\,\mathbf{1} - \boldsymbol{w})\right), & \text{if } S =]\omega_1, \omega_2[; \end{cases} \tag{22}$$

Each case is easily computed with the sign function from Eq. (20). Hence, the indices of all components $w_{m_1\ldots m_d} \in]\omega_1\,\omega_2[$ are provided by $\chi_{]\omega_1,\omega_2[}(\boldsymbol{w})$. One may additionally define the *level set* function $\mathcal{L}_S(\cdot)$ of a subset $S \subset \mathbb{R}$ as $\mathcal{L}_S(\boldsymbol{w}) := \chi_S(\boldsymbol{w}) \odot \boldsymbol{w}$, which together with the indices provides the values in the subset S.

Finding the number of indices in a level set is accomplished through consideration of the *support*

$$\text{supp}\,\chi_S := \{\boldsymbol{m} \in \mathcal{M} \;:\; \chi_S(\boldsymbol{w})_{\boldsymbol{m}} \neq 0\} \subset \mathcal{M} \tag{23}$$

of a characteristic function χ_S as the subset of those indices where it is *non-zero*. Its *cardinality* is

$$\#(\text{supp}\,\chi_S(\boldsymbol{w})) = |\text{supp}\,\chi_S(\mathbf{w})| = \langle \chi_S(\boldsymbol{w}) | \mathbf{1} \rangle_{\mathcal{T}}, \tag{24}$$

the number of non-zero positions in the characteristic function; requiring only the computation of one inner product with the Hadamard multiplicative unit.

Computing the probability of being in a level, as well as the mean and the variance is under the assumption that each index in $\boldsymbol{w}$ carries the same probability, as then the *probability* of a value of $\boldsymbol{w}$ being in a subset S is simply

$$\mathbb{P}_{\boldsymbol{w}}(S) := \frac{\#(\text{supp}\,\chi_S(\boldsymbol{w}))}{N} = \frac{\langle \chi_S(\boldsymbol{w}) | \mathbf{1} \rangle_{\mathcal{T}}}{N}. \tag{25}$$

The *mean* or *average* and *variance* of $\boldsymbol{w}$ is then

$$\mathbb{E}\,(\boldsymbol{w}) := \bar{w} := \frac{1}{N}\,\langle \boldsymbol{w} | \mathbf{1} \rangle_{\mathcal{T}}; \quad \text{var}(\boldsymbol{w}) = \frac{1}{N}\,\langle \tilde{\boldsymbol{w}} | \tilde{\boldsymbol{w}} \rangle_{\mathcal{T}}, \;\; \text{where } \tilde{\boldsymbol{w}} := \boldsymbol{w} - \bar{w}\,\mathbf{1}. \tag{26}$$

Given the level set function from above, one may also compute the conditional mean, conditioned on being in the set S:

$$\mathbb{E}\,(\boldsymbol{w}|S) := \bar{w}_{|S} := \frac{\langle \mathcal{L}_S(\boldsymbol{w}) | \mathbf{1} \rangle_{\mathcal{T}}}{\langle \chi_S(\boldsymbol{w}) | \mathbf{1} \rangle_{\mathcal{T}}}. \tag{27}$$

2.4 Auxiliary functions and algorithmic details

Here we show how to compute the auxiliary functions used in the previous Subsection 2.3. These are functions defined on the Euclidean Hadamard algebra $\mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$, and it is worth while pointing out that the iterative algorithms will be operating on the algebra, which, as we saw in Subsection 2.2, is isomorphic to an algebra of diagonal matrices, $\mathfrak{diag}(\mathbb{R}, N) \subset \mathfrak{gl}(\mathbb{R}, N)$, and these matrices are real symmetric. This means in particular that all the known algorithms for computing matrix functions [35] can be used on the Hadamard algebra as well. Of those needed here, the most basic one and the one with the simplest connection to the algebra turns out to be the inverse $\boldsymbol{w}^{\odot -1}$. Let us remark once again that the algorithms are valid in any abstract algebra, but we are mainly concerned with the example of the "function algebra" $\bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$.

The Hadamard inverse $\boldsymbol{w}^{\odot -1}$ of a Hadamard invertible $\boldsymbol{w} \in \mathcal{T}$ is needed for *inverse* iteration and other post-processing tasks. Although the Hadamard inverse is the application of the function $f : t \mapsto t^{-1}$ to $\boldsymbol{w}$ and could in principle be computed through a polynomial, it is often advantageous and much more effective to use other algorithms. The algorithm for the inverse can be given simply by referring to the quadratically convergent Newton algorithm for matrices [35] for computing inverses, i.e. we use *Newton's method* for $(P \leftarrow (\odot - 1))$ and apply it to the equation $F(\boldsymbol{v}) := \boldsymbol{v}^{\odot -1} - \boldsymbol{w} = \mathbf{0}$ to solve for $\boldsymbol{v}$, the only solution of which is $\boldsymbol{v} = \boldsymbol{w}^{\odot -1}$. Hence the iteration function to be used in Algorithm 1 with starting vector $\boldsymbol{v}_0 := \boldsymbol{w}$ is

$$\Phi_{(\odot -1)}(\boldsymbol{v}) := \boldsymbol{v} \odot (2 \cdot \mathbf{1} - \boldsymbol{w} \odot \boldsymbol{v}). \tag{28}$$

It is well known that the iteration converges *quadratically* [35], and according to what was explained in Subsection 2.1 this is true even with *truncation*.

The sign function is defined in Eq. (20). Many of the tasks explained in the previous Subsection 2.3 involve the sign function. Given the sign function, the characteristic, support, and level set functions are easily computed by the basic operations of the algebra.

From the fact that each value in a tensor $\boldsymbol{w}$ is an eigenvalue in the Hadamard algebra, and from the isomorphy with the algebra of diagonal matrices $\mathfrak{diag}(\mathbb{R}, N) \subset \mathfrak{gl}(\mathbb{R}, N)$ where each value on the diagonal is also obviously an eigenvalue, one sees that this is actually the same definition as the one for matrices [35], and is also the one which follows from general functional calculus in abstract algebras [60]. This means that $\operatorname{sign}(\boldsymbol{w})$ is the application of the function $f : t \mapsto \operatorname{sign}(t)$ to $\boldsymbol{w} \in \mathcal{T}$, and again could in principle be computed with a polynomial. To compute it via the iterative Algorithm 1, i.e. $(P \leftarrow \operatorname{sign})$, one uses the same algorithm [35] — the Roberts-Newton algorithm — as for matrices. It can be derived by applying *Newton's* method to the equation $F(\boldsymbol{v}) := \boldsymbol{v} \odot \boldsymbol{v} - \mathbf{1} = \mathbf{0}$ with starting value $\boldsymbol{v}_0 := \boldsymbol{w}$. This yields the iteration function

$$\Phi_{\operatorname{sign}}(\boldsymbol{v}) := \frac{1}{2}\,(\boldsymbol{v} + \boldsymbol{v}^{\odot -1}). \tag{29}$$

Observe that this iteration function in a slightly more general form $\Phi_{(\sqrt{\boldsymbol{w}})}(\boldsymbol{v}) := \frac{1}{2}\,(\boldsymbol{v} + \boldsymbol{v}^{\odot -1} \odot \boldsymbol{w})$ is the ancient *Babylonian* method [35] — in modern paralance Newton's method — to find the square root $\boldsymbol{w}^{1/2}$ of $\boldsymbol{w}$, as by inserting $\boldsymbol{w} = \mathbf{1}$ in the Babylonian iteration $\Phi_{(\sqrt{\boldsymbol{w}})}$, one obtains Eq. (29). This means that one is iterating to compute the square root of the unit element $\mathbf{1}$ with the specific starting value $\boldsymbol{v}_0 = \boldsymbol{w}$. We will come back to this point of view shortly.

It is known [35] that the iteration converges *quadratically*, and according to what was explained in Subsection 2.1, it does so even with *truncation*; observe that it needs the Hadamard inverse from the previous paragraph. Therefore it is not really practical as this would mean that it can only be applied to invertible $\boldsymbol{w} \in \mathcal{T} = \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$, but it also actually needs a new inverse $\boldsymbol{v}_i^{\odot -1}$ in each iteration. This last fact would computationally result in a nested iteration — using the algorithm from the previous paragraph in an inner iteration to compute the inverse $\boldsymbol{v}_i^{\odot -1}$ in each sweep of the outer iteration for the sign function — and such procedures are seldom computationally efficient.

It is thus simpler to use a device employed also for matrix sign computations [35], namely to replace the explicit inverse in Eq. (29) by one step of the iteration for the inverse in Eq. (28). The resulting Newton-Schulz algorithm [35] has the same starting point $\boldsymbol{v}_0 := \boldsymbol{w}$ as before, but uses the iteration function

$$\Phi_{\text{N-S}}(\boldsymbol{v}) := \frac{1}{2} \cdot \boldsymbol{v} \odot (3 \cdot \mathtt{1} - \boldsymbol{v}^{\odot 2}) \tag{30}$$

in the Algorithm 1. It is now a simple iteration which still converges *quadratically* [35], even with *truncation* according to the explanations in Subsection 2.1. We refer to the discussion in [35] on how to get even faster algorithms using Padé approximations and other ways to accelerate the convergence through scaling, which is especially important in the initial stages.

Coming back to the somewhat curious idea of iterating for the square root of $\mathtt{1}$, one may observe that although the defining equation $F(\boldsymbol{v}) := \boldsymbol{v} \odot \boldsymbol{v} - \mathtt{1} = \mathtt{0}$ does not cover the case that a datum with the exact value of zero occurs — a non-invertible element would not satisfy the defining equation — the Newton-Schulz iteration with the function in Eq. (30) with starting value $\boldsymbol{v}_0 := \boldsymbol{w}$ takes care of this. A value which vanishes in $\boldsymbol{v}_i$ also vanishes in $\boldsymbol{v}_{i+1} = \Phi_{\text{N-S}}(\boldsymbol{v}_i)$, due to the product with $\boldsymbol{v}_i$ in Eq. (30).

Stopping criteria in case of quadratic convergence in an iteration like in Algorithm 1 are well known. First, if one wants to solve $F(\boldsymbol{v}) = \mathtt{0}$ or a fixed point equation $F(\boldsymbol{v}) := \Phi(\boldsymbol{v}) - \boldsymbol{v} = \mathtt{0}$, a natural criterion is the size of the residuum at step i:

$$\|F(\boldsymbol{v}_i)\|_{\mathcal{T}} < \eta_F. \tag{31}$$

But this checks only how well the equation is satisfied, and not directly how accurate the iterate $\boldsymbol{v}_i$ is. Regarding this latter issue, specifically for quadratic convergence, a natural criterion [35] to check for the accuracy of $\boldsymbol{v}_i$ at step i is

$$\delta_i := \frac{\|\boldsymbol{v}_i - \boldsymbol{v}_{i-1}\|_{\mathcal{T}}}{\|\boldsymbol{v}_i\|_{\mathcal{T}}} < \eta_v. \tag{32}$$

Further, referring to the discussion specifically for the sign function in [35], there are arguments to check

$$\delta_i < \|\boldsymbol{v}_i\|_{\mathcal{T}}^p \, \eta_v \tag{33}$$

for the exponents $p = 0, 1, 2$. For $p = 0$, this is the original general criterion Eq. (32), whereas the other values of p take specific consideration of the sign function.

Eigenvalue computations are involved in the first three tasks described in Subsection 2.3. One may use any algorithm developed for large scale matrices [24, 57, 58, 65] which only uses the action of the matrix on a vector in the computation — this is the

Hadamard product in our case — and we shall here only explain the main idea and a few variations.

The simplest algorithm is power iteration ($P \leftarrow$ pow-it), and the iteration map $\Phi_{\text{pow-it}}$ is given in Algorithm 2, to be used in Algorithm 1. Assume that the datum $w_{\hat{\boldsymbol{m}}} > 0$ of $\boldsymbol{w}$ with maximum absolute value or maximum modulus is indeed a (positive) maximum. Assume also that this occurs at one unique index $\hat{\boldsymbol{m}}$, i.e. the datum $\lambda_1 := w_{\hat{\boldsymbol{m}}}$ is a simple eigenvalue, and denote the next by absolute value smaller element / eigenvalue by λ_2. Assume the further eigenvalues ordered by decreasing absolute value.

