Electronic Transactions on Numerical Analysis.
Volume 63, pp. 300–319, 2025.
© 2025, the Author(s), licensed via CC BY 4.0.
ISSN 1068–9613.
DOI: 10.1553/etna_vol63s300

ETNA
Kent State University **and**
Johann Radon Institute (RICAM)

COMPUTING A FEW EXTREME SINGULAR TRIPLETS OF A THIRD-ORDER TENSOR USING THE t-PRODUCT[*]

ANAS EL HACHIMI[†‡], KHALIDE JBILOU[‡], MOHAMED J. MAAOUNI[†], AHMED RATNANI[†], AND LOTHAR REICHEL[§]

Abstract. This paper describes software for computing a few of the largest or smallest singular triplets of a third-order tensor using the t-product. The software implements restarted partial tensor bidiagonalization techniques that were introduced by El Hachimi et al. in [Numer. Linear Algebra Appl., 31 (2024), Art. e2530]. Restarting is carried out by augmenting the available solution subspace by Ritz lateral slices or by harmonic Ritz lateral slices. The performance of our Python implementation is investigated. The software is designed for easy use in various applications.

Key words. tensor algorithms, singular triplets, t-product, Ritz augmentation, harmonic Ritz augmentation

AMS subject classifications. 15A69, 65F15, 65F50

1. Introduction.

Tensors are higher-dimensional extensions of matrices. They find applications in a variety of areas including data compression and completion, facial recognition, image restoration, and network analysis. This paper focuses on third-order tensors, which are data arrays in $\mathbb{C}^{\ell \times p \times n}$, whose multiplication is defined by the t-product. This tensor product was introduced by Kilmer et al. [7, 8] and has found many applications. The t-product allows the definition of analogues of well-known matrix factorizations for third-order tensors. For instance, the t-svd is an analogue for third-order tensors of the matrix singular value decomposition (svd).

Baglama et al. [1, 2] describe efficient algorithms for approximating a few extreme singular triplets of a large matrix. Generalizations of these algorithms for the computation of a few extreme singular triplets of a third-order tensor using the t-product are presented in [5]. This paper presents efficient Python implementations of the latter algorithms. The aim of the current work is to provide an open-source and well-documented implementation. The availability of our Python code makes it easy for scientists to use these methods in novel applications.

To enhance readability, we summarize key theoretical concepts. In addition, we present numerical experiments, runtime comparisons, and validation plots to benchmark the performance of our implementation in realistic settings. The accompanying software is modular and extensible, serving as a platform for future research in low-rank tensor approximation under the t-product framework.

2. The t-product of third-order tensors.

The t-product allows the extension of many matrix operations to third-order tensors. We will use notation from Kilmer and Martin [8] and Kolda and Bader [9].

[*] Received November 27, 2024. Accepted July 18, 2025. Published online on August 26, 2025. Recommended by A. Buccini.

[†] The UM6P Vanguard Center, Mohammed VI Polytechnic University, Green City, Morocco (anaselhachimi1997@gmail.com, {jalal.maaouni, ahmed.ratnani}@um6p.ma).

[‡] Université du Littoral Cote d'Opale, LMPA, 50 rue F. Buisson, 62228 Calais-Cedex, France (khalide.jbilou@univ-littoral.fr).

[§] Kent State University, Department of Mathematical Sciences, Kent, OH 44242, USA (reichel@math.kent.edu).

A third-order tensor is an array $\mathcal{A} = [a_{ijk}] \in \mathbb{C}^{\ell\times p\times n}$. Matrices are tensors of order two, and vectors are tensors of order one. A *slice* of a third-order tensor $\mathcal{A}$ is a section obtained by fixing any one of the three indices. Using MATLAB notation, we let $\mathcal{A}(i,:,:)$, $\mathcal{A}(:,j,:)$, and $\mathcal{A}(:,:,k)$ denote the ith horizontal, the jth lateral, and the kth frontal slices of $\mathcal{A}$, respectively. The lateral slice $\mathcal{A}(:,j,:)$ also is denoted by $\vec{\mathcal{A}}_j$, and the frontal slice $\mathcal{A}(:,:,k)$ is an $\ell \times p$ matrix, which we sometimes denote by $\mathcal{A}^{(k)}$. A *fiber* of a third-order tensor $\mathcal{A}$ is defined by fixing any two of the three indices. The fiber $\mathcal{A}(i,j,:)$ is called a *tube* of $\mathcal{A}$.

We will use capital calligraphic letters $\mathcal{A}$ to denote third-order tensors, capital letters A to identify matrices, boldface lower case letters $\boldsymbol{a}$ to denote tubes, and lower case letters a stand for scalars. Furthermore, $\mathbb{K}_n^{\ell\times p} = \mathbb{R}^{\ell\times p\times n}$ stands for the space of third-order tensors of size $\ell \times p \times n$, $\mathbb{K}_n^{\ell} = \mathbb{R}^{\ell\times 1\times n}$ denotes the space of lateral slices of size $\ell \times n$, and $\mathbb{K}_n = \mathbb{R}^{1\times 1\times n}$ stands for the space of tubes with n entries.

Given two tensors $\mathcal{A} \in \mathbb{K}_n^{\ell\times q}$ and $\mathcal{B} \in \mathbb{K}_n^{q\times p}$, their t-product is defined as

$$\mathcal{A} * \mathcal{B} = \texttt{fold}(\texttt{bcirc}(\mathcal{A})\texttt{unfold}(\mathcal{B})) \in \mathbb{K}_n^{\ell\times p},$$

where the operations `bcirc`, `fold`, and `unfold` are defined as follows: for the tensor $\mathcal{A} \in \mathbb{K}_n^{\ell\times p}$, the associated block circulant matrix is defined as

$$\texttt{bcirc}(\mathcal{A}) = \begin{bmatrix} \mathcal{A}^{(1)} & \mathcal{A}^{(n)} & \cdots & \mathcal{A}^{(2)} \\ \mathcal{A}^{(2)} & \mathcal{A}^{(1)} & \cdots & \mathcal{A}^{(3)} \\ \mathcal{A}^{(3)} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \mathcal{A}^{(n)} & \mathcal{A}^{(n-1)} & \cdots & \mathcal{A}^{(1)} \end{bmatrix} \in \mathbb{R}^{\ell n\times qn},$$

and for the tensor $\mathcal{B}$, the unfolding operation results in

$$\texttt{unfold}(\mathcal{B}) = \begin{bmatrix} \mathcal{B}^{(1)} \\ \mathcal{B}^{(2)} \\ \vdots \\ \mathcal{B}^{(n)} \end{bmatrix} \in \mathbb{R}^{qn\times p}.$$

The folding operation $\texttt{fold}(\texttt{unfold}(\mathcal{B}))$ recovers the tensor $\mathcal{B}$.

The t-product can be evaluated with the aid of the Discrete Fourier Transform (DFT). The transformed tensor $\widehat{\mathcal{A}}$ is obtained by applying the DFT along the third dimension. Using MATLAB notation,

$$\widehat{\mathcal{A}} = \texttt{fft}(\mathcal{A}, [\,], 3).$$

The inverse operation can be evaluated with the command

$$\mathcal{A} = \texttt{ifft}(\widehat{\mathcal{A}}, [\,], 3).$$

It follows that the t-product $\mathcal{C} = \mathcal{A} * \mathcal{B}$ can be evaluated by first computing the matrix products

$$\widehat{\mathcal{C}}^{(i)} = \widehat{\mathcal{A}}^{(i)}\widehat{\mathcal{B}}^{(i)}, \qquad i = 1, 2, \ldots, n,$$

where $\widehat{\mathcal{A}}^{(i)}$, $\widehat{\mathcal{B}}^{(i)}$, and $\widehat{\mathcal{C}}^{(i)}$ are the ith frontal slices of the tensors $\widehat{\mathcal{A}}$, $\widehat{\mathcal{B}}$, and $\widehat{\mathcal{C}}$, respectively, and then computing the inverse transform of $\widehat{\mathcal{C}}$; see, e.g., Kilmer et al. [7] or El Hachimi et al. [5] for further details, where it is also pointed out that the count of arithmetic floating point operations (flops) can be reduced somewhat by using some symmetry properties that arise for real tensors. These symmetries do not hold for complex tensors. The operation `conj` in the following algorithm denotes element-wise complex conjugation.

