TU Delft

Delft University of Technology

Performance of linear solvers in tensor-train format on current multicore architectures

Röhrig-Zöllner, Melven; Becklas, Manuel; Thies, Jonas; Basermann, Achim

DOI
10.1177/10943420251317994

Publication date
2025

Document Version
Final published version

Published in
International Journal of High Performance Computing Applications

Citation (APA)
Röhrig-Zöllner, M., Becklas, M., Thies, J., & Basermann, A. (2025). Performance of linear solvers in tensor-train format on current multicore architectures. *International Journal of High Performance Computing Applications*, *39*(3), 443-461. https://doi.org/10.1177/10943420251317994

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Check for updates

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
2025, Vol. 0(0) 1–19
© The Author(s) 2025

Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/10943420251317994
journals.sagepub.com/home/hpc
Sage

Performance of linear solvers in tensor-train format on current multicore architectures

Melven Röhrig-Zöllner[1], Manuel Becklas[1], Jonas Thies[2] and Achim Basermann[1]

Abstract

Tensor networks are a class of algorithms aimed at reducing the computational complexity of high-dimensional problems. They are used in an increasing number of applications, from quantum simulations to machine learning. Exploiting data parallelism in these algorithms is key to using modern hardware. However, there are several ways to map required tensor operations onto linear algebra routines ("building blocks"). Optimizing this mapping impacts the numerical behavior, so computational and numerical aspects must be considered hand-in-hand. In this paper we discuss the performance of solvers for low-rank linear systems in the tensor-train format (also known as matrix-product states). We consider three popular algorithms: TT-GMRES, MALS, and AMEn. We illustrate their computational complexity based on the example of discretizing a simple high-dimensional PDE in, for example, 50^{10} grid points. This shows that the projection to smaller subproblems for MALS and AMEn reduces the number of floating-point operations by orders of magnitude. We suggest optimizations regarding orthogonalization steps, singular value decompositions, and tensor contractions. In addition, we propose a generic preconditioner based on a TT-rank-1 approximation of the linear operator. Overall, we obtain roughly a 5× speedup over the reference algorithm for the fastest method (AMEn) on a current multicore CPU.

Keywords

Low-rank tensor algorithms, node-level performance, tensor-train format, matrix-product states, linear solvers

1. Introduction

Low-rank tensor methods provide a way to approximately solve problems that would otherwise require huge amounts of memory and computing time. Many ideas in this field arise from quantum physics. For example, the global state of a quantum system with N two-state particles can be expressed as a tensor of dimension 2^N. In this setting, interesting states are for example given by the eigenvectors of the smallest eigenvalues of the Hamiltonian of the system–a Hermitian linear operator that describes the energy of the system. Therefore, most work focuses on solving eigenvalue problems for Hermitian/symmetric operators using the DMRG method (Schollwöck, 2005; White, 1992; Wilson, 1983). However, linear systems in low-rank tensor formats also arise in interesting applications for example for solving high-dimensional or parameterized partial differential equations, see, for example, Kressner and Tobler (2011); Dahmen et al. (2015); Dolgov and Pearson (2019). In addition, linear solvers and eigenvalue solvers are closely related and many successful methods for finding eigenvalues are based on successive linear solves. This paper addresses iterative methods for solving linear systems (symmetric and non-symmetric) in the tensor-train (TT) format for the case where the individual dimensions are not tiny, that is, for systems of dimension $n^d \times n^d$ with $n \gg 2$. We employ the TT format (called matrix-product states in physics) as it is a simple and common low-rank tensor format. Most of the ideas are transferable to other low-rank tensor formats (at least to loop-free tensor networks). Our work considers the TT-GMRES algorithm (Ballani and Grasedyck, 2012;

[1]German Aerospace Center (DLR), Institute of Software Technology, Cologne, Germany
[2]Delft Institute of Applied Mathematics, Delft University of Technology, Delft, Netherlands

Corresponding author:
Melven Röhrig-Zöllner, German Aerospace Center, Linder Höhe, Cologne 51147, Germany.
Email: Melven.Roehrig-Zoellner@DLR.de

Dolgov, 2013), the modified alternating least-squares (MALS) algorithm (DMRG for linear systems) (Holtz et al., 2012; Oseledets and Dolgov, 2012), and the Alternating Minimal Energy (AMEn) algorithm (Dolgov and Savostyanov, 2014). We show numerical improvements and performance improvements of the underlying operations and focus on a single CPU node. These improvements are orthogonal to the parallelization for distributed memory systems presented in Daas et al. (2022), so the suggestions from both Daas et al. (2022) and this paper could be combined in the future. As the resulting complete linear solver requires a tight interplay of different algorithmic components, we discuss the behavior of the different numerical methods involved for the TT format. An alternative class of methods to solve linear systems in TT format consists of Riemannian optimization on the manifold of fixed-rank tensor-trains, see Kressner et al. (2016). This results in a nonlinear optimization problem and is therefore not in the scope of this paper. However, it partly requires similar underlying operations.

The paper is organized as follows: First, we start with required numerical background in Section 2 and also introduce relevant performance metrics for today's multicore computers. Then in Section 3, we discuss the involved high-level numerical algorithms TT-GMRES, MALS, and AMEn. Based on an example, we illustrate the numerical behavior of these algorithms and compare their complexity in Section 4. Afterwards, in Section 5, we analyze and optimize the underlying building blocks. We conclude in Section 6 with a short summary and open points for future work.

2. Background and notation

In this section, we provide the required background concerning numerics and performance.

2.1. Numerical background

We first introduce required matrix decompositions and a notation for the considered algorithms.

2.1.1. Matrix decompositions. As matrix decompositions are heavily used as steps in tensor-train algorithms, we repeat some basic properties of QR and SVD decompositions from the literature, see for example, Golub and Van Loan (2013); Higham (2002).

A QR-decomposition is a factorization of a matrix $M \in \mathbf{R}^{n_1 \times n_2}$ with column rank r into a matrix Q with orthonormal columns and an upper triangular part R:

$$M = QR, \qquad \text{with} \qquad Q^T Q = I, Q \in \mathbf{R}^{n_1 \times r}, R \in \mathbf{R}^{r \times n_2}. \tag{1}$$

For (numerically) rank-deficient M, one can employ a pivoted QR-decomposition

$$MP = QR, \qquad \Leftrightarrow \qquad M = QRP^{-1}, \tag{2}$$

where P is a permutation matrix. The pivoted QR-decomposition can be computed in a numerically robust way. However, it cannot (safely) be used to approximate M with a lower rank matrix based on the size of the pivot elements (diagonal entries of R) by $Q_{:,1:r'}R_{1:r',:}$, $r' < r$ as the worst case error grows with $O(2^r)$, see Higham (1990); Kawamura and Suda (2021).

The singular value decomposition (SVD) in contrast provides the best approximation of lower rank:

$$\begin{aligned} M = USV^T \qquad &\Rightarrow \qquad \|M - U_{:,1:r'}S_{1:r',1:r'}V^T_{:,1:r'}\|_F \\ &= \min_{\tilde{M},\, \mathrm{rank}(\tilde{M})=r'} \|M - \tilde{M}\|_F, \end{aligned} \tag{3}$$

where $U \in \mathbf{R}^{n_1 \times r}$, $V \in \mathbf{R}^{n_2 \times r}$ are the matrices of the orthonormal left/right singular vectors and $S = \mathrm{diag}(s_1, s_2, \ldots, s_r)$ is composed of the singular values $s_1 \geq s_2 \geq \cdots \geq s_r > 0$.

2.1.2. Tensor-train decomposition. In higher dimensions, there is no unique way to decompose a tensor into factors and to define its rank(s) (variants are e.g., the Tucker and the CANDECOMP/PARAFAC (CP) decompositions, see the review Kolda and Bader, 2009). We focus on the tensor-train format which decomposes a tensor $X \in \mathbf{R}^{n_1 \times n_2 \times \cdots \times n_d}$ into d three-dimensional sub-tensors X_1, ..., X_d:

$$\begin{aligned} X &= X_1 \bowtie X_2 \bowtie \cdots \bowtie X_d \\ \text{with} \qquad X_k &\in \mathbf{R}^{r_{k-1} \times n_k \times r_k}, r_0 = r_d = 1. \end{aligned} \tag{4}$$

Here, $(\cdot \bowtie \cdot)$ is the contraction of the last dim. of the left operand with the first dim. of the right operand:

$$X_k \bowtie X_{k+1} = \sum_i (X_k)_{:,:,i}(X_{k+1})_{i,:,:} \in \mathbf{R}^{r_{k-1} \times n_k \times n_{k+1} \times r_{k+1}}. \tag{5}$$

The TT decomposition of a given tensor is not unique: it is invariant with respect to multiplying one sub-tensor by a matrix and the next with its inverse. More precisely, for $M, N^T \in \mathbf{R}^{r_k \times r'_k}$ with $MN = I$:

$$\begin{aligned} \overline{X}_k \bowtie \overline{X}_{k+1} &= X_k \bowtie X_{k+1} \\ \text{for} \qquad \overline{X}_k &:= X_k \bowtie M, \quad \overline{X}_{k+1} := N \bowtie X_{k+1}. \end{aligned} \tag{6}$$

The smallest possible dimensions $(r_1, \ldots r_{d-1})$ that allow to represent X denote the TT ranks of X with the maximal rank $r := rank(X) = \max(r_1, \ldots, r_{d-1})$. If X has rank-1 in the TT format, we can also write it as a generalized dyadic product of a set of vectors:

$$X = (X_1)_{1,:,1} \otimes (X_2)_{1,:,1} \otimes \cdots \otimes (X_d)_{1,:,1}.$$

2.1.3. Tensor unfolding and orthogonalities. We define a general *reshape* operation to reinterpret the entries of a tensor as a tensor of different dimensions:

$$\begin{aligned} &\text{reshape}\left(X, \begin{pmatrix} \overline{n}_1 & \cdots & \overline{n}_{\overline{d}} \end{pmatrix}\right) := \overline{X} \in \mathbf{R}^{\overline{n}_1 \times \cdots \times \overline{n}_{\overline{d}}} \quad \text{with} \\ &\quad (\overline{X})_{\overline{i}_1, \ldots, \overline{i}_{\overline{d}}} = (X)_{i_1, \ldots, i_d} \\ &\quad \text{for} \quad \overline{i}_1 + \overline{i}_2 \overline{n}_1 + \overline{i}_3 \overline{n}_1 \overline{n}_2 + \cdots \\ &\qquad\quad = i_1 + i_2 n_1 + i_3 n_1 n_2 + \cdots . \end{aligned}$$

With this, we define the *left-unfolding* that combines two dimensions of a 3d tensor X_k to obtain a matrix:

$$(X_k)_{\text{left}} := \text{reshape}(X_k, \begin{pmatrix} r_{k-1} n_k & r_k \end{pmatrix}) \in \mathbf{R}^{r_{k-1} n_k \times r_k}. \tag{7}$$

Similarly, we define the *right-unfolding*:

$$(X_k)_{\text{right}} := \text{reshape}(X_k, \begin{pmatrix} r_{k-1} & n_k r_k \end{pmatrix}) \in \mathbf{R}^{r_{k-1} \times n_k r_k}. \tag{8}$$

We denote a 3d tensor X_k as *left-orthogonal* if the columns of the left-unfolding are orthonormal $((X_k)_{\text{left}}^T (X_k)_{\text{left}} = I)$ and *right-orthogonal* if the rows of the right-unfolding are orthonormal $((X_k)_{\text{right}} (X_k)_{\text{right}}^T = I)$. From the TT format, we can build an SVD of an unfolding of $X \in \mathbf{R}^{n_1 \times n_2 \cdots \times n_d}$ into a set of left and right dimensions

$$USV^T = (X)_{\text{unfold}_j} := \text{reshape}(X, \begin{pmatrix} n_1 \cdots n_j & n_{j+1} \cdots n_d \end{pmatrix}) \tag{9}$$

by left-orthogonalizing $(X_1, \ldots, X_j)$ and right-orthogonalizing $(X_{j+1}, \ldots, X_d)$:

$$X = \underbrace{\overline{X}_1 \bowtie \cdots \bowtie \overline{X}_j}_{U} \bowtie S \bowtie \underbrace{\overline{X}_{j+1} \bowtie \cdots \bowtie \overline{X}_d}_{V^T}. \tag{10}$$

2.1.4. Tensor-train vectors and operators. A tensor-train operator is a tensor in TT format where we combine pairs of dimensions $(n_i \times m_i)$ with the form $\mathcal{A} \in \mathbf{R}^{(n_1 \times m_1) \times (n_2 \times m_2) \times \cdots \times (n_d \times m_d)}$

$$\begin{aligned} &\mathcal{A} = A_1 \bowtie A_2 \bowtie \cdots A_d \\ &\text{with} \qquad A_k \in \mathbf{R}^{r^A_{k-1} \times n_k \times m_k \times r^A_k}, r^A_0 = r^A_d = 1 \end{aligned} \tag{11}$$

which defines a linear mapping: $\mathcal{A} : \mathbf{R}^{m_1 \times m_2 \times \cdots \times m_d} \to \mathbf{R}^{n_1 \times n_2 \times \cdots \times n_d}$. For the scope of this paper, we only consider quadratic regular operators ($m_k = n_k$). Again, the sub-tensor dimensions $(r^A_1, \ldots r^A_{d-1})$ denote the TT ranks of the operator with a maximal rank of $r^A := \text{rank}(\mathcal{A}) = \max(r^A_1, \ldots r^A_{d-1})$. This definition allows an efficient application of a TT operator on a TT "vector" directly in the TT format:

$$\begin{aligned} &\mathcal{A}X = Y = Y_1 \bowtie Y_2 \bowtie \cdots \bowtie Y_n \qquad \text{with} \\ &Y_k := \text{reshape} \\ &\qquad \left(\sum_i (A_k)_{:,:,i,:} (X_k)_{:,i,:,:}, \begin{pmatrix} r^A_{k-1} r_{k-1} & n_k & r^A_k r_k \end{pmatrix} \right). \end{aligned} \tag{12}$$

The resulting TT decomposition has ranks $(r^A_1 r_1, \ldots, r^A_{d-1} r_{d-1})$ and thus $\textit{rank}\,(Y) \leq r^A r$.

