Computer Physics Communications 246 (2020) 106869

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

Parallel cross interpolation for high-precision calculation of high-dimensional integrals[☆]

Sergey Dolgov[a,1], Dmitry Savostyanov[b,*,2]

[a] University of Bath, Claverton Down, BA2 7AY Bath, UK
[b] University of Brighton, Lewes Road, BN2 4GJ Brighton, UK

A R T I C L E   I N F O

Article history:
Received 1 April 2019
Received in revised form 6 August 2019
Accepted 20 August 2019
Available online 23 August 2019

Keywords:
High-dimensional integration
High precision
Tensor train format
Cross interpolation
Ising integrals
Parallel algorithms

A B S T R A C T

We propose a parallel version of the cross interpolation algorithm and apply it to calculate high-dimensional integrals motivated by Ising model in quantum physics. In contrast to mainstream approaches, such as Monte Carlo and quasi Monte Carlo, the samples calculated by our algorithm are neither random nor form a regular lattice. Instead we calculate the given function along individual dimensions (modes) and use these values to reconstruct its behaviour in the whole domain. The positions of the calculated univariate fibres are chosen adaptively for the given function. The required evaluations can be executed in parallel along each mode (variable) and over all modes.

To demonstrate the efficiency of the proposed method, we apply it to compute high-dimensional Ising susceptibility integrals, arising from asymptotic expansions for the spontaneous magnetisation in two-dimensional Ising model of ferromagnetism. We observe strong superlinear convergence of the proposed method, while the MC and qMC algorithms converge sublinearly. Using multiple precision arithmetic, we also observe exponential convergence of the proposed algorithm. Combining high-order convergence, almost perfect scalability up to hundreds of processes, and the same flexibility as MC and qMC, the proposed algorithm can be a new method of choice for problems involving high-dimensional integration, e.g. in statistics, probability, and quantum physics.

1. Introduction

High-dimensional integrals often occur in quantum mechanics [1], in statistics and probability, e.g. expectations with multivariate probability distributions [2], inverse problems with uncertainty [3], and many more. Analytical formulae for them are rarely available, hence numerical approaches become the mainstream approach. Unfortunately, high-dimensional integrals are notoriously difficult for numerical methods as well. A naïve approach, based on tensor product of one-dimensional quadrature rules, requires the total number of function evaluations N that grows exponentially with the problem dimension d, exceeding the possibilities of modern computers for $d \gtrsim 10$. This behaviour, known as the *curse of dimensionality*, motivates development of special methods for the integration in higher dimensions. Currently the most popular methods are the Monte Carlo quadrature [4], quasi Monte Carlo [5–8], Markov chain Monte Carlo [2], and their derivatives such as multilevel Monte Carlo methods [9–12]. These algorithms are rigorously studied and many theoretical results are available, including error bounds which typically do not depend on problem dimension d for problems of interest. Unfortunately, MC and qMC methods converge slowly — the relative accuracy ε depends on the number of function evaluations N_{eval} as $\varepsilon \sim N_{\text{eval}}^{-\alpha}$, where the convergence rate $\alpha = 0.5$ for MC and $0.5 \leqslant \alpha \leqslant 1$ for qMC. The numerical costs therefore grow quickly when higher precision is required, making calculations expensive, prohibitively long, or impossible. Methods based on Smolyak's sparse grids [13–15] are often used to mitigate, but cannot fully remove, the curse of dimensionality.

In this paper we consider a problem of numerical integration of a multivariate function in a simple tensor-product domain such as free space $\mathbb{R}^d$ or hypercube $[0,1]^d$. We follow the naïve approach and use a tensor product of univariate quadrature rules, hence reducing the problem to calculation and summation over the entries of a multi-dimensional array (which we call *tensor*). To overcome the curse of dimensionality, we approximate the whole array based on a few entries from it, but avoid calculating the whole array. To achieve this, we develop and use the parallel version of the tensor cross interpolation algorithm proposed by one of the authors in [16]. This algorithm interpolates the given

☆ Equal contributions. The order of authors is alphabetical.
* Corresponding author.
E-mail addresses: S.Dolgov@bath.ac.uk (S. Dolgov), d.savostyanov@brighton.ac.uk (D. Savostyanov).
1 Supported by the EPSRC, UK fellowship EP/M019004/1.
2 Supported by the EPSRC, UK grant EP/P033954/1 and the Rising Stars grant by the University of Brighton, UK.

https://doi.org/10.1016/j.cpc.2019.106869
0010-4655/© 2019 The Authors. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

array in the *tensor train* (TT) decomposition [17,18], essentially performing *separation of variables*. The array entries are evaluated along one-dimensional lines or *fibres*, each of which is formed by freezing all indices of the multivariate function and only varying one. The lines intersect forming *crosses*, and on the positions of each cross the constructed approximation *interpolates* the data exactly. The positions of the crosses, and hence the nodes of the quadrature rule, are chosen adaptively for the given function, following the maximum-volume method [19,20]. Using the interpolant, various observables, including the integral of the function, can be computed in linear in d time.

Essentially, the proposed algorithm reconstructs all n^d values of the function $f(x_1,\dots,x_d)$ on a tensor product $n\times\cdots\times n$ quadrature grid from a linear in d number of samples, which are adapted specifically to f. This adaptivity allows the proposed algorithm to locate important samples (e.g. areas of concentration of density) and reach faster convergence, compared to mainstream numerical methods, such as MC and qMC, where the positions of the samples are either not optimised, or are optimal for a wide class of functions. For the family of Ising integrals, considered in the numerical experiments section of this paper, the proposed algorithm demonstrates high-order convergence of the order of $\varepsilon \sim N_{\text{eval}}^{-7}$, clearly outperforming MC and qMC. Using multiple precision arithmetic, we were able to compute an integral in more than thousands dimensions to more than hundred decimal digits, observing exponential convergence of the proposed method. As a flexible and non-intrusive algorithm, it can become a new method of choice for problems involving numerical integration in higher dimensions.

Data-sparse algorithms based on tensor product decompositions (canonical polyadic [21], Tucker [22], tensor train (TT) [17] or Hierarchical Tucker (HT) [23]) have a long history of development [24–27], with applications in quantum physics and chemistry [28–32], signal processing [33,34], plasma modelling [35], stochastics and uncertainty quantification [36,37], and fractional calculus [38,39]. However, scalable *high performance* implementation of tensor product algorithms is a relatively new area of research. A straightforward idea is to parallelise dense tensor algebra in computations of factors of a decomposition [40]. However, this typically requires all-to-all communications which limits the scalability. Another strategy is to parallelise a tensor decomposition over different factors, or *dimensions*. One of the first examples of such approach was the parallel density matrix renormalisation group (DMRG) algorithm [41] for ground state computations in quantum physics. In mathematical community this research direction started with dimension-parallel linear solver [42] and cross algorithms in HT format [43]. The main difficulty of parallelisation over dimension is the need of *algorithmic modifications*, since state of the arts tensor algorithms were designed in intrinsically sequential way. Ideally, such modifications should not compromise numerical stability or convergence for the sake of parallel efficiency.

In this paper we develop a parallel version of the TT cross interpolation algorithm [16]. The parallel algorithm is adaptive and converges with the same rate as the sequential version. It involves only local communications with constant loading of processes, and demonstrates almost perfect scaling up to the ultimate partitioning where each process is responsible for a single direction (mode, variable). Moreover, further speedup can be achieved using OpenMP parallelisation of tensor algebra in each process.

The rest of the paper is organised as follows. In Section 2 we recall the cross interpolation method for matrices and provide necessary definitions. In Section 3 we discuss how the matrix interpolation can be applied for high-dimensional arrays (tensors). We compare the existing methods and explain why the cross interpolation algorithm proposed by one of the authors in [16] seems to be the most suitable for parallelisation over the dimensions. We then present the parallel version of this algorithm. In Section 4 we explain how the cross interpolation algorithm can be applied for numerical integration. We also introduce more formally the MC and qMC methods. In Section 5 we introduce Ising susceptibility integrals which will be our main numerical example in this paper. We demonstrate that the proposed method achieves high-order (sometimes exponential) convergence, while the convergence of MC and qMC remains sublinear. In the conclusion, we briefly summarise the results of this paper and discuss some challenges and potential directions for future work.

2. Cross interpolation: notation, definitions and algorithms

2.1. Cross interpolation of matrices

Cross interpolation is based on a simple observation: for a given $m\times n$ matrix $A = [A(i,j)]_{i,j=1}^{m,n}$ its rank-r interpolation can be recovered from its r columns $\mathcal{J} = \{\mathcal{J}^{(t)}\}_{t=1}^r$ and r rows $\mathcal{I} = \{\mathcal{I}^{(s)}\}_{s=1}^r$ as follows:

$$\begin{aligned} A(i,j) \approx \tilde{A}(i,j) &= \sum_{s=1}^{r}\sum_{t=1}^{r} A(i,\mathcal{J}^{(t)})[A(\mathcal{I},\mathcal{J})]^{-1}_{t,s}A(\mathcal{I}^{(s)},j) \\ &= A(i,\mathcal{J})[A(\mathcal{I},\mathcal{J})]^{-1}A(\mathcal{I},j). \end{aligned} \tag{1}$$

Remark 1 (*Compression*)**.** To compute the right-hand side we use only the elements of selected columns $A(i,\mathcal{J}^{(t)})$, and rows $A(\mathcal{I}^{(s)},j)$. Other elements of A are not required to construct $\tilde{A}$ and we can avoid calculating them. Thus, evaluation and storage of $\tilde{A}$ requires $(mr+nr-r^2)$ matrix elements — this is more cost-efficient than working with the whole matrix A if $r \ll \min(m,n)$.

Due to the shape of the *locus* of computed entries, shown in Fig. 1, this decomposition is known as *skeleton* [44], *pseudo-skeleton* (if the exact inverse is replaced with, say, pseudo-inverse) [45], or *cross* [46].

2.2. Notation for matrices and submatrices

Eq. (1) is understood element-wisely, i.e. holds for all possible values of free indices i and j. According to the matrix multiplication rule, the summation is performed over the summation indices from the sets $\mathcal{I}$ and $\mathcal{J}$. Note that the sums are taken over the indices which are repeated in the formula, cf. Einstein's summation convention [47]. Notation $A(\mathcal{I},\mathcal{J})$ refers to a submatrix on the intersection of rows $\mathcal{I}$ and columns $\mathcal{J}$, mimicking the intuitive syntax of programming languages like Fortran90, Matlab, R and Julia, where a vector of indices can be passed into an array to select a subsection of it, e.g. `A(1:2,1:3)` for a 2×3 leading submatrix of A. We can also use index sets $\mathbb{I}=\{1,\dots,m\}$ and $\mathbb{J}=\{1,\dots,n\}$ to refer to full columns and rows. For instance, the approximant $\tilde{A}$ in (1) is a product of three matrices:

- $m\times r$ matrix of columns $A(\mathbb{I},\mathcal{J}) = [A(i,j)]_{i\in\mathbb{I},j\in\mathcal{J}}$;
- inverse of the $r\times r$ submatrix at the intersection $A(\mathcal{I},\mathcal{J}) = [A(i,j)]_{i\in\mathcal{I},j\in\mathcal{J}}$;
- $r\times n$ matrix of rows $A(\mathcal{I},\mathbb{J}) = [A(i,j)]_{i\in\mathcal{I},j\in\mathbb{J}}$.

Embracing this notation, we will keep the same letter A for all three factors of the cross interpolation. Compared to the *CGR* notation [45,48] or *CUR* notation [49], our notation in (1) highlights that factors of the cross decomposition are submatrices of the given matrix A, which distinguishes it from SVD, QR and LU factorisations.

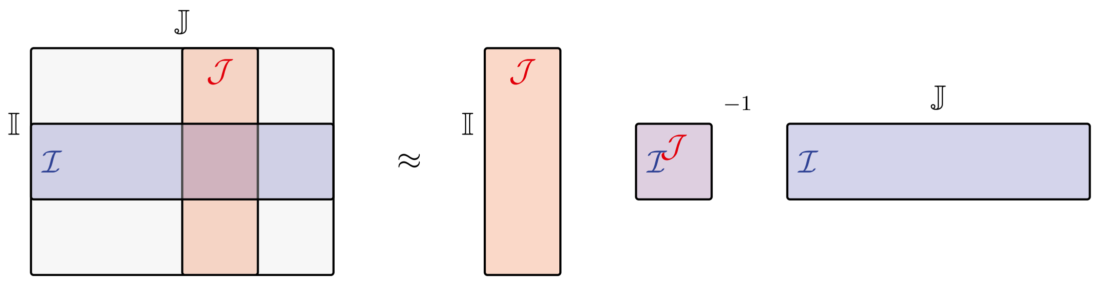


Fig. 1. Cross interpolation for matrices. The full matrix A is approximated by a low-rank decomposition $\tilde{A}$ based on a small number of columns and rows computed in A. Note that the approximation (1) is exact in the positions of computed rows and columns.

2.3. Maximum volume principle

The approximation $A \approx \tilde{A}$ is exact on the positions of computed rows $\mathcal{I}$ and columns $\mathcal{J}$, which is why we call it *interpolation*. For other entries the mismatch between A and $\tilde{A}$ can be arbitrary large in general, because the approximation $\tilde{A}$ does not use information about the matrix A apart of its few chosen columns and rows. Theoretical error upper bounds can be obtained based on additional properties of the matrix, e.g. when $A = [f(x_i, y_j)]_{i,j=1}^{m,n}$ is generated by asymptotically smooth function [50]. However, the quality of the cross approximation $\tilde{A}$ depends critically on a choice of good positions $(\mathcal{I}, \mathcal{J})$ for the cross. Good theoretical estimates are available for the maximum-volume cross, i.e. such that $A(\mathcal{I}, \mathcal{J})$ has the largest possible *volume*

$$\text{vol}\, A(\mathcal{I}, \mathcal{J}) = |\det A(\mathcal{I}, \mathcal{J})|$$

of all submatrices of this size. The maximum-volume principle for matrix approximation was first proposed in [19,20,48], and the estimates were later generalised to other norms [51,52], and rectangular submatrices [53,54].

Unfortunately, the search for a maximum-volume submatrix is NP-hard [55] and cheaper alternative algorithms are required for practical calculations with large matrices.

2.4. Practical algorithms for matrix cross interpolation

When matrix A is available in full, reliable algorithms for low-rank approximation are available, such as the famous singular value decomposition (SVD) [56], and faster rank-revealing QR [57] and LU [58] algorithms. However, these approaches are unfeasible for very large-scale matrices, e.g. those coming from high-dimensional problems, when even $\mathcal{O}(mn)$ costs become prohibitive.