Algorithm 1 The t-product with symmetry check for real tensors.

```
Input: $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$, $\mathcal{B} \in \mathbb{K}_n^{p \times q}$
Output: $\mathcal{C} \in \mathbb{K}_n^{\ell \times q}$
Compute $\widehat{\mathcal{A}}$, and $\widehat{\mathcal{B}}$.
if both $\mathcal{A}$ and $\mathcal{B}$ are real then
    for $i = 1, \dots, \left[\frac{n+1}{2}\right]$ do
        $\widehat{\mathcal{C}}^{(i)} = \widehat{\mathcal{A}}^{(i)} \widehat{\mathcal{B}}^{(i)}$;
    end for
    for $i = \left[\frac{n+1}{2}\right] + 1, \dots, n$ do
        $\widehat{\mathcal{C}}^{(i)} = \texttt{conj}(\widehat{\mathcal{C}}^{(n-i+2)})$;
    end for
else
    for $i = 1, \dots, n$ do
        $\widehat{\mathcal{C}}^{(i)} = \widehat{\mathcal{A}}^{(i)} \widehat{\mathcal{B}}^{(i)}$;
    end for
end if
Compute $\mathcal{C}$ from $\widehat{\mathcal{C}}$;
```

2.1. Computational complexity of the t-product. The dominant computational work required when evaluating the t-product of two tensors $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$ and $\mathcal{B} \in \mathbb{K}_n^{p \times q}$ by Algorithm 1 is the calculation of DFTs and of matrix-matrix products. The DFT is applied to every tube of the tensor $\mathcal{A}$, each with n components. Using the Fast Fourier Transform (FFT) algorithm, the computation of the DFT of an n-vector requires $O(n \log n)$ flops. Consequently, the application of the DFT to all tubes demands $O(\ell \cdot p \cdot n \log n)$ flops. The same flop count holds for the inverse transform. Similarly, the DFT applied to $\mathcal{B}$ requires $O(p \cdot q \cdot n \log n)$ flops. Matrix-matrix products of frontal slices have to be evaluated. This requires $O(\ell \cdot p \cdot q)$ flops for each pair of slices. When applied to all slices, the resulting flop count becomes $O(n \cdot \ell \cdot p \cdot q)$. Summing up these flop counts, we obtain that Algorithm 1 demands $O(\ell \cdot p \cdot n \cdot \log n + p \cdot q \cdot n \log n + n \cdot \ell \cdot p \cdot q)$ flops. When the tensors are large, the matrix-matrix product evaluations dominate the flop count.

Algorithm 1 requires storage of the tensors $\mathcal{A}$ and $\mathcal{B}$, and of the Fourier-transformed tensors $\widehat{\mathcal{A}}$ and $\widehat{\mathcal{B}}$, in addition to the storage of the resulting tensor $\mathcal{C}$. The latter may overwrite the tensor $\widehat{\mathcal{C}}$. Here we assume that we do not want to overwrite the tensors $\mathcal{A}$ and $\mathcal{B}$. The storage requirement then is of size $O(n \cdot (\ell \cdot p + p \cdot q + \ell \cdot q))$.

To illustrate the computational complexity of Algorithm 1, we applied the algorithm to a sequence of $n \times n \times n$ tensors for increasing values of n and measured the runtimes, which are displayed in Figure 2.1. The runtime grows roughly like $O(n^4)$. All computations shown in this paper were carried out on a desktop computer equipped with an AMD Ryzen 9 7950X 16-Core Processor (32 threads), with a base clock speed of 400 MHz and a maximum clock speed of 5.84 GHz. The system has 64 GB of RAM and runs Python 3.10.12 on a Linux-based operating system. All calculations involving tensors are averaged over 20 runs, and the tensors used in the experiments are randomly generated with reasonable conditioning to ensure numerical stability.

Figure 2.1 shows the computing time to be quite close to the curve $n \to c \cdot n^4$, where $c = 6.93 \times 10^{-11}$. However, certain data points deviate somewhat from this curve. This variation can largely be attributed to overheads present in real-world computation environments. Moreover, we found that the time required by the FFT function of Python varies significantly depending on the factorization of n.

The execution times for the Python functions fft and ifft generally follow the expected $n^3 \log(n)$ trend, and tensor-tensor multiplication follows the n^4 trend. However, variations in the timings for the fft function indicate that the timings are heavily influenced by the factorization of n; the runtime is faster when n is a product of small primes; see Figure 2.2.

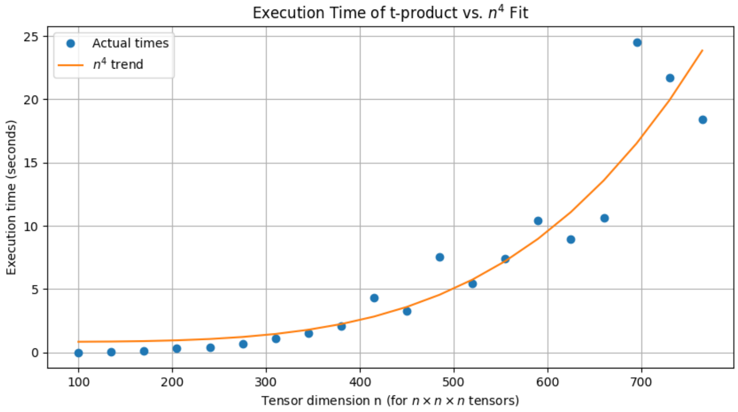

FIG. 2.1. *Runtime of Algorithm 1 as a function of n and a comparison with n^4 (red curve).*

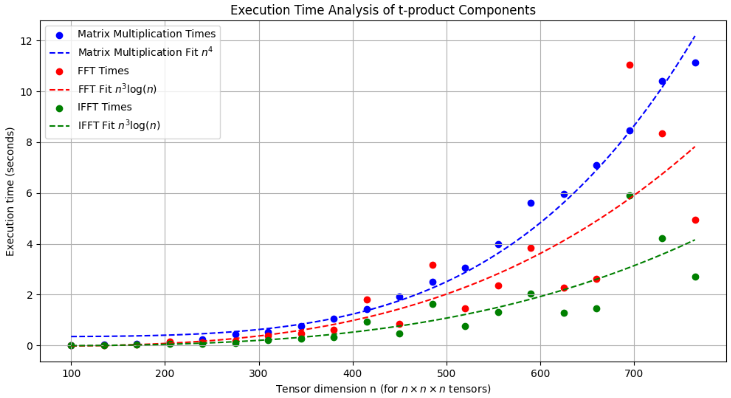

FIG. 2.2. *Detailed runtime analysis of the three significant components of the t-product evaluation. Timings for the fft and ifft functions, as well as for tensor-tensor product evaluations are displayed. The timings for the fft and ifft functions are compared with graphs for $n \to cn^3 \log(n)$, and the timings for matrix-matrix product evaluation are compared to the graph $n \to dn^4$ for suitable constants c and d.*

The fitted function for tensor-tensor multiplication is $n \to c \cdot n^4$ with $c = 3.2 \times 10^{-11}$, while the fft and ifft coefficients show greater variation, reflecting sensitivity to n. The fitted curves for these operations are shown in the figure, where the multiplication fit follows dn^4 and the fft and ifft fits follow $cn^3 \log(n)$ for suitable constants c and d.

3. The tensor singular value decomposition (t-svd). One of the attractions of the tensor t-product is that it allows for generalizations of common matrix factorizations to third-order tensors including Cholesky factorization, QR factorization, singular value decomposition (SVD), and spectral factorization; see, e.g., [4, 5, 7, 11].

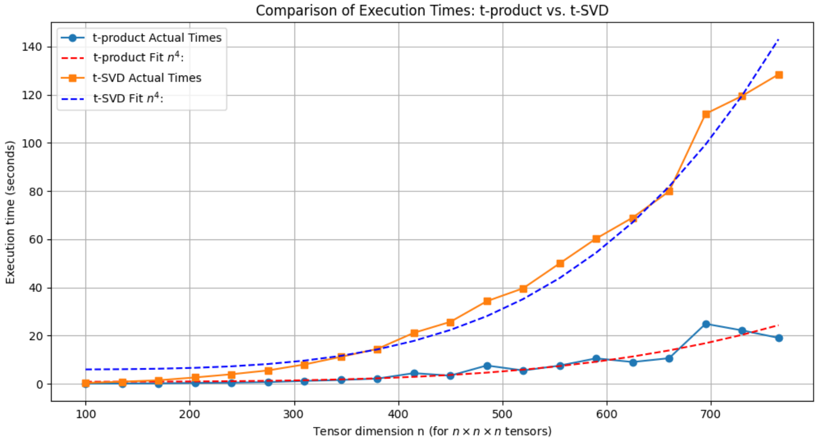

FIG. 3.1. *Comparison of the runtime for the evaluation of the t-svd and the t-product of tensors in* $\mathbb{K}_n^{n\times n}$ *for increasing values of* n. *The graphs illustrate the significantly higher computational demand of t-svd computations.*

The SVD of a tensor under the t-product, known as the t-svd, has received considerable attention due to its use in a variety of applications including tensor denoising, completion, and compression; see, e.g, [3, 10, 12]. A third-order tensor $\mathcal{A} \in \mathbb{K}_n^{\ell\times\ell}$ is said to be orthogonal if

$$\mathcal{A}^H * \mathcal{A} = \mathcal{A} * \mathcal{A}^H = \mathcal{I}_\ell,$$

where the tensor $\mathcal{A}^H$ is obtained by first transposing and conjugating each one of the frontal slices of $\mathcal{A}$, and then reversing the order of the conjugated transposed frontal slices 2 through n. The identity tensor $\mathcal{I}_\ell \in \mathbb{K}_n^{\ell\times\ell}$ has the first frontal slice equal to the identity matrix, and the remaining frontal slices are zero matrices. A third-order tensor is said to be f-diagonal if its frontal slices in the Fourier domain are diagonal matrices; see [7].

