Structure exploiting Lanczos method for Hadamard product of low-rank matrices

Computational Linear Algebra project

Name: Niccolò Discacciati
Sciper: 273520

Date: June 5, 2019

Instructors: Stefano Massei, Lana Periša

1 Introduction

Given two matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$, their Hadamard product $\mathbf{C} = \mathbf{A} \star \mathbf{B}$ is defined as the entry-wise product $c_{ij} = a_{ij}b_{ij}$ for $i = 1, \dots, m = I_1$ and $j = 1, \dots, n = I_2$. It constitutes a fundamental building block of several algorithms in scientific computing and data analysis, partly because Hadamard products of matrices correspond to products of bi-variate functions. Consider a discretization of the rectangle $\Omega = [a_1, b_1] \times [a_2, b_2]$ made by points $a_k \le \xi_{k,1} < \cdots < \xi_{k,I_k} \le b_k$ with $k \in \{1, 2\}$ and two functions $u, v : \Omega \to \mathbb{R}$. Assuming that the matrices $\mathbf{A}$ and $\mathbf{B}$ contain the evaluation of u, v on the grid points, then $\mathbf{C}$ contains the evaluation of the product uv on the same points. It is known that the degree of smoothness of the underlying function is related to the low-rank approximation properties of the matrix [1]. In particular, if u, v are sufficiently smooth, the corresponding matrices admit a meaningful low-rank approximation. The Hadamard product can also be efficiently compressed, due to regularity of uv. However, on the algebraic side, the element-wise product of low-rank matrices generally increases ranks, impacting negatively on the approximation properties. Despite this unfavorable aspect, exploiting the structure of the entry-wise product allows us to create fast matrix-vector multiplications, which turn out to be helpful in designing computationally efficient algorithms to construct low-rank approximations of the Hadamard product.
In this project we want to investigate the low-rank compression properties of the entry-wise product, aiming to construct a powerful technique to compress the resulting matrix. Particular emphasis is given to an analysis of the computational performance of the method. Note that most of the methods we present can be extended to dimensions greater than two, where matrices are suitably replaced by tensors. See [2] for the details.

2 Methods

2.1 Definitions and properties

In this report, matrices are denoted with boldface capital letters, while boldface lowercase letters are used for vectors, unless stated otherwise. Given a matrix $\mathbf{A}$, its columns are denoted with boldface lowercase letters, i.e. $\mathbf{A} = (\mathbf{a}_1 \cdots \mathbf{a}_n)$, while for single entries we use $\mathbf{A}_{ij}$ or a_{ij}[1].
Let $\mathrm{vec} : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ be the standard vectorization of a matrix, obtained from stacking its columns, i.e.

$$(\mathrm{vec}\,\mathbf{A})_i = a_{(i-1)\%m+1,\lfloor (i-1)/m\rfloor+1}, \quad \text{or} \quad \mathrm{vec}\,\mathbf{A} = (\mathbf{a}_1^T \ \cdots \ \mathbf{a}_n^T)^T.$$

Moreover, for a square matrix $\mathbf{A}$, $\mathrm{diag}(\mathbf{A})$ denotes the vector consisting of its diagonal elements, while for a vector $\mathbf{v}$, $\mathrm{diag}(\mathbf{v})$ denotes a diagonal matrix with the given vector on the diagonal, i.e.

$$(\mathrm{diag}(\mathbf{A}))_i = a_{ii},$$
$$(\mathrm{diag}(\mathbf{v}))_{ij} = v_i\delta_{ij}.$$

In this work, we use the following definitions and properties.

- *Hadamard product.* Given $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m\times n}$, the Hadamard or element-wise product is defined as

  $$(\mathbf{A} \star \mathbf{B})_{ij} = a_{ij}b_{ij},$$

  i.e.

  $$\mathbf{A} \star \mathbf{B} = \begin{pmatrix} a_{11}b_{11} & \cdots & a_{1n}b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}b_{m1} & \cdots & a_{mn}b_{mn} \end{pmatrix},$$

  and it is an element of $\mathbb{R}^{m\times n}$.