With all these definitions at hand, we can specify the main problem considered in this paper: Given a low-rank linear operator in TT format $\mathcal{A}_{\text{TT}} := (A_1, \ldots A_d)$, a low-rank right-hand side (RHS) $B_{\text{TT}} := (B_1, \ldots, B_d)$, and a desired residual tolerance ϵ_{tol}, find an approximate low-rank solution $X_{\text{TT}} := (X_1, \ldots, X_d)$ with

$$\|\mathcal{A}_{\text{TT}} X_{\text{TT}} - B_{\text{TT}}\|_F \leq \epsilon_{\text{tol}}. \tag{13}$$

2.2. Performance characteristics on today's multicore CPU systems

Today's hardware features multiple levels of parallelism and memory that we need to exploit to efficiently use the available compute capacity: A supercomputer is composed of a number of nodes connected by a network (distributed memory parallelism). Each node contains one or more multicore CPUs with access to a main memory (shared memory parallelism). However, the different CPUs access different memory domains with different speed (NUMA architecture): Usually the access to the memory banks directly connected to one CPU is faster. Each CPU "socket" consists of multiple cores ($\lesssim 100$ in 2023) with a hierarchy of caches. Inside one core, SIMD units perform identical calculations on a small vector of floating-point numbers. We focus on the performance on a single multicore CPU, but the algorithms considered can also run in parallel on a cluster (see Daas et al., 2022). Many supercomputers nowadays use GPUs which is not discussed further in this paper.

2.2.1. Roofline performance model. To obtain a simpler abstraction of the hardware, we employ the Roofline (Williams et al., 2009) performance model. The Roofline model distinguishes between computations (floating-point operations) and data transfers: The maximal performance one can achieve on a given hardware is P_{peak} [GFlop/s]. The bandwidth of data transfers from main memory is b_s [GByte/s]. If all data fits into some cache level, the appropriate cache bandwidth is used instead. These are the required hardware characteristics. The considered characteristics of one building block of an algorithm (e.g., one nested loop) are the number of required floating-point operations n_{flops} and the volume of the data transfers $V_{\text{read/write}}$. Their ratio is called computational intensity

$I_c := n_{\text{flops}}/V_{\text{read/write}}$ [Flop/Byte]. Assuming that the data transfers and operations overlap perfectly, this results in the following performance:

$$P_{\text{roofline}} = \min\left(P_{\text{peak}}, I_c b_s\right). \tag{14}$$

If the compute intensity is low ($I_c \ll P_{\text{peak}}/b_s$), the building block is *memory bound*. If in contrast the compute intensity is high ($I_c \gg P_{\text{peak}}/b_s$), the building block is *compute bound*. We specifically split the algorithms considered in this paper into smaller parts ("building blocks") because they feature not one dominating operation but are composed of multiple different blocks with different performance characteristics.

2.2.2. Memory and cache performance. In addition to the model above, some details of the memory hierarchy play a crucial role for the algorithms at hand (see Hager and Wellein (2010) for more details): First, modifying memory is often slower than reading it. In order to write to main memory, the memory region is usually first transferred to the cache, modified there and written back (write-allocate). A special CPU instruction allows to avoid this and directly stream to memory (non-temporal store). A common technique to improve the performance of data transfers is to avoid writing large (temporary) data when the algorithm can be reformulated accordingly (write-avoiding), see Carson et al. (2016). Second, the CPU caches are organized in cache lines: This means that, e.g., 8 double precision values are transferred together, always starting from a memory address divisible by the cache line size. So the data locality–e.g., which index is stored contiguously–is important.

In addition, today's CPUs use set-associative caches that allow the mapping of one memory address to a fixed set of cache lines. Due to this, memory addresses with a specific distance (e.g., 1024 double numbers) are mapped to the same cache set and the cache effectiveness is dramatically reduced when data is accessed with specific "bad" strides (*cache thrashing*). This easily occurs for tensor operations if the product of some dimensions is close to a power of two. A common solution for operations on 2d arrays is padding: adding a few ignored zero rows in a matrix such that the stride is at least a few cache lines bigger than some power of two. This becomes more complicated in higher dimensions as one either obtains a complex indexing scheme or one needs to perform calculations with zeros. This is discussed in more detail in Section 5.3.

3. Numerical algorithms

In this section, we discuss three different methods to approximately solve a linear system in TT format as in equation (13). We start with a general purpose ("global") approach based on Krylov subspace methods, TT-GMRES (Dolgov, 2013), and present some improvements for the TT format. Then, we consider the more specialized ("local") MALS (Holtz et al., 2012; Oseledets and Dolgov, 2012), which optimizes pairs of sub-tensors similar to DMRG. Afterwards, we discuss the ("more local") AMEn (Dolgov and Savostyanov, 2014) method, which iterates on one sub-tensor after another. Finally, we present a simple yet effective preconditioner in TT format to accelerate convergence of TT-GMRES.

3.1. Krylov methods: TT-GMRES

All methods that apply the linear operator $\mathcal{A}_{\text{TT}}$ on linear combinations of previously calculated directions produce solutions from the Krylov subspace $\mathbf{K}_k(\mathcal{A}_{\text{TT}}; V_{\text{TT}}) := \text{span}\{V_{\text{TT}}, \mathcal{A}_{\text{TT}} V_{\text{TT}}, \ldots, \mathcal{A}_{\text{TT}}^{k-1} V_{\text{TT}}\}$ where V_{TT} is usually the initial residual of the problem. Different Krylov subspace methods then select the "best" solution from the subspace $\mathbf{K}_k$ according to different definitions of "best," see van der Vorst (2003) and Saad (2003) for a detailed discussion. As we usually only approximate intermediate steps in TT arithmetic, we effectively employ inexact Krylov methods which are discussed thoroughly in Simoncini and Szyld (2003); van den Eshof and Sleijpen (2004). In this paper, we consider the TT-GMRES method (Dolgov, 2013) for non-symmetric problems. For symmetric operators $\mathcal{A}_{\text{TT}}$, we simply omit unneeded steps to obtain a MINRES variant. However, all considerations here effectively hold for other Krylov subspace methods as well.

3.1.1. Arithmetic operations in tensor-train format. Krylov methods require the following operations which can be performed directly in the TT format (all introduced in Oseledets, 2011): Applying the operator to a vector ($Y_{\text{TT}} \leftarrow \mathcal{A}_{\text{TT}} X_{\text{TT}}$), dot products ($\alpha \leftarrow \langle X_{\text{TT}}, Y_{\text{TT}} \rangle$) and scaled additions ($Y_{\text{TT}} \leftarrow \alpha X_{\text{TT}} + Y_{\text{TT}}$) of two tensor-trains. To reduce the computational complexity, TT truncation ($\widetilde{X}_{\text{TT}} \leftarrow \text{trunc}_\delta(X_{\text{TT}})$) approximates a tensor-train with another tensor-train with lower rank (see TT-rounding algorithm in Oseledets, 2011):

$$\|X_{\text{TT}} - \text{trunc}_\delta(X_{\text{TT}})\|_F \leq \delta.$$

With these operations, we can perform a variant of the GMRES algorithm with additional truncation operations, see Alg. 1. This idea was first discussed in Ballani and Grasedyck (2012) for the more general H-Tucker format with a slightly different projection. We employ a variant based on Dolgov (2013). The numerical stability of TT-GMRES is analyzed in more detail in Coulaud et al. (2022). We remark that we use a more strict truncation tolerance than suggested in Dolgov (2013) based on the analysis of inexact Krylov methods in Simoncini and Szyld (2003). However, in Simoncini and Szyld (2003) only inaccurate applications of the linear operator are considered (as in line 5 of Alg. 1). We also truncate in each step of the orthogonalization (line 8) and once afterwards

Algorithm 1 TT-GMRES with modified Gram-Schmidt

Input: Linear operator $\mathcal{A}_{\mathrm{TT}} : \mathbf{R}^{n_1 \times \cdots \times n_d} \to \mathbf{R}^{n_1 \times \cdots \times n_d}$ and RHS $B_{\mathrm{TT}} \in \mathbf{R}^{n_1 \times \cdots \times n_d}$, desired tolerance ϵ, max. number of iterations m, estimated condition number c

Output: Approximate solution X_{TT} with $\|B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}}\|_F \lesssim \epsilon \|B_{\mathrm{TT}}\|_F$

1: $\gamma_0 \leftarrow \|B_{\mathrm{TT}}\|_F$
2: $V_{TT,1} \leftarrow 1/\gamma_0 B_{\mathrm{TT}}$
3: **for** $i = 1, \ldots, m$ **do**
4: Choose tolerance $\delta_i = \frac{0.5\epsilon}{cm} \frac{\gamma_0}{\gamma_{i-1}}$ *(In Dolgov (2013): $\delta_i = \epsilon \frac{\gamma_0}{\gamma_{i-1}}$)*
5: $W_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\delta_i/(i+1)}(\mathcal{A}_{\mathrm{TT}} V_{\mathrm{TT},i})$
6: **for** $j = 1, \ldots, i$ **do**
7: $h_{i,j} \leftarrow \langle V_{\mathrm{TT},j}, W_{\mathrm{TT}} \rangle$ *(MINRES: $h_{i,j} = 0, j < i-1$)*
8: $W_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\delta_i/(i+1)}(W_{\mathrm{TT}} - h_{i,j} V_{\mathrm{TT},j})$
9: **end for**
10: $W_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\delta_i}(W_{\mathrm{TT}})$
11: $h_{i+1,i} \leftarrow \|W_{\mathrm{TT}}\|_F$
12: $V_{\mathrm{TT},i+1} \leftarrow 1/h_{i+1,i} W_{\mathrm{TT}}$
13: $y \leftarrow \arg\min_y \|Hy - \gamma_0 e_1\|_2$ for $H = (h_{k,l}),\ k = 1, \ldots, i+1,\ l = 1, \ldots, i$
14: $\gamma_i \leftarrow \|Hy - \gamma_0 e_1\|_2$
15: **if** $\gamma_i / \gamma_0 \leq 0.5\epsilon$ **then**
16: $X_{\mathrm{TT}} \leftarrow 0$
17: **for** $j = 1, \ldots, i$ **do**
18: $X_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\epsilon/(ci)}(X_{\mathrm{TT}} + y_j V_{\mathrm{TT},j})$
19: **end for**
20: **return**
21: **end if**
22: **end for**
23: Abort: not converged in m steps!

(line 10). We can still express the error as an error in the operator of the form (see eq. (2.2) in Simoncini and Szyld, 2003):

$$(A + E_i)v_i = V_{i+1} H_i, \qquad i = 1, \ldots, m.$$

We denote the errors of all truncation operations in one Arnoldi iteration with $\Delta w^{(0)}$ (line 5), $\Delta w^{(j)}$ (line 8) and $\Delta w^{(i+1)}$. Then, we obtain for the error:

$$E_i v_i = \Delta w^{(i+1)} + \sum_{j=0}^{i} \left(I - \sum_{k=j+1}^{i} v_k v_k^T \right) \Delta w^{(j)}.$$

For orthogonal basis vectors v_k of the Krylov subspace, the error of the Arnoldi iteration is bounded by:

$$\|E_i\| \leq \|\Delta w^{(i+1)}\|_F + \sum_{j=0}^{i} \|\Delta w^{(j)}\|_F \leq 0.5\delta_i \|w^{(i+1)}\|_F + \sum_{j=0}^{i} \frac{0.5\delta_i}{i+1} \|w^{(j)}\|_F \leq \delta_i \|A\|.$$

Of course, due to the truncations, one easily looses the orthogonality of the basis vectors v_k, see discussion in Section 3.1.2 below. Suitable tolerances δ_i require the condition number $c = \kappa(H_m)$ that we estimate using the parameter $c \approx \kappa(A)$ as suggested in Simoncini and Szyld (2003). So we obtain the following bound for the difference between the true residual vector r_* and the inexact residual vector $\widetilde{r}_*$ (see equation (5.8) in Simoncini and Szyld (2003)):

$$\|r_* - \widetilde{r}_*\|_F \leq 0.5\epsilon.$$

The factor 0.5 ensures that the true residual norm is smaller than the desired tolerance:

$$\|r_*\| = \|r_* - \widetilde{r}_* + \widetilde{r}_*\| \leq \|r_* - \widetilde{r}_*\| + \|\widetilde{r}_*\| \leq \epsilon.$$

In our experiments, we use an optimized variant (see Section 5.1) of the standard TT truncation algorithm. An alternative randomized truncation algorithm is presented in Daas et al. (2022) for truncating sums of multiple tensor-trains (e.g., only truncating in line 10 of Alg. 1 and not in line 8).