To compute a sufficiently good cross with sublinear costs, the *incomplete cross approximation* [46] algorithm was proposed, that increases the volume of the intersection matrix by alternating updates of rows $\mathcal{I}$ and columns $\mathcal{J}$. If the set of columns $\mathcal{J}$ is fixed, there is a combinatorial number of possible row sets $\mathcal{I}$ to compare. To keep costs feasible, rows $\mathcal{I}$ are updated one-by-one with a greedy algorithm first suggested by Donald Knuth [59]. Greedy updates of rows, shown in Alg. 1, continue until the volume is large enough. Then rows are fixed and columns are updated, and the algorithm alternates until vol $A(\mathcal{I}, \mathcal{J})$ is significantly large. The details of this maxvol algorithm for matrix cross interpolation are given in [20,46].

A conceptually simpler *adaptive cross approximation* (ACA) algorithm [60] follows a greedy optimisation approach by increasing the interpolating sets by one columns and row at a time. It can be seen as a Gaussian elimination with partial column pivoting [61], which is computationally cheap but may result in exponential amplification 2^r of the error. A more conservative complete pivoting is believed to be numerically stable [62], but involves a search through all matrix elements, and thus is more expensive. A good alternative is the rook pivoting [63], which searches for a pivot $(i^\star, j^\star)$ that is dominant in its own row and columns:

$$|A(i^\star, j^\star) - \tilde{A}(i^\star, j^\star)| \geqslant |A(i, j) - \tilde{A}(i, j)|, \quad \text{for all } (i, j) \text{ such that } i = i^\star \text{ or } j = j^\star \tag{2}$$

Rook pivoting avoids exponential deterioration of the error [64,65] and in practice seems to have the same asymptotical complexity as partial pivoting, thus combining the best of both worlds. We use rook pivoting in combination with random pivoting, as shown in Alg. 2.

Algorithm 1 One step of the practical row selection algorithm [59]

```
Input: Sets $(\mathcal{I}, \mathcal{J})$ of the interpolation (1)
1: $B(\mathbb{I}, \mathcal{I}) \leftarrow A(\mathbb{I}, \mathcal{J})[A(\mathcal{I}, \mathcal{J})]^{-1}$.    % $m \times n$ matrix with $B(\mathcal{I}, \mathcal{I}) = I$
2: $(i^\star, i^\dagger) \leftarrow \arg\max_{(i,j)\in\mathbb{I}\times\mathcal{I}} |B(i, j)|$    % $(i^\star, i^\dagger) \in \mathbb{I} \times \mathcal{I}$
Output: Updated row set $\mathcal{I} \leftarrow \mathcal{I} \cup \{i^\star\} \setminus \{i^\dagger\}$ with vol $A(\mathcal{I}, \mathcal{J}) \leftarrow$ vol $A(\mathcal{I}, \mathcal{J})|B(i^\star, i^\dagger)|$.
```

Algorithm 2 One step of the matrix cross interpolation algorithm

```
Input: Sets $(\mathcal{I}, \mathcal{J})$ of the interpolation (1)
1: Pick a random set of samples $\mathcal{L} = \{(i, j)\}$ and choose the one with the largest error,
       $(i^\star, j^\star) \leftarrow \arg\max_{(i,j)\in\mathcal{L}} |A(i, j) - \tilde{A}(i, j)|$
2: repeat    % column and row partial pivoting updates
3:     $(i^\star, j^\star) \leftarrow \arg\max_{i\in\mathbb{I}} |A(i, j^\star) - \tilde{A}(i, j^\star)|$
4:     $(i^\star, j^\star) \leftarrow \arg\max_{j\in\mathbb{J}} |A(i^\star, j) - \tilde{A}(i^\star, j)|$
5: until rook condition (2) is met or computational budget is exhausted
Output: Expanded index sets $\mathcal{I} \leftarrow \mathcal{I} \cup \{i^\star\}$, $\mathcal{J} \leftarrow \mathcal{J} \cup \{j^\star\}$
```

Remark 2 (*Numerical Complexity*)**.** If $|\mathcal{L}| = \mathcal{O}(m + n)$, a single rank-one update step evaluates $\mathcal{O}(m+n)$ matrix entries and performs $(m+n)r$ additional operations. Thus r steps of Algorithm 2 produce the rank-r interpolation (1) using $\mathcal{O}((m + n)r)$ matrix elements plus $\mathcal{O}((m + n)r^2)$ additional operations.

Remark 3 (*Accuracy*)**.** Algorithm 2 does not access all elements of a matrix and therefore is *heuristic*, i.e. its accuracy cannot be guaranteed in general.

Algorithm 2 is written in a very general way and many details are clearly improvable. For example, the choice of $\mathcal{L} = \{(i,j)\}$ for initial sampling can be optimised to ensure $i \notin \mathcal{I}$ and $j \notin \mathcal{J}$ since the error $A - \tilde{A}$ is zero on the positions of the cross. A variety of other heuristic tricks were proposed, e.g. Mahoney et al. [49] suggest to estimate the column and row norms of A and sample $(i,j) \in \mathcal{L}$ with probabilities proportional to these norms. The focus of this paper is not the 'best heuristic' for the matrix case, but the extension to high-dimensional problems. We refer the reader to [66] for a review of matrix low-rank approximation algorithms.

3. Cross approximation and cross interpolation in higher dimensions

3.1. Notation for tensors and multi-indices

We consider an array $A = [A(i_1,\dots,i_d)]$ with d indices i_k, $k = 1,\dots,d$, which are also called *dimensions* or *modes*. Each index assumes values $i_k \in \mathbb{I}_k = \{1,\dots,n_k\}$, where n_k is called the *mode size*. Such arrays are called *tensors* in numerical linear algebra (NLA) community [61], although we do not differentiate upper and lower indices, as it is customary for tensors in mathematical physics [47]. The total storage required for A grows exponentially with the dimension, prohibiting work with full A for large d. Hence, tensor product representations are required for all practical calculations with tensors.

At the heart of tensor product formats lies the idea of *separation of indices*. Consider grouping indices $i_1,\dots,i_k$ together and separating them from the group $i_{k+1},\dots,i_d$, thus reshaping $n_1 \times n_2 \times \dots \times n_d$ tensor A into a $(n_1 \cdots n_k) \times (n_{k+1} \cdots n_d)$ matrix

$$A^{\{k\}}(i_{\leqslant k}, i_{>k}) = A^{\{k\}}(i_1 i_2 \dots i_k; i_{k+1} \dots i_d) = A(i_1, i_2, \dots, i_d),$$

called kth matricization or unfolding of the tensor. As in (1), the equation is understood element-wisely for all possible values of all indices, i.e. $A^{\{k\}}$ differs from A only by 'shape'. Rows and columns of $A^{\{k\}}$ are enumerated by multi-indices

$$i_{\leqslant k} = i_1 i_2 \dots i_k \in \mathbb{I}_1 \times \mathbb{I}_2 \times \dots \times \mathbb{I}_k,$$
$$i_{>k} = i_{k+1} \dots i_d \in \mathbb{I}_{k+1} \times \dots \times \mathbb{I}_d.$$

To separate row and column (multi)-indices $i_{\leqslant k}$ and $i_{>k}$, we apply matrix interpolation formula (1) to $A^{\{k\}}$, yielding

$$\begin{aligned} A^{\{k\}}(i_{\leqslant k}, i_{>k}) &\approx \tilde{A}^{\{k\}}(i_{\leqslant k}, i_{>k}) \\ &= A^{\{k\}}(i_{\leqslant k}, \mathcal{I}_{>k})[A^{\{k\}}(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})]^{-1} A^{\{k\}}(\mathcal{I}_{\leqslant k}, i_{>k}). \end{aligned} \tag{3}$$

Here $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ indicate the positions of r_k rows and columns of the interpolation cross in the unfolding $A^{\{k\}}$.

3.2. Tensor train format

The use of element-wise notation allows us to drop the superscript for the unfolding, because the dimensions of matrices and tensors are unambiguous from the range of the variables within. Hence, Eq. (3) can be simplified to

$$A(i_1,\dots,i_d) \approx A(i_1,\dots,i_k,\mathcal{I}_{>k})[A(\mathcal{I}_{\leqslant k},\mathcal{I}_{>k})]^{-1}A(\mathcal{I}_{\leqslant k}, i_{k+1},\dots,i_d), \tag{4}$$

that emphasises separation of left and right groups of indices. By continuing the separation process, we arrive to the decomposition where all indices i_k are isolated:

$$\begin{aligned} A(i_1,\dots,i_d) &\approx \tilde{A}(i_1,\dots,i_d) \\ &= A(i_1,\mathcal{I}_{>1})[A(\mathcal{I}_{\leqslant 1},\mathcal{I}_{>1})]^{-1}A(\mathcal{I}_{\leqslant 1}, i_2, \mathcal{I}_{>2}) \\ &\quad \times [A(\mathcal{I}_{\leqslant 2},\mathcal{I}_{>2})]^{-1} \cdots A(\mathcal{I}_{\leqslant d-1}, i_d). \end{aligned} \tag{5}$$

This formula is a direct generalisation of skeleton/cross interpolation (1) to tensor case and is therefore called skeleton/cross tensor decomposition [18]. Note that the factors of the cross decomposition are constructed from *fibres* $A(\mathcal{I}_{\leqslant k-1}, i_k, \mathcal{I}_{>k})$ of the given tensor, while in a general tensor train (TT) decomposition [17] we deal with general factors. TT decomposition is itself a particular case of more general Hierarchical Tucker (HT) decomposition [23,67]. Cross approximation algorithms are available for HT format [68,69], as well as for more specialised tensor formats, including Tucker [22] and canonical polyadic decomposition [21].

Remark 4 (*Compression*)**.** The right-hand side of (5) involves $\sum_{k=1}^{d} r_{k-1} n_k r_k - \sum_{k=1}^{d-1} r_k^2 = \mathcal{O}(dnr^2)$ entries[3] of the tensor A.

In general, tensor cross decomposition (5) is an approximation, but not an interpolation formula for A. The following result [16, Theorem 4] provides the sufficient condition for (5) to be called tensor cross interpolation.

Theorem 1 (*Interpolation, see [16]*)**.** *If the crosses* $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ *are nested:*

$$\mathcal{I}_{\leqslant k+1} \subset \mathcal{I}_{\leqslant k} \times \mathbb{I}_{k+1}, \qquad \mathcal{I}_{>k} \subset \mathbb{I}_{k+1} \times \mathcal{I}_{>k+1}, \qquad k = 1,\dots,d-1, \tag{6}$$

formula (5) interpolates the evaluated entries of the tensor,

$$A(\mathcal{I}^{\leqslant k-1}, i_k, \mathcal{I}^{>k}) = \tilde{A}(\mathcal{I}^{\leqslant k-1}, i_k, \mathcal{I}^{>k}), \qquad k = 1,\dots,d.$$

Theorem 1 cannot be reversed, i.e. nestedness of indices is not necessary for the interpolation, as shown by the following.

Theorem 2 (*Exact Recovery of the Exact-rank Tensor*)**.** *If* rank $A^{\{k\}} = r_k$ *for all* $k = 1,\dots,d-1$, *and all submatrices* $A(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ *are non-singular, the formula (5) recovers the original tensor exactly,* $A(i_1,\dots,i_d) = \tilde{A}(i_1,\dots,i_d)$.

This theorem was first proven in [18] with the additional requirement of nestedness. In the exact-rank case the requirement of nestedness can be relaxed, as we show here.

Proof. If rank $A^{\{k\}} = r_k$, the rank-r_k interpolation (3) recovers the unfolding $A^{\{k\}}$ exactly, which means that (4) is also exact,

$$A(i_1,\dots,i_d) = A(i_1,\dots,i_k,\mathcal{I}_{>k})[A(\mathcal{I}_{\leqslant k},\mathcal{I}_{>k})]^{-1}A(\mathcal{I}_{\leqslant k}, i_{k+1},\dots,i_d). \tag{7}$$

We start by applying (7) with $k = 1$, yielding

$$A(i_1,\dots,i_d) = A(i_1,\mathcal{I}_{>1})[A(\mathcal{I}_{\leqslant 1},\mathcal{I}_{>1})]^{-1}A(\mathcal{I}_{\leqslant 1}, i_2,\dots,i_d).$$

Writing (7) with $k = 2$, we get

$$A(i_1, i_2,\ i_3,\dots,i_d) = A(i_1,i_2,\mathcal{I}_{>2})[A(\mathcal{I}_{\leqslant 2},\mathcal{I}_{>2})]^{-1}A(\mathcal{I}_{\leqslant 2}, i_3,\dots,i_d). \tag{8}$$

This is true for all $i_1 \in \mathbb{I}_1$, so it is also true for $i_1 \in \mathcal{I}_1 = \mathcal{I}_{\leqslant 1}$, because $\mathcal{I}_1 \subset \mathbb{I}_1$. Reducing (8) to $A(\mathcal{I}_{\leqslant 1}, i_2,\dots,i_d)$ and plugging into the previous equation, we obtain

$$\begin{aligned} A(i_1,\dots,i_d) &= A(i_1,\mathcal{I}_{>1})[A(\mathcal{I}_{\leqslant 1},\mathcal{I}_{>1})]^{-1}A(\mathcal{I}_{\leqslant 1}, i_2, \mathcal{I}_{>2}) \\ &\quad \times [A(\mathcal{I}_{\leqslant 2},\mathcal{I}_{>2})]^{-1}A(\mathcal{I}_{\leqslant 2}, i_3,\dots,i_d). \end{aligned}$$

Continuing in the same way for $k = 3,\dots,d-1$, we complete the proof. □

[3] In all complexity estimates we assume $n_1 \sim n_2 \sim \dots \sim n_d \sim n$ and $r_1 \sim r_2 \sim \dots \sim r_{d-1} \sim r$.

Algorithm 3 Left-to-right sweep of the ALS maxvol cross approximation algorithm [18]

```
Input: Sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ of the interpolation (5)
1: for $k = 1, \dots, d-1$ do
2:     $\mathcal{I}^\star_{\leqslant k} \leftarrow$ maxvol $[A(\mathcal{I}^\star_{\leqslant k-1} i_k, \mathcal{I}_{>k})]$     % choose $r_k$ rows in $r_{k-1}n_k \times r_k$ matrix
3: end for
Output: Updated index sets $\mathcal{I}_{\leqslant k} \leftarrow \mathcal{I}^\star_{\leqslant k}$, $k = 1, \dots, d-1$
```

If $A^{\{k\}}$'s are only approximately low-rank, the good choice of crosses $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ is important to ensure accurate approximation in (5). If all $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ are maximum-volume submatrices in respective unfoldings $A^{\{k\}}$, the lower accuracy bounds are extended from matrices [19,20,48] to the tensor case [16, Theorem 1]. Inspired by the maximum volume principle, we will now discuss practical algorithms for computation of sufficiently good crosses for the tensor cross interpolation.