- *Kronecker product.* Given $\mathbf{A} \in \mathbb{R}^{m\times n}$, $\mathbf{B} \in \mathbb{R}^{p\times q}$, the Kronecker product is defined as

  $$(\mathbf{A} \otimes \mathbf{B})_{ij} = a_{\lfloor (i-1)/p\rfloor+1,\lfloor (j-1)/q\rfloor+1}b_{(i-1)\%p+1,(j-1)\%q+1},$$

[1] Even though the notation $\mathbf{A}_{ij}$ is more general, we often use a_{ij} to ease the readability. To denote the elements of a matrix resulting from an operation, we always use the first notation, e.g., $(\mathbf{A}\ op\ \mathbf{B})_{ij}$. A similar reasoning holds for vectors.

i.e.

$$\mathbf{A}\otimes\mathbf{B}=\begin{pmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{pmatrix},$$

and it is an element of $\mathbb{R}^{mp\times nq}$.

- *Khatri-Rao product.* Given $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\mathbf{B}\in\mathbb{R}^{p\times n}$, the Khatri-Rao product is defined as

$$(\mathbf{A}\odot\mathbf{B})_{ij}=(\mathbf{a}_j\otimes\mathbf{b}_j)_i=a_{\lfloor(i-1)/p\rfloor+1,j}b_{(i-1)\%p+1,j},$$

i.e.

$$\mathbf{A}\odot\mathbf{B}=\begin{pmatrix}\mathbf{a}_1\otimes\mathbf{b}_1 & \cdots & \mathbf{a}_n\otimes\mathbf{b}_n\end{pmatrix},$$

and it is an element of $\mathbb{R}^{mp\times n}$.

- *Transpose Khatri-Rao product.* Given $\mathbf{A}\in\mathbb{R}^{m\times n}$, $\mathbf{B}\in\mathbb{R}^{m\times q}$, the transpose Khatri-Rao product is defined as

$$\mathbf{A}\odot^T\mathbf{B}=(\mathbf{A}^T\odot\mathbf{B}^T)^T, \tag{2.1}$$

and it is an element of $\mathbb{R}^{m\times nq}$.

Proposition 1. *Let $\mathbf{A},\mathbf{B}$ two matrices and $\mathbf{v}$ a vector for which the required operations are well defined. Then the following properties hold:*

- *Property 1:*

$$(\mathbf{A}\otimes\mathbf{B})\mathbf{v}=\mathrm{vec}(\mathbf{B}\mathbf{V}\mathbf{A}^T),\ \textit{where}\ \mathbf{v}=\mathrm{vec}(\mathbf{V}). \tag{2.2}$$

- *Property 2:*

$$(\mathbf{A}\odot\mathbf{B})\mathbf{v}=\mathrm{vec}(\mathbf{B}\,\mathrm{diag}(\mathbf{v})\mathbf{A}^T). \tag{2.3}$$

- *Property 3:*

$$(\mathbf{A}\odot^T\mathbf{B})\mathbf{v}=\mathrm{diag}(\mathbf{B}\mathbf{V}\mathbf{A}^T),\ \textit{where}\ \mathbf{v}=\mathrm{vec}(\mathbf{V}). \tag{2.4}$$

- *Property 4:*

$$(\mathbf{A}\otimes\mathbf{B})^T=\mathbf{A}^T\otimes\mathbf{B}^T. \tag{2.5}$$

These properties can be verified by applying the definitions of the operators, and the proof is omitted for the sake of brevity.