Algorithm 2 One step power iteration $\Phi_{\text{pow-it}}$

1: Input iterate $\boldsymbol{v}_i$; ▷ assume $\boldsymbol{v}_i$ of unit length.
2: $\boldsymbol{u} \leftarrow \boldsymbol{w} \odot \boldsymbol{v}_i$; ▷ $\boldsymbol{u} = \boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{v}_i$.
3: $\gamma \leftarrow \langle \boldsymbol{u} | \boldsymbol{u} \rangle_{\mathcal{T}}^{-1/2}$; ▷ inverse length of $\boldsymbol{u}$. $\lambda_1 \approx \gamma^{-1}$
4: $\boldsymbol{z} \leftarrow \gamma \cdot \boldsymbol{u}$; ▷ normalise output $\boldsymbol{z}$ to unit length.
5: Output $\boldsymbol{z}$;

As is well known [24, 57], the generated sequence $\boldsymbol{v}_i$ *converges linearly* to $\pm \boldsymbol{e}^{(\hat{\boldsymbol{m}})}$ with a contraction factor of $q = |\lambda_2/\lambda_1|$, and with what was said in the explanations in Subsection 2.1, it still converges with truncation and will stagnate in the vicinity of $\pm \boldsymbol{e}^{(\hat{\boldsymbol{m}})}$. Also, as in the algorithm the input $\boldsymbol{v}_i$ has unit length, the length of $\boldsymbol{u}$ — given by the inverse γ^{-1} of the scaling factor in line 3 — converges linearly to the desired $\lambda_1 = w_{\hat{\boldsymbol{m}}}$ with the same rate. Algorithmically, the eigenvalue approximation $\lambda_1 \approx \gamma^{-1}$ is a *side-effect* of Algorithm 2.

This can be immediately enhanced through the additional computation of the *Rayleigh quotient* (RQ)

$$\varrho_{\boldsymbol{w}}(\boldsymbol{v}_i) := \frac{\langle \boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{v}_i | \boldsymbol{v}_i \rangle_{\mathcal{T}}}{\langle \boldsymbol{v}_i | \boldsymbol{v}_i \rangle_{\mathcal{T}}} = \frac{\langle \boldsymbol{w} \odot \boldsymbol{v}_i | \boldsymbol{v}_i \rangle_{\mathcal{T}}}{\langle \boldsymbol{v}_i | \boldsymbol{v}_i \rangle_{\mathcal{T}}} \tag{34}$$

between line 2 and line 3 of Algorithm 2: $\varrho_{\boldsymbol{w}}(\boldsymbol{v}_i) = \langle \boldsymbol{u} | \boldsymbol{v}_i \rangle_{\mathcal{T}}$ — no need to divide by the length of the unit vector $\boldsymbol{v}_i$. The Rayleigh quotient is stationary at an eigenvalue and has thus a quadratic convergence, it usually represents a much better approximation to $\lambda_1 = w_{\hat{\boldsymbol{m}}}$ than γ^{-1}; see Algorithm 3.

Stopping criteria for eigenvalues could be done similarly to normal convergent processes, but is is possible to have a posteriori error estimates which are actual bounds specifically for eigenvalues and -vectors, e.g. see [44, 24, 57]. The simplest seems to be the so-called Krylov-Bogolyubov bound. With an approximate eigenvalue μ and approximate eigenvector $\boldsymbol{x}$ of $\boldsymbol{L}_{\boldsymbol{w}}$ it is given by [44]

$$\min_{\lambda_j \in \sigma(\boldsymbol{w})} \frac{|\lambda_j - \mu|}{|\lambda_j|} \leq \frac{\|\boldsymbol{L}_{\boldsymbol{w}} \boldsymbol{x} - \mu \, \boldsymbol{x}\|_{\mathcal{T}}}{\|\boldsymbol{x}\|_{\mathcal{T}}}. \tag{35}$$

The right-hand side of Eq. (35) is minimised by $\mu = \varrho_{\boldsymbol{w}}(\boldsymbol{x})$. A short computation [44] shows that for the substitutions $\boldsymbol{x} \leftarrow \boldsymbol{v}_i$ and $\mu \leftarrow \varrho_{\boldsymbol{w}}(\boldsymbol{v}_i)$ the right-hand side of Eq. (35) becomes $(\varrho_{\boldsymbol{w}^{\odot 2}}(\boldsymbol{v}_i) - \varrho_{\boldsymbol{w}}(\boldsymbol{v}_i)^2)^{1/2}$. As

$$\varrho_{\boldsymbol{w}^{\odot 2}}(\boldsymbol{v}_i) = \langle \boldsymbol{w}^{\odot 2} \odot \boldsymbol{v}_i | \boldsymbol{v}_i \rangle_{\mathcal{T}} = \langle \boldsymbol{w} \odot \boldsymbol{v}_i | \boldsymbol{w} \odot \boldsymbol{v}_i \rangle_{\mathcal{T}} = \langle \boldsymbol{u} | \boldsymbol{u} \rangle_{\mathcal{T}},$$

the Krylov-Bogolyubov error bound Eq. (35) can be computed as

$$\min_{\lambda_j \in \sigma(\boldsymbol{w})} \frac{|\lambda_j - \langle \boldsymbol{u} | \boldsymbol{v}_i \rangle_{\mathcal{T}}|}{|\lambda_j|} \leq \varepsilon_\lambda := (\langle \boldsymbol{u} | \boldsymbol{u} \rangle_{\mathcal{T}} - \langle \boldsymbol{u} | \boldsymbol{v}_i \rangle_{\mathcal{T}}^2)^{1/2}. \tag{36}$$

Observe that this is not an a posteriori error estimate, but an actual *bound*. It is the best possible with the available information [44], and it can be very easily computed inside the iteration to control the possible termination. Similar bounds exist for the approximate eigenvector $\boldsymbol{v}_i$. Inserting these considerations into Algorithm 2 gives Algorithm 3 for the task power iteration with RQ computation ($P \leftarrow$ pow-RQ) for the iteration map $\Phi_{\text{pow-RQ}}$ to be used in Algorithm 1, which has the more accurate RQ eigenvalue approximation $\lambda_1 \approx \varrho_{\boldsymbol{w}}$ and error bound $\varepsilon_\lambda = (\langle \boldsymbol{u}|\boldsymbol{u}\rangle_{\mathcal{T}} - \langle \boldsymbol{u}|\boldsymbol{v}_i\rangle_{\mathcal{T}}^2)^{1/2}$ included as side-effects.

Algorithm 3 One step power iteration with Rayleigh quotient (RQ): $\Phi_{\text{pow-RQ}}$

1: Input iterate $\boldsymbol{v}_i$;	▷ assume $\boldsymbol{v}_i$ of unit length.
2: $\boldsymbol{u} \leftarrow \boldsymbol{w} \odot \boldsymbol{v}_i$;	▷ $\boldsymbol{u} = \boldsymbol{L}_{\boldsymbol{w}}\boldsymbol{v}_i$.
3: $\varrho_1 \leftarrow \langle \boldsymbol{u}|\boldsymbol{v}_i\rangle_{\mathcal{T}}$;	▷ Rayleigh quotient (RQ) $\lambda_1 \approx \varrho_1$
4: $\varrho_2 \leftarrow \langle \boldsymbol{u}|\boldsymbol{u}\rangle_{\mathcal{T}}$;	▷ RQ of $\boldsymbol{w}^{\odot 2}$.
5: $\varepsilon_\lambda \leftarrow (\varrho_2 - \varrho_1^2)^{-1/2}$;	▷ error bound Eq. (36) of λ_1.
6: $\gamma \leftarrow \varrho_2^{-1/2}$;	▷ inverse length of $\boldsymbol{u}$.
7: $\boldsymbol{z} \leftarrow \gamma \cdot \boldsymbol{u}$;	▷ normalise output $\boldsymbol{z}$ to unit length.
8: Output $\boldsymbol{z}$;	

Once the determination of the eigenvalue is accurate enough such that one has an interval which contains only *one* eigenvalue, one may use the even tighter Temple-Kato bounds [44].

Starting vectors and other enhancements like deflation and Krylov subspaces are discussed here. Recalling the discussion of invariant subspaces in the paragraph on maxima and minima in Subsection 2.3, the starting vector $\boldsymbol{v}_0$ in Algorithm 1 should not have any zero in it, as this would exclude an invariant subspace from the investigation. One possibility is $\boldsymbol{v}_0 = \mathbf{1}/\sqrt{N}$, being equal in all positions. With this starting vector one has $\boldsymbol{v}_1 \propto \boldsymbol{w} \odot \mathbf{1} = \boldsymbol{w}^{\odot 1}$, and it is easy to see that what is computed are scaled versions of $\boldsymbol{w}^{\odot i}$ as eigenvector approximation $\boldsymbol{v}_i$. This gives another possibility for acceleration [27], which will be discussed later.

Another technique in eigenvalue computations is *deflation* [24, 57]: when one eigenvalue $\lambda_1 = \lambda_{\hat{\boldsymbol{m}}}$ of largest modulus and corresponding eigenvector $\boldsymbol{e}^{(\hat{\boldsymbol{m}})}$ have been located, one may want to compute a further location $\tilde{\boldsymbol{m}}$ where there is an element $w_{\tilde{\boldsymbol{m}}} = \lambda_{\tilde{\boldsymbol{m}}}$ of equal magnitude $|w_{\tilde{\boldsymbol{m}}}| = |\lambda_1| = |\lambda_{\hat{\boldsymbol{m}}}|$. For this one can use deflation, and one could change line 2 in Algorithm 2 or Algorithm 3 to $\boldsymbol{u} \leftarrow \boldsymbol{w} \odot (\mathbf{1} - \boldsymbol{e}^{(\hat{\boldsymbol{m}})}) \odot \boldsymbol{v}_i$ to obtain a new iteration map for deflation. The factor $(\mathbf{1} - \boldsymbol{e}^{(\hat{\boldsymbol{m}})})$ projects into the orthogonal complement of the invariant subspace $\text{span}\{\boldsymbol{e}^{(\hat{\boldsymbol{m}})}\}$, and thus all the eigenvalues are left unchanged, except for $\lambda_{\hat{\boldsymbol{m}}}$, which is mapped to zero. But in this special case — where we know the form of all invariant subspaces — it is even simpler to keep the same iteration map and to choose as starting vector $\boldsymbol{v}_0 := (\mathbf{1} - \boldsymbol{e}^{(\hat{\boldsymbol{m}})})/\sqrt{(N-1)}$. It has a zero at position $\hat{\boldsymbol{m}}$ and is thus in the invariant subspace $(\text{span}\{\boldsymbol{e}^{(\hat{\boldsymbol{m}})}\})^\perp$, and all iterates will stay in that subspace. Iterating in Algorithm 1 with that starting vector either with the iteration map in Algorithm 2 or in Algorithm 3 would give us either another eigenvalue $\lambda_{\tilde{\boldsymbol{m}}}$ of equal magnitude, or the eigenvalue with second largest magnitude λ_2, and the corresponding eigenvector. In this manner all desired eigenvalues can be computed via deflation.

Some other acceleration techniques should be mentioned briefly: The convergence speed in power iteration is controlled by the ratio $|\lambda_2/\lambda_1|$, where λ_2 is the next smallest eigenvalue in absolute size. Sometimes this ratio can be very close to unity. If instead with

a single vector $\boldsymbol{v}_i$ one iterates with a whole block $\boldsymbol{v}_i^{(1)}, \ldots, \boldsymbol{v}_i^{(j)}$ of j mutually orthogonal vectors — effectively a subspace — the convergence speed changes to $|\lambda_{1+j}/\lambda_1|$. One has to restore orthogonality though after each iteration sweep [57] of this block- or subspace iteration. This kind of technique also makes it possible to compute multiple eigenvalues, i.e. when the maximum occurs at several places.

Other well-known methods for eigenvalues with even faster convergence build on *Krylov* subspaces [57], here we have symmetric matrices and thus one would use the Lanczos method. These procedures rely on orthogonalisation though, and this may be problematic when combined with truncation. And certainly the Krylov subspace can be combined with the block iteration idea to give block- or subspace-Lanczos methods [45].