THEOREM 3.1 ([7]). *Let* $\mathcal{A} \in \mathbb{K}_n^{\ell\times p}$. *Then* $\mathcal{A}$ *can be factored as*

$$\mathcal{A} = \mathcal{U} * \mathcal{S} * \mathcal{V}^H, \tag{3.1}$$

where $\mathcal{U} \in \mathbb{K}_n^{\ell\times\ell}$ *and* $\mathcal{V} \in \mathbb{K}_n^{p\times p}$ *are orthogonal tensors, and* $\mathcal{S} \in \mathbb{K}_n^{\ell\times p}$ *is an f-diagonal tensor.*

The decomposition (3.1) allows $\mathcal{A}$ to be expressed as

$$\mathcal{A} = \sum_{i=1}^{\min(\ell,p)} \vec{\mathcal{U}}_i * \boldsymbol{s}_i * \vec{\mathcal{V}}_i^H = \sum_{i=1}^{\min(\ell,p)} \boldsymbol{s}_i * \vec{\mathcal{U}}_i * \vec{\mathcal{V}}_i^H,$$

where $\boldsymbol{s}_i = \mathcal{S}(i,i,:)$ represents the i^{th} singular tube and $\vec{\mathcal{U}}_i = \mathcal{U}(:,i,:)$ and $\vec{\mathcal{V}}_i = \mathcal{V}(:,i,:)$ are left and the right singular slices of $\mathcal{A}$, respectively. The singular tubes are ordered according to decreasing norm, i.e.,

$$\|\boldsymbol{s}_1\| \geq \|\boldsymbol{s}_2\| \geq \ldots \geq \|\boldsymbol{s}_{\min(\ell,p)}\| \geq 0,$$

where $\|\cdot\|$ denotes the Euclidean vector norm. We refer to $\{\boldsymbol{s}_i, \vec{\mathcal{U}}_i, \vec{\mathcal{V}}_i\}$ as the ith singular triplet of $\mathcal{A}$. The largest singular triplets are associated with singular tubes of largest norm, while

Algorithm 2 The t-svd.

```
Input: $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$
Output: $\mathcal{U} \in \mathbb{K}_n^{\ell \times \ell}$, $\mathcal{S} \in \mathbb{K}_n^{\ell \times p}$, $\mathcal{V} \in \mathbb{K}_n^{p \times p}$
Compute $\widehat{\mathcal{A}}$.
if $\mathcal{A}$ is real then
    for $i = 1, \ldots, \left[\frac{n+1}{2}\right]$ do
        $\left[\widehat{\mathcal{U}}^{(i)}, \widehat{\mathcal{S}}^{(i)}, \widehat{\mathcal{V}}^{(i)}\right] = \mathtt{svd}\left(\widehat{\mathcal{A}}^{(i)}\right)$.
    end for
    for $i = \left[\frac{n+1}{2}\right] + 1, \ldots, n$ do
        $\widehat{\mathcal{U}}^{(i)} = \mathtt{conj}\left(\widehat{\mathcal{U}}^{(n-i+2)}\right)$
        $\widehat{\mathcal{S}}^{(i)} = \mathtt{conj}\left(\widehat{\mathcal{S}}^{(n-i+2)}\right)$
        $\widehat{\mathcal{V}}^{(i)} = \mathtt{conj}\left(\widehat{\mathcal{V}}^{(n-i+2)}\right)$
    end for
else
    for $i = 1, \ldots, n$ do
        $\left[\widehat{\mathcal{U}}^{(i)}, \widehat{\mathcal{S}}^{(i)}, \widehat{\mathcal{V}}^{(i)}\right] = \mathtt{svd}\left(\widehat{\mathcal{A}}^{(i)}\right)$.
    end for
end if
Compute $\mathcal{U}$, $\mathcal{S}$, and $\mathcal{V}$ from $\widehat{\mathcal{U}}$, $\widehat{\mathcal{S}}$, and $\widehat{\mathcal{V}}$, respectively.
```

the smallest singular triplets are associated with singular tubes of smallest norm. Algorithm 2 outlines the computation of the t-svd of a third-order tensor.

The computation of the t-svd of a tensor $\mathcal{A} \in \mathbb{K}_n^{n \times n}$ requires $O(n^4)$ flops. This flop count follows from the fact that the evaluation of the singular value decomposition of an $n \times n$ matrix demands $O(n^3)$ flops. Figure 3.1 compares the runtime for the computation of tensor-tensor products and the t-svd as a function of n. Despite the fact that both computations require $O(n^4)$ flops, the evaluation of the t-svd can be seen to be much slower. We determine the leading coefficient in the fitted curves so that the dashed graphs approximate the data fairly well. The difference in timings depends on the fact that the evaluation of the t-svd of a tensor $\mathcal{A} \in \mathbb{K}_n^{n \times n}$ is much more complicated than the evaluation of the t-product $\mathcal{A} * \mathcal{A}$. Moreover, the available hardware allows efficient execution of the computations required for the latter task.

The graphs of Figure 3.1 compare the execution times for the t-product and t-svd algorithms, both of which exhibit an asymptotic complexity of $O(n^4)$. However, the constant factors differ significantly: for the t-product, the coefficient is 6.92×10^{-11}, while for the t-svd, it is 4.12×10^{-10}.

4. Partial tensor Lanczos bidiagonalization under the t-product.

The extension of matrix decompositions to tensors via the t-product finds applications in multi-dimensional data analysis. An application to face recognition that uses a database of color images of facial expressions is described in [5]. The numerical method used in this application is based on the computation of a few of the largest singular triplets of a large third-order tensor. This is achieved with the aid of partial tensor Lanczos bidiagonalization.

Let $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$ be a third-order tensor. The application of $1 \leq m \ll \min\{\ell, p, n\}$ steps of the tensor Lanczos bidiagonalization process generically yields the tensors

$$\mathcal{P}_m = \left[\vec{\mathcal{P}}_1, \dots, \vec{\mathcal{P}}_m\right] \in \mathbb{K}_n^{p \times m}, \quad \mathcal{Q}_m = \left[\vec{\mathcal{Q}}_1, \dots, \vec{\mathcal{Q}}_m\right] \in \mathbb{K}_n^{\ell \times m}, \tag{4.1}$$

as well as a bidiagonal tensor $\mathcal{B}_m \in \mathbb{K}_n^{m \times m}$. The lateral slices of $\mathcal{P}_m$ and $\mathcal{Q}_m$ form bases for the tensor Krylov subspaces

$$\begin{aligned}
\boldsymbol{K}_m(\mathcal{A}^H * \mathcal{A}, \vec{\mathcal{P}}_1) &= \text{span}\left\{\vec{\mathcal{P}}_1, \mathcal{A}^H * \mathcal{A} * \vec{\mathcal{P}}_1, \dots, (\mathcal{A}^H * \mathcal{A})^{m-1} * \vec{\mathcal{P}}_1\right\}, \\
\boldsymbol{K}_m(\mathcal{A} * \mathcal{A}^H, \vec{\mathcal{Q}}_1) &= \text{span}\left\{\vec{\mathcal{Q}}_1, \mathcal{A} * \mathcal{A}^H * \vec{\mathcal{Q}}_1, \dots, (\mathcal{A} * \mathcal{A}^H)^{m-1} * \vec{\mathcal{Q}}_1\right\},
\end{aligned}$$

respectively. Algorithm 3 describes the computation of the tensors (4.1).

Algorithm 3 Partial tensor Lanczos bidiagonalization under the t-product.