2.2 A first result

Given two matrices with low-rank structures, we want to study how the compression properties are inherited by their element-wise product. The first step aims to construct a suitable representation of the Hadamard product, starting from two matrices in a low-rank revealing format. Hence, we let $\mathbf{A},\mathbf{B}\in\mathbb{R}^{m\times n}$ be two matrices with low-rank structures

$$\mathbf{A}=\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^T,\quad \mathbf{B}=\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^T, \tag{2.6}$$

with $\mathbf{U}_A\in\mathbb{R}^{m\times k_A}$, $\mathbf{\Sigma}_A\in\mathbb{R}^{k_A\times k_A}$, $\mathbf{V}_A\in\mathbb{R}^{n\times k_B}$, $\mathbf{U}_B\in\mathbb{R}^{m\times k_B}$, $\mathbf{\Sigma}_B\in\mathbb{R}^{k_B\times k_B}$, $\mathbf{V}_B\in\mathbb{R}^{n\times k_B}$, for some $k_A,k_B\ll\min\{m,n\}$. Such approximations can be computed in multiple ways, with a popular technique relying on the singular value decomposition (SVD) and

Theorem 1 (Eckart-Young [3])**.** *Let $\mathbf{A}\in\mathbb{R}^{m\times n}$ be given. Let its SVD be $\mathbf{A}=\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ with singular values $\{\sigma_i\}_{i=1}^{\min\{m,n\}}$ in descending order. Set $\mathbf{A}_j=\mathbf{U}_j\mathbf{\Sigma}_j\mathbf{V}_j$, where $\mathbf{\Sigma}_j=\mathrm{diag}(\{\sigma_1,\dots,\sigma_j\})$, and $\mathbf{U}_j,\mathbf{V}_j$ are formed by the first j columns of $\mathbf{U},\mathbf{V}$. Then*

$$\left(\sum_{i=j+1}^{\min\{m,n\}}\sigma_i^2\right)^{1/2}=\|\mathbf{A}-\mathbf{A}_j\|_F=\min\left\{\|\mathbf{A}-\mathbf{B}\|_F\,|\,\mathrm{rank}(\mathbf{B})\le j\right\}.$$

Thus, the SVD constitutes an important tool in compression theory. In particular, (2.6) can be computed from the SVD, truncating the smallest singular values and the corresponding left and right vectors.
The Hadamard product admits a meaningful representation, as stated in the following

Proposition 2. *Let* $\mathbf{A}, \mathbf{B}$ *be two matrices having the structure* (2.6)*. Then*

$$\mathbf{A} \star \mathbf{B} = \left(\mathbf{U}_A^T \odot \mathbf{U}_B^T\right)^T \left(\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B\right) \left(\mathbf{V}_A^T \odot \mathbf{V}_B^T\right), \tag{2.7}$$

or, equivalently,

$$\mathbf{A} \star \mathbf{B} = \left(\mathbf{U}_A \odot^T \mathbf{U}_B\right) \left(\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B\right) \left(\mathbf{V}_A \odot^T \mathbf{V}_B\right)^T. \tag{2.8}$$

Proof. The proof follows directly from definitions of the involved operators. The (i,j) component of the left-hand-side of (2.7) can be written as

$$a_{ij}b_{ij} = \sum_{r=1}^{k_A}\sum_{x=1}^{k_A} u_{ir}^A \sigma_{rx}^A v_{jx}^A \sum_{s=1}^{k_B}\sum_{y=1}^{k_B} u_{is}^B \sigma_{sy}^B v_{jy}^B = \sum_{r=1}^{k_A}\sum_{x=1}^{k_A}\sum_{s=1}^{k_B}\sum_{y=1}^{k_B} u_{ir}^A \sigma_{rx}^A v_{jx}^A u_{is}^B \sigma_{sy}^B v_{jy}^B. \tag{2.9}$$

The (i,j) component of the right-hand-side of (2.7) is

$$\begin{aligned} &\sum_{k=1}^{k_Ak_B}\sum_{l=1}^{k_Ak_B} \left(\mathbf{U}_A^T \odot \mathbf{U}_B^T\right)_{ik}^T \left(\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B\right)_{kl} \left(\mathbf{V}_A^T \odot \mathbf{V}_B^T\right)_{lj} \\ &= \sum_{k=1}^{k_Ak_B}\sum_{l=1}^{k_Ak_B} u_{i,\lfloor (k-1)/k_B\rfloor+1}^A u_{i,(k-1)\%k_B+1}^B \sigma_{\lfloor (k-1)/k_B\rfloor+1,\lfloor (l-1)/k_B\rfloor+1}^A \\ &\qquad \sigma_{(k-1)\%k_B+1,(l-1)\%k_B+1}^B v_{j,\lfloor (l-1)/k_B\rfloor+1}^A v_{j,(l-1)\%k_B+1}^B. \end{aligned} \tag{2.10}$$