3.1.2. Improved Gram-Schmidt orthogonalization. Above, we assumed that the resulting Krylov basis vectors are orthogonal. However, as the modified Gram-Schmidt orthogonalization is only applied approximately (truncations in line 8 and 10), this assumption is usually violated. As a result, the true residual norm might not be smaller than the prescribed tolerance ϵ even

though the approximate residuals converge. To compensate for the inaccurate orthogonalization, one can prescribe a more strict truncation tolerance as discussed in Section 6 of Simoncini and Szyld (2003). Another common approach consists in re-orthogonalization: We employ the following specialized variant of a modified Gram-Schmidt iteration.

In the TT format calculating a scalar product is much faster than a truncated scaled addition (axpby) as discussed in more detail in Section 5. So we can perform additional scalar products to reorder Gram-Schmidt iterations and perform selective re-orthogonalization as shown in Alg. 2 to increase the robustness. This omits subtracting directions that are already almost orthogonal in order to avoid growing the TT-ranks. See Leon et al. (2012) and the references therein for a detailed discussion on different Gram-Schmidt orthogonalization schemes. Here, we again need to use sufficiently small truncation tolerances to fulfill the requirements of the outer inexact GMRES method. The factor $2i$ is an estimate for the number of inner iterations (line 9) as usually orthogonalizing "twice is enough" (Giraud et al., 2005; Parlett, 1998). And we choose $k_{\max} = 4$ in all our experiments as this was sufficient for the cases we investigated.

Algorithm 2 Selective iterated modified Gram-Schmidt (SIMGS) in TT format

Input: Orthonormal previous directions $V_{\mathrm{TT},j}$, $j = 1, \ldots, i$, new direction W_{TT}, tolerance δ_i, max. re-orthogonlization iterations $k_{\max}$

Output: New normalized direction $V_{\mathrm{TT},i+1}$ with $W_{\mathrm{TT}} \approx \sum_{j=1}^{i+1} h_j V_{\mathrm{TT},j}$, and $|\langle V_{\mathrm{TT},i+1}, V_{\mathrm{TT},j}\rangle| \lesssim \delta_i$, $j = 1, \ldots, i$

1: $W_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\delta_i/(2i+1)}\,(W_{\mathrm{TT}})$
2: $h_j \leftarrow 0$, $j = 1, \ldots i$
3: **for** $k = 1, \ldots, k_{\max}$ **do**
4: Calculate $g_j \leftarrow \langle V_{\mathrm{TT},j}, W_{\mathrm{TT}}\rangle / \|W_{\mathrm{TT}}\|_F$, $j = 1, \ldots i$
5: **if** $\|g\|_\infty \leq \delta_i$ **break**
6: **for** $j = \arg\max_l |g_l|$ and $|g_j| > \delta_i$ **do**
7: $\beta \leftarrow \langle V_{\mathrm{TT},j}, W_{\mathrm{TT}}\rangle$
8: $W_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\delta_i/(2i+1)}(W_{\mathrm{TT}} - \beta V_{\mathrm{TT},j})$
9: $h_j \leftarrow h_j + \beta, \qquad g_j \leftarrow 0$
10: **end for**
11: **end for**
12: $W_{\mathrm{TT}} \leftarrow \mathrm{trunc}_{0.5\delta_i}\,(W_{\mathrm{TT}})$
13: $h_{i+1} \leftarrow \|W_{\mathrm{TT}}\|_F, \qquad V_{\mathrm{TT},i+1} \leftarrow W_{\mathrm{TT}}/h_{i+1}$

3.1.3. Tensor-train ranks for problems with a displacement structure. Even with truncations after each tensor-train addition, the tensor-train ranks can grow exponentially in the worst case:

$$\mathrm{rank}(V_{\mathrm{TT},i+1}) \leq \mathrm{rank}(\mathcal{A}_{\mathrm{TT}})\mathrm{rank}(V_{\mathrm{TT},i}) + \sum_{j=0}^{i} \mathrm{rank}(V_{\mathrm{TT},j}).$$

We observe only a much smaller growth for some special linear operators $\mathcal{A}_{\mathrm{TT}}$. In particular, we consider linear operators with a displacement/Laplace structure:

$$(\mathcal{A}X)_{\mathrm{unfold}_j} = A_{j,\mathrm{left}} X_{\mathrm{unfold}_j} + X_{\mathrm{unfold}_j} A_{j,\mathrm{right}}, \quad j = 1, \ldots, d-1.$$

For those operators, the rank of the solution is bounded if the right-hand side also has small rank as discussed for several tensor formats in Shi and Townsend (2021). We obtain the following expression for applying the operator k times:

$$(\mathcal{A}^k X)_{\mathrm{unfold}_j} = \sum_{i=0}^{k} \binom{k}{i} A_{j,\mathrm{left}}^{k-i} X_{\mathrm{unfold}_j} A_{j,\mathrm{right}}^{i}.$$

Similarly, for any matrix polynomial p_k of degree k, we get

$$(p_k(\mathcal{A})X)_{\mathrm{unfold}_j} = \sum_{i=0}^{k} \overline{p}_{k-i}(A_{j,\mathrm{left}}) X_{\mathrm{unfold}_j} \widehat{p}_i(A_{j,\mathrm{right}})$$

with appropriate sets of polynomials $\overline{p}_i$ and $\widehat{p}_i$. As the jth TT rank is just the rank of the jth unfolding, this results in at most a linear growth of ranks of Krylov subspace basis vectors:

$$\mathrm{rank}(V_{\mathrm{TT},i}) \leq (i+1)\mathrm{rank}(V_{\mathrm{TT},0}). \tag{15}$$

However, we will see in Section 4.1 that this only holds in exact arithmetic. If we do not calculate the Krylov basis accurately enough, the TT ranks might again grow exponentially.

3.2. Modified alternating least-squares (MALS)

Krylov methods like TT-GMRES consider the linear operator as a black box. However, we can also exploit the tensor-train structure of the problem and project it onto the subspace of one or several sub-tensors. This is the idea of the ALS and MALS methods discussed in Holtz et al.

(2012). In principle, MALS is identical to the famous DMRG method (Schollwöck, 2005; White, 1992) for eigenvalue problems from quantum physics. Here, we describe it from the point of view of numerical linear algebra. Using all but two sub-tensors of the current approximate solution, we define the operator:

$$\begin{aligned}\mathcal{V}_{j,2} &: \mathbf{R}^{r_{j-1}\times n_j\times n_{j+1}\times r_{j+1}} \to \mathbf{R}^{n_1\times\cdots\times n_d}, \quad \text{with}\\ \mathcal{V}_{j,2}Y &= X_1 \bowtie \cdots \bowtie X_{j-1} \bowtie Y \bowtie X_{j+2}\\ &\quad \bowtie \cdots \bowtie X_d.\end{aligned} \tag{16}$$

If sub-tensors (X_1, …X_{j-1}) are left-orthogonal and sub-tensors (X_{j+2}, …X_d) are right-orthogonal, the operator $\mathcal{V}_{j,2}$ has orthonormal columns: $\mathcal{V}_{j,2}^T\mathcal{V}_{j,2} = I$. Using $\mathcal{V}_{j,2}$, we can project the problem onto the subspace of the sub-tensors $\{X_k, k<j \bigvee k>j+1\}$. This results in a smaller problem of the form:

$$\mathcal{V}_{j,2}^T\mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2}Y = \mathcal{V}_{j,2}^T B_{\mathrm{TT}}. \tag{17}$$

For s.p.d. operators A_{TT}, this much smaller problem typically has a better condition number ($\kappa(\mathcal{V}_{j,2}^T\,\mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2}) \leq \kappa(\mathcal{A}_{\mathrm{TT}})$), and its solution minimizes the error in the induced operator norm:

$$\begin{aligned} Y &= \arg\min_Y \left\|\mathcal{V}_{j,2}Y - \mathcal{X}_{\mathrm{TT}}^*\right\|_{\mathcal{A}_{\mathrm{TT}}}^2\\ \text{for} \qquad Y &= \left(\mathcal{V}_{j,2}^T\mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2}\right)^{-1}\mathcal{V}_{j,2}^T B_{\mathrm{TT}}, \end{aligned}$$

where $\mathcal{X}_{\mathrm{TT}}^*$ denotes the true solution. For non-symmetric operators $\mathcal{A}_{\mathrm{TT}}$, this projection (Ritz-Galerkin) is often still successful (Dolgov and Savostyanov, 2014), but one might also consider a different projection for the left and the right-hand side of the operator (Petrov-Galerkin):

$$\overline{\mathcal{V}}_j^T\mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2}Y = \overline{\mathcal{V}}_j^T B_{\mathrm{TT}}.$$

Here, $\overline{\mathcal{V}}_j$ should have the same dimensions as $\mathcal{V}_{j,2}$ to ensure that projected problem is square (and thus usually easier to solve). A possible non-symmetric approach is

$$\overline{\mathcal{V}}_j Z \approx \mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2} \qquad \text{with} \qquad \overline{\mathcal{V}}_j^T\overline{\mathcal{V}}_j = I.$$

This is only possible approximately if $\overline{\mathcal{V}}_j$ should have low rank in the TT format again. The solution of the projected problem then approximately minimizes the residual in the Frobenius norm (similar to GMRES). In this paper, we will not further discuss this approach as we focus on the performance of the operations involved, but other variants of projections are possible.

Algorithm 3 TT-MALS

Input: Linear operator $\mathcal{A}_{\mathrm{TT}} : \mathbf{R}^{n_1\times\cdots\times n_d} \to \mathbf{R}^{n_1\times\cdots\times n_d}$ and RHS $B_{\mathrm{TT}} \in \mathbf{R}^{n_1\times\cdots\times n_d}$, initial guess $X_{\mathrm{TT}} \in \mathbf{R}^{n_1\times\cdots\times n_d}$, desired tolerance ϵ, max. number of sweeps m

Output: Approximate solution X_{TT} with $\|B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}}X_{\mathrm{TT}}\|_F \lesssim \epsilon\,\|B_{\mathrm{TT}}\|_F$

Right-orthogonalize $X_d \ldots X_3$
for $i_{\mathrm{Sweep}} = 1, \ldots, m$ **do**
  $j_{\mathrm{start}} \leftarrow 1$ **if** $i_{\mathrm{Sweep}} = 1$ **else** 2
  for $j = j_{\mathrm{start}}, \ldots, d-1$ **do**
    Left-orthogonalize X_{j-1} **if** $j > 1$
    Setup projection operator $\mathcal{V}_{j,2}$ using (16)
    Solve local problem $\mathcal{V}_{j,2}^T\mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2}Y = \mathcal{V}_{j,2}^T B_{\mathrm{TT}}$ with initial guess $X_j \bowtie X_{j+1}$
    Update $X_j \bowtie X_{j+1} \leftarrow Y$
  end for
  if $\|B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}}X_{\mathrm{TT}}\|_F \leq \epsilon\|B_{\mathrm{TT}}\|$ **then**
    return
  end if
  for $j = d-2, \ldots, 1$ **do**
    Right-orthogonalize X_{j+2}
    Setup projection operator $\mathcal{V}_{j,2}$ using Equation 16
    Solve local problem $\mathcal{V}_{j,2}^T\mathcal{A}_{\mathrm{TT}}\mathcal{V}_{j,2}Y = \mathcal{V}^T B_{\mathrm{TT}}$ with initial guess $X_j \bowtie X_{j+1}$
    Update $X_j \bowtie X_{j+1} \leftarrow Y$
  end for
  if $\|B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}}X_{\mathrm{TT}}\|_F \leq \epsilon\|B_{\mathrm{TT}}\|$ **then**
    return
  end if
end for
Abort: not converged in m sweeps!

To solve the global problem, the MALS algorithm sweeps from the first two to the last two dimensions and back, see Alg. 3. This can be interpreted as a moving subspace correction as discussed in Oseledets et al. (2018). In contrast, the ALS algorithm only projects the problem onto the subspace of all but one sub-tensor. This yields smaller local problems but provides no mechanism to increase the TT-rank of the approximate solution. So in the simplest form, it can only converge for special initial guesses that already have the same TT-rank as the desired approximate solution. In Section 3.3, we discuss a possible way to avoid this problem.