Input: $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$, initial unit-norm slice $\vec{\mathcal{P}}_1 \in \mathbb{K}_n^{p}$, number of steps $m \in \mathbb{N}^*$.
Output: Tensors $\mathcal{P}_m \in \mathbb{K}_n^{p \times m}$ and $\mathcal{Q}_m \in \mathbb{K}_n^{\ell \times m}$ with orthonormal lateral slices, upper bidiagonal tensor $\mathcal{B}_m \in \mathbb{K}_n^{m \times m}$, and residual tensor $\vec{\mathcal{R}}_m \in \mathbb{K}_n^{p}$.

```
$\mathcal{P}_1 \leftarrow \left[\vec{\mathcal{P}}_1\right]$
$\vec{\mathcal{Q}}_1 \leftarrow \mathcal{A} * \vec{\mathcal{P}}_1$
$\left[\vec{\mathcal{Q}}_1, \boldsymbol{\alpha}_1\right] \leftarrow$ Normalize$(\vec{\mathcal{Q}}_1)$
$\mathcal{Q}_1 \leftarrow \left[\vec{\mathcal{Q}}_1\right]$
$\mathcal{B}_m(1,1,:) \leftarrow \boldsymbol{\alpha}_1$
for $i \leftarrow 1$ to $m$ do
    $\vec{\mathcal{R}}_i \leftarrow \mathcal{A}^H * \vec{\mathcal{Q}}_i - \boldsymbol{\alpha}_i * \vec{\mathcal{P}}_i$
    $\vec{\mathcal{R}}_i \leftarrow \vec{\mathcal{R}}_i - \mathcal{P}_i * (\mathcal{P}_i^H * \vec{\mathcal{R}}_i)$
    if $i < m$ then
        $\left[\vec{\mathcal{P}}_{i+1}, \boldsymbol{\beta}_i\right] \leftarrow$ Normalize$(\vec{\mathcal{R}}_i)$
        $\mathcal{P}_{i+1} \leftarrow \left[\mathcal{P}_i, \vec{\mathcal{P}}_{i+1}\right]$
        $\mathcal{B}_m(i, i+1, :) \leftarrow \boldsymbol{\beta}_i$
        $\vec{\mathcal{Q}}_{i+1} \leftarrow \mathcal{A} * \vec{\mathcal{P}}_{i+1} - \boldsymbol{\beta}_i * \vec{\mathcal{Q}}_i$
        $\vec{\mathcal{Q}}_{i+1} \leftarrow \vec{\mathcal{Q}}_{i+1} - \mathcal{Q}_i * (\mathcal{Q}_i^H * \vec{\mathcal{Q}}_{i+1})$
        $\left[\vec{\mathcal{Q}}_{i+1}, \boldsymbol{\alpha}_{i+1}\right] \leftarrow$ Normalize$(\vec{\mathcal{Q}}_{i+1})$
        $\mathcal{Q}_{i+1} \leftarrow \left[\mathcal{Q}_i, \vec{\mathcal{Q}}_{i+1}\right]$
        $\mathcal{B}_m(i+1, i+1, :) \leftarrow \boldsymbol{\alpha}_{i+1}$
    end if
end for
```

In detail, the Lanczos bidiagonalization process applied to the tensor $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$ is based on the equations

$$\mathcal{A} * \mathcal{P}_m = \mathcal{Q}_m * \mathcal{B}_m, \tag{4.2}$$

$$\mathcal{A}^H * \mathcal{Q}_m = \mathcal{P}_m * \mathcal{B}_m^H + \vec{\mathcal{R}}_m * \vec{\mathcal{E}}_m^H, \tag{4.3}$$

where the tensor slice $\vec{\mathcal{R}}_m$ is orthogonal to all lateral slices of $\mathcal{P}_m$, and $\vec{\mathcal{E}}_m \in \mathbb{K}_n^m$ denotes the canonical lateral slice whose elements are zero except for the first element of the mth tube, which equals 1. The tensor $\mathcal{B}_m$ is upper bidiagonal,

$$\mathcal{B}_m = \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\beta}_1 & & & \\ & \boldsymbol{\alpha}_2 & \boldsymbol{\beta}_2 & & \\ & & \ddots & \ddots & \\ & & & \boldsymbol{\alpha}_{m-1} & \boldsymbol{\beta}_{m-1} \\ & & & & \boldsymbol{\alpha}_m \end{bmatrix} \in \mathbb{K}_n^{m\times m},$$

where the $\boldsymbol{\alpha}_i$ and $\boldsymbol{\beta}_i$ denote coefficients that are determined by the algorithm. Normalization of the "remainder slice" $\vec{\mathcal{R}}_m$ in (4.3) gives

$$\vec{\mathcal{R}}_m = \vec{\mathcal{P}}_m * \boldsymbol{\beta}_m, \tag{4.4}$$

where $\vec{\mathcal{P}}_{m+1} \in \mathbb{K}_m^p$ is of unit norm and $\boldsymbol{\beta}_m$ is a tube; see [5]. Equation (4.3) now can be expressed as

$$\mathcal{A}^H * \mathcal{Q}_m = \mathcal{P}_{m+1} * \mathcal{B}_{m,m+1}^H,$$

where the tensor $\mathcal{P}_{m+1}$ is obtained by appending the lateral slice $\vec{\mathcal{P}}_{m+1}$ to $\mathcal{P}_m$, namely, $\mathcal{P}_{m+1} = \left[\mathcal{P}_m, \vec{\mathcal{P}}_{m+1}\right]$. Moreover, the tensor $\mathcal{B}_{m,m+1}$ is defined by

$$\mathcal{B}_{m,m+1} = \begin{bmatrix} \mathcal{B}_m & \\ & \boldsymbol{\beta}_{m+1} \end{bmatrix} \in \mathbb{K}_n^{m\times(m+1)}.$$

The following plots illustrate the finite-precision behavior of the identities (4.2) and (4.3). Let $\mathcal{A} \in \mathbb{K}_n^{n\times n}$, and let m denote the number of steps of Algorithm 3. Figure 4.1 displays the Frobenius norm of the residual

$$\|\mathcal{A} * \mathcal{P}_m - \mathcal{Q}_m * \mathcal{B}_m\|_F$$

for various tensor dimensions n and Lanczos bidiagonalization steps m. The surface appears visually flat because the error remains numerically close to machine precision for all tested values. This indicates that equation (4.2) holds with high accuracy in finite-precision arithmetic. The flatness of the surface suggests backward stability and internal consistency of the partial tensor Krylov decompositions (4.2) computed by Algorithm 3.

Figure 4.2 depicts the Frobenius norm of the residual in equation (4.3), namely

$$\|\mathcal{A}^H * \mathcal{Q}_m - \mathcal{P}_m * \mathcal{B}_m^H + \vec{\mathcal{R}}_m * \vec{\mathcal{E}}_m^H\|_F.$$

In contrast to Figure 4.1, this plot reveals more variation of the error, in particular for small values of m.

While the norms shown in Figures 4.1 and 4.2 are small, we stress that these plots do not imply that the tensors $\mathcal{B}_m$, $\mathcal{P}_m$, and $\mathcal{Q}_m$ are computed with high accuracy. Rather, they validate that the relations among these tensors are satisfied to a high degree of precision.

Finally, Figure 4.3 displays the execution time in seconds required by Algorithm 3 as a function of the tensor dimension n and the number of Lanczos bidiagonalization steps m. The execution time increases with both parameters, as can be expected. The sharp rise in the surface along both axes reflects the cubic complexity in the tensor dimension n and the linear complexity with respect to the number of steps m. This is consistent with the cost of repeated tensor-matrix multiplications and orthogonalizations carried out by the algorithm.

Notably, the execution time remains under 4 seconds also for the largest tested configuration (n = 300, m = 5). This indicates that the method is computationally feasible for moderately sized problems and illustrates the usefulness of partial tensor Lanczos bidiagonalization under the t-product for real-world applications. As with the previous figures, the flat region near the lower-left corner of Figure 4.3 corresponds to small problem sizes for which the computational cost is negligible.