3.3. Practical algorithms for tensor cross interpolation

In this section we provide a brief overview of tensor cross interpolation algorithms for TT format and compare them.

3.3.1. ALS maxvol algorithm [18]

The algorithm in the pioneering paper [18] is a direct generalisation of the matrix cross interpolation algorithm from [46] to the tensor case. Starting from a selection of crosses $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$, it updates them one-by-one using the maximum-volume principle. The left-to-right sequence of updates, called *sweep*, is shown in Alg. 3. It is followed by a similar right-to-left sweep and the algorithm sweeps back and forth through the TT cores until convergence. This pattern of updates is often referred to as ALS, coming from *alternating least squares* or *alternating linear scheme*, although the abbreviation is often used in a broader sense.

Remark 5 (*Nestedness in Alg. 3*)**.** The nestedness condition (6) is not preserved during the sweep in Alg. 3. Consider the moment when the left-to-right sweep reaches position k in the train and replaces previous $\mathcal{I}_{\leqslant k}$ with the updated rows $\mathcal{I}^\star_{\leqslant k}$. The nestedness $\mathcal{I}^\star_{\leqslant k} \subset \mathcal{I}^\star_{\leqslant k-1} \times \mathbb{I}_k$ is ensured by construction, so the nestedness of rows is maintained from the left side up to the current active core. However the nestedness of rows in the right part of the train is lost, $\mathcal{I}_{\leqslant k+1} \not\subset \mathcal{I}^\star_{\leqslant k} \times \mathbb{I}_{k+1}$, because $\mathcal{I}_{\leqslant k+1}$ have not yet been updated. The nestedness is recovered when the sweep reaches the end of the train, so the output $\tilde{A}$ of Alg. 3 interpolates the given tensor A.

The main limitation of this algorithm is that it cannot update the ranks r_k of the interpolation, and therefore its success relies on two assumptions, both of which are not easy to ensure in practice:

1. the ranks r_k of the interpolation $\tilde{A}$ are not underestimated to ensure that a good accuracy $|A - \tilde{A}|$ is achievable; and
2. the ranks r_k of the interpolation $\tilde{A}$ are not overestimated and non-singular submatrices $A(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ can be chosen at the initialisation step.

3.3.2. DMRG maxvol algorithm [70]

To allow rank adaptation, we can consider a *superblock* $A(\mathcal{I}_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$ as $r_{k-1}n_k \times n_{k+1}r_{k+1}$ matrix. If we can compute the superblock in full, its low-rank decomposition can be computed by standard algorithms e.g. SVD [56]. This allows us to adapt the rank r_k in accordance with the desired accuracy and compute the good interpolation sets $(I_{\leqslant k}, \mathcal{I}_{>k})$ from the factors of SVD decomposition, as shown in Alg. 4.

Algorithm 4 Left-to-right sweep of the DMRG maxvol cross approximation algorithm [70]

```
Input: Sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ of the interpolation (5), accuracy threshold $\varepsilon$
1: for $k = 1, \dots, d-1$ do
2:     $B \leftarrow A(\mathcal{I}_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$     % compute superblock as $r_{k-1}n_k \times n_{k+1}r_{k+1}$ matrix
3:     $USV^T \leftarrow \mathrm{svd}_\varepsilon(B)$  % compute truncated SVD with accuracy $\varepsilon$
4:     $r_k \leftarrow$ rank $(USV^T)$; $\mathcal{I}^\star_{\leqslant k} \leftarrow$ maxvol $U$; $\mathcal{I}^\star_{>k} \leftarrow$ maxvol $V$
5: end for
Output: Updated index sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k}) \leftarrow (\mathcal{I}^\star_{\leqslant k}, \mathcal{I}^\star_{>k})$, $k = 1, \dots, d-1$
```

Algorithm 5 Left-to-right sweep of the DMRG greedy cross interpolation algorithm [16]

```
Input: Sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ of the interpolation (5)
1: for $k = 1, \dots, d-1$ do
2:     Apply Alg. 2 to the superblock $A(\mathcal{I}^\star_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$ seen as $r_{k-1}n_k \times n_{k+1}r_{k+1}$ matrix. Find a new pivot $(i^\star_{\leqslant k}, i^\star_{>k})$
3:     $\mathcal{I}^\star_{\leqslant k} \leftarrow \mathcal{I}_{\leqslant k} \cup \{i^\star_{\leqslant k}\}$; $\mathcal{I}^\star_{>k} \leftarrow \mathcal{I}_{>k} \cup \{i^\star_{>k}\}$
4: end for
Output: Updated index sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k}) \leftarrow (\mathcal{I}^\star_{\leqslant k}, \mathcal{I}^\star_{>k})$, $k = 1, \dots, d-1$
```

Density matrix renormalisation group (DMRG) [71] and related *matrix product states* (MPS) [72,73] algorithms were developed in quantum physics community to find the ground state of a quantum spin system. The ranks of the ground state are not known in advance, which makes the rank adaptation crucial for the success of the method. Then the DMRG/MPS format was rediscovered in numerical linear algebra as the TT format [17], it was applied to a variety of problems including signal processing [33,34], partial and fractional differential equations [38,74,75], modelling of ionospheric plasma [35] and simulation of NMR [30]. Tailoring DMRG framework to compute interpolation and integration of high-dimensional functions is yet another example of extreme power and flexibility of algorithms, which can be understood, analysed and applied beyond the boundaries of the area where they were discovered.

Remark 6 (*Nestedness in Alg. 4*)**.** Similar to Alg. 3, the DMRG Alg. 4 does not preserve nestedness (6) during the sweep, but recovers it at the end of each sweep. Therefore, the output of Alg. 4 interpolates the initial tensor on all positions $(\mathcal{I}_{\leqslant k-1}, i_k, \mathcal{I}_{>k})$, $k = 1, \dots, d$.

Unfortunately, Alg. 4 is moderately expensive — it evaluates $\mathcal{O}(dn^2r^2)$ points of the given tensor and interpolates only $\mathcal{O}(dnr^2)$ of them.

3.3.3. DMRG greedy algorithm [16]

Calculation of the superblock $A(\mathcal{I}_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$ requires $\mathcal{O}(r^2n^2)$ function evaluations. This may be too expensive, particularly when we aim for high precision and hence employ large mode sizes n_k for accurate quadratures and expect large ranks r_k to achieve accurate interpolation (5). To reduce costs we can replace maxvol optimisation step by greedy cross interpolation step, as proposed in [16] and shown in Alg. 5 and Fig. 2. The algorithm sweeps back and forth the tensor train (5) and attempts to add one cross to each set $(\mathcal{I}_{\leqslant}, \mathcal{I}_{>k})$ at a time.

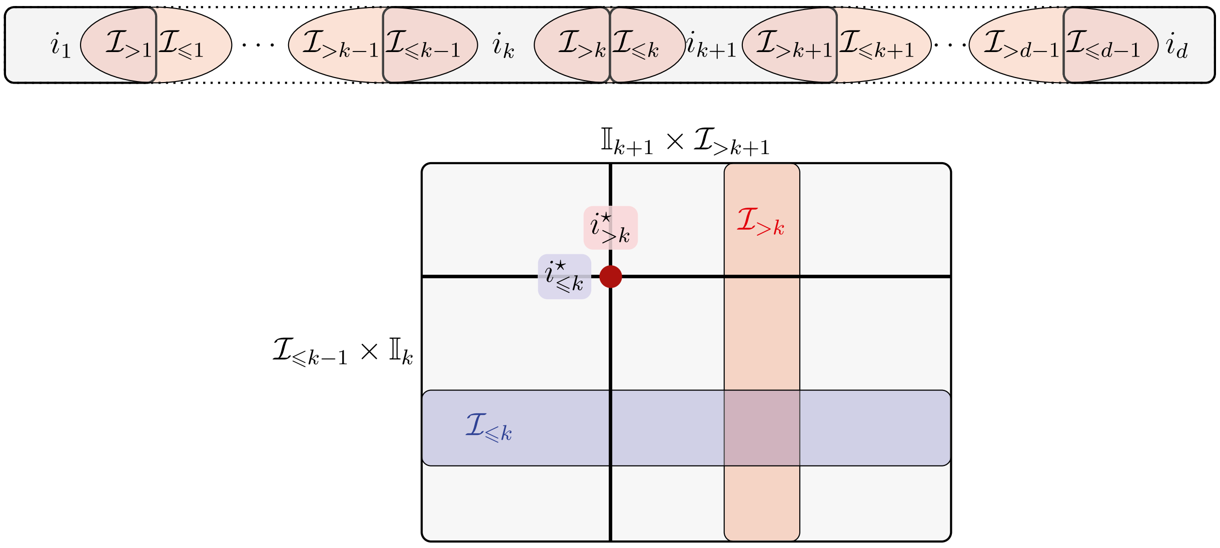

i_1
$\mathcal{I}_{>1}$
$\mathcal{I}_{\leqslant 1}$
$\mathcal{I}_{>k-1}$
$\mathcal{I}_{\leqslant k-1}$
i_k
$\mathcal{I}_{>k}$
$\mathcal{I}_{\leqslant k}$
i_{k+1}
$\mathcal{I}_{>k+1}$
$\mathcal{I}_{\leqslant k+1}$
$\mathcal{I}_{>d-1}$
$\mathcal{I}_{\leqslant d-1}$
i_d
$\mathbb{I}_{k+1} \times \mathcal{I}_{>k+1}$
$\mathcal{I}_{\leqslant k-1} \times \mathbb{I}_k$
$i^\star_{>k}$
$i^\star_{\leqslant k}$
$\mathcal{I}_{>k}$
$\mathcal{I}_{\leqslant k}$

Fig. 2. Cross interpolation algorithm [16] searches for a new pivot $(i^\star_{\leqslant k}, i^\star_{>k})$ in each superblock $A(\mathcal{I}_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$.

Remark 7 (*Nestedness in Alg. 5*)**.** By construction, Alg. 5 preserves nestedness (6) at each internal step of the sweep. The output of Alg. 5 interpolates the initial tensor on all positions $(\mathcal{I}_{\leqslant k-1}, i_k, \mathcal{I}_{>k})$, $k = 1, \dots, d$.

Remark 8 (*Complexity of Alg. 5*)**.** Computation of TT interpolation (5) by Alg. 5 requires $\mathcal{O}(dnr^2)$ evaluations of tensor elements and $\mathcal{O}(dnr^3)$ additional operations. The actual number of function evaluations, $N_{\text{eval}} \lesssim Cdnr^2$, depends on the number of rook pivoting steps in Alg. 2. In all experiments in this paper we keep $C \leqslant 3$.

To the best of our knowledge, Alg. 5 is one of the fastest tensor interpolation algorithms currently available in public domain. As all algorithms considered in this section, Alg. 5 allows trivial parallelisation along each mode, which means that $\mathcal{O}(n)$ tensor entries forming each fibre can be evaluated in parallel. However, Alg. 5 also maintains nestedness on each internal step, which makes it suitable for parallelisation over all modes. Indeed, since no particular step violates the nestedness, all rank-one updates can be performed in parallel, as will be explained in the following section.

3.4. Dimension parallel tensor cross interpolation algorithm

Traditional ALS algorithm is carried out sequentially over tensor factors. However, it was noticed that this dependence is more technical than essential. A concurrency in ALS type algorithms is a matter of active research. As observed in [41], the DMRG algorithm for ground state computations can be executed in parallel over subsets of TT blocks with only a little deterioration of the convergence. Later a dimension parallel version of the HT-ALS for linear equations was developed [42]. In a non-adaptive HT-Cross method the samples and the factors can also be reconstructed in parallel [43].

In this section we show that the adaptive Alg. 5 allows a natural parallelisation over dimensions. From Line 3 of Alg. 5 we see that two consecutive steps k and $k+1$ are connected by only one new pivot $i^\star_{\leqslant k}$, which expands the left index set $\mathcal{I}_{\leqslant k}$. For the sake of scalability we can accept a slight restriction of the search space and replace expanded index sets $\mathcal{I}^\star_{\leqslant k-1}$ with the sets $\mathcal{I}_{\leqslant k-1}$ taken from the previous sweep, see Fig. 3 (top). This allows us to search for new pivots in Line 2 of Alg. 5 in a superblock $A(\mathcal{I}_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$ with the old index sets, which can be done in parallel over all different bonds $k = 1, \dots, d-1$. Different processes find their new pivots $(i^\star_{\leqslant k}, i^\star_{>k})$ independently, and then communicate them and expand index sets before the next whole *sweep*, rather than each internal *step* as in Alg. 5. Since the superblocks owned by different processes overlap only for the neighbouring processes (e.g. the index i_k belongs to only $(k-1)$th and kth superblocks), only the neighbouring processes need to communicate: the multi-index $i^\star_{\leqslant k}$ is sent from kth to $(k+1)$th process, and $i^\star_{>k+1}$ is sent from $(k+1)$th to kth process, see Fig. 3 (bottom).

Although the proposed restriction might potentially lead to a different (sub-optimal) pivot selection, we observed no noticeable change of convergence between the sequential and parallel versions in our numerical experiments.

If fewer than $d-1$ processes are available, each process can be given several consecutive superblocks. The algorithm becomes similar to parallel DMRG algorithm of S. White [41], see Alg. 6: each process performs the sequential sweep as in Alg. 5 over its local part of the TT decomposition, and after that the neighbouring processes exchange new pivots in exactly the same way as described above.

Remark 9. This dimension parallel procedure can be hybridised with multi-threaded local computations, which consist of the evaluation of different samples in Alg. 2 and additional linear algebra operations.

Assuming balanced splitting over P processes, we conclude that each process performs $\mathcal{O}(dnr^2/P)$ evaluations of tensor elements and $\mathcal{O}(dnr^3/P)$ additional operations. Moreover, the tuples $i^\star_{\leqslant k}$ and $i^\star_{>k}$ consist of at most $d-1$ integers, which need to be communicated with neighbours using two messages in each of r iterations, resulting in a total communication volume of $\mathcal{O}(dr)$. Convergence checks require a global communication between all processors, amounting to $\mathcal{O}(r \log P)$ single-word messages in total.