3.2.1. Inner solver: TT-GMRES. The projected problem (17) again has a structure similar to (13) but with just two dimensions. More specifically, after some contractions, the projected operator has the form

$$\mathcal{V}_{j,2}^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}_{j,2} = \overline{A}_{j-1,\mathrm{left}} \bowtie A_j \bowtie A_{j+1} \bowtie \overline{A}_{j+2,\mathrm{right}}$$

with $\overline{A}_{j-1,\mathrm{left}} \in \mathbf{R}^{r_{j-1} \times r_{j-1} \times r_{j-1}^A}$ and $\overline{A}_{j+2,\mathrm{right}} \in \mathbf{R}^{r_{j+1}^A \times r_{j+1} \times r_{j+1}}$. And the initial guess as well as the required solution is in the form:

$$Y = X_j \bowtie X_{j+1} \in \mathbf{R}^{r_{j-1} \times n_j \times n_{j+1} \times r_{j+1}}.$$

Using tensor contractions, we can also express the projected right-hand side as:

$$\mathcal{V}_{j,2}^T B_{\mathrm{TT}} = \overline{B}_{j,\mathrm{left}} \bowtie \overline{B}_{j+1,\mathrm{right}}.$$

As long as the rank of Y during the iteration is much smaller than $r_{j-1} n_j$, respectively $n_{j+1} r_{j+1}$, it is usually beneficial to use the factorized form for Y. This results in an inner-outer scheme with an outer MALS and an inner TT-GMRES algorithm. From our observation, this also yields slightly smaller ranks in the outer MALS than using standard GMRES with the dense form of Y and a subsequent factorization.

Remark 1. *In the first MALS sweeps, it is not necessary to solve the inner problem very accurately. So one can use a larger relative tolerance for the inner iteration than for the outer iteration. The same yields in the last MALS sweeps (close to the solution). Combining both aspects, we employ a relative tolerance of:*

$$\overline{\epsilon}_{inner} = \max\left(\epsilon_{inner}, \epsilon \frac{\|B_{\mathrm{TT}}\|}{\|\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}\|}\right),$$
$$\text{with} \quad \epsilon_{inner} := \sqrt{\epsilon}.$$

Remark 2. *One might wonder if an inner-outer iteration scheme with outer (flexible) TT-GMRES and inner MALS (as preconditioner) might also work. In our experiments, this results in super-linear (up to exponential) growth of the TT-ranks in the Arnoldi iteration. This can be explained by the fact that the displacement structure of the linear operator is not retained through this form of a varying preconditioner.*

For symmetric problems, our implementation switches to TT-MINRES and for positive definite problems, one can also employ a tensor-train variant of the CG algorithm.

3.3. AMEn method

For the MALS method above, always two sub-tensors are considered at once in the inner problem. One can also consider only one sub-tensor (ALS) at a time using the projection operator

$$\begin{aligned} &\mathcal{V}_{j,1} : \mathbf{R}^{r_{j-1} \times n_j \times r_j} \to \mathbf{R}^{n_1 \times \cdots \times n_d}, \quad \text{with} \\ &\mathcal{V}_{j,1} Y = X_1 \bowtie \cdots \bowtie X_{j-1} \bowtie Y \bowtie X_{j+1} \\ &\qquad \bowtie \cdots \bowtie X_d. \end{aligned} \tag{18}$$

This results in a smaller local operator. By contracting sub-tensors of $\mathcal{V}_{j,1}$ and $\mathcal{A}_{\mathrm{TT}}$, one obtains:

$$\overline{\mathcal{A}}_{\mathrm{TT}} := \mathcal{V}_{j,1}^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}_{j,1} = \overline{A}_{j-1,\mathrm{left}} \bowtie A_j \bowtie \overline{A}_{j+1,\mathrm{right}}. \tag{19}$$

However, one needs a way to adapt the rank of the approximate solution and to ensure convergence. An early approach from physics was just to increase the rank through adding random directions. A more sophisticated method is introduced in Dolgov and Savostyanov (2014) named AMEn (alternating minimal energy). The main idea is to enrich the subspace after each inner iteration with directions obtained from the current residual tensor. For this, the current residual tensor is projected onto the subspace of the left sub-tensors of the current approximation when sweeping left-to-right, respectively the right sub-tensors when sweeping right-to-left. The corresponding projection operators are given by:

$$\begin{aligned} &\mathcal{V}_{j,d-j} : \mathbf{R}^{r_{j-1} \times n_j \times \cdots \times n_d} \to \mathbf{R}^{n_1 \times \cdots \times n_d}, \quad \text{with} \\ &\quad \mathcal{V}_{j,d-j} Y = X_1 \bowtie \cdots \bowtie X_{j-1} \bowtie Y, \\ &\mathcal{V}_{1,j} : \mathbf{R}^{n_1 \times \cdots \times n_j \times r_{j+1}} \to \mathbf{R}^{n_1 \times \cdots \times n_d}, \quad \text{with} \\ &\quad \mathcal{V}_{1,j} Y = Y \bowtie X_{j+1} \bowtie \cdots \bowtie X_d. \end{aligned} \tag{20}$$

For s.p.d. operators, this results in a steepest descent step for which global convergence is shown in Dolgov and Savostyanov (2014). In practice, it is often sufficient to approximate a few directions of the residual tensor and add them to the current subspace to obtain fast convergence. The standard form of the AMEn algorithm is depicted in Alg. 4 (based on the SVD variant in Dolgov and Savostyanov, 2014). The step to update the residual $R_{\mathrm{TT}} = \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - X_{\mathrm{TT}}$ (line 9) requires saving all intermediate matrices from left- respectively right-orthogonalization of the sub-tensors R_1, …, R_d of R_{TT}. To calculate k suitable directions to enrich the subspace in the left-to-right sweep (line 15), we left-orthogonalize Z_j w.r.t. X_j, such that:

Algorithm 4 TT-AMEn

Input: Linear operator $\mathcal{A}_{\mathrm{TT}}$, RHS B_{TT} and initial guess X_{TT}, desired tolerance ϵ, max. number of sweeps m, number of enrichment directions k

Output: Approximate solution X_{TT} with $\|B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}}\|_F \lesssim \epsilon \|B_{\mathrm{TT}}\|_F$

1: Right-orthogonalize $X_d \dots X_2$
2: Calculate residual $R_{\mathrm{TT}} \leftarrow \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}$
3: Right-orthogonalize $R_d \dots R_2$ and let $Z_{\mathrm{TT}} \leftarrow R_{\mathrm{TT}}$
4: **for** $i_{\mathrm{Sweep}} = 1, \dots, m$ **do**
5: **for** $j = 1, \dots, d-1$ **do**
6: Setup projection operators $\mathcal{V}_{j,1}, \mathcal{V}_{j,d-j}$ using (18) and (20)
7: Solve local problem $\mathcal{V}_{j,1}^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}_{j,1} Y = \mathcal{V}_{j,1}^T B_{\mathrm{TT}}$ with initial guess X_j
8: Update $X_j \leftarrow Y$
9: Update R_j s.t. $R_{\mathrm{TT}} = \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}$
10: **if** $\|R_{\mathrm{TT}}\|_F \le \epsilon \|B_{\mathrm{TT}}\|$ **then**
11: **return**
12: **end if**
13: Update Z_j, s.t. $Z_j \bowtie \dots \bowtie Z_d = \mathcal{V}_{j,d-j}^T R_{\mathrm{TT}}$
14: Left-orthogonalize X_j (updating X_{j+1} s.t. $X_j \bowtie X_{j+1}$ remains the same)
15: Left-orthogonalize Z_j w.r.t. X_j
16: Append $Z_{:,:,1:k} \bowtie 0$ to $X_j \bowtie X_{j+1}$
17: **end for**
18: Similarly sweep from d to 2.
19: **end for**
20: Abort: not converged in m sweeps!

$$USV^T = (I - (X_j)_{\mathrm{unfold}_l} (X_j)_{\mathrm{unfold}_l}^T)(R_j)_{\mathrm{unfold}_l}, \quad (Z_j)_{\mathrm{unfold}_l} = U.$$

This results in the method AMEn + SVD from Dolgov and Savostyanov (2014). To append the directions (line 16), we concatenate the tensors, such that $\overline{r}_j = r_j + k$ and $(\overline{X}_j)_{:,:,r_j+1:r_j+k} = Z_{:,:,1:k}$ and $(\overline{X}_{j+1})_{r_j+1:r_j+k,:,:} = 0$. The same is done in the right-to-left sweep with mirrored dimensions.

As can be seen from Alg. 4, calculating the required directions from the residual mostly involves updating the subtensors of the residual in every step of the sweep. This is not significantly more work than calculating the residual in the first place. Still, the complete algorithm is so cheap that the residual calculation accounts for a significant part of the overall runtime. That is why Dolgov and Savostyanov (2014) discusses several more heuristic ways to determine suitable enrichment directions.

3.3.1. TT-AMEn + ALS. The most promising variant from Dolgov and Savostyanov (2014) is based on an ALS-like approximation of the residual. As the resulting complete algorithm is not explicitly shown there, we illustrate the required steps Alg. 5 and discuss important implementation details.

First, to really decrease the work, one needs a way to check the error without using the global residual norm. In Dolgov and Savostyanov (2014), this is not further discussed but in the code used for the numerical experiments of Dolgov and Savostyanov (2014), the global residual error is estimated using the projected residuals (before solving the local problem), assuming that

$$\frac{1}{2\sqrt{d-1}} \max_j \left(\left\| \mathcal{V}_{j,1}^T (\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}) \right\|_F \right) \lesssim \|\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}\|.$$

The factor 1/2 is just a heuristic way to ensure that the global residual norm is smaller than the tolerance.

Second, one needs a cheap way to enrich the subspace. For this, a rough approximation of the residual is usually sufficient which can be obtained by a fixed-rank ALS iteration (*ALS (l)*). This results in the following update of the *j*th subtensor of the current approximation of the residual $\widetilde{R}_{\mathrm{TT}}$:

$$\begin{aligned} \widetilde{R}_j' &:= \mathcal{W}_{j,1}^T (\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}) \\ &= \mathcal{W}_{j,1}^T \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - \mathcal{W}_{j,1}^T B_{\mathrm{TT}}, \quad \text{where} \end{aligned} \tag{21}$$

$$\begin{aligned} &\mathcal{W}_{j,1} : \mathbf{R}^{l \times n_j \times l} \to \mathbf{R}^{n_1 \times \dots \times n_d}, \quad \mathcal{W}_{j,1} \\ &Y = \widetilde{R}_1 \bowtie \dots \widetilde{R}_{j-1} \bowtie Y \bowtie \widetilde{R}_{j+1} \bowtie \dots \bowtie \widetilde{R}_d. \end{aligned} \tag{22}$$

Here, $\widetilde{R}_1, \dots, \widetilde{R}_{j-1}$ must be left-orthogonal and $\widetilde{R}_{j+1}, \dots, \widetilde{R}_d$ right-orthogonal. We remark that the approximation $\widetilde{R}_{\mathrm{TT}}$ cannot be used to check convergence as $\|\widetilde{R}_{\mathrm{TT}}\|_F \le \|R_{\mathrm{TT}}\|_F$.

Algorithm 5 TT-AMEn+ALS

Input: Linear operator $\mathcal{A}_{\mathrm{TT}}$, RHS B_{TT}, initial guess X_{TT}, desired tolerance ϵ, max. number of sweeps m, enrichment rank k, inner tolerance ϵ_{inner}, approx. residual rank l

Output: Approximate solution X_{TT} with $\|B_{\mathrm{TT}} - \mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}}\|_F \lesssim \epsilon \|B_{\mathrm{TT}}\|_F$

Right-orthogonalize $X_d \dots X_2$
Initialize approx. residual $\tilde{R}_1, \dots, \tilde{R}_d$ with rank l
Right-orthogonalize $\tilde{R}_d \dots \tilde{R}_2$ and let $\tilde{Z}_{\mathrm{TT}} \leftarrow \tilde{R}_{\mathrm{TT}}$
for $i_{\text{Sweep}} = 1, \dots, m$ **do**
  for $j = 1, \dots, d$ **do**
    Setup projection operators $\mathcal{V}_{j,1}, \mathcal{V}_{j,d-j}$ using (18) and (20)
    if $i_{\text{Sweep}} = 1 \vee j \neq 1$ **then**
      Calculate local residual norm: $\delta_j \leftarrow \|\mathcal{V}_{j,1}^T (\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}})\|_F$
      Adapt tolerance: $\bar{\epsilon}_{\text{inner,abs}} \leftarrow \max(\delta_j \, \epsilon_{\text{inner}}, \ \|\mathcal{V}_{j,1}^T B_{\mathrm{TT}}\|_F \, \epsilon / (2\sqrt{d-1}))$
      Approximately solve local problem $\mathcal{V}_{j,1}^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}_{j,1} Y = \mathcal{V}_{j,1}^T B_{\mathrm{TT}}$ with initial guess X_j up to abs. tolerance $\bar{\epsilon}_{\text{inner,abs}}$
      Update $X_j \leftarrow Y$
      Setup residual projection operator $\mathcal{W}_{j,1}$ using (22)
      Update $\tilde{R}_j \leftarrow \mathcal{W}_{j,1}^T (\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}})$
    end if
    if $j < d$ **then**
      Update $\tilde{Z}_j$, s.t. $\tilde{Z}_j \bowtie \dots \bowtie \tilde{Z}_d = \mathcal{V}_{j,d-j}^T \tilde{R}_{\mathrm{TT}}$
      Left-orthogonalize $\tilde{R}_j$ (updating $\tilde{R}_{j+1}$ s.t. $\tilde{R}_j \bowtie \tilde{R}_{j+1}$ remains the same)
      Left-orthogonalize X_j (updating X_{j+1} s.t. $X_j \bowtie X_{j+1}$ remains the same)
      Left-orthogonalize $\tilde{Z}_j$ w.r.t. X_j
      Append $\tilde{Z}_{:,:,1:k} \bowtie 0$ to $X_j \bowtie X_{j+1}$
    end if
  end for
  if $\max_j(\delta_j) \leq \frac{\epsilon}{2\sqrt{d-1}}$ **then**
    return
  end if
  Similarly sweep from d to 1 and check convergence.
end for
Abort: not converged in m sweeps!