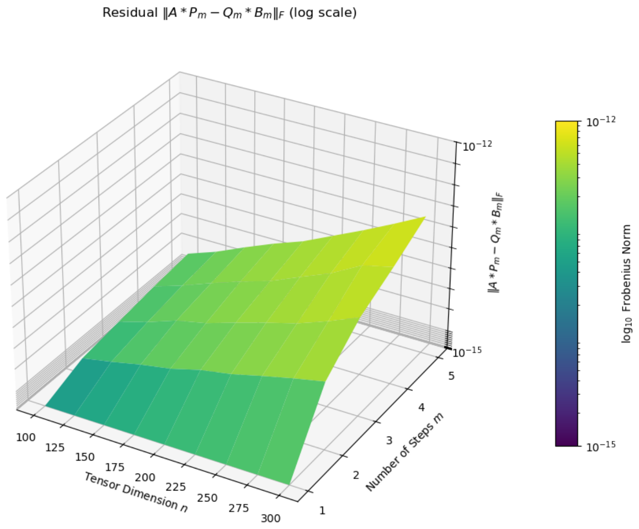

FIG. 4.1. *Frobenius norm of the difference of the computed right-hand side and left-hand side of equation (4.2) for tensors $\mathcal{A}$ of different sizes n and a few steps m with Algorithm 3.*

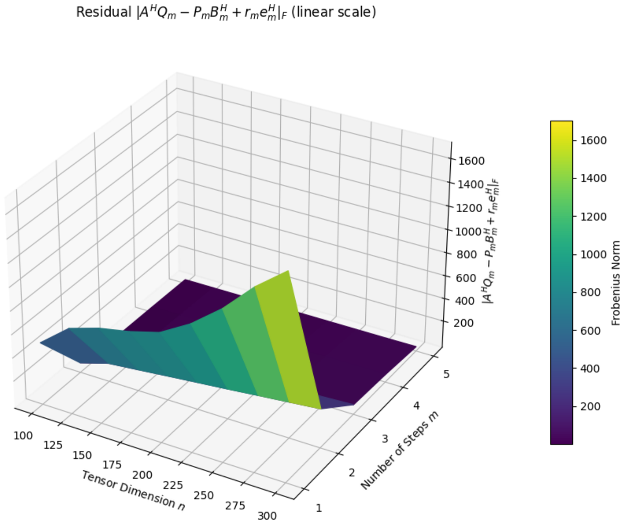

FIG. 4.2. *Frobenius norm of the difference of the computed right-hand side and left-hand side of equation (4.3) for tensors $\mathcal{A}$ of different sizes n and a few steps m with Algorithm 3.*

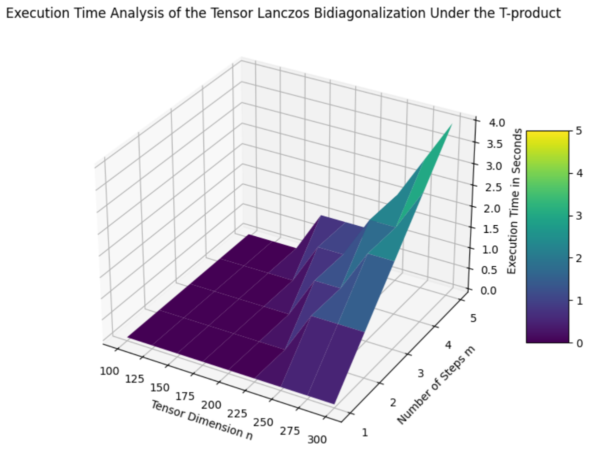

FIG. 4.3. *Execution time (in seconds) of Algorithm 3 for various values of the tensor size n and number of steps m.*

5. Approximation of a few singular triplets. Let $\ell \geq p$, and consider a large third-order tensor $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$. We approximate the first k singular triplets $\left\{ \boldsymbol{s}_{i,m}^{\mathcal{A}}, \vec{\mathcal{U}}_{i,m}^{\mathcal{A}}, \vec{\mathcal{V}}_{i,m}^{\mathcal{A}} \right\}_{i=1}^{k}$ of $\mathcal{A}$ by

$$\boldsymbol{s}_{i,m}^{\mathcal{A}} = \boldsymbol{s}_i, \qquad \vec{\mathcal{U}}_{i,m}^{\mathcal{A}} = \mathcal{Q}_m * \vec{\mathcal{U}}_i, \qquad \vec{\mathcal{V}}_{i,m}^{\mathcal{A}} = \mathcal{P}_m * \vec{\mathcal{V}}_i, \tag{5.1}$$

where $\left\{ \boldsymbol{s}_i, \vec{\mathcal{U}}_i, \vec{\mathcal{V}}_i \right\}_{i=1}^{m}$ are the singular triplets of $\mathcal{B}_m$. They satisfy the relations

$$\mathcal{B}_m * \vec{\mathcal{V}}_i = \vec{\mathcal{U}}_i * \boldsymbol{s}_i, \qquad \mathcal{B}_m^H * \vec{\mathcal{U}}_i = \vec{\mathcal{V}}_i * \boldsymbol{s}_i.$$

It is shown in [5] that the approximate singular triplets satisfy

$$\mathcal{A} * \vec{\mathcal{V}}_{i,m}^{\mathcal{A}} = \vec{\mathcal{U}}_{i,m}^{\mathcal{A}} * \boldsymbol{s}_{i,m}^{\mathcal{A}}, \qquad \mathcal{A}^H * \vec{\mathcal{U}}_{i,m}^{\mathcal{A}} = \vec{\mathcal{V}}_{i,m}^{\mathcal{A}} * \boldsymbol{s}_{i,m}^{\mathcal{A}} + \boldsymbol{\beta}_m * \vec{\mathcal{P}}_{m+1} * \vec{\mathcal{E}}_m^H * \vec{\mathcal{U}}_i.$$

It is desirable that the residual term $\boldsymbol{\beta}_m * \vec{\mathcal{P}}_{m+1} * \vec{\mathcal{E}}_m^H * \vec{\mathcal{U}}_i$ has small Frobenius norm. This requirement suggests that the inequalities

$$\left\| \boldsymbol{\beta}_m * \vec{\mathcal{P}}_{m+1} * \vec{\mathcal{E}}_m^H * \vec{\mathcal{U}}_i \right\|_F \leq \varepsilon \left(\boldsymbol{s}_{1,m}^{\mathcal{A}} \right)^{(1)}, \quad i = 1, \ldots, k,$$

be satisfied for some small value of $\varepsilon > 0$. Here $\left(\boldsymbol{s}_{1,m}^{\mathcal{A}} \right)^{(1)}$ denotes the first entry of the vector $\boldsymbol{s}_{1,m}^{\mathcal{A}}$. If these inequalities do not hold for a user-specified value of ε, then techniques outlined in the following sections can be used to improve the approximations of the singular triplets. Further details can be found in [5, 6].

5.1. Augmentation by Ritz lateral slices. Consider a large third-order tensor $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$ with $\ell \geq p$. If m steps of Algorithm 3 are executed and the first $k < n$ singular triplets of $\mathcal{A}$ are to be approximated, then we apply a scheme that uses Ritz lateral slices.

DEFINITION 5.1. *Let $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$, and let equations (4.2) and (4.3) hold for some $m > 0$. Define the ith lateral Ritz slice corresponding to $\mathcal{A}^H * \mathcal{A}$ as the ith approximated right lateral slice $\vec{\mathcal{V}}_{i,m}^{\mathcal{A}}$, as specified in equation (5.1) associated with the Ritz tube $(\boldsymbol{s}_{i,m}^{\mathcal{A}})^2$.*

The approach of the above definition is used when tensor Lanczos bidiagonalization alone does not yield a sufficiently accurate approximation. This approach constructs modified equations that are similar to (4.2) and (4.3) and replaces the first k lateral slices of the right basis of the tensor Krylov subspace with the first k lateral Ritz slices.

THEOREM 5.2 ([5]). *Apply m steps of Algorithm 3 to $\mathcal{A}$. If $\boldsymbol{\beta}_m$ in (4.4) is non-zero for $k < m$, then the following relations hold:*

$$\mathcal{A} * \widetilde{\mathcal{P}}_{k+1} = \widetilde{\mathcal{Q}}_{k+1} * \widetilde{\mathcal{B}}_{k+1}, \tag{5.2}$$

$$\mathcal{A}^H * \widetilde{\mathcal{Q}}_{k+1} = \widetilde{\mathcal{P}}_{k+1} * \widetilde{\mathcal{B}}_{k+1}^H + \widetilde{\boldsymbol{\beta}}_{k+1} * \vec{\widetilde{\mathcal{P}}}_{k+2} * \vec{\mathcal{E}}_{k+1}^H, \tag{5.3}$$

where $\widetilde{\mathcal{P}}_{k+1}$ and $\widetilde{\mathcal{Q}}_{k+1}$ are tensors in $\mathbb{K}_n^{p \times (k+1)}$ and $\mathbb{K}_n^{\ell \times (k+1)}$, respectively, with orthonormal lateral slices, and $\widetilde{\mathcal{B}}_{k+1}$ is an upper triangular tensor in $\mathbb{K}_n^{(k+1) \times (k+1)}$.

For further details on the construction of tensors and lateral slices in equations (5.2) and (5.3), we refer to [5]. The efficacy of this method hinges on that $\widetilde{\boldsymbol{\beta}}_{k+1}$ is non-vanishing.

5.2. Augmentation by harmonic Ritz lateral slices. Augmentation by harmonic Ritz lateral slices is carried out when approximating the k last singular triplets of a third-order tensor $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$ with $\ell \geq p$; see, e.g., the numerical examples in [5].