Setting $k = (r-1)k_B + s$, $l = (x-1)k_B + y$, (2.10) simplifies to

$$\sum_{r=1}^{k_A}\sum_{s=1}^{k_B}\sum_{x=1}^{k_A}\sum_{y=1}^{k_B} u_{ir}^A u_{is}^B \sigma_{rx}^A \sigma_{sy}^B v_{jx}^A v_{jy}^B,$$

which is exactly the expression (2.9). The representation (2.8) follows from the definition of the transpose Khatri-Rao product (2.1). □

2.3 Fast matrix-vector multiplications

As shown in the following Sections, the representation (2.7) does not capture the low-rank properties of the Hadamard product. In (2.6), the ranks of $\mathbf{A}$ and $\mathbf{B}$ are captured by the matrices $\mathbf{\Sigma}_A$ and $\mathbf{\Sigma}_B$ respectively, and are at most k_A and k_B. According to (2.7), the rank of the Hadamard product is captured by $\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B$, and can be up to k_Ak_B. Thus, the ranks of $\mathbf{A}$ and $\mathbf{B}$ are (in general) multiplied, and the good compression properties of $\mathbf{A} \star \mathbf{B}$ can be lost. Hence, this *curse of dimensionality* should be avoided in order to efficiently compress the Hadamard product.
However, (2.7) is helpful to compute efficient multiplications between the Hadamard product $\mathbf{A} \star \mathbf{B}$ and a vector $\mathbf{v}$. Such a procedure, that avoids the explicit construction of the matrix $\mathbf{A} \star \mathbf{B}$, can be divided in three steps.

- *Step 1.* Compute $\mathbf{w}_1 = \left(\mathbf{V}_A^T \odot \mathbf{V}_B^T\right)\mathbf{v}$.
  From (2.3) we have that $\mathbf{w}_1 = \mathrm{vec}(\mathbf{V}_B^T \,\mathrm{diag}(\mathbf{v})\mathbf{V}_A)$. In this context it is enough to compute the matrix $\mathbf{W}_1$, obtained by reshaping $\mathbf{w}_1$ into a $k_B \times k_A$ matrix. Since

  $$\mathbf{w}_1 = \mathrm{vec}\,\mathbf{W}_1, \tag{2.11}$$

  we have

  $$\mathbf{W}_1 = \mathbf{V}_B^T \,\mathrm{diag}(\mathbf{v})\mathbf{V}_A.$$

  The computational cost of this step is $\mathcal{O}(nk_Ak_B)$, since it consists of the multiplication of the matrices $\mathbf{V}_B^T \in \mathbb{R}^{k_B \times n}$ and $\mathrm{diag}(\mathbf{v})\mathbf{V}_A \in \mathbb{R}^{n \times k_A}$. Note that the product of a diagonal and a full matrix can be computed efficiently at a cost that is negligible in this framework.

- *Step 2.* Compute $\mathbf{w}_2 = (\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B)\,\mathbf{w}_1$.
  Using (2.2) and (2.11), we have that
  $$\mathbf{w}_2 = \mathrm{vec}(\mathbf{\Sigma}_B \mathbf{W}_1 \mathbf{\Sigma}_A^T).$$
  Similarly to Step 1, we set
  $$\mathbf{w}_2 = \mathrm{vec}\,\mathbf{W}_2. \tag{2.12}$$
  so that
  $$\mathbf{W}_2 = \mathbf{\Sigma}_B^T \mathbf{W}_1 \mathbf{\Sigma}_A^T.$$
  Since two matrix-matrix multiplications are performed, the cost at most of order $\mathcal{O}(k_B k_A^2 + k_A k_B^2)$.