As for the projected local problem, the required additional terms can be updated successively in each step of the sweep. In our implementation, we use B_{TT} as initial guess for the approximate residual $\tilde{R}_{\mathrm{TT}}$ and truncate it, respectively extend it by random directions to obtain the desired rank l. In Section 5.5, we show numerical experiments with both the full AMEn and the AMEn + ALS algorithm.

3.4. Preconditioning

For iterative solvers of linear systems, it is a common approach to employ a preconditioner to obtain much faster convergence. Of course, we can precondition all previously discussed algorithms. However, some additional aspects should be considered when preconditioning linear solvers in the TT format. On the one hand, for MALS and AMEn, we can employ *local preconditioners* for the projected operators $\mathcal{V}_j^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}_j$ with $\mathcal{V}_j = \mathcal{V}_{j,2}$ or $\mathcal{V}_j = \mathcal{V}_{j,1}$. In this case,

- one needs to calculate a different preconditioner in every step of the sweep,
- often only a few local iterations are performed,
- one may need to contract, for example, $\mathcal{V}_{j,1}^T \mathcal{A}_{\mathrm{TT}} \mathcal{V}_{j,1} = \overline{A}_{j-1,\text{left}} \bowtie A_j \bowtie \overline{A}_{j+1,\text{right}}$ (costly).

On the other hand, we could employ a global preconditioner which is

- either applied to tensor-train "vectors" (TT-GMRES),
- or directly to the tensor-train operator $\mathcal{A}_{\mathrm{TT}}$ (TT-GMRES, MALS, AMEn),

- but is not tailored to the local problems for MALS/AMEn.

Furthermore, a *global preconditioner* (and also *local preconditioners* for MALS) should retain the low-rank of the solution and right-hand side. To emphasize this: just the fact that $\kappa(\mathcal{P}\mathcal{A}_{\mathrm{TT}}) \ll \kappa(\mathcal{A}_{\mathrm{TT}})$ does not imply that $\mathrm{rank}_{\mathrm{TT}}(\mathcal{P}\mathcal{B}_{\mathrm{TT}}) \leq \mathrm{rank}_{\mathrm{TT}}(B_{\mathrm{TT}})$ for left-preconditioning with $\mathcal{P}$. And similarly the same problem occurs for the approximate solution X_{TT} for right-preconditioning.

3.4.1. TT-rank-1 preconditioner. Considering the desired properties, we suggest a simple, *global* rank-1 preconditioner that usually is cheap to calculate (for operators of sufficiently low rank). The two-sided variant can be constructed as follows: First, approximate the operator with a tensor-train of rank 1 using TT-truncation: $\widetilde{\mathcal{A}}_{\mathrm{TT}} = \widetilde{A}_1 \otimes \widetilde{A}_2 \otimes \cdots \otimes \widetilde{A}_d = \mathrm{trunc}_{r=1}(\mathcal{A}_{\mathrm{TT}})$. Then, perform an SVD for each sub-matrix ($U_j S_j V_j^T = \widetilde{A}_j, j = 1, \ldots, d$) to construct the TT operators for the left and the right:

$$\mathcal{P}_{\mathrm{TT,left}} = (S_1^{-1/2} U_1^T) \otimes \cdots \otimes (S_d^{-1/2} U_d^T),$$
$$\mathcal{P}_{\mathrm{TT,right}} = (V_1 S_1^{-1/2}) \otimes \cdots \otimes (V_d S_d^{-1/2}).$$

This yields the preconditioned system:

$$\left(\mathcal{P}_{\mathrm{TT,left}} \mathcal{A}_{\mathrm{TT}} \mathcal{P}_{\mathrm{TT,right}}\right) Y_{\mathrm{TT}} = \mathcal{P}_{\mathrm{TT,left}} B_{\mathrm{TT}}$$
$$\text{with} \qquad X_{\mathrm{TT}} = \mathcal{P}_{\mathrm{TT,right}} Y_{\mathrm{TT}}.$$

For a symmetric operator $\mathcal{A}_{\mathrm{TT}}$ where for each sub-tensor $(A_j)_{:,k,l,:} = (A_j)_{:,l,k,:}$, the preconditioned operator is still symmetric. And as the preconditioner has rank one, it does not lead to higher ranks for the right-hand side or the exact solution X_{TT}. However, if the operator has a displacement structure, this is not preserved for the preconditioned operator.

4. Comparison of algorithms

In the following we present numerical experiments for linear systems in TT format. We use the pitts library (Röhrig-Zöllner and Becklas, 2024) which also contains the setup and output for all results shown in this paper. For illustrating the numerical behavior, we consider a simple multidimensional convection-diffusion equation. It is discretized using a finite difference stencil, for example, in 1d the operator is

$$\frac{\mathrm{tridiag}(-1, 2, -1)}{h_j^2} + \frac{c}{\sqrt{d}} \frac{\mathrm{tridiag}(0, 1, -1)}{h_j},$$
$$\text{for } h_j = \frac{1}{n_j + 1}.$$

Here, c denotes a convection constant and the convection direction is diagonal through all dimensions. As right-hand side, we use a vector of all ones (rank 1) or just a tensor-train with chosen rank and random sub-tensors. All cases use double-precision calculations and a desired residual tolerance of $\epsilon = 10^{-8}$.

4.1. Behavior of tensor-train ranks in the calculation

First, to understand the computational complexity of the different methods, we show the behavior of the TT ranks during the calculation. As shown in Figure 1(a), the improved orthogonalization in the TT-GMRES algorithms (SIMGS, Alg. 2) results in slightly slower rank growth in the Krylov basis than standard MGS; Both show the expected linear growth (for operators with a displacement structure). A naive MGS implementation (all truncations performed with tolerance δ_i in Alg. 1) results in exploding ranks after a few iterations. Furthermore, the calculated solutions of the MGS variants are not within the desired residual tolerance (with $\|\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}\|_F \approx 10\epsilon$). With the TT-rank-1 preconditioner, the algorithm converges after less than half the number of iterations, but higher ranks occur (as the preconditioned operator does not have a displacement structure). Similarly, a GMRES restart also results in higher TT ranks. Overall, the TT-GMRES has a computational complexity that is cubic in the maximal TT rank ($r_{\max}$) of the Krylov basis: $O(dnr_{\max}^3 + dn^2 r_{\max}^2 (r^A)^2)$. For the MALS method, we observe a (linearly) growing rank of the approximation solution in the iteration up to the rank (r) of the resulting approximate solution, see Figure 1(b). However, in the inner TT-GMRES iterations, the Krylov basis ranks behave like after a GMRES restart after the first sweep. So significantly higher ranks occur in the inner iteration. The effect on the computational complexity is only quadratic for MALS (only two sub-tensors in the inner problem): $O(dnrr_{\max}^2 + dn^2 r r_{\max} (r^A)^2)$. The behavior of the rank of the approximate solution for AMEn is similar to MALS. But as the local problem consists only of one sub-tensor, no higher intermediate ranks can occur in the inner iteration. So only $O(dnr^3 + dn^2 r^2 (r^A)^2)$ operations are needed assuming that the enrichment rank k is chosen appropriately.

4.2. Computational complexity of the different methods

The TT ranks explain the results in Figure 2: TT-GMRES needs 10–100× more operations than MALS. MALS needs 10–100× more operations than AMEn. For larger individual dimensions (n_j), MALS with inner TT-GMRES needs fewer operations than with inner MALS. In addition, applying the operator becomes more costly, and it is beneficial to use a sparse matrix format for the sub-tensors of the operator (sparse TT-Op variant in Figure 2(a)). We also measured the behavior using the quantics TT format (QTT, Khoromskij, 2011) where we just convert the operator to a 2^d tensor. For our test case

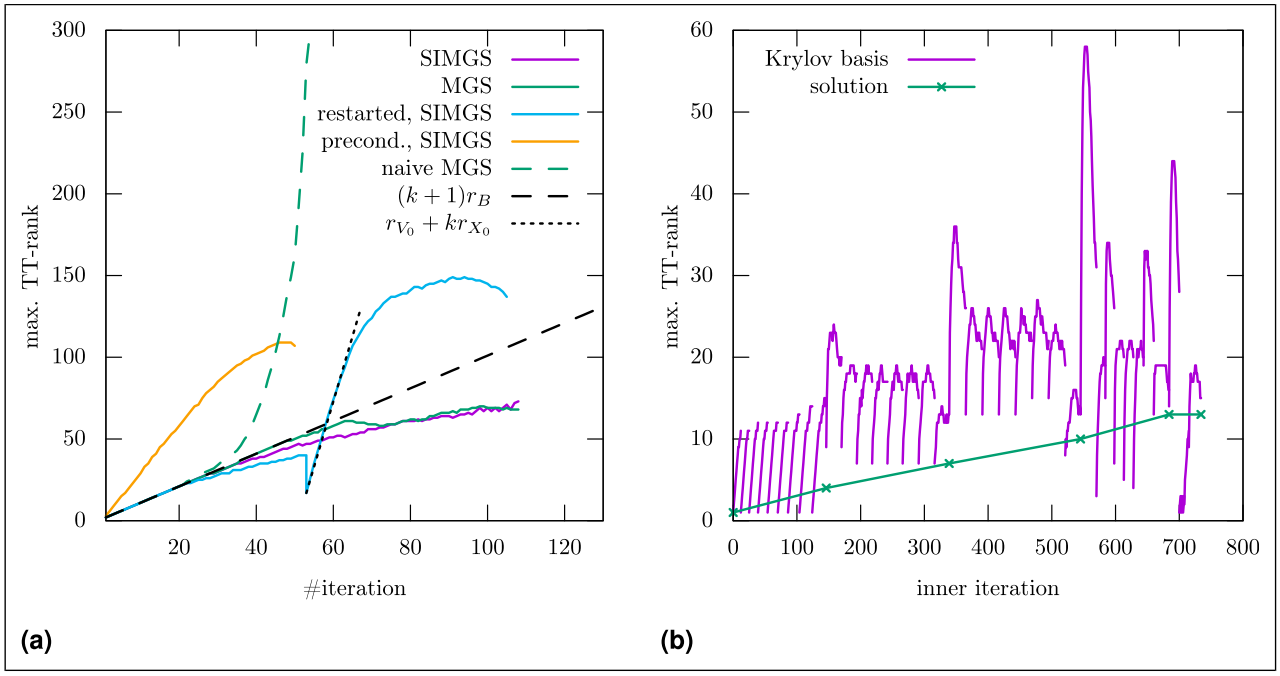

Figure 1. Tensor-train ranks for the Krylov basis, respectively the approximate solution for a 20^{10} convection-diffusion problem (c = 10) and RHS B_{TT} of ones. For TT-GMRES (left), both MGS variants lead to inaccurate solutions that are not within the desired residual tolerance in contrast to all cases with SIMGS. Overall, more accurate orthogonalization (SIMGS) without restart and preconditioning features the lowest maximal ranks during the calculation. For MALS (right), the solution ranks only increase slowly with each sweep (as intended), but the Krylov basis vectors of the inner iteration again yield higher ranks. (a) Krylov basis ranks for TT-GMRES. (b) Ranks for precond. MALS with inner TT-GMRES.