DEFINITION 5.3 ([5]). *Let $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$. Assume that equations (4.2) and (4.3) hold for a tensor Lanczos bidiagonalization determined by m steps of Algorithm 3. Let $1 \leq j < m$. If the pair $(\breve{\theta}_j, \vec{\breve{\omega}}_j) \in \mathbb{K}_n \times \mathbb{K}_n^m$ solves the equation*

$$\left((\mathcal{B}_m^H * \mathcal{B}_m)^2 + \boldsymbol{\alpha}_m^2 * \boldsymbol{\beta}_m^2 * \vec{\mathcal{E}}_m * \vec{\mathcal{E}}_m^H \right) * \vec{\breve{\omega}}_j = \breve{\theta}_j * \mathcal{B}_m^H * \mathcal{B}_m * \vec{\breve{\omega}}_j,$$

then the pair $(\breve{\theta}_j, \mathcal{P}_m * \vec{\breve{\omega}}_j)$ is said to be the jth harmonic Ritz pair associated with $\mathcal{A}$.

The above definition yields new tensor relations that are analogous to (4.2) and (4.3) using harmonic Ritz lateral slices obtained from $\mathcal{A}^H * \mathcal{A}$.

THEOREM 5.4 ([5]). *Apply m steps of Algorithm 3 to $\mathcal{A}$ and assume that the tensor $\mathcal{B}_m$ so determined is invertible. Then for $k = 1, \ldots, m-1$, we have*

$$\mathcal{A} * \breve{\mathcal{P}}_{k+1} = \breve{\mathcal{Q}}_{k+1} * \breve{\mathcal{B}}_{k+1},$$
$$\mathcal{A}^H * \breve{\mathcal{Q}}_{k+1} = \breve{\mathcal{P}}_{k+1} * \breve{\mathcal{B}}_{k+1}^H + \breve{\boldsymbol{\beta}}_{k+1} * \vec{\breve{\mathcal{P}}}_{k+2} * \vec{\mathcal{E}}_{k+1}^H,$$

where $\breve{\mathcal{P}}_{k+1}$ and $\breve{\mathcal{Q}}_{k+1}$ are tensors with orthonormal lateral slices, and $\breve{\mathcal{B}}_{k+1}$ is an upper triangular tensor.

For details on the construction of the tensors in Theorem 5.4 and for proofs, we refer to [5]. This method to determine harmonic Ritz lateral slices performs well when the tensor $\mathcal{B}_m$ is not very ill-conditioned.

We use the tensor Lanczos bidiagonalization Ritz (t-lbr) algorithm to compute a few extreme singular triplets of a large tensor; see Algorithm 4. This algorithm determines Ritz or harmonic Ritz lateral slices for computing a few of the largest or smallest singular triplets of a tensor. The choice of augmentation by Ritz or harmonic Ritz lateral slices depends on the condition number, $\kappa(\mathcal{B}_m)$, of $\mathcal{B}_m$. Section 5.3 discusses and illustrates properties of the t-lbr algorithm.

Algorithm 4 Partial tensor Lanczos bidiagonalization Ritz (t-lbr) algorithm for computing a few singular triplets.

Input: $\mathcal{A} \in \mathbb{K}_n^{\ell \times p}$, number of Lanczos steps m, initial unit vector $\vec{\mathcal{P}}_1 \in \mathbb{K}_n^p$, number of desired singular triplets k, tolerance δ, machine epsilon ϵ, augmentation type ('Ritz' or 'Harm').
Output: The k desired singular triplets of $\mathcal{A}$: $\{(\sigma_i, \vec{\mathcal{U}}_i, \vec{\mathcal{V}}_i)\}_{i=1}^k$.

```
 1: Compute partial Lanczos bidiagonalization of $\mathcal{A}$ by Algorithm 3.
 2: Compute the t-svd of $\mathcal{B}_m$.
 3: Check convergence of the $k$ desired singular triplets. If all triplets are within tolerance $\delta$,
    then terminate the computations.
 4: if type is 'Ritz' or $\kappa(\mathcal{B}_m) > \epsilon^{-1/2}$ then
 5:     Compute Ritz augmentation using $\mathcal{B}_m$.
 6: else if type is 'Harm' and $\kappa(\mathcal{B}_m) \leq \epsilon^{-1/2}$ then
 7:     Compute harmonic Ritz augmentation using $\mathcal{B}_{m,m+1}$.
 8: end if
 9: Append additional columns and rows to tensors to expand them to full size.
10: Repeat from step 2 if necessary.
```

5.3. Numerical examples of Ritz augmentation.

This section presents numerical experiments that illustrate the performance of Algorithm 4 using Ritz augmentation to approximate selected singular triplets of third-order tensors.

5.3.1. Approximating the first four singular triplets.

We first illustrate the use of Ritz augmentation for approximating the four largest singular triplets of third-order tensors $\mathcal{A} \in \mathbb{K}^{n \times 100 \times 3}$. The tensor dimension n varies from 250 to 1750, and the number of Lanczos steps m ranges from 7 to 15. For each pair (n, m), Algorithm 4 is executed 20 times with a convergence tolerance of 10^{-6}.

Figure 5.1 illustrates the Frobenius norm of the relative error of the computed approximate singular tubes. The color gradient encodes the magnitude of the error, with cooler colors indicating higher accuracy. The results show that increasing the number of Lanczos steps m improves the quality of the approximation. This improvement, however, comes at the cost of increased computational effort.

As shown in Figure 5.1, the error tends to stabilize for $n \geq 1000$ and larger values of m, confirming the effectiveness of the augmentation strategy in capturing dominant spectral components of large-scale tensors. These observations illustrate the capability of Algorithm 4 to accurately approximate the leading singular triplets by carrying out a modest number of Lanczos bidiagonalization steps.

5.3.2. Convergence behavior of the t-lbr algorithm.

We analyze the convergence characteristics of the t-lbr algorithm for a variety of tensor dimensions and numbers of Lanczos bidiagonalization steps. Our aim is to understand how the number of iterations of the algorithm varies with these parameters when approximating the four largest singular triplets of tensors of size $(n \times 100 \times 3)$.

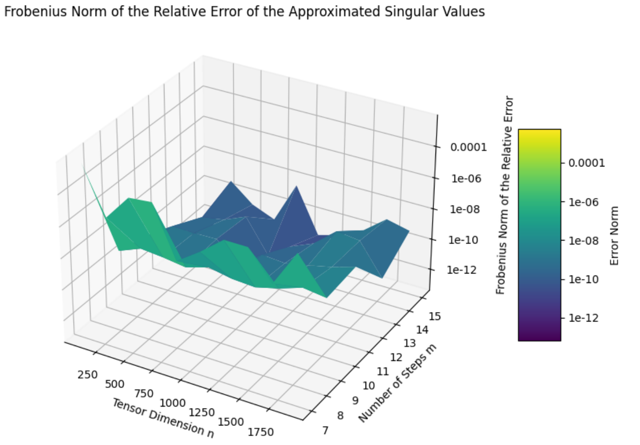

FIG. 5.1. *Frobenius norm of the relative error of computed approximate singular tubes for a variety of tensor dimensions n and Lanczos step numbers m.*

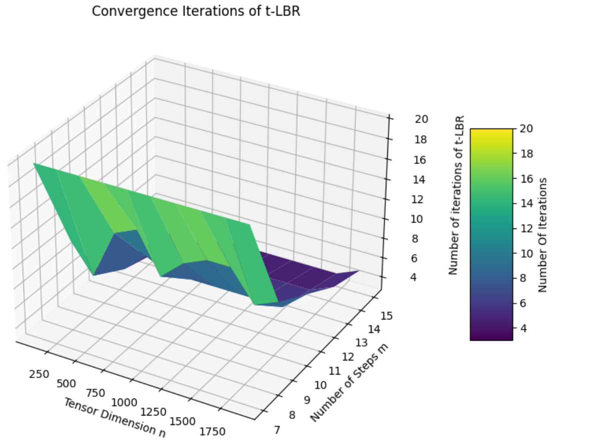

FIG. 5.2. *Number of t-lbr iterations required to satisfy the stopping criterion for various tensor sizes and Lanczos step counts.*

The convergence tolerance is set to 10^{-6}, and the algorithm is allowed to iterate until this criterion is satisfied. For each pair (n, m), we record the number of iterations required to satisfy the stopping criterion. Figure 5.2 displays the number of iterations required to satisfy the stopping criterion. As expected, the number of iterations increases with the tensor dimension n. This trend reflects the increased effort required to capture spectral information of larger tensors.