"Exponentiating" the power iteration can be easily achieved via the clever idea of [27]. Recall that with the starting vector $\boldsymbol{v}_0 := \mathbf{1}/\sqrt{N}$, power iteration would compute $\boldsymbol{v}_1 \propto \boldsymbol{w}^{\odot 1} = \boldsymbol{w}^{\odot 1} \odot \mathbf{1}$, a scaled version of $\boldsymbol{w}$, corresponding to the action of $\boldsymbol{w}^{\odot 1}$ on $\mathbf{1}$. Starting actually with $\boldsymbol{v}_0 \propto \boldsymbol{w}$ and changing line 2 in Algorithm 2 or Algorithm 3 to $\boldsymbol{u} := \boldsymbol{v}_i^{\odot 2}$, one has $\boldsymbol{v}_1 \propto \boldsymbol{w}^{\odot 2} = \boldsymbol{w}^{\odot 2} \odot \mathbf{1}$, i.e. the action of $\boldsymbol{w}^{\odot 2}$ on $\mathbf{1}$. In the next iteration one has $\boldsymbol{v}_2 \propto \boldsymbol{w}^{\odot 4}$, and in iteration i one has $\boldsymbol{v}_i \propto \boldsymbol{w}^{\odot 2^i}$. Reformulating Algorithm 3 for this new task of "exponentiated" power iteration ($P \leftarrow$ exp-pow) for the new iteration function $\Phi_{\text{exp-pow}}$ results in Algorithm 4, to be used in Algorithm 1 with starting vector $\boldsymbol{v}_0 := \boldsymbol{w}/\|\boldsymbol{w}\|_{\mathcal{T}}$. We also need to keep the auxiliary vector $\boldsymbol{y} = \boldsymbol{w}^{\odot 2}$.

Algorithm 4 One step "exponentiated" power iteration with RQ: $\Phi_{\text{exp-pow}}$

1: Input iterate $\boldsymbol{v}_i$;	▷ assume $\boldsymbol{v}_i \propto \boldsymbol{w}^{\odot 2^i}$ of unit length.
2: $\boldsymbol{u} \leftarrow \boldsymbol{v}_i \odot \boldsymbol{v}_i$;	▷ $\boldsymbol{u} \propto \boldsymbol{L}_{\boldsymbol{w}}^{\odot 2^{i+1}} \mathbf{1} = \boldsymbol{w}^{\odot 2^{i+1}}$.
3: $\varrho_1 \leftarrow \langle \boldsymbol{w} \vert \boldsymbol{u} \rangle_{\mathcal{T}}$;	▷ Rayleigh quotient (RQ) $\lambda_1 \approx \varrho_1$
4: $\varrho_2 \leftarrow \langle \boldsymbol{y} \vert \boldsymbol{u} \rangle_{\mathcal{T}}$;	▷ RQ of $\boldsymbol{w}^{\odot 2}$.
5: $\varepsilon_\lambda \leftarrow (\varrho_2 - \varrho_1^2)^{-1/2}$;	▷ error bound Eq. (36) of λ_1.
6: $\gamma \leftarrow \langle \boldsymbol{u} \vert \boldsymbol{u} \rangle_{\mathcal{T}}^{-1/2}$;	▷ inverse length of $\boldsymbol{u}$.
7: $\boldsymbol{z} \leftarrow \gamma \cdot \boldsymbol{u}$;	▷ normalise output $\boldsymbol{z}$ to unit length.
8: Output $\boldsymbol{z}$;	

Thus the eigenvalue approximation is through the Rayleigh quotient $\varrho_1 = \langle \boldsymbol{w} \vert \boldsymbol{u} \rangle_{\mathcal{T}} = \langle \boldsymbol{w} \vert \boldsymbol{v}_i^{\odot 2} \rangle_{\mathcal{T}} = \langle \boldsymbol{w} \odot \boldsymbol{v}_i \vert \boldsymbol{v}_i \rangle_{\mathcal{T}}$ in line 3 of Algorithm 4. As $\boldsymbol{y} = \boldsymbol{w}^{\odot 2}$, on line 4 the quantity $\varrho_2 = \langle \boldsymbol{y} \vert \boldsymbol{u} \rangle_{\mathcal{T}} = \langle \boldsymbol{w}^{\odot 2} \vert \boldsymbol{v}_i^{\odot 2} \rangle_{\mathcal{T}} = \langle \boldsymbol{w}^{\odot 2} \odot \boldsymbol{v}_i \vert \boldsymbol{v}_i \rangle_{\mathcal{T}}$ computes the RQ of $\boldsymbol{w}^{\odot 2}$. The "eigenvector" $\boldsymbol{v}_i \propto \boldsymbol{w}^{\odot 2^i}$ converges with the rate $|\lambda_2/\lambda_1|^{2^i}$, and hence one achieves *exponential* convergence. This acceleration technique can be used in all eigenvalue computations described here.

Transformed eigenvalue computations are a well-known device to find e.g. the maximum of the function $\boldsymbol{y} = f(\boldsymbol{w})$. After computing $f(\boldsymbol{w})$, one proceeds as before, but with $\boldsymbol{y}$ instead of $\boldsymbol{w}$. Some functions $f(t)$ are simple enough so that the computation of $f(\boldsymbol{w})$ is combined with the iteration. Recall that this transforms all eigenvalues $\lambda_{\boldsymbol{w}}$ to $\lambda_{\boldsymbol{y}} = f(\lambda_{\boldsymbol{w}})$ according to spectral calculus, leaving the eigenvectors unchanged.

Shifting and inverse shifting are functions needed to access other than the point of largest modulus in the spectrum of $\boldsymbol{L}_{\boldsymbol{w}}$ resp. $\boldsymbol{w}$. These functions are the shift $f : t \mapsto t + \beta$ and the inverse shift $f : t \mapsto (t - \rho)^{-1}$.

The simple function of shifting $f(t) = t + \beta$ is needed if the actual element of maximum modulus is a minimum of $\boldsymbol{w}$; then this is a maximum of $-\boldsymbol{w}$. In the iteration this will

be picked up by the Rayleigh-quotient being negative: $\varrho_{\boldsymbol{w}} < 0$. Also note that for this case — where the maximum has smaller absolute value than the minimum — one may *shift* the tensor by a value of $\beta = -\varrho_{\boldsymbol{w}} \approx -\lambda_1$ to $\hat{\boldsymbol{w}} := \boldsymbol{w} + \beta \cdot \mathbf{1}$ so that the maximum of $\hat{\boldsymbol{w}}$ has larger absolute value than the minimum of $\hat{\boldsymbol{w}}$. One then iterates with $\hat{\boldsymbol{w}}$. After determining the maximum of $\hat{\boldsymbol{w}}$, we may subtract the number $\beta > 0$ again, to obtain the maximum of $\boldsymbol{w}$. The index of the maximum of $\hat{\boldsymbol{w}}$ is of course the same as the one for the maximum of $\boldsymbol{w}$. Completely analogous manipulations can be performed to find a minimum which is not of maximum modulus. These and similar techniques are well known from eigenvalue calculations of large / sparse symmetric matrices [24, 57, 58, 65].

The inverse shift $f(t) = (t - \rho)^{-1}$ is needed to access intermediate points in the spectrum $\sigma(\boldsymbol{w})$. Here one finds the data in $\boldsymbol{w}$ closest to ρ, as they have the maximum modulus under the transformation. Thus one computes first $\boldsymbol{y} := (\boldsymbol{w} - \rho \cdot \mathbf{1})^{\odot -1}$ approximately through Algorithm 1 with the iteration function Eq. (28) — with $\check{\boldsymbol{w}} := \boldsymbol{w} - \rho \cdot \mathbf{1}$ instead of $\boldsymbol{w}$ — followed by the eigenvector computation with iteration function as in Algorithm 4, but with starting vector $\boldsymbol{v}_0 := \boldsymbol{y}/\|\boldsymbol{y}\|_{\mathcal{T}}$.

3 Tensor formats

In this section we review definitions and properties of frequently used tensor formats. Many such formats are used in quantum physics under the name *tensor networks*, see [64, 59, 22, 50, 4, 3]. We only look at the *canonical polyadic* (CP), the *Tucker*, and the *Tensor Train* (TT) format. The CP [36] and *Tucker* [62] formats have been well known for a long time and are therefore very popular. The TT format was originally developed in quantum physics and chemistry as "matrix product states" (MPS), see [64] and references therein, and rediscovered in [55, 56] as tensor train.

A new class of tensor formats, which we do not consider here, is the hierarchical tensor (HT) format. It was introduced in [30], and further considered in [28].

We note that the sets of low-rank tensors of fixed rank in the CP format are not closed for $d > 2$, whereas in the Tucker, TT, or HT tensor formats these sets are closed. Therefore the minimisation problem Eq. (39) for the best approximation in a low-rank format has a solution, which can be computed by, for example, an appropriately modified Newton method. The computations in the TT and HT formats are based on the *singular value decomposition* (SVD) [53, 28], and in the Tucker format on the *higher order SVD* (HOSVD) [63]. In all tensor formats the tensor rank doubles for addition and squares for the Hadamard product. To avoid unnecessary and harmful rank growth, the rank is usually truncated, e.g. by ALS-like or other optimisation algorithms [14, 21].

A tensor format is described by a parameter vector space $\mathcal{P} = \bigtimes_{\nu=1}^{d} \mathcal{P}_\nu$, where $\mathcal{P}_\nu = \mathbb{R}^{d_\nu}$, and a multilinear map $U : \mathcal{P} \to \mathcal{T}$ into the tensor space $\mathcal{T} := \bigotimes_{\ell=1}^{d} \mathbb{R}^{M_\ell}$. For practical implementations of high dimensional problems we need to distinguish between a tensor $\boldsymbol{w} \in \mathcal{T}$ and its tensor format representation $\boldsymbol{P} \in \mathcal{P}$, where $\boldsymbol{w} = U(\boldsymbol{P})$. There are many possibilities to define tensor formats. Here, we consider the canonical (CP), the Tucker, and the tensor train (TT) format. In the following we briefly repeat definitions and properties of these formats [31]. We also note that the reader can invent his own tensor format, which especially well fits to his needs.

The CP format is cheap, it is simpler than the Tucker or TT format, but, compared to others, there are no reliable algorithms to compute CP decompositions for $d > 2$ [31, 39]. The Tucker format has stable algorithms [38], but the storage and complexity costs are $\mathcal{O}(d\, r\, n + r^d)$, i.e. they grow exponentially with d. The TT format is a bit more

complicated, but does not have this disadvantage. CP and Tucker rank-structured tensor formats have been applied in chemometrics and in signal processing [61, 7].

3.1 The canonical polyadic tensor format

The canonical representation of multivariate functions [36] was introduced in 1927. A

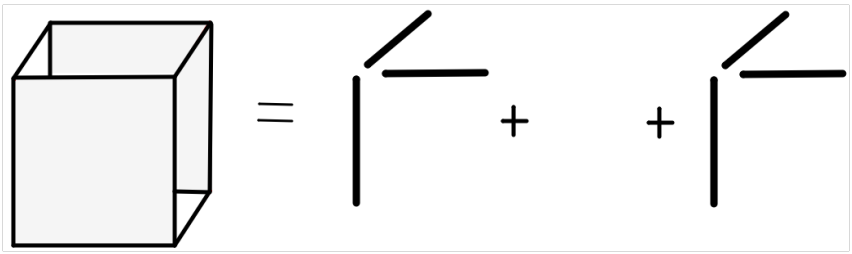

Figure 1: Schema of the CP tensor decomposition of a 3D tensor.

schema of the CP tensor format for $d = 3$ is shown in Figure 1. The full tensor $\boldsymbol{w} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is shown on the left side of the equality sign, and its decomposition on the right. The lines denote the vectors w_{i1}, w_{i2}, and w_{i3} for $i = 1, \ldots, r$, respectively.

Definition 3.1 (Canonical Polyadic (CP) Tensor Format)**.** *The canonical polyadic tensor format in $\mathcal{T}$ for variable r is defined by the multilinear mapping*

$$U_{\mathrm{CP},r} : \mathcal{P}_{\mathrm{CP},r} := \bigtimes_{\nu=1}^{d} \mathcal{P}_\nu^r \;\to\; \mathcal{T}, \quad \mathcal{P}_\nu = \mathbb{R}^{M_\nu}, \tag{37}$$

$$\mathcal{P}_{\mathrm{CP},r} \ni \boldsymbol{P} := (\boldsymbol{w}_i^{(\nu)} : 1 \le i \le r, \quad 1 \le \nu \le d) \;\mapsto\; U_{\mathrm{CP},r}(\boldsymbol{P}) := \boldsymbol{w} = \sum_{i=1}^{r} \bigotimes_{\nu=1}^{d} \boldsymbol{w}_i^{(\nu)} \in \mathcal{T}.$$

We call the sum of elementary tensors $\boldsymbol{w} := U_{\mathrm{CP},r}(\boldsymbol{P})$ a tensor represented in the canonical tensor format with r terms. The system of vectors $\boldsymbol{P} = (\boldsymbol{w}_i^{(\nu)} \in \mathbb{R}^{M_\nu} : 1 \le i \le r, \quad 1 \le \nu \le d)$ is a representation system *of $\boldsymbol{w}$ with* representation rank *r. One may think of $\boldsymbol{P}$ as of a vector valued $r \times d$ matrix with the vector $\boldsymbol{w}_i^{(\nu)} \in \mathbb{R}^{M_\nu}$ at index position (i, ν).*

The storage requirement for $\boldsymbol{w} = U_{\mathrm{CP},r}(\boldsymbol{P})$ is $r \times \sum_{\nu=1}^{d} M_\nu$, and in the simple case $M_1 = \cdots = M_d = n$ it is $\mathcal{O}(r\,d\,n)$.