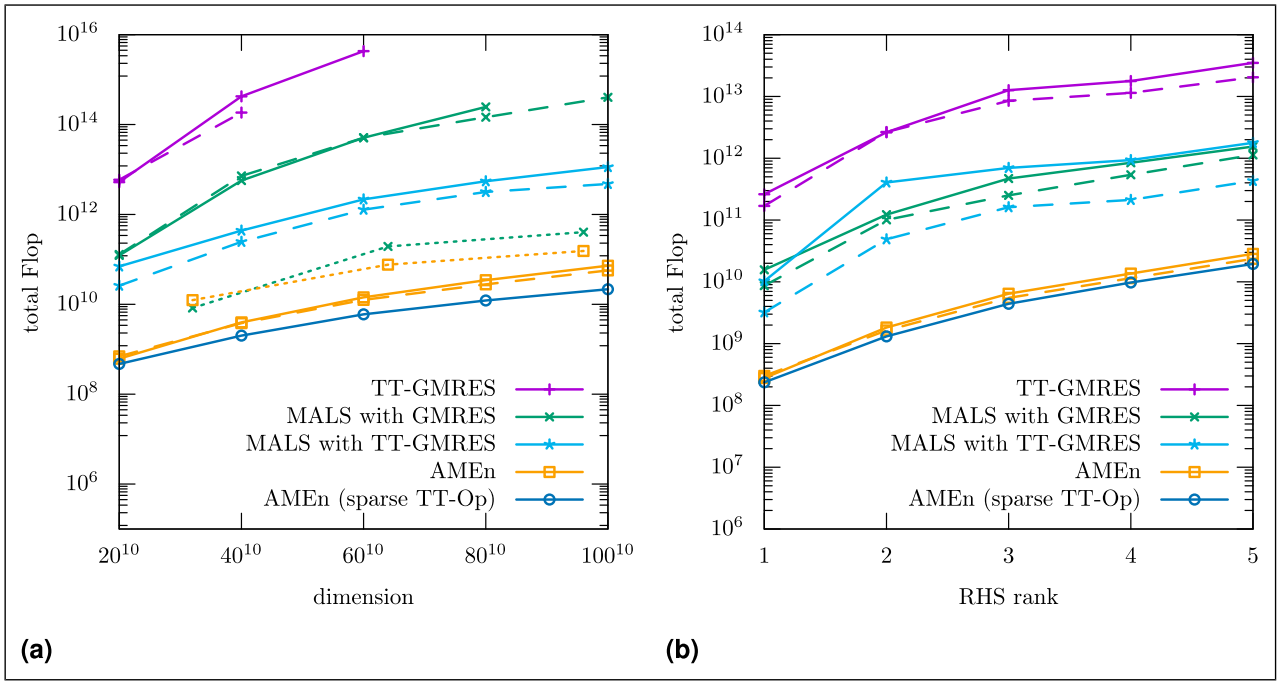

Figure 2. Number of floating-point operations measured using likwid (Treibig et al., 2010) for a convection-diffusion problem (c = 10). Dashed lines use the TT-rank-1 preconditioner. Dotted lines first transform the problem to the QTT format (Khoromskij, 2011). In all cases, AMEn requires orders of magnitude fewer operations than MALS and TT-GMRES. (a) Varying dimensions (RHS B_{TT} of ones) (b) Random RHS B_{TT} with varying ranks (dimension 20^{10}).

here, QTT-MALS needs fewer operations than TT-MALS (ranks in inner iteration can grow at most by a factor of 2). However, QTT-AMEn needs more operations than TT-AMEn here. Looking at the approximate solution, it has TT ranks (…, r, $2r$, $3r$, $4r$, …, $4r$, $3r$, $2r$, r, …) where r is the rank of the approximate solution without transformation to QTT format. So the QTT format is not beneficial here as the solution is not well approximable with small QTT ranks.

5. Performance of algorithmic building blocks

In the following, we discuss the required basic tensor-train operations ("building blocks") for the algorithms from Section 3. We focus on the node-level performance on a single multicore CPU with some remarks for a distributed parallel implementation. We only consider those operations where we see a significant improvement over the standard implementation as introduced in Oseledets (2011). These are in particular left-/right-orthogonalization with/without truncation, TT addition with subsequent truncation (TT-axpby + trunc), and faster contractions.

5.1. Replacing costly SVDs and pivoted QR decompositions with faster but less accurate alternatives

In Röhrig-Zöllner et al. (2022), the authors present a significantly faster implementation for decomposing a large dense tensor in the TT format (TT-SVD) using a Q-less tall-skinny QR (TSQR) algorithm. For the TT-SVD, one starts with a decomposition of the form:

$$\min_{B,\,Q^T} \left\| (X)_{\text{unfold}_{d-1}} - BQ^T \right\|_F \qquad \text{with} \qquad Q^TQ = I.$$

$(X)_{\text{unfold}_{d-1}}$ is a tall-skinny matrix, and the standard algorithm uses a truncated SVD to build B and Q:

$$(X)_{\text{unfold}_{d-1}} = USV^T \qquad \text{and} \qquad B = U_{:,1:r_1}S_{1:r_1,1:r_1}, \quad Q = V_{:,1:r_1}.$$

The optimized algorithm uses the following steps instead (grayed-out matrices are not calculated):

$$(X)_{\text{unfold}_{d-1}} = \overline{Q}R, \quad R \approx \overline{U}SV^T, \qquad \text{and} \qquad B = (X)_{\text{unfold}_{d-1}}V_{:,1:r_1}, \quad Q = V_{:,1:r_1}.$$

As V has orthonormal columns, the matrix B can be calculated accurately this way.

For the left-/right-orthogonalization (see first loop in TT-rounding in Oseledets, 2011), we unfortunately need slightly different operations, for example, in the left-to-right QR sweep:

$$(X_j)_{\text{left}} = QB, \qquad \text{and} \qquad (X'_j)_{\text{left}} = Q, \qquad X'_{j+1} = B \bowtie X_{j+1}.$$

And very similarly for a left-to-right SVD sweep:

$$\min_{Q,B} \|(X_j)_{\text{left}} - QB\|_F \qquad \text{and} \qquad (X'_j)_{\text{left}} = Q, \qquad X'_{j+1} = B \bowtie X_{j+1}.$$

The only difference is that the pure orthogonalization is exact up to the numerical rank whereas the truncation intentionally cuts off with a given tolerance. We can use the same trick as for the TT-SVD in a slightly different way here. For the orthogonalization, one obtains with pivoting matrix P:

$$(X_j)_{\text{left}}P = QR \qquad \text{and} \qquad \widetilde{X}'_j = X_j \bowtie (PR^{-1}), \qquad X'_{j+1} = (RP^T) \bowtie X_{j+1}. \tag{23}$$

And again similarly for the truncation:

$$(X_j)_{\text{left}} = QR, \quad R = USV^T \qquad \text{and} \qquad \widetilde{X}'_j = X_j \bowtie (VS^{-1}), \quad X'_{j+1} = (SV^T) \bowtie X_{j+1}. \tag{24}$$

There is a difference to the TT-SVD here: in both cases, the faster formulation might introduce a significant numerical error ($\|\widetilde{X}'_j - X'_j\|_F$). For $\kappa(R) \gg 1$, respectively $\kappa(S) \gg 1$, applying R^{-1}, respectively VS^{-1}, can be numerically unstable. For the SVD-sweep, one can see that the error is only large in "unimportant" directions and should not affect the accuracy of the complete tensor-train: Assuming X_1, …, X_{j-1} are left-orthogonal and X_{j+1}, …, X_d right-orthogonal, then:

$$\begin{aligned} \|\widetilde{X}_{\text{TT}} - X_{\text{TT}}\|_F &= \|(\widetilde{X}'_j - X'_j) \bowtie X'_{j+1}\|_F \\ &= \|(\widetilde{X}'_j - X'_j) \bowtie S\|_F. \end{aligned}$$

This is not ensured for the orthogonalization step. However, this shows that the error should be weighted by the singular values of the corresponding unfolding of the approximated tensor. So for the truncation, the left-to-right QR sweep is followed by a right-to-left SVD sweep with:

$$\widetilde{X}''_j = \widetilde{X}'_j \bowtie (\overline{U}\,\overline{S}), \qquad \text{for} \qquad \overline{U}\,\overline{S}\,\overline{V}^T = (\overline{X}_{j+1})_{\text{right}}.$$

We suggest checking the error a posteriori using the orthogonalization error weighted by the singular values of the jth unfolding:

$$\left\| \overline{S}^2 - (\widetilde{X}''_j)^T_{\text{left}} (\widetilde{X}''_j)_{\text{left}} \right\|_\infty \overset{?}{\approx} \epsilon.$$

In our numerical tests this was always the case even if $\|I - (\widetilde{X}'_j)^T_{\text{left}} (\widetilde{X}'_j)_{\text{left}}\|_\infty \gg \epsilon$. If this a posteriori check fails, one can still recalculate the orthogonalization with the standard algorithm.

5.2. Exploiting orthogonalities in TT-axpby + trunc

To add tensors in TT format ($Z_{\mathrm{TT}} = \alpha X_{\mathrm{TT}} + \beta Y_{\mathrm{TT}} = Z_1 \bowtie \cdots \bowtie Z_d$), one combines their sub-tensors:

$$
\begin{aligned}
(Z_1)_{1,:,:} &= \begin{pmatrix} (X_1)_{1,:,:} & (Y_1)_{1,:,:} \end{pmatrix}, \\
(Z_j)_{:,i,:} &= \begin{pmatrix} (X_j)_{:,i,:} & 0 \\ 0 & (Y_j)_{:,i,:} \end{pmatrix}, \text{ for } i=1,\ldots,n_j, j=2,\ldots,d-1 \\
(Z_d)_{:,:,1} &= \begin{pmatrix} \alpha(X_d)_{:,:,1} \\ \beta(Y_d)_{:,:,1} \end{pmatrix}.
\end{aligned}
$$

This operation is needed in the TT-GMRES algorithm as well as in MALS and the standard AMEn. For AMEn, the operation is performed step-by-step in each step of a sweep. In all cases, the sub-tensors of X_{TT} and Y_{TT} are already left- or right-orthogonal. And after the addition, one performs a left- or right-orthogonalization of the Z_{TT} for a subsequent truncation step. We can exploit the block non-zero structure and pre-existing orthogonalities. In the following, we assume $\mathrm{rank}_{\mathrm{TT}}(X_{\mathrm{TT}}) \geq \mathrm{rank}_{\mathrm{TT}}(Y_{\mathrm{TT}})$ and that X_1, …, X_{d-1} are left-orthogonal. Then, we can compute left-orthogonal sub-tensors for $\overline{Z}_1, \ldots, \overline{Z}_{d-1}$ with smaller QR-decompositions $Q_1 R_1$, …, $Q_{d-1} R_{d-1}$ than for the standard algorithm. For the first sub-tensor, we obtain:

$$
\begin{aligned}
(Z_1)_{1,:,:} &= \underbrace{\begin{pmatrix} (X_1)_{1,:,:} & Q_1 \end{pmatrix}}_{(\overline{Z}_1)_{1,:,:}} \begin{pmatrix} I & M_1 \\ 0 & R_1 \end{pmatrix} \text{with} \\
M_1 &= (X_1)_{\mathrm{left}}^T (Y_1)_{\mathrm{left}}, \\
Q_1 R_1 &= (I - (X_1)_{\mathrm{left}} (X_1)_{\mathrm{left}}^T)(Y_1)_{\mathrm{left}}.
\end{aligned}
\tag{25}
$$

For the next sub-tensors $j = 2, \ldots, d-1$, we obtain:

$$
\begin{aligned}
(Z'_j)_{:,i,:} &= \begin{pmatrix} I & M_{j-1} \\ 0 & R_{j-1} \end{pmatrix} \begin{pmatrix} (X_j)_{:,i,:} & 0 \\ 0 & (Y_j)_{:,i,:} \end{pmatrix} && \text{for} \quad i = 1, \ldots, n_j \\
&= \underbrace{\begin{pmatrix} (X_j)_{:,i,:} & (Q_j)_{:,i,:} \end{pmatrix}}_{(\overline{Z}_j)_{:,i,:}} \begin{pmatrix} I & M_j \\ 0 & R_j \end{pmatrix} && \text{with} \quad M_j = \overline{X}_j^T \overline{Y}_j, \\
& && \quad\; Q_j R_j = (I - \overline{X}_j \overline{X}_j^T) \overline{Y}_j,
\end{aligned}
\tag{26}
$$

where we simplified the notation by introducing:

$$
\overline{X}_j := \left(\begin{pmatrix} I \\ 0 \end{pmatrix} \bowtie X_j \right)_{\mathrm{left}}, \qquad \overline{Y}_j = \left(\begin{pmatrix} M_{j-1} \\ R_{j-1} \end{pmatrix} \bowtie Y_j \right)_{\mathrm{left}}.
$$

The last sub-tensor simply results in:

$$
(\overline{Z}_d)_{:,:,1} = \begin{pmatrix} I & M_{d-1} \\ 0 & R_{d-1} \end{pmatrix} \begin{pmatrix} \alpha(X_d)_{:,:,1} \\ \beta(Y_d)_{:,:,1} \end{pmatrix}. \tag{27}
$$

5.2.1. Stable residual calculation with inaccurate orthogonalization. For (standard) AMEn and MALS, we update a left- respectively right-orthogonal representation of $\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} =: Y_{\mathrm{TT}}$ in each step and calculate the residual $B_{\mathrm{TT}} - Y_{\mathrm{TT}}$ from it reusing the orthogonality. This is susceptible to numerical errors as for the solution all directions should cancel out. More specifically, we observed relative errors of up to $\epsilon = \sim 10^{-3}$ when the residual norm was close to the machine precision.