The parallelisation over the modes proposed in Alg. 6 can scale well for the number of processes $P \lesssim d$. It requires only a small number of global communications and lends itself well to distributed-memory 'cluster' architectures and MPI-based implementation. In contrast, the parallelisation along each mode

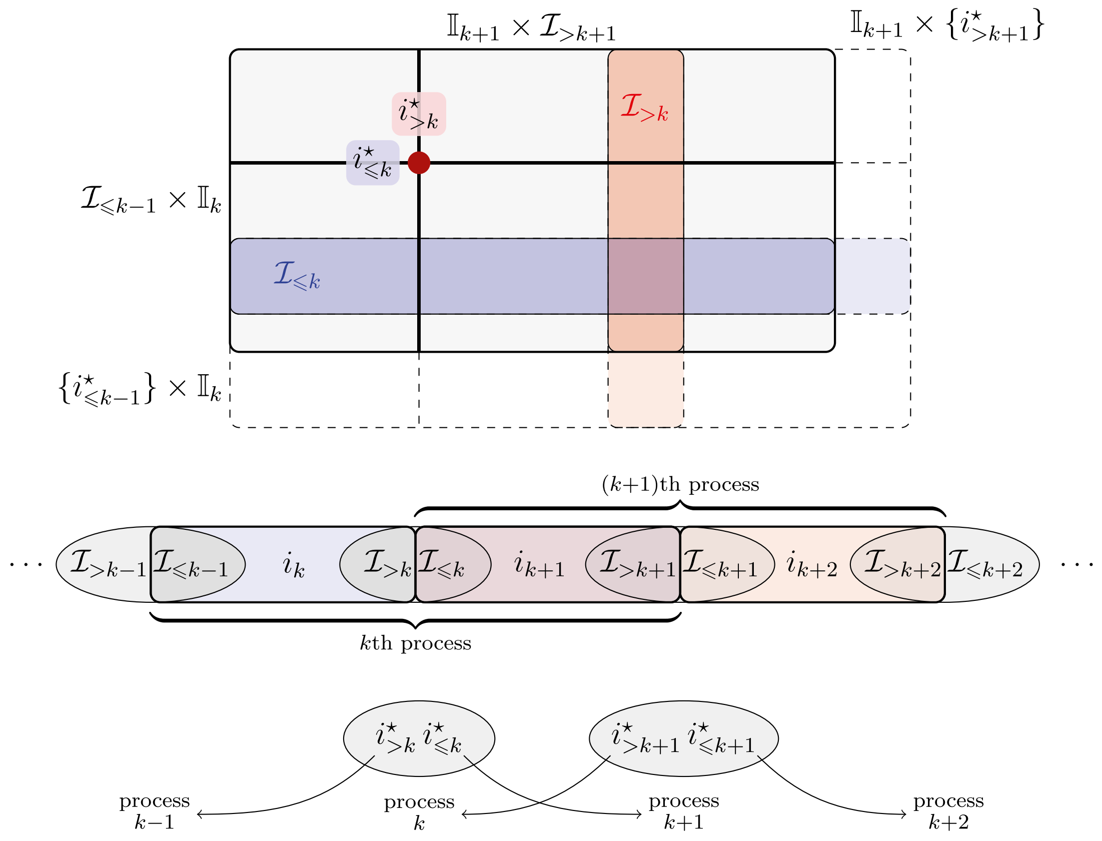

$\mathbb{I}_{k+1} \times \mathcal{I}_{>k+1}$
$\mathbb{I}_{k+1} \times \{i^\star_{>k+1}\}$
$i^\star_{>k}$
$i^\star_{\leqslant k}$
$\mathcal{I}_{>k}$
$\mathcal{I}_{\leqslant k-1} \times \mathbb{I}_k$
$\mathcal{I}_{\leqslant k}$
$\{i^\star_{\leqslant k-1}\} \times \mathbb{I}_k$
$(k+1)$th process
$\cdots$ $\mathcal{I}_{>k-1}$ $\mathcal{I}_{\leqslant k-1}$ i_k $\mathcal{I}_{>k}$ $\mathcal{I}_{\leqslant k}$ i_{k+1} $\mathcal{I}_{>k+1}$ $\mathcal{I}_{\leqslant k+1}$ i_{k+2} $\mathcal{I}_{>k+2}$ $\mathcal{I}_{\leqslant k+2}$ $\cdots$
kth process
$i^\star_{>k}$ $i^\star_{\leqslant k}$
$i^\star_{>k+1}$ $i^\star_{\leqslant k+1}$
process $k-1$
process k
process $k+1$
process $k+2$

Fig. 3. Parallel version of the cross interpolation algorithm. Top: excluding $i^\star_{\leqslant k-1}$ and $i^\star_{>k+1}$ from the row and column sets during the pivot search disentangles different steps in Alg. 5, with mild effect on the pivoting efficiency. Bottom: searching of pivots in different superblocks in parallel implies local data overlap and next-neighbour communication between processors.

Algorithm 6 Dimension parallel DMRG greedy cross interpolation algorithm

```
Input: Sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ of the interpolation (5)
1: Deduce the range $[k_{\text{beg}}, k_{\text{end}})$ of superblocks belonging to the process $p$.
2: for $k = k_{\text{beg}}, \dots, k_{\text{end}} - 1$ do          % in parallel over p
3:     Apply Alg. 2 to the superblock $A(\mathcal{I}^\star_{\leqslant k-1} i_k, i_{k+1}\mathcal{I}_{>k+1})$ seen as $r_{k-1}n_k \times n_{k+1}r_{k+1}$ matrix. Find a new pivot $(i^\star_{\leqslant k}, i^\star_{>k})$
4:     $\mathcal{I}^\star_{\leqslant k} \leftarrow \mathcal{I}_{\leqslant k} \cup \{i^\star_{\leqslant k}\}$; $\mathcal{I}^\star_{>k} \leftarrow \mathcal{I}_{>k} \cup \{i^\star_{>k}\}$
5: end for
6: Send $i^\star_{\leqslant k_{\text{end}}-1}$ to process $p+1$, receive $i^\star_{\leqslant k_{\text{beg}}-1}$ from process $p-1$.
7: Send $i^\star_{>k_{\text{beg}}}$ to process $p-1$, receive $i^\star_{>k_{\text{end}}}$ from process $p+1$.
8: Update $\mathcal{I}^\star_{\leqslant k_{\text{beg}}-1} \leftarrow \mathcal{I}_{\leqslant k_{\text{beg}}-1} \cup \{i^\star_{\leqslant k_{\text{beg}}-1}\}$; $\mathcal{I}^\star_{>k_{\text{end}}} \leftarrow \mathcal{I}_{>k_{\text{end}}} \cup \{i^\star_{>k_{\text{end}}}\}$.
Output: Updated index sets $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k}) \leftarrow (\mathcal{I}^\star_{\leqslant k}, \mathcal{I}^\star_{>k})$, $k = 1, \dots, d-1$
```

requires all workers to access the shared block of memory where the fibre (or superblock) is stored. Hence, this level of parallelisation is best for shared-memory architectures, such as cores and/or threads of a CPU/GPU processor and OpenMP-based implementation. It scales efficiently when the number of cores/threads sharing the same memory is $T \lesssim n$.

In our algorithm we combine both MPI and OpenMP parallelisation to achieve the best performance.

4. High-dimensional integration

In this section we review quadrature rules for the numerical integration in high dimensions. We aim at computing an integral

$$I = \int_{[0,1]^d} f(x_1, \dots, x_d)\mathrm{d}x_1 \cdots \mathrm{d}x_d = \int_{[0,1]^d} f(\mathbf{x})\mathrm{d}\mathbf{x},$$

of a continuous function $f(\mathbf{x})$ on a rectangular domain $[0, 1]^d$. The exact integral is approximated by a quadrature

$$I \approx \tilde{I} = \sum_{i=1}^{N_{\text{eval}}} w_i f(\mathbf{x}_i),$$

where N_{eval} nodes $\{\mathbf{x}_i\}$ and weights $\{w_i\}$ are chosen to reduce the error $|I - \tilde{I}|$. Below we consider several examples of the quadrature rules.

4.1. Full tensor product quadratures

One of the simplest strategies is to rely on an appropriate one-dimensional quadrature rule (e.g. Gauss–Legendre, tanh–sinh), defined by the nodes $\{t_i\}_{i=1}^n \subset [0, 1]$ and the weights $\{w_i\}_{i=1}^n$. The tensor product quadrature approximates each of the one-dimensional integrals independently,

$$\tilde{I} = \sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} w_{i_1} \cdots w_{i_d} f(t_{i_1}, \dots, t_{i_d}). \tag{9}$$

The main advantage of the tensor product quadrature is the fast convergence in n, which stems from the fast convergence of the one-dimensional Gauss–Legendre rule. For example, if a function $f(x)$, $x \in [-1, 1]$, is analytically extensible to a Bernstein ellipse $\mathcal{E}_\rho = \{z \in \mathbb{C} : |z-1| + |z+1| \leqslant \rho + \frac{1}{\rho}\}$ of radius $\rho > 1$, the Gauss–Legendre quadrature converges with exponential rate, $|I - \tilde{I}| = \mathcal{O}(\rho^{-n})$ [76]. However, direct application of (9) is prohibitively expensive in high dimensions, as the total number of quadrature nodes $N_{\text{eval}} = n^d$ grows exponentially with d.

4.2. Quadratures based on tensor product interpolation

4.2.1. Algorithm

To make the calculations in high dimensions feasible and benefit from the fast convergence of the Gauss–Legendre quadrature, we may replace the full tensor $F = [f(t_{i_1}, \dots, t_{i_d})]$ in (9) with a cheaper approximation, reducing the complexity from exponential in d to a manageable polynomial cost. Specifically, in this paper we replace F with tensor product interpolation (5), computed e.g. by Alg. 6,

$$\begin{aligned} f(t_{i_1}, \dots, t_{i_d}) &= A(i_1, \dots, i_d) \\ &\approx \tilde{A}(i_1, \dots, i_d) = A(i_1, \mathcal{I}_{>1})[A(\mathcal{I}_{\leqslant 1}, \mathcal{I}_{>1})]^{-1} \\ &\quad \times A(\mathcal{I}_{\leqslant 1}, i_2, \mathcal{I}_{>2}) \cdots A(\mathcal{I}_{\leqslant d-1}, i_d). \end{aligned} \tag{10}$$

Plugging (10) in (9), we can split the summations and treat each mode individually, breaking the curse of dimensionality. The result is now given as a product of $(2d-1)$ matrices,

$$\begin{aligned} \tilde{I} = &\left(\sum_{i_1=1}^{n} w_{i_1} A(i_1, \mathcal{I}_{>1})\right) [A(\mathcal{I}_{\leqslant 1}, \mathcal{I}_{>1})]^{-1} \\ &\times \left(\sum_{i_2=1}^{n} w_{i_2} A(\mathcal{I}_{\leqslant 1}, i_2, \mathcal{I}_{>2})\right) \cdots \left(\sum_{i_d=1}^{n} w_{i_d} A(\mathcal{I}_{\leqslant d-1}, i_d)\right). \end{aligned} \tag{11}$$

In practical calculations, we adapt the interpolation (10) with a step of Alg. 6 and re-calculate the approximate integral using (11), as explained in Alg. 7. The stopping criterion in Alg. 7 is based on internal convergence, i.e. we consider the result accurate to the desired precision ε when the desired number of leading digits no longer changes after the update cycle or (better yet) several sequential updates. We will also terminate the calculations if all applications of Alg. 6 in one or several sequential iterations of Alg. 7 fail to find a new pivot (i.e. an element with significantly large residual).

We discovered in numerical experiments that the proposed algorithm converges faster if it is applied to a tensor where the function values are pre-multiplied with quadrature weights,

$$B(i_1, \dots, i_d) = w_{i_1} \cdots w_{i_d} \cdot f(t_{i_1}, \dots, t_{i_d}).$$

This often leads to lower TT ranks/error compared to the approximation of $f(t_{i_1}, \dots, t_{i_d})$ if the function has a complicated structure near the boundaries. In this case, the boundary elements, multiplied by small cumulative products of the quadrature weights, become less influential to both the quadrature and the cross interpolation algorithm.

4.2.2. Complexity

After r steps of Alg. 7 we obtain the interpolation (10) with ranks $r_k \leqslant r$. It is composed of blocks $A(\mathcal{I}_{\leqslant k-1}, \mathbb{I}_k, \mathcal{I}_{>k})$, $k = 0, \dots, d$, each of which consists of the values of function $f(\mathbf{x})$ at points $\mathbf{x}_i = (t_{i_1}, \dots, t_{i_d})$ with $(i_1, \dots, i_{k-1}) \in \mathcal{I}_{\leqslant k-1}$, $i_k \in \mathbb{I}_k$ and $(i_{k+1}, \dots, i_d) \in \mathcal{I}_{>k}$. In total, Eq. (10) requires $\mathcal{O}(dnr^2)$ evaluations of function $f(\mathbf{x})$ and $\mathcal{O}(dnr^3)$ additional operations, as explained by Remarks 4 and 8. Note that each function evaluation requires at least linear number of operations (to engage all input values), which makes the overall complexity at least quadratic in d.

Algorithm 7 High-dimensional integration using tensor cross interpolation

```
Input: A function to integrate $f(\mathbf{x}) = f(x_1, \dots, x_d)$ and domain for integration, e.g. $[0, 1]^d$
1: Choose desired relative accuracy $\varepsilon$ of integration.
2: Choose appropriate quadrature nodes and weights $\{t_{i_k}, w_{i_k}\}_{i_k=1}^{n_k}$ in each mode $k = 1, \dots, d$.
3: Choose initial element $(i_1, \dots, i_d) \in \mathbb{I}_1 \times \cdots \times \mathbb{I}_d$, and use it to initialise interpolation sets $\mathcal{I}_{\leqslant k} = \{(i_1, \dots, i_k)\}$ and $\mathcal{I}_{>k} = \{(i_{k+1}, \dots, i_d)\}$ for $k = 1, \dots, d-1$.
4: repeat                                   % rank-one updates of interpolation (10)
5:     Calculate approximate integral $\tilde{I}$ by (11) using current sets $(\mathcal{I}^\star_{\leqslant k}, \mathcal{I}^\star_{>k})$
6:     Apply Alg. 6 to update sets $\mathcal{I}^\star_{\leqslant k} \leftarrow \mathcal{I}_{\leqslant k} \cup \{i^\star_{\leqslant k}\}$; $\mathcal{I}^\star_{>k} \leftarrow \mathcal{I}_{>k} \cup \{i^\star_{>k}\}$ for $k = 1, \dots, d-1$
7:     Re-calculate approximate integral $\tilde{I}^\star$ by (11) using updated sets $(\mathcal{I}^\star_{\leqslant k}, \mathcal{I}^\star_{>k})$
8: until $|\tilde{I} - \tilde{I}^\star| \leqslant \varepsilon |\tilde{I}^\star|$                % internal convergence
Output: approximate integral $\tilde{I}$ as shown in (11).
```

Evaluation of the integral by (11) requires $\mathcal{O}(dnr^2) + \mathcal{O}(r^3)$ operations. When calculations are performed in parallel on a distributed-memory platform, the matrix multiplications in (11) have to be arranged in a balanced tree way to reduce the parallel depth of the algorithm from linear to logarithmic in the number of processors.