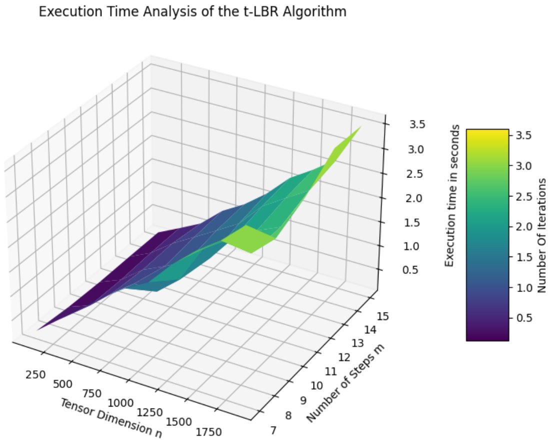

FIG. 5.3. *Execution time required by the t-lbr algorithm as a function of the tensor dimension n and the number of Lanczos steps m.*

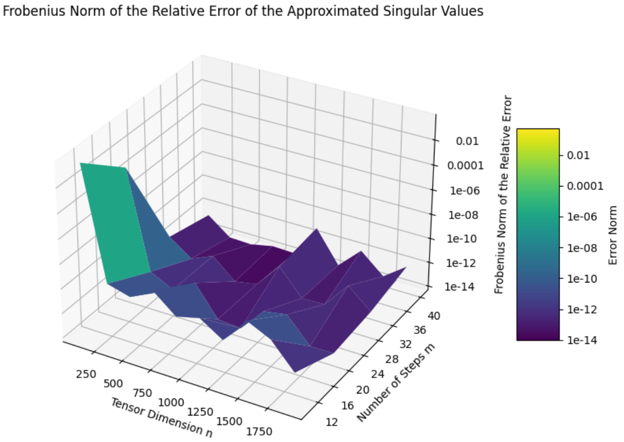

FIG. 5.4. *Frobenius norm of the relative error in approximating the last four singular triplets of tensors of size* $(n \times 100 \times 3)$. *Accuracy improves with increased Lanczos steps, although convergence is slower than for dominant triplets.*

5.3.3. Execution time of the t-lbr algorithm. To assess the computational cost, we measure the execution time (in seconds) of the t-lbr algorithm for the same tensor configurations as above. Each run targets the approximation of the four largest singular triplets with a convergence tolerance of 10^{-6} and a maximum of 20 Lanczos bidiagonalization steps.

Figure 5.3 displays the execution time to increase steadily with both the tensor dimension n and the number of Lanczos bidiagonalization steps m. This behavior aligns with the algorithmic complexity, which is dominated by repeated tensor-matrix product evaluations and orthogonalization steps.

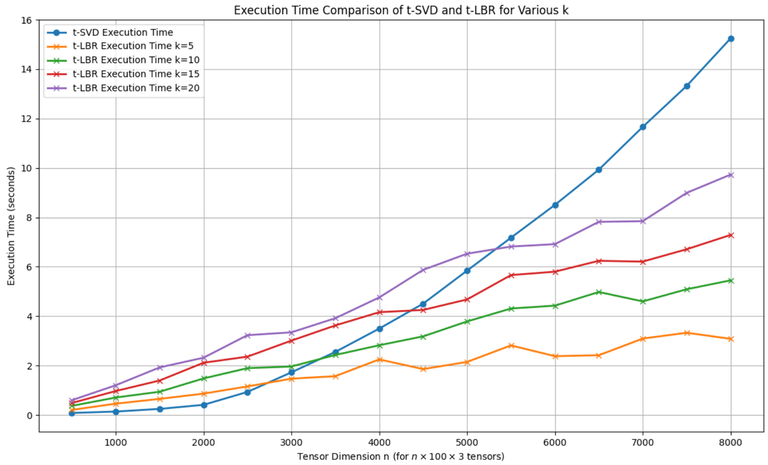

FIG. 6.1. *Execution time comparison of the t-svd and t-lbr algorithms for various tensor sizes n, and different numbers of singular triplets k.*

5.3.4. Approximating the last four singular triplets. Finally, we examine the accuracy of the t-lbr algorithm when computing the last four singular triplets of third-order tensors $\mathcal{A} \in \mathbb{K}^{n\times 100\times 3}$. In this experiment, the tensor dimension n again varies from 250 to 1750. The algorithm is allowed up to 40 Lanczos steps with a convergence tolerance of 10^{-6}.

Figure 5.4 displays the Frobenius norm of the relative error in the computed singular triplets. Compared to the approximation of the first four singular triplets, we can see that it is significantly more demanding to compute accurate approximations of the smallest singular triplets. The figure shows that more Lanczos bidiagonalization steps are required to achieve comparable accuracy, especially for large tensors. These results illustrate that it can be difficult to capture the smallest singular triplets. This is also true for matrices when the smallest singular values are close.

6. Execution time comparison of t-svd and t-lbr.

This section compares the computational performance of the t-svd and t-lbr algorithms for third-order tensors of size $\mathcal{A} \in \mathbb{R}^{n\times 100\times 3}$. We would like to illustrate how the runtime of each method scales with the tensor dimension n, and under what conditions the Lanczos bidiagonalization-based method is more efficient than computing the full t-svd.

Execution times are measured for tensor sizes with $n \in \{500, 1000, \dots, 8000\}$. For each tensor size, we evaluate the performance of the t-lbr algorithm when computing k singular triplets with $k \in \{5, 10, 15, 20\}$ and compare it to the runtime required to evaluate the full t-svd decomposition of the same tensor.

Figure 6.1 displays the runtime of t-svd to grow rapidly with increasing n, as the method computes all singular triplets of the tensor. In contrast, the t-lbr algorithm requires significantly less runtime, especially when only a small number of singular triplets are requested. We can observe the following key behaviors:

- For small n, t-svd can be competitive due to the efficient underlying implementation of the Python function `svd`.
- For moderate to large values of n, t-lbr outperforms t-svd consistently when k is small (e.g., $k = 5$ or $k = 10$).

- The crossover point when t-lbr becomes faster than t-svd depends on the value of k. For smaller k, the benefit of using t-lbr manifests itself at smaller n.
- As k increases, the gap between the performance of t-svd and t-lbr narrows, and for very large k, the advantage of t-lbr is reduced.

Our timings show that for large-scale tensor problems for which only a few leading singular triplets are needed, t-lbr is more efficient than t-svd. This situation is common in applications such as low-rank tensor approximation, data reduction, and signal compression, where computing the full decomposition is both unnecessary and computationally prohibitive.

7. Conclusion.

This paper discusses the performance of efficient algorithms for computing a few of the largest or smallest singular triplets of large third-order tensors under the t-product framework. By extending matrix-based bidiagonalization techniques, such as Ritz and harmonic Ritz augmentation, to the tensor setting, competitive numerical methods for computing a few extremal singular triplets are obtained.

The tensor Lanczos bidiagonalization algorithm (t-lbr) leverages the structure of the t-product and exploits the tensor Krylov subspace to build low-rank approximations of third-order tensors. We demonstrate the accuracy and efficiency of this approach through a detailed numerical experiments. Future work includes adaptation of these methods considered in this paper to more general tensor products and exploration of preconditioning strategies.

8. Software.

This section describes the software that accompanies the paper[1]. The software provided is a Python package. This package is structured into three levels, each containing modules and functions related to tensor operations. Below is a detailed organization of the functions by modules:

8.1. Level 1: Basic tensor operations.

8.1.1. `tensor_basic_operations` Module.

This module provides basic tensor operations:

- `tensor_transpose(tensor)`: Transpose the first two dimensions of a tensor and apply complex conjugate.
- `normalize_tensor(tensor)`: Normalize a tensor using FFT and inverse FFT.
- `tensor_norm(tensor)`: Compute the norm of a tensor.
- `diagonal_tensor(slice_tensor)`: Create a diagonal tensor from a given lateral slice.
- `upper_triangular_tensor(tensor)`: Convert each frontal slice of a tensor to its upper triangular form.

8.1.2. `tensor_generation` module.

This module focuses on generating specific types of tensors:

- `generate_canonical_slice(n_dim, p_dim, tube_index)`: Generate a canonical lateral slice tensor.
- `generate_identity_tensor(shape)`: Generate an identity tensor with the specified shape.
- `generate_one_tube(length)`: Generate a one-tube tensor where all elements are equal to one.

[1] Available at https://etna.ricam.oeaw.ac.at/volumes/2021-2030/vol63/addition/p300.php.

8.2. Level 2: Operations with two tensors.

8.2.1. `tensor_pair_operations` module. This module deals with operations involving two tensors:

- `t_product(tensor1, tensor2, fft_tensor1=True, fft_tensor2=True)`: Perform the tensor-tensor product using FFT.
- `t_qr(tensor)`: Perform the QR decomposition of a tensor using FFT.
- `t_svd(tensor)`: Perform the singular value decomposition (SVD) of a tensor using FFT.
- `t_orthogonalize(target_tensor, basis_tensor)`: Orthogonalize a tensor with respect to another tensor.
- `t_elementwise_left_division(tensor1, tensor2)`: Perform element-wise left division of tensors using a least-squares solution.