3.1.1 Basic operations with the canonical format

We denote the set of all tensors $\boldsymbol{w} \in \mathcal{T}$ of rank r by $\mathcal{T}^r$. The set $\mathcal{T}^r$ is a cone, i.e. $\boldsymbol{w} \in \mathcal{T}^r$ implies $\alpha \cdot \boldsymbol{w} \in \mathcal{T}^r$ for $\alpha \in \mathbb{R}$, and at the same time $\mathcal{T}^r$ is not a vector space as for $\boldsymbol{w}_1, \boldsymbol{w}_2 \in \mathcal{T}^r$ one has in general $\boldsymbol{w}_1 + \boldsymbol{w}_2 \notin \mathcal{T}^r$, but $\boldsymbol{w}_1 + \boldsymbol{w}_2 \in \mathcal{T}^{2r}$ [21].

A complete description of fundamental operations in the canonical tensor format and a their numerical cost can be found in [31]. For recent algorithms in the canonical tensor format we refer to [14, 15, 16, 17].

Multiplication by a scalar $\alpha \in \mathbb{R}$ could be done for a $\boldsymbol{w} = U_{\mathrm{CP},r}(\boldsymbol{P})$ as in Eq. (37) by multiplying all of the vectors $\{\boldsymbol{w}_j^{(\nu)}, i = 1, \ldots, r\}$ for any ν by α. But to spread the effect equally and keep the vectors balanced in size, we recommend to define $\alpha_\nu := \sqrt[d]{|\alpha|}$

for all $\nu > 1$, and $\alpha_1 := \mathrm{sign}(\alpha)\sqrt[d]{|\alpha|}$. Then, with a computational cost of $\mathcal{O}(r\,n\,d)$, and without changing the rank,

$$\alpha \cdot \boldsymbol{w} = \sum_{j=1}^{r} \alpha \bigotimes_{\nu=1}^{d} \boldsymbol{w}_j^{(\nu)} = \sum_{j=1}^{r} \bigotimes_{\nu=1}^{d} (\alpha_\nu \boldsymbol{w}_j^{(\nu)}) = \sum_{j=1}^{r} \bigotimes_{\nu=1}^{d} \tilde{\boldsymbol{w}}_j^{(\nu)}, \quad \text{with } \tilde{\boldsymbol{w}}_j^{(\nu)} = \alpha_\nu \boldsymbol{w}_j^{(\nu)}.$$

The sum of two tensors in the CP format $\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{v}$ can be written as follows

$$\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{v} = \left(\sum_{j=1}^{r_u} \bigotimes_{\nu=1}^{d} \boldsymbol{u}_j^{(\nu)} \right) + \left(\sum_{k=1}^{r_v} \bigotimes_{\mu=1}^{d} \boldsymbol{v}_k^{(\mu)} \right) = \sum_{j=1}^{r_u+r_v} \bigotimes_{\nu=1}^{d} \boldsymbol{w}_j^{(\nu)},$$

where $\boldsymbol{w}_j^{(\nu)} := \boldsymbol{u}_j^{(\nu)}$ for $j \le r_u$ and $\boldsymbol{w}_j^{(\nu)} := \boldsymbol{v}_j^{(\nu)}$ for $r_u < j \le r_u + r_v$. The result generally has rank $r_u + r_v$, and as this operation requires only concatenation of memory it has only a computing cost of $\mathcal{O}(1)$.

The Hadamard product $\boldsymbol{w} = \boldsymbol{u} \odot \boldsymbol{v}$ can be written as follows

$$\boldsymbol{w} = \boldsymbol{u} \odot \boldsymbol{v} = \left(\sum_{j=1}^{r_u} \bigotimes_{\nu=1}^{d} \boldsymbol{u}_j^{(\nu)} \right) \odot \left(\sum_{k=1}^{r_v} \bigotimes_{\nu=1}^{d} \boldsymbol{v}_k^{(\nu)} \right) = \sum_{j=1}^{r_u} \sum_{k=1}^{r_v} \bigotimes_{\nu=1}^{d} \left(\boldsymbol{u}_j^{(\nu)} \odot \boldsymbol{v}_k^{(\nu)} \right).$$

The new rank is generally $r_u \times r_v$, and the computational cost is $\mathcal{O}(r_u\, r_v n\, d)$ arithmetic operations.

The Euclidean inner product is computed as follows:

$$\langle \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}} = \langle \sum_{j=1}^{r_u} \bigotimes_{\nu=1}^{d} \boldsymbol{u}_j^{(\nu)} | \sum_{k=1}^{r_v} \bigotimes_{\nu=1}^{d} \boldsymbol{v}_k^{(\nu)} \rangle_{\mathcal{T}} = \sum_{j=1}^{r_u} \sum_{k=1}^{r_v} \prod_{\nu=1}^{d} \langle \boldsymbol{u}_j^{(\nu)} | \boldsymbol{v}_k^{(\nu)} \rangle_{\mathcal{P}_\nu}.$$

The computational cost of the inner product is $\mathcal{O}(r_u\, r_v\, n\, d)$.

We note that in all operations the numerical cost grows only linearly with d, but the representation rank of the resulting tensors may increase. Therefore, a rank truncation procedure is needed, which approximates a given tensor represented in the canonical format with lower rank tensors up to a given accuracy.

3.1.2 Rank truncation in the CP format

Let $\boldsymbol{w}$ be a tensor of rank R. Truncating $\boldsymbol{w}$ to a new rank $r < R$ is a fundamental problem [14]. This problem can be formulated as follows:

$$\text{find a } \boldsymbol{w}^* \text{ with rank } r \text{ such that } \forall \boldsymbol{u} \text{ with rank } r \ : \ \|\boldsymbol{w} - \boldsymbol{w}^*\| \le \|\boldsymbol{w} - \boldsymbol{u}\|.$$

Typical methods to solve this problem are the ALS-method and the Gauss-Newton-method. The ALS method may show slow convergence, see [14, 21] and references therein. The Gauss-Newton method for $d \ge 3$ requires some additional assumptions, and also may not show any convergence at all [14].

It is known, see for instance pp. 91–92 in [39], that the class of rank-r CP tensors is a non-closed set in the corresponding tensor product space for $d > 2$. Therefore there is no $\boldsymbol{w}^*$ as above, and one may look for an ε-solution, i.e. minimising within a deviation of ε from the infimum:

Definition 3.2 (Approximation Problem)**.** *For a given tensor $\boldsymbol{w}$ in CP format with rank R and $\varepsilon > 0$ we are looking for minimal $r_\varepsilon < R$ and a tensor $\boldsymbol{w}^*$ of rank r_ε in CP format, such that:*

$$\|\boldsymbol{w} - \boldsymbol{w}^*\| \leq \varepsilon \|\boldsymbol{w}\|.$$

This problem is discussed in [14, 17].

3.2 The Tucker tensor format

The Tucker tensor format was introduced in [62]. A schema for the Tucker tensor format

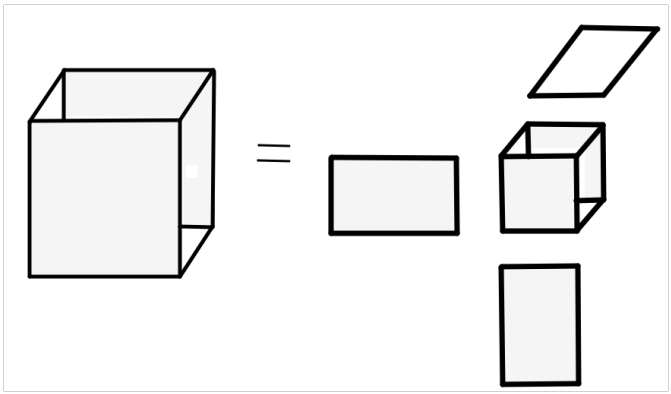

Figure 2: Schema of the Tucker decomposition of a 3D tensor.

for $d = 3$ is shown in Figure 2. The full tensor $\boldsymbol{w} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is shown on the left side of the equality sign, and its decomposition on the right. The small cube on the right side in the Tucker format denotes a core-tensor $\hat{\boldsymbol{w}}$ of size $r_1 \times r_2 \times r_3$, and the rectangles denote matrices of column vectors $\boldsymbol{U}_1 := [\boldsymbol{w}_1^{(1)}, \ldots, \boldsymbol{w}_{r_1}^{(1)}] \in \mathbb{R}^{n_1 \times r_1}$, $\boldsymbol{U}_2 := [\boldsymbol{w}_1^{(2)}, \ldots, \boldsymbol{w}_{r_2}^{(2)}] \in \mathbb{R}^{n_2 \times r_2}$, $\boldsymbol{U}_3 := [\boldsymbol{w}_1^{(3)}, \ldots, \boldsymbol{w}_{r_3}^{(3)}] \in \mathbb{R}^{n_3 \times r_3}$, In this way one has

$$\boldsymbol{w} = \hat{\boldsymbol{w}} \times_1 \boldsymbol{U}_1^{\mathsf{T}} \times_2 \boldsymbol{U}_2^{\mathsf{T}} \times_3 \boldsymbol{U}_3^{\mathsf{T}},$$

where the symbol $\times_k$ denotes the contraction of the k-th tensor index of $\hat{\boldsymbol{w}}$ with the first index of the matrix $\boldsymbol{U}_k^{\mathsf{T}}$ in Figure 2.

Definition 3.3 (Tucker Tensor Format)**.** *The Tucker tensor format in $\mathcal{T}$ with rank parameter $\boldsymbol{r} = (r_1, \ldots, r_d)$ is defined by the multilinear mapping*

$$\begin{aligned} U_{\mathrm{T},\boldsymbol{r}} : \mathcal{P}_{\mathrm{T},\boldsymbol{r}} := \bigtimes_{\nu=1}^{d} \mathcal{P}_\nu^{r_\nu} \times \mathcal{P}_c \to \mathcal{T}, \quad \mathcal{P}_\nu = \mathbb{R}^{M_\nu}\ (\nu = 1, \ldots, d), \quad \mathcal{P}_c = \mathbb{R}^{r_1} \otimes \cdots \otimes \mathbb{R}^{r_d}, \\ \mathcal{P}_{\mathrm{T},\boldsymbol{r}} \ni \boldsymbol{P} := \Big(\boldsymbol{w}_{\nu_i}^{(i)}, \hat{\boldsymbol{w}} = (\hat{w}_{\nu_1,\ldots,\nu_d}) \in \mathcal{P}_c : 1 \leq \nu_i \leq r_i, 1 \leq i \leq d\Big) \qquad (38) \\ \mapsto \quad U_{\mathrm{T},\boldsymbol{r}}(\boldsymbol{P}) := \boldsymbol{w} = \sum_{\boldsymbol{\nu}}^{\boldsymbol{r}} \hat{w}_{\nu_1,\ldots,\nu_d} \bigotimes_{i=1}^{d} \boldsymbol{w}_{\nu_i}^{(i)} \ \in \mathcal{T}, \end{aligned}$$

where $\{\boldsymbol{w}_{\nu_i}^{(i)}\}_{\nu_i=1}^{r_i} \subset \mathbb{R}^{M_i}$ represents a set of orthonormal vectors for $i = 1, \ldots, d$ and $\hat{\boldsymbol{w}} = (\hat{w}_{\nu_1,\ldots,\nu_d}) = (\hat{w}_{\boldsymbol{\nu}}) \in \mathcal{P}_c$ is the so-called Tucker core tensor. We call the sum of elementary tensors $\boldsymbol{w} = U_{\mathrm{T},\boldsymbol{r}}(\boldsymbol{P})$ a tensor represented in the Tucker tensor format with $\prod_{i=1}^{d} r_i$ terms.