Although a single evaluation of (11) does not increase the asymptotic estimate of the cost of interpolation, re-calculation of the integral at each step of Alg. 7 increases the costs to $\mathcal{O}(dnr^3) + \mathcal{O}(r^4)$. This term may become dominant if the function evaluation requires $\mathcal{O}(d)$ operations and $r \gg d$, in which case we can update the integral on each step using Sherman–Morrison–Woodbury formula [77–79], reducing the numerical costs to those of interpolation.

4.2.3. Accuracy

To provide an analytic estimate for the error of the integration $|I - \tilde{I}|$, we need to answer the following questions:

1. Is function $f(\mathbf{x})$ separable, i.e. does it admit an accurate representation in the TT format with moderate ranks?
   This question was answered for particular classes of functions, see for example [80–82], and the accuracy ε of the best separable approximation of rank r is shown to grow logarithmically, $r = \mathcal{O}(\log \varepsilon^{-1})$.
2. Is it possible to reach the same asymptotic convergence rate if the function is represented by the interpolation (10) with the best choice of interpolation crosses $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$?
   This question is answered positively by the analysis in [16] and [83].
3. Is it possible to reach the same asymptotic convergence with the interpolation crosses $(\mathcal{I}_{\leqslant k}, \mathcal{I}_{>k})$ found by a practical algorithm with (low) polynomial complexity in d, r and n?
   The heuristic nature of Alg. 6 does not allow us to answer this question in this work. We hope that further research will lead to tensor product algorithms combining fast practical convergence with better theoretical properties.

4.3. Monte Carlo and quasi Monte Carlo techniques

The Monte Carlo quadrature is a statistical method which is based on the central limit theorem. It introduces random nodes $\{\mathbf{x}_i\}_{i=1}^{N_{\text{eval}}}$ drawn from a uniform distribution on $[0, 1]^d$, and the

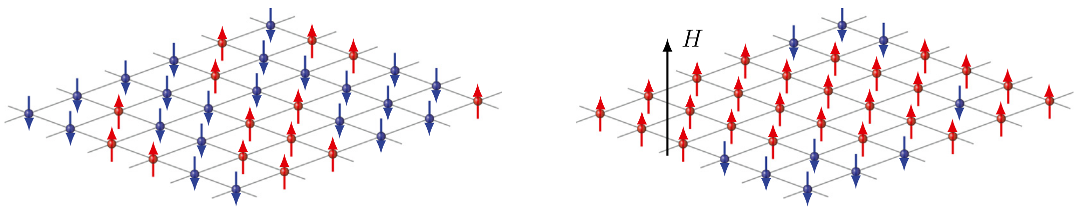


Fig. 4. Two-dimensional Ising model shown as a square lattice of interacting spins. Normally, one would expect to observe individual spins in both states $\sigma_{i,j} \in \{\uparrow, \downarrow\}$ with equal probability (as on the left panel). Spins also would align with the direction of external magnetic field (as shown on the right panel). Surprisingly, ferromagnetics will also exhibit collective large-distance behaviour (e.g. spontaneous magnetisation) at $H = 0$ for sub-critical temperatures $T < T_c$. Theoretical explanation of this fact was first proposed by Lars Onsager in 1944.

integral is approximated by an average of the values of the function at these nodes and all weights equal,

$$\tilde{I} = \frac{1}{N_{\mathrm{eval}}} \sum_{i=1}^{N_{\mathrm{eval}}} f(\mathbf{x}_i). \tag{12}$$

The integration error depends on the variance of $f(\mathbf{x})$ (treated as a random field after randomisation of the coordinates x), $|I - \tilde{I}|^2 \leqslant \frac{\mathrm{var}(f)}{N_{\mathrm{eval}}}$. Provided that the variance is independent of the dimension, so is the error. However, the decay rate of $N_{\mathrm{eval}}^{-0.5}$ is often prohibitively slow, especially if a high accuracy is needed.

Quasi Monte Carlo (qMC) [5,6] is another family of equal-weight quadrature rules (that is, $w_i = 1/N_{\mathrm{eval}}$ for all $i = 1, \ldots, N_{\mathrm{eval}}$), but the nodes are chosen semi-deterministically. Firstly, one constructs a deterministic lattice rule, defined by a generating vector $\mathbf{q} = (q_1, \ldots, q_d)$. The lattice is optimised to minimise the worst-case error component by component [7,84]. The quadrature nodes are then computed as shifted multiples of the generating vector,

$$\mathbf{x}_i = \mathrm{frac}\left(\frac{i}{N_{\mathrm{eval}}}\mathbf{q} + \mathbf{s}\right), \qquad i = 1, \ldots, N_{\mathrm{eval}}. \tag{13}$$

Here $\mathbf{s} = (s_1, \ldots, s_d)$ is a vector of random shifts, distributed uniformly on [0, 1], and frac(x) denotes the fractional part of x. Standard qMC rules provide a convergence rate $\mathcal{O}(N_{\mathrm{eval}}^{-\alpha})$, with $0.5 \leqslant \alpha \leqslant 1$. Under certain assumptions on the function, the rate can be proven to be close to 1, and the constant to be independent of d. There exist higher order qMC rules [8] which can achieve faster convergence, but at a price of more sophisticated lattice construction algorithms and stronger assumptions on the function.

The shifts $\mathbf{s}$ make the quadrature (13) unbiased, and they also allow to estimate the quadrature error. We repeat qMC experiments using the same generating vector $\mathbf{q}$ but S different shifts. Thus we obtain S sets of nodes (13), and use (12) to calculate the estimators $\tilde{I}_j$, $j = 1, \ldots, S$. Now the error can be estimated as the empirical standard deviation,

$$\varepsilon \approx \frac{1}{\langle \tilde{I} \rangle} \sqrt{\frac{1}{S-1} \sum_{j=1}^{S} \left(\tilde{I}_j - \langle \tilde{I} \rangle\right)^2}, \qquad \langle \tilde{I} \rangle = \frac{1}{S} \sum_{j=1}^{S} \tilde{I}_j. \tag{14}$$

For the MC experiment we estimate the standard deviation in a similar way by repeating the experiments S times.

5. Numerical experiments

5.1. Ising integrals

To demonstrate the efficiency of the proposed approach, we apply tensor product interpolation to calculate high-dimensional integrals of so-called Ising class [85]. They are motivated by the famous 2D Ising model, explaining spontaneous magnetisation in ferromagnetic materials. It describes a ferromagnet as a rectangular $M \times N$ grid of spin-$\frac{1}{2}$ particles where each spin $\sigma_{i,j}$ can be observed in one of two possible states, $\sigma_{i,j} \in \{+\frac{1}{2}, -\frac{1}{2}\} = \{\uparrow, \downarrow\}$. The energy of configuration $\sigma = \{\sigma_{i,j}\}_{\substack{i=1,\ldots,M \\ j=1,\ldots,N}}$ in magnetic field H is given as follows:

$$E(\sigma) = \underbrace{-\sum_{i,j} \sigma_{i,j}\sigma_{i,j+1} - \sum_{i,j} \sigma_{i,j}\sigma_{i+1,j}}_{\text{next neighbour interaction}} \underbrace{- H \sum_{i,j} \sigma_{i,j}}_{\text{response to magnetic field}}.$$

The probability of each configuration is given by the Gibbs measure $\exp(-E(\sigma)/kT)/Z$, where T denotes the temperature and $Z(T, H) = \sum_{\sigma} \exp(-E(\sigma)/kT)$ is known as partition function. Assuming temperature and volume are constant, the Helmholtz free energy of the system is $F = -kT \log Z(T, H)$, and energy per particle is $f(T, H) = \lim_{\substack{M\to\infty \\ N\to\infty}} F(T, H)/(MN)$. We may be interested in *spontaneous magnetisation* $m_0(T) = -\left.\frac{\partial f}{\partial H}\right|_{H=0}$ and zero-field *magnetic susceptibility* $\chi_0(T) = -\left.\frac{\partial^2 f}{\partial H^2}\right|_{H=0}$. Susceptibility is particularly interesting as it relates to long-distance spin–spin correlation and hence can explain collective behaviour in a ferromagnetic system connected by only next-neighbour interactions as shown in Fig. 4.

The 2D Ising model was first solved by Lars Onsager in 1944, who has never published the results. The solution for the magnetisation was published by Yang [86], and the susceptibility was calculated by Wu, McCoy, Tracy and Barouch [87] as

$$kT\chi_{0,\pm}(T) = C_{0,\pm}\,|1 - T/T_c|^{-7/4} + C_{1,\pm}\,|1 - T/T_c|^{-3/4} + \mathcal{O}(1), \tag{15}$$

where T_c denotes critical (Curie) temperature, which for the square and isotropic lattice is given by $kT_c = 2/\ln(1+\sqrt{2})$, and $\pm$ refers to $T \to T_c$ from above $(+)$ or below $(-)$. The coefficients of the asymptotic expansion are given as infinite series,

$$C_{0,+} \sim C_{1,+} \sim \sum_{d=1,3,\ldots} \frac{\pi D_d}{(2\pi)^d}, \qquad C_{0,-} \sim C_{1,-} \sim \sum_{d=2,4,\ldots} \frac{\pi D_d}{(2\pi)^d}, \tag{16}$$

where D_d's are $(d-1)$-dimensional integrals, which can be written as shown below [85]:

$$C_d = 2\int_{[0,1]^{d-1}} B_d(x_2, \ldots, x_d)\mathrm{d}x_2 \cdots \mathrm{d}x_d, \tag{17}$$

$$D_d = 2\int_{[0,1]^{d-1}} A_d(x_2, \ldots, x_d)B_d(x_2, \ldots, x_d)\mathrm{d}x_2 \cdots \mathrm{d}x_d, \tag{18}$$

$$E_d = 2\int_{[0,1]^{d-1}} A_d(x_2, \ldots, x_d)\mathrm{d}x_2 \cdots \mathrm{d}x_d, \tag{19}$$

with

$$A_d(x_2, \ldots, x_d) = \prod_{1 \leqslant i < j \leqslant d} \left(\frac{1 - x_{i+1}\cdots x_j}{1 + x_{i+1}\cdots x_j}\right)^2,$$

$$B_d(x_2,\dots,x_d) = \left(1+\sum_{k=2}^{d} x_2\cdots x_k\right)^{-1}\left(1+\sum_{k=2}^{d} x_k\cdots x_d\right)^{-1}.$$

Bailey et al. [85] first suggested the computational / experimental mathematics approach to this problem — they took up a challenge to calculate D_d's numerically with high accuracy and then use inverse symbolic calculator [88] to conjecture the values in closed form as a linear combination of physically relevant constants. The integrals C_d and E_d were introduced as 'structurally similar', but simpler versions of D_d in assumption that their evaluation may lead to certain insights. Indeed, all C_d's were analytically reduced to two-dimensional integrals and resolved numerically to extreme precision [85]. Based on numerical results, Bailey and co-authors were able to conjecture and then prove that $C_\infty = \lim_{d\to\infty} = 2e^{-2\gamma}$, where γ is the Euler–Mascheroni constant. [85]. Using dimension reduction techniques, they calculated D_d and E_d for $d \leqslant 4$ in closed form in terms of Riemann zeta function and Dirichlet L-function and conjectured the analytic value for E_5.

Ten years later, Erik Panzer developed the code to symbolically evaluate all E_d's in terms of alternating multizeta functions [89], which he used to calculate integrals E_1 to E_8, thus confirming the conjecture of Bailey et al. for E_5. The complexity of symbolic evaluation grows rapidly with d, e.g. E_8 required 28 CPU hours and 30 GB of memory. Hence, this method cannot be seen as a practical way for evaluation of E_d's for $d > 10$. Also, no clear indication is given in [89] on whether the proposed method can be applied to more complex integrals D_d.

To the best of our knowledge, apart of results listed above, no further progress has been made in calculation of Ising integrals, meaning that accurate evaluation of Ising susceptibility integrals D_d remains an open problem for all except relatively small d.

In this work we pick up the baton and apply the proposed algorithm 6 to calculate D_d's with high accuracy for $d \lesssim 1000$. However, we will first verify our algorithm by applying it to calculate the values of C_d's treating them as high-dimensional integrals, and verifying the accuracy against the results calculated by Bailey et al. in [85].

5.2. Experiment setup for double-, quadruple- and high-precision calculations

Following Bailey [85], we evaluate the integrals numerically using tensor product of one-dimensional Gauss–Legendre quadratures, as explained in Section 4.2. The number of quadrature points in each direction, n, is chosen adaptively to reach the desired accuracy. Since functions A_d and B_d are infinitely smooth, the Gauss–Legendre quadrature for C_d, D_d and E_d converges exponentially, and we can expect the number of accurate digits to grow linearly with n.

The parallel implementation of the proposed algorithm is implemented in Fortran by authors.

Double-precision calculations are implemented using GNU Fortran compiler with BLAS and Lapack libraries from Intel MKL.

For quadruple-precision calculations we compile the same code using a compiler option `-fdefault-real-8`, that sets the default size for `double precision` to 16 bytes and increases precision to approximately 33 decimal digits. We compiled the reference implementation of BLAS and Lapack libraries with the same parameter to reach quadruple precision in the whole calculation.

For high-precision calculations we use the MPFUN2015 library [90,91]. We use the version of MPFUN2015 utilising the MPFR library,[4] which is several times faster than the version implemented fully in Fortran. We had to rewrite reference implementation of necessary BLAS libraries to use the `mp_real` data type offered by MPFUN. The code itself was compiled using the same compilers and options as for double precision calculations. The MPFUN2015 library was set up to provide accuracy of 120 decimal digits.