In general, when combining inaccurate orthogonalization with the optimized TT-axpby + trunc algorithm, we need to adjust the formulas in (25) and (26) to compensate for the loss of orthogonality. Assuming $\overline{X}_j^T \overline{X}_j \approx I$, we suggest to employ a Cholesky decomposition to correct the projection in (26) (and similarly in (25)) by using:

$$
Q'_j R'_j = \left(I - \overline{X}_j (L_j L_j^T)^{-1} \overline{X}_j^T \right) \overline{Y}_j, \quad \text{with} \quad L_j L_j^T = \overline{X}_j^T \overline{X}_j. \tag{28}
$$

This emulates re-orthogonalization but is faster if $\overline{Y}_j$ has fewer columns than $\overline{X}_j$ as re-orthogonalization would require to update the matrix $\overline{X}'_j = \overline{X}_j (L_j)^{-T}$ instead.

5.3. Faster contractions: inner iteration of AMEn

In AMEn, one needs to apply the projected TT operator from (19) to a dense tensor:

$$
\begin{aligned}
Z &= \overline{\mathcal{A}}_{\mathrm{TT},j} Y \quad \text{with} \\
\overline{\mathcal{A}}_{\mathrm{TT},j} &= \overline{A}_{j-1,\mathrm{left}} \bowtie A_j \bowtie \overline{A}_{j+1,\mathrm{right}}, \qquad Y, Z \in \mathbf{R}^{r_{j-1} \times n_j \times r_j}.
\end{aligned}
$$

We can calculate this with the following contractions assuming that the ordering of dimensions denoted by : is equal on the left- and right-hand sides of the equations:

$$(Y')_{:,:,:,:} \leftarrow \sum_{i} (\overline{A}_{j+1,\text{right}})_{:,:,i} Y_{:,:,i},$$
$$(Y'')_{:,:,:,:} \leftarrow \sum_{i_1,i_2} (A_j)_{:,:,i_1,i_2} (Y')_{i_2,:,:,i_1},$$
$$Z_{:,:,:} \leftarrow \sum_{i_1,i_2} (\overline{A}_{j-1,\text{left}})_{:,i_1,i_2} (Y'')_{i_2,:,:,i_1}.$$

If we reorder the array dimensions of the sub-tensors of $\overline{\mathcal{A}}_{\text{TT}}$ appropriately and let $(\widehat{A}_1, \widehat{A}_2, \widehat{A}_3)$ denote the reordered tensors, we can use the following steps instead:

$$(\widehat{Y}')_{:,:,:,k} \leftarrow \sum_{i} (\widehat{A}_3)_{:,i,k} Y_{:,:,i} \quad \text{for } k = 1,\ldots,r^A_j,$$
$$(\widehat{Y}'')_{:,:,:,k} \leftarrow \sum_{i_1,i_2} (\widehat{A}_2)_{:,i_1,i_2,k} (\widehat{Y}')_{:,:,i_1,i_2} \quad \text{for } k = 1,\ldots,r^A_{j-1},$$
$$Z_{:,:,:} \leftarrow \sum_{i_1,i_2} (\widehat{A}_1)_{:,i_1,i_2} (\widehat{Y}'')_{:,:,i_1,i_2}.$$

The benefit of this reordering is that we can combine the contracted and "free" dimensions ((i_1, i_2) and $(:, :)$) to obtain fewer large dimensions which reduces overhead in the implementation. In addition, we employ a column-major storage with a padding to obtain array strides that are multiples of the cache line length but not high powers of 2 to avoid cache thrashing. Another approach consists in using optimized tensor contractions as discussed in detail in Springer and Bientinesi (2018) which may reduce the overhead due to several small dimensions.

5.4. Resulting building block performance

Here, we show experiments on a single CPU socket of an AMD EPYC 7773X ("Zen 3 V-Cache") with 64 cores and the Intel oneMKL (Intel, 2023) as underlying BLAS/LAPACK library.[1] We use the Q-less TSQR implementation discussed in Röhrig-Zöllner et al. (2022).

Figure 3(a) shows timings for different variants of the TT-axpby + trunc operation. We observe significant speedup through replacing SVDs and pivoted QR decompositions by a fast Q-less TSQR implementation: For the QR orthogonalization sweep alone, the runtime improves by a factor of ~4.5 for the largest case (difference between colored an black lines). For the subsequent SVD

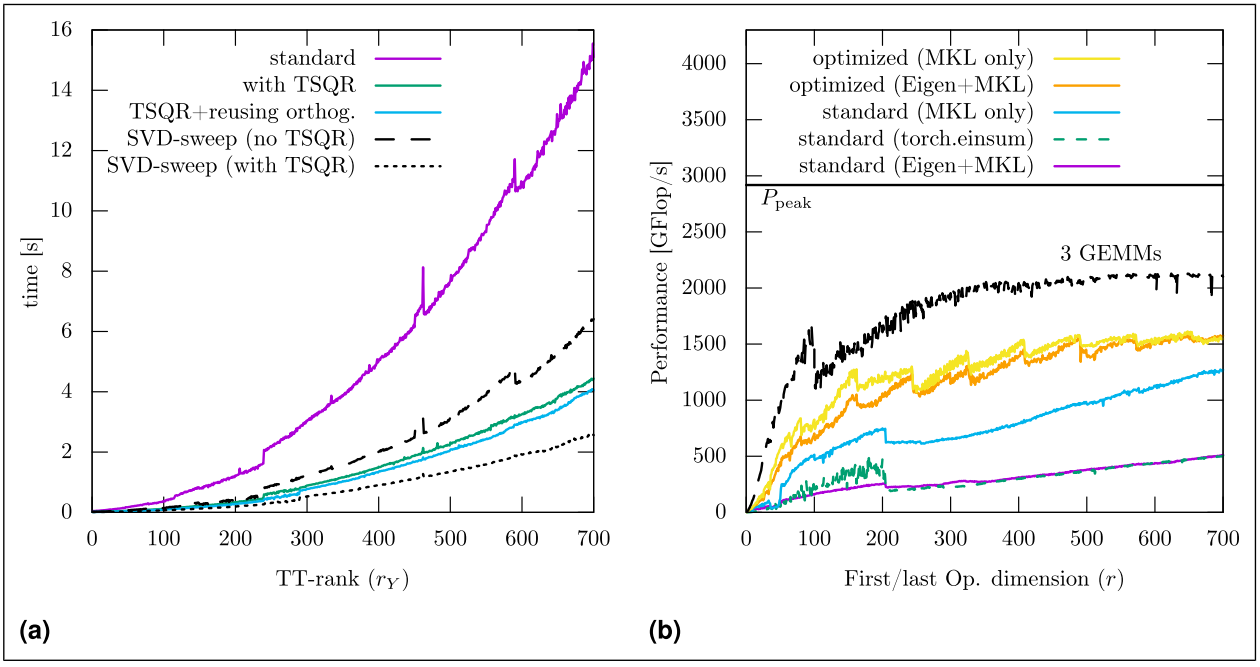

Figure 3. Effect of building block optimizations: For adding two tensors in the tensor-train format (left), we obtain a speedup of ~3.5 by mapping the calculation onto faster linear algebra operations as explained in Section 5.1 and Section 5.2. For applying the linear operator of the inner problem in AMEn (right), we obtain a speedup of ~3 through directly calling optimized BLAS routines and through reordering array dimensions. (a) Timings for the truncated addition of two tensor-trains (dim. 5010, ranks $r_X = 50$ and $r_Y = 1, \ldots, 700$). (b) Performance of the contractions for multiplying a 3d TT-operator with a dense tensor (dim.: $r \times 50 \times r$).

sweep, the runtime improves by a factor of ∼2.5. Overall speedup is about ∼3.5. We only see a small effect through reusing the orthogonality in the example here. It uses random tensor-trains as inputs, so the rank of the result is the sum of the individual ranks. In practice, for example, for the Arnoldi iteration or for calculating the residual in AMEn, the rank of the resulting tensor-train is often smaller which leads to less work in the SVD sweep and thus a bigger effect through reusing orthogonalities in the preceding QR sweep.

For the contraction in the inner iteration of AMEn, we illustrate the performance of different variants in Figure 3(b). The operation consists of 3 tensor contractions, one of them is memory-bound and the other two are compute-bound for the chosen dimensions (for $r \gtrsim 100$). The dotted line shows the performance of 3 equivalent matrix-matrix multiplications (GEMM). In particular for small dimension r, there is a significant improvement through reordering and combining the dimensions as discussed in Section 5.3. Without appropriate padding, there are performance drops whenever the r is close to a number dividable by, for example, 128 (not shown here). Overall, there is also a significant difference between using Eigen (Guennebaud and Jacob, 2010) with MKL as backend and directly calling MKL through the cblas interface, possibly because Eigen explicitly initializes the result to zero. The standard implementation used here loops over a lot of possibly small GEMM operations. We also show results using the function `torch.einsum` in pytorch (Paszke et al., 2019, version 2.4.0) with opt_einsum (Smith and Gray, 2018) for the unoptimized ordering of dimensions with MKL 2022.1 as backend. The function `torch.einsum` results in similar performance to the unoptimized variant with Eigen and MKL. This underlines that we choose the optimal contraction order and that the suggested optimization of the data layout speeds up the computation further compared to common tensor libraries such as pytorch.

5.5. Complete TT-AMEn algorithm

For the complete algorithm for solving a linear system, we focus on the AMEn method as it needs at least an order of magnitude fewer operations than the other methods. As shown in Figure 4, the full AMEn variant needs approximately twice the time of the ALS variant. This is due to calculating the global residual $\mathcal{A}_{\mathrm{TT}} X_{\mathrm{TT}} - B_{\mathrm{TT}}$ which is almost as costly as calculating X_{TT} itself. We also show timings obtained with the ttpy implementation (ALS variant) from Dolgov and Savostyanov (2014) for which we linked with Intel MKL for the underlying operations. Through using optimized building blocks and the suggested TT-rank-1 preconditioner, we speed up the calculation by a

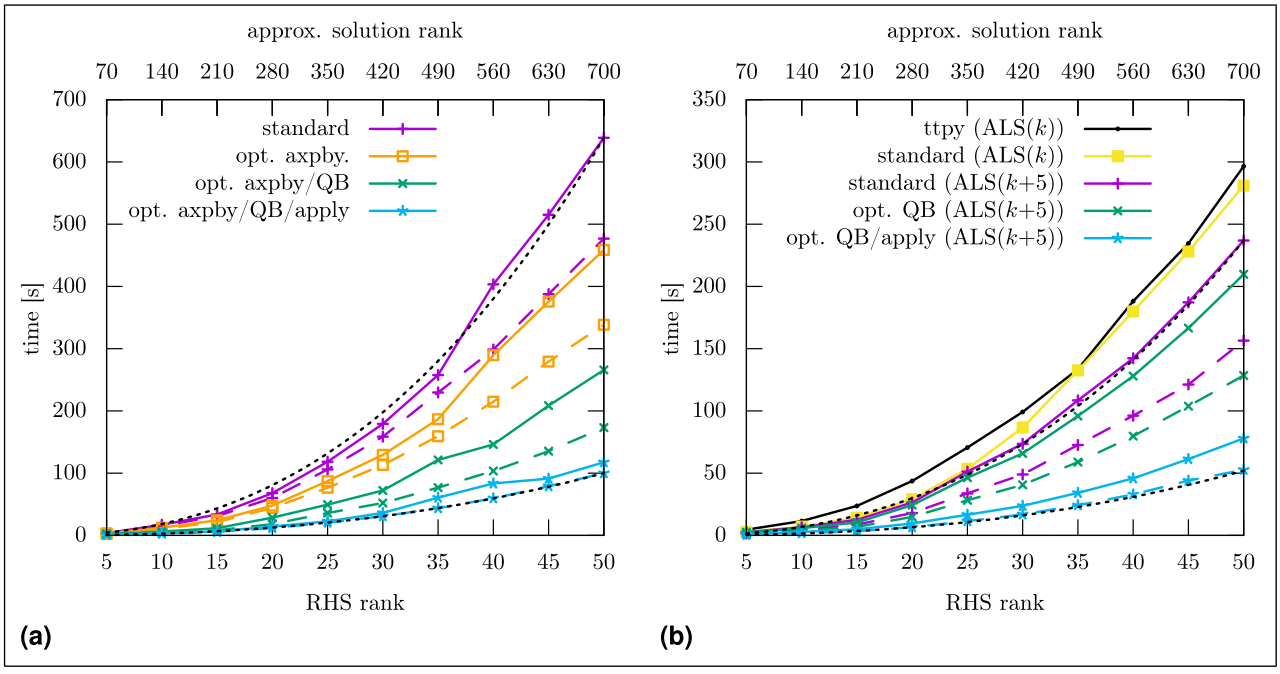

Figure 4. Timings for TT-AMEn for solving a linear system from a 50^{10} convection-diffusion problem ($c = 10$) and random RHS B_{TT} with varying ranks. Dashed lines use the TT-rank-1 preconditioner. Dotted black lines illustrate the asymptotic complexity using the formula $c\,(0.35\,(r/700)^3 + 0.65\,(r/700)^2)$. The heuristic ALS variant (right) is about twice as fast as the full variant (left). For both variants, the time-to-solution is reduced by a factor of ∼5 by combining all suggested optimizations. (a) Full variant (AMEn + SVD) (b) ALS variant (AMEn + ALS).

factor of ∼5 for both the full and the ALS variants. In our tests with the ALS variant, we obtain better convergence and time-to-solution by using a slightly better approximation of the residual (cases with *ALS* ($k + 5$) where k denotes the AMEn enrichment rank and k = rank (B_{TT})). The TT-rank-1 preconditioner reduces the required number of inner GMRES iterations by about a factor of two here: from ∼1530 to ∼750 for the SVD variant and from ∼1580 to ∼850 for the ALS variant for the largest case. However, the number of outer iterations (sweeps) stays the same and a significant part of the runtime is spent in the outer iteration. Therefore, the preconditioner only speeds up the total runtime by about a factor of 1.2–1.5.