Originally, it was applied for tensor decompositions of multidimensional arrays in chemometrics. The Tucker tensor format provides a stable algorithm for decomposition of full-size tensors. The higher order singular value decomposition (HOSVD) and the Tucker ALS algorithm for orthogonal Tucker approximation of higher order tensors were introduced in [8]. The storage cost for the Tucker tensor is bounded by $\mathcal{O}(d\,r\,n + r^d)$, with $r := \max_\ell r_\ell$.

3.2.1 Basic operations with the Tucker format

Let us assume $\boldsymbol{w} = U_{\mathrm{T},\boldsymbol{r}}(\boldsymbol{P})$ in Tucker format as in Eq. (38).

Multiplication by a scalar α can simply be done by multiplying the core tensor $\hat{\boldsymbol{w}}$:

$$\alpha \cdot \boldsymbol{w} = \sum_{\boldsymbol{\nu}}^{\boldsymbol{r}} (\alpha \hat{w}_{\nu_1,\ldots,\nu_d}) \bigotimes_{i=1}^{d} \boldsymbol{w}_{\nu_i}^{(i)}.$$

The computational cost is $\mathcal{O}(\prod_{i=1}^d r_i)$ (or simpler $\mathcal{O}(r^d)$) operations, and no change in rank and storage.

Addition of two Tucker tensors is computed as follows:

$$\begin{aligned}\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{v} &= \sum_{\boldsymbol{\mu}}^{\boldsymbol{\rho}} \hat{u}_{\boldsymbol{\mu}} \bigotimes_{i=1}^{d} \boldsymbol{u}_{\mu_i}^{(i)} + \sum_{\boldsymbol{\nu}}^{\boldsymbol{r}} \hat{v}_{\boldsymbol{\nu}} \bigotimes_{i=1}^{d} \boldsymbol{v}_{\nu_i}^{(i)} \\ &= \sum_{\boldsymbol{\mu}}^{\boldsymbol{\rho}+\boldsymbol{r}} \hat{w}_{\boldsymbol{\mu}} \bigotimes_{i=1}^{d} \boldsymbol{w}_{\mu_i}^{(i)}, \quad \text{— with } \boldsymbol{\mu} = (\mu_1,\ldots,\mu_d), \text{ and similarly } \boldsymbol{\nu}, \boldsymbol{\rho}, \boldsymbol{r} \text{ —}\end{aligned}$$

and where the core tensor $\hat{\boldsymbol{w}} = (\hat{w}_{\mu_1,\ldots,\mu_d}) \in \mathbb{R}^{(\rho_1+r_1)} \otimes \ldots \otimes \mathbb{R}^{(\rho_d+r_d)}$ of order d consists of the two blocks of core tensors $\hat{\boldsymbol{u}}$ and $\hat{\boldsymbol{v}}$, i.e. $(\hat{u}_{\mu_1,\ldots,\mu_d}), (\hat{v}_{\mu_1,\ldots,\mu_d})$ on the main diagonal. In more detail

$$\hat{w}_{\mu_1,\ldots,\mu_d} = \begin{cases} \hat{u}_{\mu_1,\ldots,\mu_d} & \text{if } \mu_1 \le \rho_1, \ldots, \mu_d \le \rho_d \\ \hat{v}_{\mu_1-\rho_1,\ldots,\mu_d-\rho_d} & \text{if } 0 < \mu_1 - \rho_1 \le r_1, \ldots, 0 < \mu_d - \rho_d \le r_d \\ 0 & \text{otherwise.} \end{cases}$$

Similarly, the vectors $\boldsymbol{w}_{\mu_i}^{(i)} \in \mathbb{R}^{M_i}$ are constructed from $\boldsymbol{u}_{\mu_i}^{(i)} \in \mathbb{R}^{M_i}$ and $\boldsymbol{v}_{\nu_i}^{(i)} \in \mathbb{R}^{M_i}$ as follows for $i = 1,\ldots,d$ and $\mu_i = 1,\ldots,\rho_i + r_i$:

$$\boldsymbol{w}_{\mu_i}^{(i)} = \begin{cases} \boldsymbol{u}_{\mu_i}^{(i)} & \text{if } \mu_i \le \rho_i \\ \boldsymbol{v}_{\mu_i-\rho_i}^{(i)} & \text{if } 0 < \mu_i - \rho_i \le r_i. \end{cases}$$

For details see [31, 39]. The computational cost is again $\mathcal{O}(1)$ since only more memory needs to be allocated, but the rank generally increases to the sum of the individual ranks of $\boldsymbol{u}$ and $\boldsymbol{v}$.

The Hadamard product of two tensors given in Tucker format is computed as follows:

$$\boldsymbol{u} \odot \boldsymbol{v} := \sum_{\nu}^{\boldsymbol{r}_u} \sum_{\mu}^{\boldsymbol{r}_v} \hat{u}_{\nu_1,\ldots,\nu_d} \hat{v}_{\mu_1,\ldots,\mu_d} \bigotimes_{i=1}^{d} \left(\boldsymbol{u}_{\nu_i}^{(i)} \odot \boldsymbol{v}_{\mu_i}^{(i)} \right).$$

Note [37] that the new Tucker core has size r^{2d}, i.e. the ranks get multiplied. The storage cost of the Hadamard product is $\mathcal{O}(d\,r^2\,n + r^{2d})$.

The Euclidean inner product is computed as follows:

$$\langle \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}} := \sum_{\nu}^{\boldsymbol{r}_u} \sum_{\mu}^{\boldsymbol{r}_v} \hat{u}_{\nu_1,\ldots,\nu_d} \hat{v}_{\mu_1,\ldots,\mu_d} \prod_{i=1}^{d} \langle \boldsymbol{u}_{\mu_i}^{(i)} | \boldsymbol{v}_{\nu_i}^{(i)} \rangle_{\mathcal{P}_i}.$$

The overall computational complexity of the inner product is $\mathcal{O}(d\, n\, r^2 + r^{2d})$.

3.2.2 Rank truncation in the Tucker format

To truncate the Tucker tensor rank from $2r$ or r^2 back to r the high order SVD (HOSVD) algorithm together with ALS iterations are applied [8]. As the set of Tucker tensors of fixed rank is closed, the best approximation exist, and there is an algorithm with quadratic convergence in the energy norm. One may compare this to the matrix case, where the SVD yields a best low-rank approximation. The truncated tensor $\boldsymbol{W}^*$ resulting from the HOSVD could be sub-optimal, but nevertheless the following inequality holds

$$\|\boldsymbol{W} - \boldsymbol{W}^*\| \le \sqrt{d} \cdot \min_{\boldsymbol{V} \in U_{\mathrm{T},\boldsymbol{r}}(\boldsymbol{P})} \|\boldsymbol{W} - \boldsymbol{V}\|. \tag{39}$$

For some improved estimates see [63].

The most time-consuming part of the Tucker algorithm is the HOSVD procedure. Namely, the computation of the initial guess using the SVD of the matrix unfolding of the original tensor [8]. The numerical cost of Tucker decomposition for full size tensors is $\mathcal{O}(n^{d+1})$.

3.3 The Tensor Train format

The tensor train (TT) format is described in [53, 51, 31, 39]. As already noted, it was originally developed in quantum chemistry as "matrix product states" (MPS), see [64] and references therein, and rediscovered later [55, 56]. For a motivation, let us start with a well-known example [1].

Example 3.4. *Consider the d-dimensional Laplacian operator discretised with standard finite differences over a uniform tensor grid with n degrees of freedom in each direction. It has the Kronecker (canonical) rank-d representation:*

$$\boldsymbol{A} = \boldsymbol{A} \otimes \overbrace{\boldsymbol{I} \otimes \cdots \otimes \boldsymbol{I}}^{d-1 \quad \text{times}} + \boldsymbol{I} \otimes \boldsymbol{A} \otimes \boldsymbol{I} \cdots \otimes \boldsymbol{I} + \cdots + \boldsymbol{I} \otimes \cdots \otimes \boldsymbol{I} \otimes \boldsymbol{A} \in \mathbb{R}^{n^d \times n^d}$$

with $\boldsymbol{A} = \mathrm{tridiag}\{-1, 2, -1\} \in \mathbb{R}^{n \times n}$, and $\boldsymbol{I}$ being the $n \times n$ identity. However, the same operator in the TT format is explicitly representable with all TT ranks equal to 2 for any dimension [1],

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}, & \boldsymbol{I} \end{pmatrix} \bowtie \overbrace{\begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{I} \end{pmatrix} \bowtie \cdots \bowtie \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{I} \end{pmatrix}}^{d-2 \quad \text{times}} \bowtie \begin{pmatrix} \boldsymbol{I} \\ \boldsymbol{A} \end{pmatrix},$$

where the "strong Kronecker" product operation "$\bowtie$" is defined as a regular matrix product for the first level of TT cores, i.e. the block matrices, and the inner blocks are multiplied by means of the Kronecker or tensor product; for example

$$\begin{pmatrix} \boldsymbol{A}, & \boldsymbol{I} \end{pmatrix} \bowtie \begin{pmatrix} \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{A} & \boldsymbol{I} \end{pmatrix} = \begin{pmatrix} \boldsymbol{A} \otimes \boldsymbol{I} + \boldsymbol{I} \otimes \boldsymbol{A}, & \boldsymbol{I} \otimes \boldsymbol{I} \end{pmatrix}.$$

In the element-wise matrix product notation, the Laplace operator reads

$$\boldsymbol{A}[\boldsymbol{i},\boldsymbol{j}] = \begin{pmatrix}\boldsymbol{A}[i_1,j_1] & \boldsymbol{I}[i_1,j_1]\end{pmatrix}\begin{pmatrix}\boldsymbol{I}[i_2,j_2] & 0\\ \boldsymbol{A}[i_2,j_2] & \boldsymbol{I}[i_2,j_2]\end{pmatrix}\cdots\begin{pmatrix}\boldsymbol{I}[i_d,j_d]\\ \boldsymbol{A}[i_d,j_d]\end{pmatrix}.$$

Thus, the d-dimensional Laplacian operator is separable with TT ranks equal to 2, and each TT component (core) is defined by a one-dimensional Laplacian.

Definition 3.5 (TT-Format, TT-Representation, TT-Ranks)**.** *The* TT-tensor format *is for variable* TT-representation ranks $\boldsymbol{r} = (r_0,\ldots,r_d) \in \mathbb{N}^{d+1}$ *— with* $r_0 = r_d = 1$ *and under the assumption that* $d > 2$ *— defined by the following multilinear mapping*

$$\begin{aligned} U_{\mathrm{TT}} : \mathcal{P}_{\mathrm{TT},\boldsymbol{r}} := \bigtimes_{\nu=1}^{d} \mathcal{P}_\nu^{r_{\nu-1}\times r_\nu} \to \mathcal{T}, \quad \mathcal{P}_\nu = \mathbb{R}^{M_\nu}\ (\nu = 1,\ldots,d), \qquad (40)\\ \mathcal{P}_{\mathrm{TT},\boldsymbol{r}} \ni \boldsymbol{P} = (\boldsymbol{W}^{(\nu)} = (\boldsymbol{w}^{(\nu)}_{j_{\nu-1}j_\nu}) \in \mathcal{P}_\nu^{r_{\nu-1}\times r_\nu} : 1 \le j_\nu \le r_\nu, 1 \le \nu \le d)\\ \mapsto U_{\mathrm{TT}}(\boldsymbol{P}) := \boldsymbol{w} = \sum_{j_0=1}^{r_0}\cdots\sum_{j_d=1}^{r_d}\bigotimes_{\nu=1}^{d}\boldsymbol{w}^{(\nu)}_{j_{\nu-1}j_\nu} \in \mathcal{T}. \end{aligned}$$

We call $\boldsymbol{w} := (w_{i_1\ldots i_d}) = U_{\mathrm{TT},\boldsymbol{r}}(\boldsymbol{P})$ *a tensor represented in the train tensor format. Note that the TT-cores* $\boldsymbol{W}^{(\nu)}$ *may be viewed as a vector valued* $r_{\nu-1}\times r_\nu$ *matrix with the vector* $\boldsymbol{w}^{(\nu)}_{j_{\nu-1}j_\nu} \in \mathbb{R}^{M_\nu}$ *with the components* $w^{(\nu)}_{j_{\nu-1}j_\nu}[i_\nu] : 1 \le i_\nu \le M_\nu$ *at index position* $(j_{\nu-1}, j_\nu)$*. The representation in components is then*

$$(w_{i_1\ldots i_d}) = \sum_{j_0=1}^{r_0}\cdots\sum_{j_d=1}^{r_d} w^{(1)}_{j_0j_1}[i_1]\cdots w^{(\nu)}_{j_{\nu-1}j_\nu}[i_\nu]\cdots w^{(d)}_{j_{d-1}j_d}[i_d] \qquad (41)$$