The experiments were performed on two computers:

- at the University of Bath: this research made use of the Balena High Performance Computing (HPC) Service. Each node on Balena contains an Intel Xeon E5-2650 v2 CPU with 16 cores, running at 2.6 GHz. A single job can occupy up to 32 nodes for 5 days.
- at the University of Brighton: the development, testing and numerical experiments were made possible by use of a dedicated workstation. The workstation has two Intel Xeon E5-2650 v4 CPUs with 12 cores and 2 threads each, running at 2.2 GHz. It is also equipped with 0.5 TB of operating memory, which proved essential for large-scale calculations reported below.

5.3. Verification and benchmarking of the cross interpolation algorithm

Bailey et al. [85] found analytic transformation to convert $(d-1)$-dimensional integrals C_d to two-dimensional form. Using this simple representation, they calculated C_d's to 1000 decimal digits for $d \leqslant 1024$. They conjectured that $C_\infty = \lim_{d\to\infty} C_d = 2e^{-2\gamma}$, where γ is the Euler–Mascheroni constant. This result was then proven analytically in [85, Theorem 2].

We compute C_{1024} directly as a $(d-1)$-dimensional integral using the proposed tensor product interpolation algorithm, and compare the numerical result with the one obtained by Bailey [85]. The comparison is shown in Fig. 5. For double and quadruple precision calculations we observe an expected stagnation at the level of 15 and 32 decimal digits, respectively. When multiple precision calculations are used, the proposed algorithm seemingly provides exponential convergence for the integral C_{1024}. As we can see in Fig. 5, the observed convergence of relative accuracy ε agrees well with the assumption $\varepsilon \sim \exp(-\sqrt{N_{\text{eval}}})$. Since the number of samples evaluated by the cross interpolation algorithm is $N_{\text{eval}} \sim dnr^2$, and d, n remain constant, this allows us to conjecture that $\varepsilon \sim \exp(-r)$, i.e. the relative accuracy improves exponentially with the average TT rank r.

This observation shows that the underlying multivariate function admits an accurate low-rank representation, and, more importantly, that using the proposed algorithm a good interpolation can be constructed and the integral can be approximated much faster than by other currently known techniques, such as MC and qMC algorithms. Based on this example, we are optimistic that the challenging theoretical questions discussed in Section 4.2.3 can be answered positively for functions considered in Ising susceptibility theory, and hopefully a wider class of functions as well.

It should be noted that although the use of quadruple and multiple precision calculations comes at a small extra cost in terms of number of points (it is sufficient to double the mode size n to double the number of accurate digits), it leads to significant overhead in terms of CPU time, since the quadruple and multiple precision calculations are not optimised to the same degree as native double precision calculations and BLAS libraries. This is the reason why we report the convergence behaviour both as a function of the number of evaluated points, and of the CPU time, as shown in Fig. 5.

[4] The GNU MPFR library is a C library for multiple-precision floating-point computations in (arbitrary) high precision. It is free and available on most platforms. The details can be obtained from www.mpfr.org.

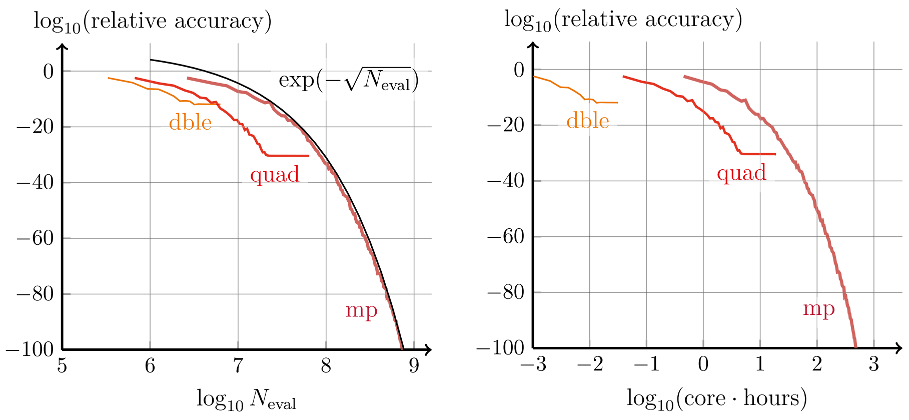


Fig. 5. Convergence of cross interpolation for calculation of C_{1024} in double, quadruple and multiple precision. Cross interpolation algorithm uses tensor product of one-dimensional Gaussian quadrature rules with $n = 33$ points for double-precision, $n = 65$ points for quadruple-precision and $n = 257$ points for multiple-precision calculations. The results are verified against the 1000-digit result reported in [85]. The relative accuracy is shown w.r.t. number of function evaluations (left) and w.r.t. CPU time (right). We can clearly see that the proposed method converges exponentially.

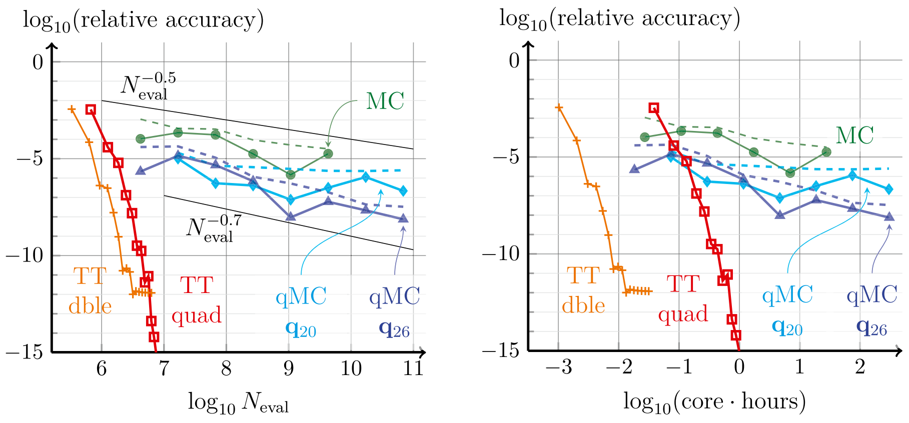


Fig. 6. Integral C_{1024} calculated by TT cross interpolation (Alg. 6), Monte Carlo (MC), and quasi Monte Carlo (qMC). Cross interpolation algorithm uses tensor product of one-dimensional Gaussian quadrature rules with $n = 33$ points for double-precision and $n = 65$ points for quadruple-precision calculations. QMC algorithm uses lattice generating vectors $\mathbf{q}_{20}$ and $\mathbf{q}_{26}$ minimising the worst-case error on 2^{20} and 2^{26} points, respectively. Solid lines: errors of numerical methods verified against the result of Bailey et al. [85]. Dashed lines: relative standard deviation estimates (14) of MC and qMC with number of repetitions $S = 16$. Left: relative accuracy w.r.t. different numbers of function evaluations N_{eval}. Right: relative accuracy w.r.t. total CPU time.

5.4. Convergence and comparison with quasi Monte Carlo

In Fig. 6 the proposed algorithm is compared with the state of the art Monte Carlo (MC) and Quasi MC approaches (see Section 4.3). For the MC quadrature we use uniformly distributed samples on $[0, 1]^d$.

For the qMC algorithm a particular care must be taken when choosing the correct lattice. Frances Kuo's website[5] provides a large collection of pre-generated lattices which were generated by optimising the worst case error with product weights k^{-2}, motivated by stochastic PDEs. The integrals considered in this paper do not exhibit the same decay, hence we decided to use the lattice with equal weights. We used the component by component algorithm from Dirk Nuyens's website[6] and constructed generating vectors $\mathbf{q}_{20}$ and $\mathbf{q}_{26}$ by minimising the worst case error on 2^{20} and 2^{26} points respectively. Notice that the lattice generated from $\mathbf{q}_{20}$ starts repeating when the number of points exceeds 2^{20}, leading to a visible stagnation of the $\mathbf{q}_{20}$ quadrature error in Fig. 6. This is why we created lattice $\mathbf{q}_{26}$ which remains convergent and allows to scale the computations up to billions of points. It has to be noted that optimising a lattice is rather expensive — the CBC algorithm took several days to produce $\mathbf{q}_{26}$ (this cost is not included in further analysis).

As in the previous subsection, we calculate C_{1024} and compare our results against the 1000-digit accurate value computed in [85]. These errors are plotted on solid lines in Fig. 6. We also show by dashed lines the relative empirical standard deviation for MC and qMC algorithms as described in (14). Notice that the true error exhibits a higher fluctuation for different N_{eval}, although the overall convergence trend coincides with that for the standard deviation.

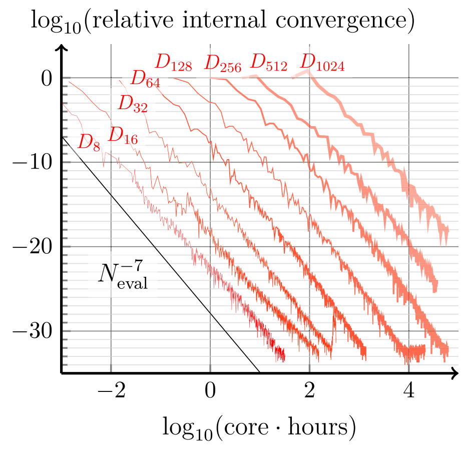
We see that the MC method converges with the rate $N_{\text{eval}}^{-0.5}$ as expected from the CLT, while the qMC method (with $\mathbf{q}_{26}$) exhibits a higher rate $N_{\text{eval}}^{-0.7}$. The TT decomposition has a much richer approximation capacity, and provides a sub-exponential convergence, as shown also in Fig. 5. When all calculations are performed in double precision, TT cross interpolation is always faster than MC and qMC methods. Switching to quadruple precision increases the TT time significantly, since we lose optimisations of the Intel MKL library, but the rapid convergence still makes it the fastest method for high accuracy.

5.5. Evaluation of ising susceptibility integrals

Now we attempt to compute original Ising susceptibility integrals D_d given by (18). Computing D_d's for large d is much

5 http://web.maths.unsw.edu.au/~fkuo/.

6 https://people.cs.kuleuven.be/~dirk.nuyens/qmc-generators/.



D_8	$\approx 1.8959911856917860437277009899220$	$\times 10^{-05}$
D_{16}	$\approx 4.2801588294649145858508255168059$	$\times 10^{-11}$
D_{32}	$\approx 2.1812588849921849857069379650793$	$\times 10^{-22}$
D_{64}	$\approx 5.6650255053330577357994408887987$	$\times 10^{-45}$
D_{128}	$\approx 3.8211244448448883068502878086763$	$\times 10^{-90}$
D_{256}	$\approx 1.7384804312816219652099879244884$	$\times 10^{-180}$
D_{512}	$\approx 3.5985459774976059367726O$	$\times 10^{-361}$
D_{1024}	$\approx 1.54184797470070618$	$\times 10^{-722}$

Fig. 7. Evaluation of the Ising susceptibility integrals D_d given by (18). The results are computed by the cross interpolation algorithm in quadruple precision using tensor product of one-dimensional Gaussian quadrature rules with $n = 129$ points (for D_8 to D_{256}) and $n = 65$ points (for D_{512} and D_{1024}). Left: convergence of cross interpolation algorithm measured by the relative internal convergence, as a function of total CPU time spent on the calculation. Right: values of the D_d's calculated by the proposed algorithm.

more challenging than evaluating C_d's, for two reasons. Firstly, each evaluation of the integrand takes $\mathcal{O}(d)$ operations for C_d, but $\mathcal{O}(d^2)$ for D_d. Secondly, all C_d's can be analytically reduced to two dimensional integrals, while for D_d's reduction performed in [85] only reduces the dimensionality by one in special cases. Using a combination of analytic transforms and Gaussian tensor-product quadratures, Bailey and collaborators calculated D_5 to 500 decimal digits using 18h on 256 CPUs of IBM Power5 nodes at the Lawrence Berkeley National Laboratory. They also produced D_6 to almost 100 decimal digits. Using qMC algorithm, they also calculated D_7 and D_8 to 5 decimal digits. Further integrals D_d were not made available.

We apply the proposed tensor interpolation algorithm to calculate D_d's in the original form (18) as $(d-1)$-dimensional integrals. We use the quadruple-precision version of the code and aim to calculate integrals $D_8, D_{16}, D_{32}, \dots, D_{1024}$ to about 30 decimal digits, which is measured by the internal convergence. The convergence plots are shown in Fig. 7. The convergence rate is approximately of order 7 for all considered integrals; noting a slight bent of the curve for D_{256} we are hopeful that exponential convergence could have been revealed if calculations were allowed to run longer and reach higher accuracy.

By looking at the values of D_d's in Fig. 7 it is easy to note that they decay exponentially. This was noted by Bailey et al. who proved [85, Thm. 3] that $\mathcal{O}(14^{-d}) \leqslant D_d \leqslant \mathcal{O}(4^{-d})$. They conjectured that as $d \to \infty$, $D_d \sim \Delta^{-d}$, and estimated $\Delta \approx 5$ based on a few available to them values D_d. Based on our values D_{128} and D_{256} shown in Fig. 7, we improve this estimate to

$$\Delta \approx 5.0792202086636783360436879567820. \tag{20}$$

5.6. Evaluation of susceptibility coefficients

We can now come back to the asymptotic expansion of magnetic susceptibility (15) and the coefficients $C_{0,\pm}$ and $C_{1,\pm}$ represented via the sums (16),

$$\Sigma_+ = \sum_{d=1,3,\dots} \frac{\pi D_d}{(2\pi)^d}, \qquad \Sigma_- = \sum_{d=2,4,\dots} \frac{\pi D_d}{(2\pi)^d},$$

Noting that $D_d \sim \Delta^{-d}$ with Δ given by (20), the coefficients in the sums $\Sigma_\pm$ decay geometrically, $D_d/(2\pi)^d \sim (2\pi\Delta)^{-d} \approx \pi(31.9)^{-d}$. This means that each next term in the sums is approximately 1000 times smaller than the previous one. Hence, if we want to evaluate Σ_+ to, say, 50 decimate digits, we could start with $D_1 = 2$, then evaluate D_3 to 47 digits (noting that the analytical value is known since [85]), calculate D_5 to 45 digits and so on. We used the proposed algorithm to evaluate D_d's for $d \leqslant 45$ with sufficient precision (judged by the internal convergence of the algorithm), and obtained the sums to 50 decimal digits as follows:

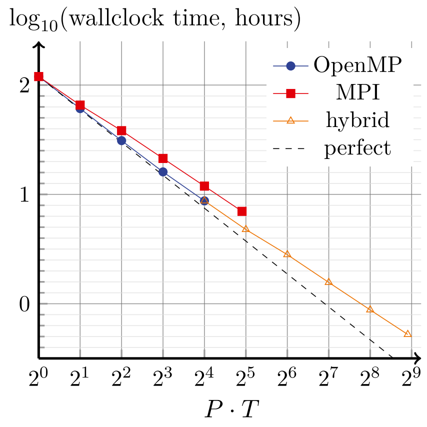
$$\begin{aligned} \Sigma_+ &= \sum_{d=1,3,\dots} \frac{\pi D_d}{(2\pi)^d} \\ &\approx 1.0008152604402126471194763630472102369375349255977 \\ \Sigma_- &= \sum_{d=2,4,\dots} \frac{\pi D_d}{(2\pi)^d} \\ &\approx 0.026551297359252325321072273129862563625255686544007 \end{aligned} \tag{21}$$

As noted in [85], these coefficients were also computed by Nickel [92] to 40 accurate decimal digits by high-order integration of differential equation for Painlevé function of the third kind in [87, Eq. (2.36)]. Bailey et al. [85], using the values of D_d obtained by quasi Monte Carlo algorithm, were able to match the results of Nickel to 20 decimal digits. Using tensor product interpolation Algorithm 6, we are able to reach higher accuracy for D_d's and confirm that the first 40 digits in our results fully agree with the results obtained by Nickel.