8.3. Level 3: Advanced tensor algorithms.

8.3.1. `tensor_algorithms` module. This module contains advanced tensor algorithms:

- `lanczos_tensor(A, m_b, iteration, k, j, a_fft=None, a_trans_fft=None, v=None, w=None, b=None, f=None)`: Perform Lanczos bidiagonalization on a tensor.
- `t_lbr(A, k, tol=1e-07, maxit=1000, m_b=None)`: Perform Tensor Lanczos Bidiagonalization Ritz (t-LBR) Algorithm for computing singular triplets.
- `t_lbr_last(A, k, tol=0.0001, maxit=1000, augmentation_type='ritz', m_b=None)`: Perform Tensor Lanczos Bidiagonalization with optional Ritz or Harmonic Ritz augmentation for the last singular triplets.

8.4. Function description. This section provides a detailed technical description of all functions available in the library. Each function is described in Table 8.1 outlining its parameters, including the field names, descriptions, and the accepted values.

TABLE 8.1
Descriptions and accepted values for the fields of the functions in the library.

Function	Field	Description of the Field	Accepted Values
`tensor_transpose`	`tensor`	Input tensor to be transposed.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
`normalize_tensor`	`tensor`	Input tensor to be normalized.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
`tensor_norm`	`tensor`	Input tensor for norm calculation.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
`diagonal_tensor`	`tensor`	A lateral slice tensor to be used for diagonalization.	$\mathbb{C}^{n \times 1 \times n}$
`upper_triangular_tensor`	`tensor`	Input tensor to be converted to upper triangular form.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$

TABLE 8.1 (CONTINUED)
Descriptions and accepted values for the fields of the functions in the library.

Function	**Field**	**Description of the Field**	**Accepted Values**
`generate_identity_tensor`	`shape`	The shape of the tensor to be generated.	$(n_1, n_2, n_3) \in \mathbb{N}^3$
`generate_one_tube`	`length`	The length of the third dimension of the tensor.	$\ell \in \mathbb{N}$
`generate_canonical_slice`	`n_dim`	The first dimension of the tensor.	$n \in \mathbb{N}$
	`p_dim`	The third dimension of the tensor.	$p \in \mathbb{N}$
	`tube_index`	Index of the tube where the first element is set to 1.	$i \in \{0, 1, \ldots, n-1\}$
`t_product`	`tensor1`	The first input tensor.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	`tensor2`	The second input tensor.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	`fft_tensor1`	Flag to determine whether to apply FFT to the first tensor.	`True`, `False` (default: `True`)
	`fft_tensor2`	Flag to determine whether to apply FFT to the second tensor.	`True`, `False` (default: `True`)
`t_qr`	`tensor`	The input tensor for QR decomposition.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
`t_svd`	`tensor`	The input tensor for singular value decomposition (SVD).	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
`t_orthogonalize`	`target_tensor`	The tensor to be orthogonalized.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	`basis_tensor`	The basis tensor for orthogonalization.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$

TABLE 8.1 (CONTINUED)
Descriptions and accepted values for the fields of the functions in the library.

Function	Field	Description of the Field	Accepted Values
`t_elementwise_left_division`	`tensor1`	The numerator tensor (input for left division).	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	`tensor2`	The denominator tensor (input for left division).	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
`lanczos_tensor`	`A`	Input tensor for Lanczos bidiagonalization.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	`m_b`	Maximum number of bidiagonalization steps.	$m \in \mathbb{N}$
	`iteration`	Current iteration number.	$i \in \mathbb{N}$
	`k`	Starting column index.	$k \in \mathbb{N}$
	`j`	Current column index.	$j \in \mathbb{N}$
	`a_fft`	FFT of the input tensor A (optional).	$\mathbb{C}^{n_1 \times n_2 \times n_3}$ (default: `None`)
	`a_trans_fft`	FFT of the transpose of A (optional).	$\mathbb{C}^{n_2 \times n_1 \times n_3}$ (default: `None`)
	`v`	Tensor V in the bidiagonalization (optional).	$\mathbb{C}^{n_2 \times m_b \times n_3}$ (default: `None`)
	`w`	Tensor W in the bidiagonalization (optional).	$\mathbb{C}^{n_1 \times m_b \times n_3}$ (default: `None`)
	`b`	Bidiagonal tensor B (optional).	$\mathbb{C}^{m_b \times m_b \times n_3}$ (default: `None`)
`t_lbr`	`A`	Input tensor.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	`k`	Number of desired singular triplets.	$0 \leq k \leq n_1, k \in \mathbb{N}$
	`tol`	Tolerance for convergence.	$tol \in \mathbb{R}$ (default: $1e^{-07}$)
	`maxit`	Maximum number of iterations.	$maxit \in \mathbb{N}$ (default: 1000)
	`m_b`	Number of Lanczos steps (optional).	$m \in \mathbb{N}$ (default: `None`)

TABLE 8.1 (CONTINUED)
Descriptions and accepted values for the fields of the functions in the library.

Function	**Field**	**Description of the Field**	**Accepted Values**
t_lbr_last	A	Input tensor.	$\mathbb{C}^{n_1 \times n_2 \times n_3}$
	k	Number of desired singular triplets.	$0 \leq k \leq n_1, k \in \mathbb{N}$
	tol	Tolerance for convergence.	$tol \in \mathbb{R}$ (default: 0.0001)
	maxit	Maximum number of iterations.	$maxit \in \mathbb{N}$ (default: 1000)
	augmentation_type	Type of augmentation (‘ritz’ or ‘harm’).	'ritz','harm' (default: 'ritz')
	m_b	Number of Lanczos steps (optional).	$m \in \mathbb{N}$ (default: None)

Supplementary material. The accompanying software is available at https://etna.ricam.oeaw.ac.at/volumes/2021-2030/vol63/addition/p300.php in form of a compressed file entitled quicksvd.zip. Installation details are discussed in the file README.md

REFERENCES

[1] J. Baglama and L. Reichel, *Augmented implicitly restarted Lanczos bidiagonalization methods*, SIAM J. Sci. Comput., 27 (2005), pp. 19–42.

[2] J. Baglama, L. Reichel, and B. W. Lewis, *irbla: Fast truncated singular value component analysis,* Software Package. https://cran.r-project.org/web/packages/irlba/index.html

[3] A. H. Bentbib, A. E. Hachimi, K. Jbilou, and A. Ratnani, *A tensor regularized nuclear norm method for image and video completion*, J. Optim. Theory Appl., 192 (2022), pp. 401–425.

[4] A. El Hachimi, K. Jbilou, A. Ratnani, and L. Reichel, *Spectral computation with third-order tensors using the t-product*, Appl. Numer. Math., 193 (2023), pp. 1–21.

[5] ———, *A tensor bidiagonalization method for higher-order singular value decomposition with applications*, Numer. Linear Algebra Appl., 31 (2024), Art. e2530, 31 pages.

[6] M. Hached, K. Jbilou, C. Koukouvinos, and M. Mitrouli, *A multidimensional principal component analysis via the c-product Golub–Kahan-SVD for classification and face recognition*, Mathematics, 9 (2021), Art. 1249, 17 pages.

[7] M. E. Kilmer, K. Braman, N. Hao, and R. C. Hoover, *Third-order tensors as operators on matrices: a theoretical and computational framework with applications in imaging*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 148–172.

[8] M. E. Kilmer and C. D. Martin, *Factorization strategies for third-order tensors*, Linear Algebra Appl., 435 (2011), pp. 641–658.

[9] T. G. Kolda and B. W. Bader, *Tensor decompositions and applications*, SIAM Rev., 51 (2009), pp. 455–500.

[10] C. Lu, J. Feng, Y. Chen, W. Liu, Z. Lin, and S. Yan, *Tensor robust principal component analysis with a new tensor nuclear norm*, IEEE Trans. Pattern Anal. Mach. Intell., 42 (2019), pp. 925–938.

[11] L. Reichel and U. O. Ugwu, *Weighted tensor Golub–Kahan–Tikhonov-type methods applied to image processing using a t-product*, J. Comput. Appl. Math., 415 (2022), Art. 114488, 21 pages.

[12] Z. Zhang, G. Ely, S. Aeron, N. Hao, and M. Kilmer, *Novel methods for multilinear data completion and de-noising based on tensor-SVD*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition 2014, IEEE Conference Proceedings, Los Alamitos, 2014, pp. 3842–3849.