Alternatively, each TT-core $\boldsymbol{W}^{(\nu)}$ *may be seen as a vector of* $r_{\nu-1}\times r_\nu$ *matrices* $\boldsymbol{W}^{(\nu)}_{i_\nu}$ *of length* M_ν*, i.e.* $\boldsymbol{W}^{(\nu)} = (\boldsymbol{W}^{(\nu)}_{i_\nu}) : 1 \le i_\nu \le M_\nu$*. Then the representation Eq. (41) reads*

$$(w_{i_1\ldots i_d}) = \prod_{\nu=1}^{d}\boldsymbol{W}^{(\nu)}_{i_\nu}, \qquad (42)$$

which explains the name matrix product state*. Observe that the first matrix is always a row vector as* $r_0 = 1$*, and the last matrix is always a column vector as* $r_d = 1$*. The matrix components* $\boldsymbol{W}^{(\nu)}_{i_\nu}$ *of the TT-cores are also called "carriages" or "waggons" with "wheels"* i_ν *at the bottom, coupled to the next "carriage" or "waggon" via the matrix product. This explains the* tensor train *name. If one notes more carefully* $\boldsymbol{W}^{(\nu)} \in \mathcal{P}_\nu^{r_{\nu-1}\times r_\nu} = \mathbb{R}^{r_{\nu-1}}\otimes\mathbb{R}^{M_\nu}\otimes\mathbb{R}^{r_\nu}$*, then Eq. (42) can be written more concisely as*

$$\boldsymbol{w} = U_{\mathrm{TT}}(\boldsymbol{P}) = \boldsymbol{W}^{(1)}\times_3^1\boldsymbol{W}^{(2)}\times_3^1\cdots\times_3^1\boldsymbol{W}^{(d)}, \qquad (43)$$

where $\boldsymbol{U}\times_k^\ell\boldsymbol{V}$ *is a contraction of the* k*-th index of* $\boldsymbol{U}$ *with the* ℓ*-th index of* $\boldsymbol{V}$*, where one often writes just* $\times_k$ *for* $\times_k^1$*. Thus in Eq. (43) the contractions leave the indices from the* $\mathbb{R}^{M_\nu}$ *untouched, so that the tensor* $\boldsymbol{w}$ *is formed.*

Each TT-core (or *block*) $\boldsymbol{W}^{(\nu)}$ is defined by $r_{\nu-1}\times r_\nu\times M_\nu$ numbers. Assuming $n = M_\nu$ for all $\nu = 1,\ldots,d$, the total number of entries scales as $\mathcal{O}(d\,n\,r^2)$, which is tractable as long as $r = \max\{r_k\}$ is moderate.

A pictorial representation of the schema for the TT tensor format is shown in Figure 3. It shows d connected waggons with one wheel. The waggons denote the TT-cores, and

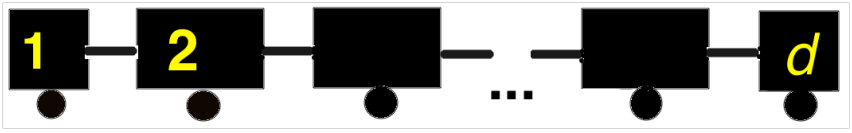

Figure 3: Schema of the TT tensor decomposition. The waggons denote the TT cores and each wheel denotes the index i_ν. Each waggon is connected with neighbours by indices $j_{\nu-1}$ and j_ν.

each wheel denotes the index i_ν. The waggons for $\nu = 2, \dots, (d-1)$ are connected with their neighbours by two indices $j_{\nu-1}$ and j_ν. The first and the last waggons are connected by only one index, namely j_1 and j_{d-1} respectively. Since by the convention in Definition 3.5 above, $r_0 = r_d = 1$, the indices j_0 and j_d run from 1 to 1, i.e. are purely formal.

The waggon or carriage 2 — $\boldsymbol{W}^{(2)}$ — is a tensor of degree 3, described by three indices j_1 (the left hitch), i_2 (the wheel), and j_2 (the right hitch). Multiplication of the third TT-core with the second and forth cores means the tensor contraction by the indices j_1 and j_2. If we perform tensor contraction of all TT-cores over the indices $j_1, \dots, j_{d-1}$, and disregard the purely formal constant indices j_0 and j_d, then the indices $j_0, \dots, j_d$ — the hitches — will disappear, and only the indices $i_1, \dots, i_d$ — the wheels — will be left.

3.3.1 Basic operations with the TT format

We follow to the work of Oseledets [51] and list the major properties of the TT-tensor format.

The multiplication with scalar α could be simply done by multiplying one of the TT-cores $\boldsymbol{W}^{(\nu)}$ in the representation Eq. (43) for any ν in $\boldsymbol{w} = \boldsymbol{W}^{(1)} \times_3^1 \boldsymbol{W}^{(2)} \times_3^1 \cdots \times_3^1 \boldsymbol{W}^{(d)}$. But to balance the effect better, define $\alpha_\nu := \sqrt[d]{|\alpha|}$ for all $\nu > 1$, and $\alpha_1 := \operatorname{sign}(\alpha) \sqrt[d]{|\alpha|}$. Then $\tilde{\boldsymbol{w}} = \alpha \cdot \boldsymbol{w}$ is given by

$$\tilde{\boldsymbol{w}} = (\alpha_1 \cdot \boldsymbol{W}^{(1)}) \times_3^1 (\alpha_2 \cdot \boldsymbol{W}^{(2)}) \times_3^1 \cdots \times_3^1 (\alpha_d \cdot \boldsymbol{W}^{(d)}) = \tilde{\boldsymbol{W}}^{(1)} \times_3^1 \cdots \times_3^1 \tilde{\boldsymbol{W}}^{(d)}.$$

The new cores are given by $\tilde{\boldsymbol{W}}^{(\nu)} = (\tilde{\boldsymbol{W}}^{(\nu)}_{i_\nu}) = (\alpha_\nu \boldsymbol{W}^{(\nu)}_{i_\nu})$, a sequence of new "carriage" matrices. The computational complexity is $\mathcal{O}(d\, n\, r^2)$.

Addition of two TT-tensors Assume two tensors $\boldsymbol{u}$ and $\boldsymbol{v}$ are given in the TT-tensor format as in Eq. (42), i.e. $(u_{i_1 \dots i_d}) = \prod_{\nu=1}^d \boldsymbol{U}^{(\nu)}_{i_\nu}$ and $(v_{i_1 \dots i_d}) = \prod_{\nu=1}^d \boldsymbol{V}^{(\nu)}_{i_\nu}$. The sum $\boldsymbol{w} = \boldsymbol{u} + \boldsymbol{v}$ is given by the new cores $\boldsymbol{W}^{(\nu)}_{i_\nu}$ such that $(w_{i_1 \dots i_d}) = \prod_{\nu=1}^d \boldsymbol{W}^{(\nu)}_{i_\nu}$, where

$$\boldsymbol{W}^{(\nu)}_{i_\nu} = \begin{pmatrix} \boldsymbol{U}^{(\nu)}_{i_\nu} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}^{(\nu)}_{i_\nu} \end{pmatrix}, \quad 1 \le i_\nu \le r_\nu, 2 \le \nu \le d-1.$$

and the first and the last cores will be

$$\boldsymbol{W}^{(1)}_{i_1} = \begin{pmatrix} \boldsymbol{U}^{(1)}_{i_1} & \boldsymbol{V}^{(1)}_{i_1} \end{pmatrix} \quad \text{and} \quad \boldsymbol{W}^{(d)}_{i_d} = \begin{pmatrix} \boldsymbol{U}^{(d)}_{i_d} \\ \boldsymbol{V}^{(d)}_{i_d} \end{pmatrix}.$$

As only storage may have to be concatenated, the computational cost is $\mathcal{O}(1)$, but as the carriages resp. TT-cores grow, the final rank will generally be the sum of the ranks.

The Hadamard product $\boldsymbol{w} = \boldsymbol{u} \odot \boldsymbol{v}$ in the TT format is computed as follows. Assume two tensors $\boldsymbol{u}$ and $\boldsymbol{v}$ are given in the TT tensor format as in Eq. (42), i.e. $(u_{i_1 \dots i_d}) = \prod_{\nu=1}^{d} \boldsymbol{U}_{i_\nu}^{(\nu)}$ and $(v_{i_1 \dots i_d}) = \prod_{\nu=1}^{d} \boldsymbol{V}_{i_\nu}^{(\nu)}$. The Hadamard product is

$$(w_{i_1 \dots i_d}) = (u_{i_1 \dots i_d} \cdot v_{i_1 \dots i_d}).$$

The tensor $\boldsymbol{w}$ has also the TT-tensor format, namely with the new cores

$$\boldsymbol{W}_{i_\nu}^{(\nu)} = \boldsymbol{U}_{i_\nu}^{(\nu)} \otimes_{\mathrm{K}} \boldsymbol{V}_{i_\nu}^{(\nu)}, \quad 1 \le i_\nu \le r_\nu, 1 \le \nu \le d,$$

where $\otimes_{\mathrm{K}}$ is the Kronecker product of two matrices [31]. The rank of $\boldsymbol{W}^{(\nu)} = (\boldsymbol{W}_{i_\nu}^{(\nu)})$ is the product of the ranks of the TT-cores $\boldsymbol{U}^{(\nu)}$ and $\boldsymbol{V}^{(\nu)}$.

The Euclidean inner product of two tensors in the TT-format as in Eq. (40)

$$\boldsymbol{u} = \sum_{j_0=1}^{r_0^u} \cdots \sum_{j_d=1}^{r_d^u} \bigotimes_{\nu=1}^{d} \boldsymbol{u}_{j_{\nu-1} j_\nu}^{(\nu)}, \qquad \boldsymbol{v} = \sum_{j_0=1}^{r_0^v} \cdots \sum_{j_d=1}^{r_d^v} \bigotimes_{\nu=1}^{d} \boldsymbol{v}_{j_{\nu-1} j_\nu}^{(\nu)},$$

with ranks $\boldsymbol{r}^u$ and $\boldsymbol{r}^v$ can be computed as follows:

$$\langle \boldsymbol{u} | \boldsymbol{v} \rangle_{\mathcal{T}} = \sum_{j_0=1}^{r_0^u} \cdots \sum_{j_d=1}^{r_d^u} \sum_{i_0=1}^{r_0^v} \cdots \sum_{i_d=1}^{r_d^v} \prod_{\nu=1}^{d} \langle \boldsymbol{u}_{j_{\nu-1} j_\nu}^{(\nu)} | \boldsymbol{v}_{i_{\nu-1} i_\nu}^{(\nu)} \rangle_{\mathcal{P}_\nu}.$$

The computational complexity is $\mathcal{O}(d\, n\, r^4)$, and can be reduced further [54].

3.3.2 Rank truncation in the TT format

The rank truncation opearation is based on the SVD algorithm and requires $\mathcal{O}(d\, n\, r^3)$ operations [25]. The TT-rounding algorithm (p. 2305 in [56]) is based on QR decomposition and costs $\mathcal{O}(d\, n\, r^3)$.

Corollary 2.4 in [56] states that for a given tensor $\boldsymbol{w}$ and rank bounds r_k, the best approximation to $\boldsymbol{w}$ in the Frobenius norm with TT-ranks bounded by r_k always exist (denote it by $\boldsymbol{w}^*$), and the TT-approximation $\boldsymbol{u}$ computed by the TT-SVD algorithm (p. 2301 in [56]) is quasi-optimal:

$$\|\boldsymbol{w} - \boldsymbol{u}\|_F \le \sqrt{d-1} \|\boldsymbol{w} - \boldsymbol{w}^*\|_F. \tag{44}$$

In [42] the authors suggested a new re-compression randomised algorithm for Tucker and TT tensor formats.