5.7. Performance and scalability

In Fig. 8 we benchmark the algorithm for different numbers of processes and threads using MPI, OpenMP and hybrid parallelisation. The first two lines in Fig. 8 (left) show the CPU time for OpenMP-only parallelisation of local computations (i.e. essentially Alg. 5 with no dimension parallelisation), and for MPI-only approach where all local computations are performed in one thread, but different chunks of the TT decomposition are assigned to different processes (Alg. 6). Moreover, the hybrid approach always uses $T = 16$ threads for local operations, and different numbers of processes P for parallelisation over dimension. In Fig. 8 (left) we report the product of the number of processes and the number of threads in each process.

The integral D_{32} is taken over 31 variables, which means there are 30 rank-update steps to perform simultaneously (recall that TT ranks separate the dimensions, so there is one less rank than dimensions). Hence, our MPI parallelisation can scale only up to 30 processors. Using the hybrid framework, we accelerate the computing further up to a maximum of 512 cores, available on the Balena cluster per one job. We notice a very good scaling, since the cost of communicating $\mathcal{O}(rd + r \log P)$ bytes is much



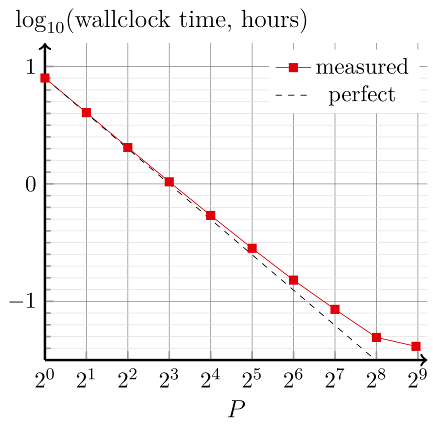


Fig. 8. Left: strong scaling for D_{32} for different numbers of processes P and numbers of threads T, quadruple precision with $n = 129$. Right: strong MPI scaling (1 OpenMP thread per process) for D_{512}, double precision with $n = 33$.

smaller than the cost of computing $\mathcal{O}(dnr^2/P)$ tensor elements. A slight deviation from the linear scaling for the largest numbers of processes is due to load imbalance, as different TT blocks pick up different ranks in the course of the cross algorithm.

This is demonstrated further in Fig. 8 (right), where we approximate a function for the D_{512} integral. The maximal number of processes 510 allows us to use only $T = 1$ OpenMP thread, and instead vary the number of MPI processes P in the entire range. We see that the time is closer to the perfect scaling due to better balancing when each process owns more TT blocks. Even better scaling could be expected for D_{2^p+2} integrals, where the same number of TT blocks could be assigned to each of 2^p processes. Nevertheless, even in a deliberately unbalanced situation (which is more practical though), the algorithm scales almost linearly up to the maximum computing capacity available at the given machine.

Finally, we should note that even though with the proposed algorithm 6 we enjoy fast convergence, the numerical costs remain quite high. For example, calculation of D_{1024} to 18 decimal digits (see Fig. 7) took about 4 days on 512 nodes of Balena supercomputer at the University of Bath, consuming approximately a megawatt hour of energy. Based on our preliminary experiments with qMC, and assuming that the convergence rate $\varepsilon \sim N_{\text{eval}}^{-0.7}$ will not deteriorate, we estimate that to reach the same accuracy with qMC we would need approximately 10^{13} years of calculations and 10^9 terawatt hours of energy — which exceeds the age of the Universe ($\approx 1.3 \cdot 10^{10}$ years) and annual world energy consumption ($\approx 1.5 \cdot 10^5$ TWh in 2014) by three orders of magnitude.

6. Conclusion

The problem of high-dimensional integration is a particularly important and challenging area. Motivated by risk simulation in finance and engineering, this problem was actively researched and resulted in Monte Carlo Metropolis algorithm [4] considered as one of top 10 algorithms of the 20th century [93]. The use of random samples in the MC algorithm allows to break away from tensor-product quadratures and hence avoid the curse of dimensionality, seemingly inevitable in higher dimensions. The flexibility and simplicity of MC was spoiled by its slow convergence, motivating the further development, until the arrival of the quasi Monte Carlo algorithm [5,6]. QMC lattices can be optimised for a class of functions (e.g. those appearing from stochastic PDEs [7,8]), and demonstrate faster convergence, which currently makes qMC the method of choice in areas of sPDEs, finance and risk modelling, engineering, etc. However, the convergence is still not too fast, particularly considering that in practice many end users can make sub-optimal choices in choosing/creating the correct qMC lattice for their problems, leading to excessive numerical costs and increasing negative effect of the HPC industry on the environment.

The curse of dimensionality turns therefore in a challenge of precision. Although admittedly many practical problems (e.g. in areas of stochastic inference or machine learning) do not require precision above one or two decimal digits, many applications (e.g. engineering, theoretical quantum physics, quantum computations) need the answer to be precise to ten(s) or hundred(s) of decimal digits, which cannot be achieved (or leads to excessive costs in terms of energy and CPU time) using mainstream MC/qMC approaches. In this paper we address this challenge by development of a new algorithm, based on tensor decompositions. We are pleased to see that the idea of the decompositional approach to matrix computation [94], which was also recognised as a top 10 algorithm of 20th century [93], can break the curse of dimensionality — arguably one of the main challenges of numerical mathematics since 1960s [95] and till the present day.

Tensor product algorithms have undergone very rapid development during the last 15 years, including progress in theory, publicly available algorithmic implementations, and growth of areas of applications. Using the idea of separation of variables, tensor methods give a new hope in lifting the curse of dimensionality and drastically reducing the computational burden associated with high-dimensional problems in a number of areas from quantum physics and chemistry to stochastics, signal processing and data analysis. In this paper we applied tensor cross interpolation algorithm [16] to reconstruct the behaviour of the given high-dimensional function from a few samples and to numerically integrate it. Our research proposes a new step in development of tensor product algorithms, by combining the algorithmic power provided by data-sparse low-rank tensor product representations, and the efficient parallel implementation utilising the potential of modern HPC systems.

The Ising susceptibility integrals, which we use in this paper to demonstrate the efficiency of the proposed method, are important not only because of their applications in the quantum theory of ferromagnetism [87], but also as a convenient benchmark for testing and comparing numerical algorithms and analytic approaches. Bailey, Borwein and Crandall [85] approached this problem from many different directions, and their results mark the state of the art of what can be achieved using the algorithms and methods of the 20th century. This is not an easy competition, and we are pleased that our algorithm stands up for it: we are able to reproduce the values calculated in [85] and also to improve the precision of physically relevant integrals from 5–6 to 18–32 decimal digits in dimensions $d \lesssim 1000$. Using multiple precision library developed by David Bailey [91], we were able to reach precision of over 100 decimal digits which revealed sub-exponential convergence of our algorithm $\varepsilon \sim \exp(-\sqrt{N_{\text{eval}}})$ for one of the considered integrals. The potential to converge sub-exponentially w.r.t. the number of function evaluations clearly distinguish the proposed method from MC/qMC algorithms, which usually demonstrate sublinear convergence $\varepsilon \sim N_{\text{eval}}^{-\alpha}$ with $0 \leqslant \alpha \leqslant 1$.

The use of multiple precision arithmetic comes at a significant price. Even though MPFUN2015 [91] and other arbitrary precision libraries [90] are well optimised, the lack of optimisation at CPU level and vectorisation at the level of BLAS operations slows the calculations down, increasing the challenge of high precision. From programming point of view, extra steps are needed to rewrite BLAS and Lapack functions in multiple precision. Although this problem is mitigated in more modern languages (such as Matlab, Python and Julia), they do not always provide enough control of parallelisation at both the distributed-memory (MPI) and shared-memory (OpenMP) levels. This is why for the development and demonstration stage we decided to implement the algorithm in Fortran, although it is clear that further work is required to provide simpler user-friendly interfaces to high-level languages mentioned above.

The context of numerical integration is particularly convenient because the final answer is simply a number, allowing us to objectively evaluate and compare the quality of different algorithms for the given problem. High accuracy of the produced results shows that for the considered examples the proposed method is superior to MC and qMC algorithms. However it must be noted that tensor cross interpolation does not just compute the integral, but indeed reconstructs the whole function in the high-dimensional tensor-product domain and represents it in TT form. When the compact representation of the function is available, it can be post-processed (e.g. interactively) to produce projections, nonlinear functionals (e.g. high-order moments), etc. This approach can be compared to calculation with functions using Chebyshëv polynomials [96], and integrating Chebyshëv interpolation together with the tensor cross interpolation seems to be a natural direction for further work, continuing the existing work in two and three-dimensions [97,98].

The most important direction of development of this work is without doubt the application of the proposed method to larger variety of applications. Many problems motivating precise high-dimensional integration are listed in [90]; we can extend this list by mentioning applications in multivariate probability [99], stochastics [37,100], and optimal control [101,102]. We are hopeful that the proposed tensor cross interpolation algorithm will demonstrate fast convergence in these applications and eventually becomes a method of choice for high-dimensional integration.

Acknowledgements

Authors appreciate financial support of EPSRC, UK which makes this work possible — grants EP/M019004/1 (S.D.) and EP/P033954/1 (D.S.). Numerical calculations for this paper were performed on a dedicated server, provided to D.S. by the University of Brighton, UK (Rising Stars grant), and on Balena High Performance Computing (HPC) Service at the University of Bath, UK.

Software

The Fortran implementation of algorithms from this paper is made by both authors and publicly available at:

- github.com/savostyanov/ttcross.

References

[1] H.-D. Meyer, U. Manthe, L.S. Cederbaum, Chem. Phys. Lett. 165 (1990) 73–78.

[2] S. Brooks, A. Gelman, G. Jones, X.-L. Meng (Eds.), HandBook of Markov Chain Monte Carlo, CRC Press, 2011.

[3] A.M. Stuart, Acta Numer. 19 (2010) 451–559, http://dx.doi.org/10.1017/S0962492910000061.

[4] N. Metropolis, S. Ulam, J. Am. Stat. Assoc. 44 (247) (1949) 335–341, http://dx.doi.org/10.1080/01621459.1949.10483310.

[5] H. Niederreiter, Bull. AMS 84 (6) (1978) 957–1041.

[6] W.J. Morokoff, R.E. Caflisch, J. Comput. Phys. 122 (2) (1995) 218–230, http://dx.doi.org/10.1006/jcph.1995.1209.

[7] I.G. Graham, F. Kuo, D. Nuyens, R. Scheichl, I.H. Sloan, J. Comput. Phys. 230 (10) (2011) 3668–3694, http://dx.doi.org/10.1016/j.jcp.2011.01.023.

[8] J. Dick, F.Y. Kuo, Q.T.L. Gia, D. Nuyens, C. Schwab, SIAM J. Num. Anal. 52 (6) (2014) 2676–2702, http://dx.doi.org/10.1137/130943984.

[9] A. Barth, C. Schwab, N. Zollinger, Numer. Math. 119 (2011) 123–161, http://dx.doi.org/10.1007/s00211-011-0377-0.

[10] F.Y. Kuo, C. Schwab, I.H. Sloan, Found. Comput. Math. 15 (2) (2015) 411–449, http://dx.doi.org/10.1007/s10208-014-9237-5.

[11] F. Kuo, R. Scheichl, C. Schwab, I. Sloan, E. Ullmann, Math. Comp. 86 (2017) 2827–2860, http://dx.doi.org/10.1090/mcom/3207.

[12] F. Nobile, L. Tamellini, F. Tesei, R. Tempone, Sparse Grids and Applications - Stuttgart 2014, Springer International Publishing, 2016, pp. 191–220.

[13] S.A. Smolyak, Dokl. Akad. Nauk SSSR 148 (5) (1963) 1042–1053, transl.: Soviet Math. Dokl. 4 (1963) 240-243.

[14] H.-J. Bungatrz, M. Griebel, Acta Numer. 13 (1) (2004) 147–269, http://dx.doi.org/10.1017/S0962492904000182.

[15] M. Bieri, C. Schwab, Comput. Methods Appl. Mech. Engrg. 198 (13–14) (2009) 1149–1170, http://dx.doi.org/10.1016/j.cma.2008.08.019.

[16] D.V. Savostyanov, Linear Algebra Appl. 458 (2014) 217–244, http://dx.doi.org/10.1016/j.laa.2014.06.006.

[17] I.V. Oseledets, SIAM J. Sci. Comput. 33 (5) (2011) 2295–2317, http://dx.doi.org/10.1137/090752286.

[18] I.V. Oseledets, E.E. Tyrtyshnikov, Linear Algebra Appl. 432 (1) (2010) 70–88, http://dx.doi.org/10.1016/j.laa.2009.07.024.

[19] S.A. Goreinov, N.L. Zamarashkin, E.E. Tyrtyshnikov, Math. Notes 62 (4) (1997) 515–519, http://dx.doi.org/10.1007/BF02358985.

[20] S.A. Goreinov, I.V. Oseledets, D.V. Savostyanov, E.E. Tyrtyshnikov, N.L. Zamarashkin, in: V. Olshevsky, E. Tyrtyshnikov (Eds.), Matrix Methods: Theory, Algorithms, Applications, World Scientific, Hackensack, NY, 2010, pp. 247–256.