6. Conclusion and future work

In this paper, we discussed the complexity and the performance of linear solvers in tensor-train format. In particular, we considered three different common methods, namely TT-GMRES, MALS (DMRG approach for linear systems) and AMEn, and tested their behavior for a simple, non-symmetric discretization of a convection-diffusion equation. Concerning the complexity in terms of floating-point operations, we illustrated that AMEn can be about 100× faster than MALS, which in turn can be about 100× faster than TT-GMRES. These results already include an optimized orthogonalization scheme for the Arnoldi iteration in the TT-GMRES method which is also used as inner iteration of the MALS method.

Concerning the performance, we focussed on the required building blocks on a many-core CPU. We suggested three improvements over the standard implementation: (a) exploiting orthogonalities in the TT-addition with subsequent truncation, (b) using a Q-less tall-skinny QR (TSQR) implementation to speed up costly singular value and QR decompositions, and (c) optimizing the memory layout/ordering of required tensor-contraction sequences for applying the tensor-train operator in the inner iteration. As improvements (a) and (b) lead to less robust underlying linear algebra operations, we discussed their accuracy in the context of the required tensor-train operations. In addition, we presented a simple generic preconditioner based on a tensor-train rank-1 approximation of the operator. Overall, we obtained a speedup of about 5× over the reference implementation on a 64-core CPU.

For future work, we want to investigate building block improvements for other tensor-network algorithms which are often based on very similar underlying operations.

Acknowledgements

Our former student Rebekka-Sarah Hennig investigated the behavior of tensor-train Krylov methods in her Master's thesis, which is available at https://elib.dlr.de/143341/1/master-endversion.pdf.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was partially performed as part of the Helmholtz School for Data Science in Life, Earth and Energy (HDS-LEE) and received funding from the Helmholtz Association of German Research Centres. This work was also partially funded by the Quantum Computing Initiative of the German Aerospace Center (DLR) and the German Federal Ministry for Economic Affairs and Climate Action (Bundesministerium für Wirtschaft und Klimaschutz); see https://qci.dlr.de/projects/qutenet.

ORCID iDs

Melven Röhrig-Zöllner https://orcid.org/0000-0001-9851-5886
Jonas Thies https://orcid.org/0000-0001-9231-9999
Achim Basermann https://orcid.org/0000-0003-3637-3231

Note

1. We also obtain qualitatively similar results on a 16-core Intel Xeon Gold 6246. We use a workaround for running Intel oneMKL on AMD hardware defining the function mkl_serv_intel_cpu_true as discussed in https://danieldk.eu/Posts/2020-08-31-MKL-Zen.html.

References

Al Daas H, Ballard G, Cazeaux P, et al. (2023) Randomized algorithms for rounding in the tensor-train format. *SIAM Journal on Scientific Computing* 45(1): A74–A95. DOI: 10.1137/21m1451191.

Ballani J and Grasedyck L (2012) A projection method to solve linear systems in tensor format. *Numerical Linear Algebra with Applications* 20(1): 27–43. DOI: 10.1002/nla.1818.

Carson E, Demmel J, Grigori L, et al. (2016) Write-avoiding algorithms. In: 2016 IEEE international parallel and distributed processing symposium (IPDPS), Chicago, IL, 23–27 May 2016, 648–658. IEEE. DOI: 10.1109/ipdps.2016.114.

Coulaud O, Giraud L and Iannacito M (2022) A robust gmres algorithm in tensor train format. ArXiv Preprint arXiv: 2210.14533.

Daas HA, Ballard G and Benner P (2022) Parallel algorithms for tensor train arithmetic. *SIAM Journal on Scientific Computing* 44(1): C25–C53. DOI: 10.1137/20m1387158.

Dahmen W, DeVore R, Grasedyck L, et al. (2015) Tensor-sparsity of solutions to high-dimensional elliptic partial differential equations. *Foundations of Computational Mathematics* 16(4): 813–874. DOI: 10.1007/s10208-015-9265-9.

Dolgov SV (2013) TT-GMRES: solution to a linear system in the structured tensor format. *Russian Journal of Numerical*

Analysis and Mathematical Modelling 28(2): 149–172. DOI: 10.1515/rnam-2013-0009.

Dolgov S and Pearson JW (2019) Preconditioners and tensor product solvers for optimal control problems from chemotaxis. *SIAM Journal on Scientific Computing* 41(6): B1228–B1253. DOI: 10.1137/18m1198041.

Dolgov SV and Savostyanov DV (2014) Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing* 36(5): A2248–A2271. DOI: 10.1137/140953289.

Giraud L, Langou J, Rozložník M, et al. (2005) Rounding error analysis of the classical gram-schmidt orthogonalization process. *Numerische Mathematik* 101(1): 87–100. DOI: 10.1007/s00211-005-0615-4.

Golub GH and Van Loan CF (2013) *Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences*. 4th edition. Baltimore, MD: The Johns Hopkins University Press. Literaturangaben. DOI: 10.56021/9781421407944.

Guennebaud G and Jacob B (2010) Eigen v3. Version 3.3.9. https://eigen.tuxfamily.org.

Hager G and Wellein G (2010) *Introduction to High Performance Computing for Scientists and Engineers*. Boca Raton, FL: CRC Press. DOI: 10.1201/ebk1439811924.

Higham NJ (1990) *Analysis of the Cholesky Decomposition of a Semi-Definite Matrix*. Oxford, UK: Clarendon Press, pp. 161–185.

Higham NJ (2002) Accuracy and stability of numerical algorithms. In: *Number 80 in Other Titles in Applied Mathematics*. 2nd edition. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Holtz S, Rohwedder T and Schneider R (2012) The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing* 34(2): A683–A713. DOI: 10.1137/100818893.

Intel (2023) Intel(R) oneAPI math kernel library (oneMKL). https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html.Version2023.2.

Kawamura H and Suda R (2021) Least upper bound of truncation error of low-rank matrix approximation algorithm using QR decomposition with pivoting. *Japan Journal of Industrial and Applied Mathematics* 38(3): 757–779. DOI: 10.1007/s13160-021-00459-x.

Khoromskij BN (2011) $O(d \log N)$-quantics approximation of n-d tensors in high-dimensional numerical modeling. *Constructive Approximation* 34(2): 257–280. DOI: 10.1007/s00365-011-9131-1.

Kolda TG and Bader BW (2009) Tensor decompositions and applications. *SIAM Review* 51(3): 455–500. DOI: 10.1137/07070111x.

Kressner D and Tobler C (2011) Low-rank tensor krylov subspace methods for parametrized linear systems. *SIAM Journal on Matrix Analysis and Applications* 32(4): 1288–1316. DOI: 10.1137/100799010.

Kressner D, Steinlechner M and Vandereycken B (2016) Preconditioned low-rank riemannian optimization for linear systems with tensor product structure. *SIAM Journal on Scientific Computing* 38(4): A2018–A2044. DOI: 10.1137/15m1032909.

Leon SJ, Björck Å and Gander W (2012) Gram-Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications* 20(3): 492–532. DOI: 10.1002/nla.1839.

Oseledets IV (2011) Tensor-train decomposition. *SIAM Journal on Scientific Computing* 33(5): 2295–2317. DOI: 10.1137/090752286.

Oseledets IV and Dolgov SV (2012) Solution of linear systems and matrix inversion in the TT-format. *SIAM Journal on Scientific Computing* 34(5): A2718–A2739. DOI: 10.1137/110833142.

Oseledets IV, Rakhuba MV and Uschmajew A (2018) Alternating least squares as moving subspace correction. *SIAM Journal on Numerical Analysis* 56(6): 3459–3479. DOI: 10.1137/17m1148712.

Parlett BN (1998) *The Symmetric Eigenvalue Problem*. Philadelphia, PA: Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9781611971163.

Paszke A, Gross S, Massa F, et al. (2019) *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. Red Hook, NY: Curran Associates Inc. DOI: 10.5555/3454287.3455008.

Röhrig-Zöllner M and Becklas MJ (2024) PITTS - parallel iterative tensor-train solvers. DOI: 10.5281/zenodo.13762681. URL: https://github.com/melven/pitts.

Röhrig-Zöllner M, Thies J and Basermann A (2022) Performance of the low-rank TT-SVD for large dense tensors on modern MultiCore CPUs. *SIAM Journal on Scientific Computing* 44(4): C287–C309. DOI: 10.1137/21m1395545.

Saad Y (2003) *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA: Society for Industrial and Applied Mathematics. DOI: 10.1137/1.9780898718003.

Schollwöck U (2005) The density-matrix renormalization group. *Reviews of Modern Physics* 77(1): 259–315. DOI: 10.1103/revmodphys.77.259.

Shi T and Townsend A (2021) On the compressibility of tensors. *SIAM Journal on Matrix Analysis and Applications* 42(1): 275–298. DOI: 10.1137/20m1316639.

Simoncini V and Szyld DB (2003) Theory of inexact Krylov subspace methods and applications to scientific computing. *SIAM Journal on Scientific Computing* 25(2): 454–477. DOI: 10.1137/s1064827502406415.

Smith DGA and Gray J (2018) opt_einsum - a python package for optimizing contraction order for einsum-like expressions. *Journal of Open Source Software* 3(26): 753. DOI: 10.21105/joss.00753.

Springer P and Bientinesi P (2018) Design of a high-performance GEMM-like tensor–tensor multiplication. *ACM Transactions on Mathematical Software* 44(3): 1–29. DOI: 10.1145/3157733.

Treibig J, Hager G and Wellein G (2010) LIKWID: a lightweight performance-oriented tool suite for x86 multicore environments. In: 2010 39th international conference on parallel processing workshops, San Diego, CA, 13–16 September 2010. IEEE. DOI: 10.1109/icppw.2010.38.

van den Eshof J and Sleijpen GLG (2004) Inexact krylov subspace methods for linear systems. *SIAM Journal on Matrix Analysis and Applications* 26(1): 125–153. DOI: 10.1137/s0895479802403459.

van der Vorst HA (2003) *Iterative Krylov Methods for Large Linear Systems*. Cambridge, UK: Cambridge University Press. DOI: 10.1017/CBO9780511615115.

White SR (1992) Density matrix formulation for quantum renormalization groups. *Physical Review Letters* 69(19): 2863–2866. DOI: 10.1103/PhysRevLett.69.2863.

Williams S, Waterman A and Patterson D (2009) Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM* 52(4): 65–76. DOI: 10.1145/1498765.1498785.

Wilson KG (1983) The renormalization group and critical phenomena. *Reviews of Modern Physics* 55(3): 583–600. DOI: 10.1103/RevModPhys.55.583.

Author biographies

Melven Röhrig-Zöllner studied Computational Engineering Science at the RWTH Aachen University. Since 2014, he works as a researcher in the High-Performance Computing department of the Institute of Software Technology of the German Aerospace Center (DLR). His research focuses on the performance of numerical methods, in particular in the field of numerical linear algebra. He also supports scientific software development for HPC systems in the DLR, for example, concerning software engineering practices and testing parallel codes.

Manuel Becklas received his Bachelor's degree in mathematics at the University of Cologne in 2023. He currently pursues his Master in Innovation and Research in Informatics at the Polytechnic University of Catalonia. He worked as a student at the German Aerospace Center in Cologne in the Institute of Software Technology from 2022 to 2023. His scientific interests include high-performance computing, modern C++, linear algebra and machine learning theory.

Jonas Thies is an assistant professor and HPC advisor at the Delft University of Technology. He received his PhD in Mathematics from the University of Groningen in 2011, a Masters degree in Scientific Computing from KTH in 2006, and a Bachelors degree in Computational Engineering from the University of Erlangen Nuremberg in 2003. His research and teaching activities focus on parallel numerical algorithms, high-performance computing, and applications like CFD, oceanography and quantum physics.

Dr.-Ing. Achim Basermann is head of the High-Performance Computing Department at German Aerospace Center's (DLR) Institute of Software Technology. In 1995, he obtained his Ph.D. in Electrical Engineering from RWTH Aachen University followed by a postdoctoral position in computer science at Jülich Research Centre, Central Institute for Applied Mathematics. From 1997 to 2009 he led a team of HPC application experts at the C&C Research Laboratories, NEC Laboratories Europe, before joining DLR. Dr. Basermann's current research is focused on extremely parallel linear algebra algorithms, performance engineering for many-core and GPU clusters, data partitioning methods, optimization tools for CFD simulation, high-performance data analytics, scientific machine learning and quantum computing.