4 Numerical examples

The algorithms for post-processing high-dimensional data, or large volumes of data are shown on a few illustrative examples. Obviously, in any such application the actual computing times and even the possibility of executing such algorithms depend not only on that the huge amounts of data can be compressed to reasonable size, but also on the possibility of executing the algebraic operations with a reasonable speed on the compressed data. Additionally, as the compression may deteriorate in the course of the computation, one must assume that the intermediate results can be compressed to reasonable size, and that this re-compression can be done efficiently. All this depends very much on the data, and currently there are only very few general results to predict when this might or might not occur. Some results in this direction are already contained in [20, 12, 13] for the low-rank TT-format which was also used in our computations.

4.1 Finding the value and location of the maximum element

Already in Table 1 in Section 1 one could see an example from [14], illustrating the kinds of savings which are possible when compared to the method of inspecting every element. Here is another very similar example, again the solution to the standard finite-difference discretisation of the same d-dimensional Poisson equation as detailed in Subsection 1.2 with $n = 100$ discretisation points in every direction. This is the special stationary case $\kappa(\omega, x) \equiv 1$ of Eq. (2) in Subsection 1.1. We are looking for the maximum of the discrete solution.

d	# elements.: $N = n^d$	$\approx$ years [a] inspect. N	actual 3 GHz CPU time [s] of Algorithm 3
25	10^{50}	10^{33}	0.16
50	10^{100}	10^{83}	0.34
100	10^{200}	10^{133}	0.74
200	10^{400}	10^{233}	1.58
400	10^{800}	10^{433}	3.17
800	10^{1600}	10^{1233}	6.26

Age of the universe: $\approx 1.4 \times 10^{10}$ years.

Number of hadrons (elementary particles) in the universe $\approx 10^{80}$.

Table 2: Computing times (4th column) on 3 GHz CPU to find maximum.

Here a 3 GHz CPU was used, so assume — unrealistically — that in each cycle one element could be inspected. Then the third column in Table 2 gives the required number of years to do so on such a machine, whereas the first columns give the dimension and the second column the full number n^d of elements of the solution. Obviously this task could be parallelised, but no amount of conceivable computing power in form of more CPUs could beat the exponential growth of $N = n^d$. Again full storage and the naïve algorithm of inspecting every element are — as far as we know — beyond any possibility in this universe. We used low-rank TT-format storage, and the Algorithm 3. In no case did it need more than 20 iterations. The actual computing times on a 3 GHz CPU are shown in the fourth column, and they behave like $\mathcal{O}(n\,d)$.

4.2 Finding level sets

We show two examples, the first is the same as in the previous Subsection 4.1, wheres the second one is a solution to a stochastic partial differential equation (SPDE).

Level set of the solution to a high-dimensional Poisson equation is the first example. The data vector is the same one as used in Subsection 4.1. But now we want to find the indices of the elements which are in the set $S =]-\infty, 0.25[$, and according to Subsection 2.3 one has to compute the characteristic function χ_S as given in Eq. (22).

For this one needs the sign function given in Eq. (20), computed with the Newton-Schulz iteration function in Eq. (30). The results are shown in Table 3. The first three columns are naturally the same as in Table 2, and the fourth column are the actual computing times in seconds on a 3 GHz CPU, which behave like $\mathcal{O}(n\,d)$.

Level sets of the solution to a stochastic partial differential equation (SPDE) is the next example. This example is described in a bit more detail, see also [20].

d	# elements.: $N = n^d$	$\approx$ years [a] inspect. N	actual 3 GHz CPU iteration time [s] for Newton-Schulz
25	10^{50}	10^{33}	0.63
50	10^{100}	10^{83}	1.30
100	10^{200}	10^{133}	2.66
200	10^{400}	10^{233}	6.03
400	10^{800}	10^{433}	12.97
800	10^{1600}	10^{1233}	25.32

Table 3: Computing times (4th column) on 3 GHz CPU to compute χ_S.

Two scientific software libraries *sglib* and *TensorCalculus* [66, 18] were used. With the *sglib* procedures we discretise the SPDE Eq. (45), and with the *TensorCalculus* we solve the obtained tensor equation and compute the quantities of interests.

Consider the following stationary case of Eq. (2) in Subsection 1.1 a diffusion equation with uncertain diffusion coefficient $\kappa(\omega, x)$:

$$\left.\begin{aligned} -\nabla(\kappa(\omega,x)\nabla u(\omega,x)) &= f(\omega,x) \quad \text{in } \mathcal{G}, \\ u(\omega,x) &= 0 \quad \text{on } \partial\mathcal{G}, \end{aligned}\right\} \text{ a.s. in } \omega \in \Omega, \tag{45}$$

Here $\mathcal{G}$ is the two-dimensional L-shaped domain $[-1,1]^2 \setminus [0,1]^2$, and a triangular mesh with 557 mesh points was used for the spatial piece-wise linear finite element discretisation, and including the boundary values, the discrete spatial solution needs $M_d = 557$ storage locations.

After a stochastic discretisation with the Karhunen-Loève and Polynomial Chaos Expansion, and applying the stochastic Galerkin method as in [43, 46, 12, 13, 20, 19], we can compute the solution $u(\omega, x)$ in a chosen tensor representation.

The random field $\kappa(\omega, x)$ was taken to have a shifted log-normal distribution for, i.e. $\log(\kappa(\omega, x) - 1.1)$ has a normal distribution with parameters $\{\mu = 0.5, \sigma^2 = 1.0\}$. The isotropic and homogeneous covariance function is of Gaussian type with covariance lengths $\ell_x = \ell_y = 0.3$.

Ten Karhunen-Loève terms were taken to represent the mean zero random part of the field $\kappa(\omega, x)$, and for the polynomial chaos expansion (PCE) multivariate Hermite polynomials of maximum second degree were taken.

For the right-hand side $f(\omega, x)$ a β-distribution $\{4, 2\}$ was chosen for the random field. The covariance function is also of Gaussian type with covariance lengths $\ell_x = \ell_y = 0.6$. Again ten terms in the Karhunen-Loève expansion for the zero mean random part of $f(\omega, x)$ with a maximum second degree polynomial chaos were taken.

The total stochastic dimension of the solution $u(\omega, x)$ is 20 — ten random variables from the ten terms of the Karhunen-Loève expansion of the coefficient $\kappa(\omega, x)$, plus ten terms of the Karhunen-Loève expansion of the right hand side $f(\omega, x)$ — i.e. the multi-index α will consist of 20 indices ($\alpha = (\alpha_1, ..., \alpha_{20})$), and the solution will be represented as a function of 20 independent random variables. Together with the extra dimension from the spatial discretisation — which is lumped into one — one has a discrete version of a function $u(\omega, x)$ on $d = 21$ dimensions. With second degree polynomials, there are $M_s = 3$ polynomials for each random variable, thus the stochastic number degrees of freedom is $N_s = M_s^{20} = 3^{20} = 3,486,784,401 \approx 3.5 \times 10^9$. Thus the total number of entries for the full

tensor is $N_d \times N_s = N_d \times M_s^{20} = 557 \times 3,486,784,401 = 1,942,138,911,357 \approx 2 \times 10^{12}$, and the discrete solution $\boldsymbol{u} \in \mathbb{R}^{557} \otimes \bigotimes_{\mu=1}^{20} \mathbb{R}^3 \cong \mathbb{R}^{1,942,138,911,357}$. Thus, even for this relatively coarse discretisation, full storage of just the solution vector would require approximately 16 TB of memory.

For the numerical solution, a low-rank tensor solver was used [48, 12, 13, 20, 19], and the solution was represented in the CP-format Eq. (37) with a rank of $r = 231$. Thus

$$\boldsymbol{u} = \sum_{j=1}^{231} \boldsymbol{u}_{j0} \otimes \bigotimes_{\mu=1}^{20} \boldsymbol{u}_{j\mu} \in \mathbb{R}^{557} \otimes \bigotimes_{\mu=1}^{20} \mathbb{R}^3,$$

and the storage of the solution vector requires only $231 \times (557 + 20 \times 3) = 142,527$ storage locations, i.e. less than 1.15 MB.

$b/\|\mathbf{u}\|_\infty$	rank χ_S	max it rank	$k_{\max}$ its	error
0.2	12	24	12	2.9×10^{-8}
0.4	12	20	20	1.9×10^{-7}
0.6	8	16	12	1.6×10^{-7}
0.8	8	15	8	1.2×10^{-7}

Table 4: Computing $\chi_S(\boldsymbol{u})$.

Finally, we computed $\chi_S(\boldsymbol{u})$ for $S =]-\infty, b[$ by computing $\operatorname{sign}(b\|\boldsymbol{u}\|_\infty \mathbf{1} - \boldsymbol{u})$ for $b \in \{0.2, 0.4, 0.6, 0.8\}$, which is shown in the first column of Table 4. The final representation ranks of $\chi_S(\boldsymbol{u})$ resp. $\operatorname{sign}(b\|\boldsymbol{u}\|_\infty \mathbf{1} - \boldsymbol{u})$ are given in the second column. In this numerical example, the ranks are in all cases smaller then 13. The sign-function was computed with the iteration from Algorithm 1. The iteration function is given in Eq. (30), and the iteration terminated after $k_{\max}$ steps, shown in column four. The maximal representation rank of the iterates is documented in the third column. The error $\|\mathbf{1} - \boldsymbol{u}_{k_{\max}} \odot \boldsymbol{u}_{k_{\max}}\| / \|(b\|\boldsymbol{u}\|_\infty \mathbf{1} - \boldsymbol{u})\|$ is given in the last column. Each computation — each row of the table — took less than 10 min on a 3 GHz CPU.

5 Conclusion

Large volumes of data are used resp. generated in an increasing number of instances in science, technology, and business; some of which were mentioned in Subsection 1.1. Often this data is in the form of a sample of a high dimensional function, or it can be reshaped in this way. It may well happen that it is not possible to store such data in its entirety, so that it has to be compressed in some way. And it is usually not enough to compress these large data sets, efficient numerical algorithms are required to post-process such data. All these post-processing tasks may be trivial in low dimensions and for small data sets, but they become non-trivial for high-dimensional data, say $d > 5$, which is in some compressed format. We have introduced some numerical methods which may partially close this gap.

Here we have formulated algorithms for such post-processing, which only use the structure of an abstract commutative algebra. The Hadamard algebra is an example of such an algebra, and it can be defined for all real valued data sets. The Hadamard product of two tensors turned out to be a key tool in approximating the variance, pointwise inverse, level sets, frequency, and the maximal element of huge multi-dimensional data sets.

In our example application, the compression is low-rank tensor approximation, and we showed how to perform the algebra operations in some example low-rank tensor formats. We assumed that the initial data set has low-rank tensor representation, and showed that this low-rank property can be preserved during the whole computing process.

The choice of the tensor format is not crucial; it could be, for example, the canonical polyadic format, the Tucker format, the tensor train format, or any other. The only requirement is that it should be possible to compute the operations of the Hadamard algebra in a reasonable (linear in n) time, and the existence of stable rank truncation algorithms to truncate possibly large intermediate ranks. The algorithms needed were among others the multiplicative inverse and the sign function, and this could be fashioned after the algorithms for matrix algebras. Finally, the algorithms are demonstrated on some high dimensional data which comes from the solution of high dimensional or parametric resp. stochastic elliptic partial differential equations.

References

[1] V. A. Kazeev and B. Khoromskij, *Low-rank explicit QTT representation of the Laplace operator and its inverse*, SIAM Journal on Matrix Analysis and Applications **33** (2012), 742–758, doi:10.1137/100820479.

[2] U. Benedikt, H. Auer, M. Espig, W. Hackbusch, and A. A. Auer, *Tensor representation techniques in post-Hartree–Fock methods: matrix product state tensor format*, Molecular Physics **111** (2013), no. 16–17, 2398–2413, doi:10.1080/00268976.2013.798433.

[3] J. Biamonte and V. Bergholm, *Quantum tensor networks in a nutshell* [online], arXiv: 1708.00006 [quant-ph], July 2017, Available from: https://arxiv.org/abs/1708.00006.

[4] J. C. Bridgeman and C. T. Chubb, *Hand-waving and interpretive dance: An introductory course on tensor networks*, J. Phys. A: Math. Theor. **50** (2017), 223001, doi:10.1088/1751-8121/aa6dc3.

[5] G. Brumfiel, *Down the Petabyte highway*, Nature **469** (2011), 282–283, doi:10.1038/469282a.

[6] S. R. Chinnamsetty, M. Espig, B. N. Khoromskij, W. Hackbusch, and H.-J. Flad, *Tensor product approximation with optimal rank in quantum chemistry*, The Journal of chemical physics **127** (2007), no. 8, 084110.