[21] D.V. Savostyanov, Numer. Math. Theor. Meth. Appl. 2 (4) (2009) 439–444, http://dx.doi.org/10.4208/nmtma.2009.m9006s.

[22] I.V. Oseledets, D.V. Savostianov, E.E. Tyrtyshnikov, SIAM J. Matrix Anal. Appl. 30 (3) (2008) 939–956, http://dx.doi.org/10.1137/060655894.

[23] W. Hackbusch, S. Kühn, J. Fourier Anal. Appl. 15 (5) (2009) 706–722, http://dx.doi.org/10.1007/s00041-009-9094-9.

[24] T.G. Kolda, B.W. Bader, SIAM Rev. 51 (3) (2009) 455–500, http://dx.doi.org/10.1137/07070111X.

[25] W. Hackbusch, Tensor Spaces and Numerical Tensor Calculus, Springer–Verlag, Berlin, 2012.

[26] B.N. Khoromskij, ESAIM: Proc. 48 (2015) 1–28, http://dx.doi.org/10.1051/proc/201448001.

[27] J. Ballani, L. Grasedyck, M. Kluge, Extraction of Quantifiable Information from Complex Systems, in: Lecture Notes in Computational Science and Engineering, vol. 102, Springer, 2014, pp. 195–210, http://dx.doi.org/10.1007/978-3-319-08159-5_10.

[28] H.-J. Flad, B.N. Khoromskij, D.V. Savostyanov, E.E. Tyrtyshnikov, Rus. J. Numer. Anal. Math. Model. 23 (4) (2008) 329–344, http://dx.doi.org/10.1515/RJNAMM.2008.020.

[29] I.V. Oseledets, D.V. Savostyanov, E.E. Tyrtyshnikov, Numer. Linear Algebra Appl. 17 (6) (2010) 935–952, http://dx.doi.org/10.1002/nla.682.

[30] D.V. Savostyanov, S.V. Dolgov, J.M. Werner, I. Kuprov, Phys. Rev. B 90 (2014) 085139, http://dx.doi.org/10.1103/PhysRevB.90.085139.

[31] S. Dolgov, B. Khoromskij, Numer. Linear Algebra Appl. 22 (2) (2015) 197–219, http://dx.doi.org/10.1002/nla.1942.

[32] S.V. Dolgov, D.V. Savostyanov, Numerical Mathematics and Advanced Applications — ENUMATH 2013, Vol. 103, 2015, pp. 335–343, http://dx.doi.org/10.1007/978-3-319-10705-9_33.
[33] S.V. Dolgov, B.N. Khoromskij, D.V. Savostyanov, J. Fourier Anal. Appl. 18 (5) (2012) 915–953, http://dx.doi.org/10.1007/s00041-012-9227-4.
[34] D.V. Savostyanov, Linear Algebra Appl. 436 (9) (2012) 3215–3224, http://dx.doi.org/10.1016/j.laa.2011.11.008.
[35] S.V. Dolgov, A.P. Smirnov, E.E. Tyrtyshnikov, J. Comput. Phys. 263 (2014) 268–282, http://dx.doi.org/10.1016/j.jcp.2014.01.029.
[36] Z. Zheng, X. Yang, I.V. Oseledets, G.E. Karniadakis, L. Daniel, IEEE Trans. Comput.-aided Des. Integr. Circuits Syst. 34 (1) (2015) 63–76, http://dx.doi.org/10.1109/TCAD.2014.2369505.
[37] S. Dolgov, B.N. Khoromskij, A. Litvinenko, H.G. Matthies, SIAM J. Uncertain. Quantif. 3 (1) (2015) 1109–1135, http://dx.doi.org/10.1137/140972536.
[38] J.A. Roberts, D.V. Savostyanov, E.E. Tyrtyshnikov, J. Comput. Appl. Math. 260 (2014) 434–448, http://dx.doi.org/10.1016/j.cam.2013.10.025.
[39] S. Dolgov, J.W. Pearson, D.V. Savostyanov, M. Stoll, Appl. Math. Comput. 273 (2016) 604–623, http://dx.doi.org/10.1016/j.amc.2015.09.042.
[40] E. Solomonik, D. Matthews, J.R. Hammond, J.F. Stanton, J. Demmel, J. Parallel Distrib. Comput. 74 (12) (2014) 3176–3190, http://dx.doi.org/10.1016/j.jpdc.2014.06.002.
[41] E. Stoudenmire, S. White, Phys. Rev. B 87 (15) (2013) 155137, http://dx.doi.org/10.1103/PhysRevB.87.155137.
[42] S. Etter, SIAM J. Sci. Comput. 38 (4) (2016) A2585–A2609, http://dx.doi.org/10.1137/15M1038852.
[43] L. Grasedyck, R. Kriemann, C. Löbbert, A. Nägel, G. Wittum, K. Xylouris, Comput. Vis. Sci. 17 (2) (2015) 67–78, http://dx.doi.org/10.1007/s00791-015-0247-x.
[44] F.R. Gantmacher, The Theory of Matrices, Clelsea, NY, 1959.
[45] S.A. Goreinov, E.E. Tyrtyshnikov, N.L. Zamarashkin, Rep. Russ. Acad. Sci. 342 (2) (1995) 151–152.
[46] E.E. Tyrtyshnikov, Computing 64 (4) (2000) 367–380, http://dx.doi.org/10.1007/s006070070031.
[47] A. Einstein, Ann. Phys. 354 (7) (1916) 769–822, http://dx.doi.org/10.1002/andp.19163540702.
[48] S.A. Goreinov, E.E. Tyrtyshnikov, N.L. Zamarashkin, Linear Algebra Appl. 261 (1997) 1–21, http://dx.doi.org/10.1016/S0024-3795(96)00301-1.
[49] P. Drineas, R. Kannan, M.W. Mahoney, SIAM J. Comput. 36 (1) (2006) 184–206, http://dx.doi.org/10.1137/S0097539704442702.
[50] E.E. Tyrtyshnikov, Calcolo 33 (1) (1996) 47–57, http://dx.doi.org/10.1007/BF02575706.
[51] J. Schneider, J. Approx. Theory 162 (2010) 1685–1700, http://dx.doi.org/10.1016/j.jat.2010.04.012.
[52] S.A. Goreinov, E.E. Tyrtyshnikov, Dokl. Math. 83 (3) (2011) 374–375, http://dx.doi.org/10.1134/S1064562411030355.
[53] N.L. Zamarashkin, A.I. Osinsky, Dokl. Math. 94 (3) (2016) 643–645, http://dx.doi.org/10.1134/S1064562416060156.
[54] A.Y. Mikhalev, I.V. Oseledets, Linear Algebra Appl. 538 (2018) 187–211, http://dx.doi.org/10.1016/j.laa.2017.10.014.
[55] J.J. Bartholdi III, Oper. Res. Lett. 1 (5) (1982) 190–193.
[56] G. Golub, W. Kahan, SIAM J. Numer. Anal. 2 (2) (1965) 205–224.
[57] M. Gu, C. Eisenstat, SIAM J. Sci. Comput. 17 (4) (1996) 848–869, http://dx.doi.org/10.1137/0917055.
[58] C.-T. Pan, Linear Algebra Appl. 316 (1–3) (2000) 199–222, http://dx.doi.org/10.1016/S0024-3795(00)00120-8.
[59] D.E. Knuth, Linear Multilinear Algebra 17 (1985) 1–4, http://dx.doi.org/10.1080/03081088508817636.
[60] M. Bebendorf, Numer. Math. 86 (4) (2000) 565–589, http://dx.doi.org/10.1007/pl00005410.
[61] G.H. Golub, C.F. Van Loan, Matrix Computations, Johns Hopkins University Press, Baltimore, MD, 2013.
[62] J.H. Wilkinson, J. Assoc. Comput. Mach. 8 (1961) 281–330, http://dx.doi.org/10.1145/321075.321076.
[63] L. Neal, G. Poole, Linear Algebra Appl. 173 (1992) 239–264, http://dx.doi.org/10.1016/0024-3795(92)90432-A.
[64] L.V. Foster, J. Comput. Appl. Math. 86 (1) (1997) 177–194, http://dx.doi.org/10.1016/S0377-0427(97)00154-4.
[65] G. Poole, L. Neal, J. Comput. Appl. Math. 123 (2000) 353–369, http://dx.doi.org/10.1016/S0377-0427(00)00406-4.
[66] K.N. Kumar, J. Schneider, Linear Multilinear Algebra 65 (11) (2017) 2212–2244, http://dx.doi.org/10.1080/03081087.2016.1267104.
[67] L. Grasedyck, SIAM J. Matrix Anal. Appl. 31 (4) (2010) 2029–2054, http://dx.doi.org/10.1137/090764189.
[68] J. Ballani, L. Grasedyck, M. Kluge, Linear Algebra Appl. 428 (2013) 639–657, http://dx.doi.org/10.1016/j.laa.2011.08.010.
[69] J. Ballani, L. Grasedyck, SIAM/ASA J. Uncertain. Quantif. 3 (1) (2015) 852–872, http://dx.doi.org/10.1137/140960980.
[70] D.V. Savostyanov, I.V. Oseledets, Proceedings of 7th International Workshop on Multidimensional Systems (NDS), IEEE, 2011, http://dx.doi.org/10.1109/nDS.2011.6076873.
[71] S.R. White, Phys. Rev. Lett. 69 (19) (1992) 2863–2866, http://dx.doi.org/10.1103/PhysRevLett.69.2863.
[72] M. Fannes, B. Nachtergaele, R. Werner, Comm. Math. Phys. 144 (3) (1992) 443–490, http://dx.doi.org/10.1007/BF02099178.
[73] A. Klümper, A. Schadschneider, J. Zittartz, Europhys. Lett. 24 (4) (1993) 293–297, http://dx.doi.org/10.1209/0295-5075/24/4/010.
[74] I.V. Oseledets, S.V. Dolgov, SIAM J. Sci. Comput. 34 (5) (2012) A2718–A2739, http://dx.doi.org/10.1137/110833142.
[75] S.V. Dolgov, B.N. Khoromskij, I.V. Oseledets, SIAM J. Sci. Comput. 34 (6) (2012) A3016–A3038, http://dx.doi.org/10.1137/120864210.
[76] E. Tadmor, SIAM J. Numer. Anal. 23 (1986) 1–23.
[77] J. Sherman, W.J. Morrison, Ann. Math. Stat. 21 (1) (1950) 124–127, http://dx.doi.org/10.1214/aoms/1177729893.
[78] M.A. Woodbury, Statistical Research Group, in: Memo. Rep., vol. 42, Princeton University, Princeton, N.J, 1950.
[79] W.W. Hager, SIAM Rev. 31 (2) (1989) 221–239, http://dx.doi.org/10.1137/1031049.
[80] E.E. Tyrtyshnikov, Linear Algebra Appl. 379 (2004) 423–437, http://dx.doi.org/10.1016/j.laa.2003.08.013.
[81] W. Hackbusch, D. Braess, IMA J. Numer. Anal. 25 (4) (2005) 685–697.
[82] W. Hackbusch, B.N. Khoromskij, Computing 76 (3–4) (2006) 177–202, http://dx.doi.org/10.1007/s00607-005-0144-0.
[83] A.I. Osinsky, Comput. Math. Math. Phys. 59 (2) (2019) 201–206, http://dx.doi.org/10.1134/S096554251902012X.
[84] J. Dick, F.Y. Kuo, I.H. Sloan, Acta Numer. 22 (2013) 133–288, http://dx.doi.org/10.1017/S0962492913000044.
[85] D.H. Bailey, J.M. Borwein, R.E. Crandall, J. Phys. A: Math. Gen. 39 (2006) 12271–12302, http://dx.doi.org/10.1088/0305-4470/39/40/001.
[86] C.N. Yang, Phys. Rev. 85 (1952) 808, http://dx.doi.org/10.1103/PhysRev.85.808.
[87] T.T. Wu, B.M. McCoy, C.A. Tracy, E. Barouch, Phys. Rev. B 13 (1) (1976) 316, http://dx.doi.org/10.1103/PhysRevB.13.316.
[88] D.H. Bailey, Comput. Sci. Eng. 2 (1) (2000) 24–28, http://dx.doi.org/10.1109/5992.814653.
[89] E. Panzer, Comput. Phys. Comm. 188 (2015) 148–166, http://dx.doi.org/10.1016/j.cpc.2014.10.019.
[90] D.H. Bailey, J.M. Borwein, Mathematics 3 (2015) 337–367, http://dx.doi.org/10.3390/math3020337.
[91] D.H. Bailey, MPFUN2015: A thread-safe arbitrary precision computation package. https://www.davidhbailey.com/dhbpapers/mpfun2015.pdf.
[92] B. Nickel, J. Phys. A 32 (21) (1999) 3889, http://dx.doi.org/10.1088/0305-4470/32/21/303.
[93] J. Dongarra, F. Sullivan, Comput. Sci. Eng. 2 (1) (2000) 22–23.
[94] G.W. Stewart, Comput. Sci. Eng. 2 (1) (2000) 50–59, http://dx.doi.org/10.1109/5992.814658.
[95] R.E. Bellman, Dynamic Programming, Princeton University Press, 1957.
[96] L.N. Trefethen, Approximation Theory and Approximation Practice, SIAM, 2013.
[97] A. Townsend, L.N. Trefethen, SIAM J. Sci. Comput. 35 (6) (2013) C495–C518, http://dx.doi.org/10.1137/130908002.
[98] B. Hashemi, L.N. Trefethen, SIAM J. Sci. Comput. 39 (5) (2017) C341–C363, http://dx.doi.org/10.1137/16M1083803.
[99] S. Dolgov, K. Anaya-Izquierdo, C. Fox, R. Scheichl, Approximation and sampling of multivariate probability distributions in the tensor train decomposition, arXiv preprint 1810.01212 2018. URL http://arxiv.org/abs/1810.01212.
[100] S. Dolgov, R. Scheichl, SIAM/ASA J. Uncertain. Quantif. 7 (1) (2019) 260–291, http://dx.doi.org/10.1137/17M1138881.
[101] S. Dolgov, J.W. Pearson, Preconditioners and tensor product solvers for optimal control problems from chemotaxis, arXiv preprint 1806.08539 2018. URL http://arxiv.org/abs/1806.08539.
[102] D. Quiñones Valles, S. Dolgov, D. Savostyanov, in: C. Constanda, P.J. Harris (Eds.), Integral Methods in Science and Engineering, Birkhäuser, Cham, 2019, pp. 367–379, http://dx.doi.org/10.1007/978-3-030-16077-7_29.