[7] A. Cichocki, , and S. Sh. Amari, *Adaptive blind signal and image processing: Learning algorithms and applications*, Wiley, 2002.

[8] L. De Lathauwer, B. De Moor, and J. Vandewalle, *A multilinear singular value decomposition*, SIAM J. Matrix Anal. Appl. **21** (2000), 1253–1278.

[9] S. Dolgov, B. Khoromskij, D. Savostyanov, and I. Oseledets, *Computation of extreme eigenvalues in higher dimensions using block tensor train formats*, Comp. Phys. Communications **185** (2014), no. 4, 1207–1216.

[10] S. V. Dolgov, B. N. Khoromskij, and D. V. Savostyanov, *Superfast Fourier transform using QTT approximation*, J. Chem. Phys. **18** (2012), no. 5, 915–953.

[11] S. Dolgov and B. Khoromskij, *Simultaneous state-time approximation of the chemical master equation using tensor product formats*, Numerical Linear Algebra with Applications **22** (2015), no. 2, 197–219, doi:10.1002/nla.1942.

[12] S. Dolgov, B. N. Khoromskij, A. Litvinenko, and H. G. Matthies, *Computation of the response surface in the tensor train data format*, arXiv:1406.2816 (2014).

[13] S. Dolgov, B. N. Khoromskij, A. Litvinenko, and H. G. Matthies, *Polynomial chaos expansion of random coefficients and the solution of stochastic partial differential equations in the tensor train format*, SIAM/ASA Journal on Uncertainty Quantification **3** (2015), no. 1, 1109–1135, doi:10.1137/140972536.

[14] M. Espig, *Effiziente Bestapproximation mittels Summen von Elementartensoren in hohen Dimensionen*, Ph.D. thesis, Universität Leipzig, Germany, 2008.

[15] M. Espig, L. Grasedyck, and W. Hackbusch, *Black box low tensor rank approximation using fibre-crosses*, Constructive approximation (2009).

[16] M. Espig and W. Hackbusch, *A regularized Newton method for the efficient approximation of tensors represented in the canonical tensor format*, Numerische Mathematik (2012), doi:10.1007/s00211-012-0465-9.

[17] M. Espig, W. Hackbusch, T. Rohwedder, and R. Schneider, *Variational calculus with sums of elementary tensors of fixed rank*, Numerische Mathematik (2012), doi:10.1007/s00211-012-0464-x.

[18] M. Espig, M. Schuster, A. Killaitis, N. Waldren, P. Waehnert, S. Handschuh, and H. Auer, *Tensorcalculus, c++ library*, 2012, Available from: http://gitorious.org/tensorcalculus.

[19] M. Espig, W. Hackbusch, A. Litvinenko, H. G. Matthies, and P. Wähnert, *Efficient low-rank approximation of the stochastic Galerkin matrix in tensor formats.*, Computers & Mathematics with Applications **67** (2014), 818–829, doi:10.1016/j.camwa.2012.10.08.

[20] M. Espig, W. Hackbusch, A. Litvinenko, H. G. Matthies, and E. Zander, *Efficient analysis of high dimensional data in tensor formats*, Sparse Grids and its Applications (J. Garcke and M. Griebel, eds.), LNCSE, vol. 88, Springer-Verlag, 2013, pp. 31–56, doi:10.1007/978-3-642-31703-3_2.

[21] M. Espig, W. Hackbusch, T. Rohwedder, and R. Schneider, *Variational calculus with sums of elementary tensors of fixed rank*, Numerische Mathematik **122** (2012), no. 3, 469–488, doi:10.1007/s00211-012-0464-x.

[22] G. Evenbly and G. Vidal, *Tensor network states and geometry*, J Stat Phys **145** (2011), 891–918, doi:10.1007/s10955-011-0237-4.

[23] V. Eyring, P. M. Cox, G. M. Flato, P. J. Gleckler, G. Abramowitz, P. Caldwell, W. D. Collins, B. K. Gier, A. D. Hall, F. M. Hoffman, G. C. Hurtt, A. Jahn, C. D. Jones, S. A. Klein, J. P. Krasting, L. Kwiatkowski, R. Lorenz, E. Maloney, G. A. Meehl, A. G. Pendergrass, R. Pincus, A. C. Ruane, J. L. Russell, B. M. Sanderson, B. D. Santer, S. C. Sherwood, I. R. Simpson, R. J. Stouffer, and M. S. Williamson, *Taking climate model evaluation to the next level*, Nature Climate Change **9** (2019), no. 2, 102–110, doi:10.1038/s41558-018-0355-y.

[24] G. Golub and C. F. van Loan, *Matrix computations*, Johns Hopkins University Press, Baltimore, MD, 1996.

[25] L. Grasedyck and W. Hackbusch, *An introduction to hierarchical (H-) rank and TT-rank of tensors with examples*, Comput. Methods Appl. Math. **11** (2011), no. 3, 291–304, doi:10.2478/cmam-2011-0016.

[26] L. Grasedyck, D. Kressner, and C. Tobler, *A literature survey of low-rank tensor approximation techniques*, GAMM-Mitteilungen **36** (2013), no. 1, 53–78.

[27] L. Grasedyck and C. Löbbert, *Determine largest element*, Oral communication, March 2019.

[28] L. Grasedyck, *Hierarchical singular value decomposition of tensors*, SIAM Journal on Matrix Analysis and Applications **31** (2010), 2029–2054, doi:10.1137/090764189.

[29] R. Grossman and M. Mazzucco, *DataSpace: A data web for the exploratory analysis and mining of data*, Computing in Science & Engineering **4** (2002), 44–51, doi:10.1109/MCISE.2002.1014979.

[30] W. Hackbusch and S. Kühn, *A new scheme for the tensor representation*, Journal of Fourier Analysis and Applications **15** (2009), no. 5, 706–722, doi:10.1007/s00041-009-9094-9.

[31] W. Hackbusch, *Tensor Spaces and Numerical Tensor Calculus*, Springer, Berlin, 2012, doi:10.1007/978-3-642-28027-6.

[32] W. Hackbusch and B. N. Khoromskij, *Tensor-product approximation to operators and functions in high dimensions*, J. Complexity **23** (2007), no. 4-6, 697–714. MR MR2372023 (2008k:65042)

[33] W. Hackbusch, B. N. Khoromskij, S. Sauter, and E. E. Tyrtyshnikov, *Use of tensor formats in elliptic eigenvalue problems*, Numerical Linear Algebra with Applications **19** (2012), no. 1, 133–151, doi:10.1002/nla.793.

[34] W. Hackbusch, B. N. Khoromskij, and E. E. Tyrtyshnikov, *Approximate iterations for structured matrices*, Numerische Mathematik **109** (2008), no. 3, 365–383, doi:10.1007/s00211-008-0143-0.

[35] N. Higham, *Functions of matrices — theory and computation*, SIAM, Philadelphia, PA, 2008.

[36] F. L. Hitchcock, *The expression of a tensor or a polyadic as a sum of products*, J. Math. Physics **6** (1927), 164–189.

[37] V. Khoromskaia and B. N. Khoromskij, *Tensor numerical methods in quantum chemistry*, Walter de Gruyter GmbH & Co KG, 2018.

[38] B. N. Khoromskij and V. Khoromskaia, *Low rank Tucker-type tensor approximation to classical potentials*, Cent. Eur. J. Math. **5** (2007), no. 3, 523–550 (electronic). MR MR2322828 (2008c:65118)

[39] B. N. Khoromskij, *Tensor numerical methods in scientific computing*, Walter de Gruyter GmbH & Co KG, 2018.

[40] T. Kolda, *Orthogonal tensor decompositions*, SIAM J. Matrix Anal. Appl. **23** (2001), 243–255.

[41] T. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Review **51** (2009), no. 3, 455–500.

[42] D. Kressner and L. Perisa, *Recompression of Hadamard products of tensors in Tucker format*, SIAM Journal on Scientific Computing **39** (2017), no. 5, A1879–A1902, doi:10.1137/16M1093896.

[43] H. G. Matthies and A. Keese, *Galerkin methods for linear and nonlinear elliptic stochastic partial differential equations*, Computer Methods in Applied Mechanics and Engineering **194** (2005), no. 12–16, 1295–1331, doi:10.1016/j.cma.2004.05.027.

[44] H. G. Matthies, *Computable error bounds for the generalized symmetric eigenproblem*, Communications in Applied Numerical Methods **1** (1985), 33–38, doi:10.1002/cnm.1630010107.

[45] H. G. Matthies, *A subspace Lanczos method for the generalized symmetric eigenproblem*, Computers and Structures **21** (1985), 319–325, doi:10.1016/0045-7949(85)90252-4.

[46] H. G. Matthies, *Stochastic finite elements: Computational approaches to stochastic partial differential equations*, Zeitschrift für angewandte Mathematik und Mechanik **88** (2008), no. 11, 849–873, doi:10.1002/zamm.200800095.

[47] H. G. Matthies and R. Ohayon, *Analysis of parametric models — linear methods and approximations* [online], arXiv: 1806.01101 [math.NA], 2018, Available from: http://arxiv.org/1806.01101.

[48] H. G. Matthies and E. Zander, *Solving stochastic systems with low-rank tensor compression*, Linear Algebra and its Applications **436** (2012), 3819–3838, doi:10.1016/j.laa.2011.04.017.

[49] W. Nowak and A. Litvinenko, *Kriging and spatial design accelerated by orders of magnitude: combining low-rank covariance approximations with FFT-techniques*, Mathematical Geosciences **45** (2013), no. 4, 411–435.

[50] R. Orús, *A practical introduction to tensor networks: Matrix product states and projected entangled pair states*, Annals of Physics **349** (2014), 117–158, doi:10.1016/j.aop.2014.06.013.

[51] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Scientific Computing **33** (2011), no. 5, 2295–2317.

[52] I. V. Oseledets, D. V. Savostyanov, and E. E. Tyrtyshnikov, *Linear algebra for tensor problems*, Computing **85** (2009), 169–188, doi:10.1007/s00607-009-0047-6.

[53] I. V. Oseledets and E. Tyrtyshnikov, *Breaking the curse of dimensionality, or how to use SVD in many dimensions*, SIAM J. Scientific Computing **31** (2009), no. 5, 3744–3759.

[54] I. Oseledets, *Matlab TT-toolbox, version 2.2*, 2011, Available from: http://spring.inm.ras.ru/osel/?page_id=24.

[55] I. Oseledets and E. Tyrtyshnikov, *TT-cross approximation for multidimensional arrays*, Linear Algebra and its Applications **432** (2010), 70–88, doi:10.1016/j.laa.2009.07.024.

[56] I. V. Oseledets, *Tensor-train decomposition*, SIAM J. Sci. Comput. **33** (2011), no. 5, 2295–2317, doi:10.1137/090752286.

[57] B. N. Parlett, *The symmetric eigenvalue problem*, SIAM, Philadelphia, PA, 1998.

[58] Y. Saad, *Numerical methods for large eigenvalue problems: Theory and algorithms*, Manchester University Press, Manchester, 1992.

[59] S. Sachdev, *Tensor networks—a new tool for old problems*, Physics **2** (2009), 90, doi:10.1103/Physics.2.90.

[60] I. E. Segal and R. A. Kunze, *Integrals and operators*, 2^{nd} ed., Springer, Berlin, 1978.

[61] A. Smilde, R. Bro, and P. Geladi, *Multi-way analysis with applications in the chemical sciences*, Wiley, 2004.

[62] L. R. Tucker, *Some mathematical notes on three-mode factor analysis*, Psychometrika **31** (1966), 279–311.

[63] N. Vannieuwenhoven, R. Vandebril, and K. Meerbergen, *A new truncation strategy for the higher-order singular value decomposition*, SIAM Journal on Scientific Computing **34** (2012), no. 2, A1027–A1052, doi:10.1137/110836067.

[64] G. Vidal, *Efficient classical simulation of slightly entangled quantum computations*, Phys. Rev. Lett. **91** (2003), no. 14, doi:10.1103/PhysRevLett.91.147902.

[65] D. Watkins, *The matrix eigenvalue problem: GR and Krylov subspace methods*, SIAM, Philadelphia, PA, 2007.

[66] E. Zander, *SGLib - A Matlab Toolbox for Stochastic Galerkin Methods*, February 2010, Available from: http://github.com/ezander/sglib/.

14_tensor_post-proc.tex,v 1.14 2019/05/31 22:38:57 hgm Exp