- *Step 3.* Compute $\mathbf{w}_3 = \left(\mathbf{U}_A^T \odot \mathbf{U}_B^T\right)^T \mathbf{w}_2 = \left(\mathbf{U}_A \odot^T \mathbf{U}_B\right) \mathbf{w}_2$.
  Using (2.4) and (2.12), we have that
  $$\mathbf{w}_3 = \mathrm{diag}(\mathbf{U}_B \mathbf{W}_2 \mathbf{U}_A^T).$$
  Since two matrix-vector multiplications are performed, the cost should scale quadratically with m. Exploiting the fact that only the diagonal elements are needed, the computational effort reduces to a multiplication between $\mathbf{W}_2 \in \mathbb{R}^{k_B \times k_A}$ and $\mathbf{U}_A^T \in \mathbb{R}^{k_A \times m}$ and it is of order $\mathcal{O}(k_A k_B m)$.

Hence, we are able to compute a matrix-vector multiplication in

$$\mathcal{C}_1 = \mathcal{O}(n k_A k_B + k_B k_A^2 + k_A k_B^2 + k_A k_B m),$$

in comparison with

$$\mathcal{C}_2 = \mathcal{O}(mn),$$

if the standard implementation is considered. If the low-rank approximations are efficiently performed, a clear speedup is present. Assuming that $m = n$ and $k_A = k_B = k$, the two costs reduce to $\mathcal{C}_1 = \mathcal{O}(nk^2 + k^3)$ and $\mathcal{C}_2 = \mathcal{O}(n^2)$ respectively, and our implementation is preferred over the standard one when $k^2 < n$, at least asymptotically.
Even though the procedure has been described to compute $(\mathbf{A} \star \mathbf{B})\mathbf{v}$, the same algorithm can be exploited to compute $(\mathbf{A} \star \mathbf{B})^T \mathbf{w}$. From (2.7) and (2.5), we have

$$(\mathbf{A} \star \mathbf{B})^T = \left(\mathbf{V}_A^T \odot \mathbf{V}_B^T\right)^T \left(\mathbf{\Sigma}_A^T \otimes \mathbf{\Sigma}_B^T\right) \left(\mathbf{U}_A^T \odot \mathbf{U}_B^T\right),$$

so that the previous reasoning can be applied with the following substitutions

$$\mathbf{U}_A \leftarrow \mathbf{V}_A,\ \mathbf{\Sigma}_A \leftarrow \mathbf{\Sigma}_A^T,\ \mathbf{V}_A \leftarrow \mathbf{U}_A, \quad \mathbf{U}_B \leftarrow \mathbf{V}_B,\ \mathbf{\Sigma}_B \leftarrow \mathbf{\Sigma}_B^T,\ \mathbf{V}_B \leftarrow \mathbf{U}_B, \quad \mathbf{v} \leftarrow \mathbf{w}.$$

2.4 Computing the SVD

Recalling Theorem 1, the SVD allows to construct a low-rank approximation that is optimal in the sense of the Frobenius norm. Hence, we want to be able to compute the SVD for the Hadamard product in an efficient way. Letting $\mathbf{A} \in \mathbb{R}^{m \times n}$, we can construct its SVD exploiting the eigendecomposition of $\mathbf{A}^T\mathbf{A}$ or $\mathbf{A}\mathbf{A}^T$. Specifically, if $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, we have that

$$\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{\Sigma}^T\mathbf{\Sigma}\mathbf{V}^T = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T,$$

so that the squared singular values (resp. right singular vectors) are the eigenvalues (resp. eigenvectors) of $\mathbf{A}^T\mathbf{A}$. The left singular vectors can be retrieved by

$$\mathbf{U} = \mathbf{A}\mathbf{V}\mathbf{\Sigma}^{-1}, \tag{2.13}$$

where $\mathbf{\Sigma}^{-1}$ contains on the main diagonal the reciprocal of the nonzero elements of $\mathbf{\Sigma}$ and has a suitable block of zeros if $m \neq n$, necessary to match the dimensions. Similarly,

$$\mathbf{A}\mathbf{A}^T = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^T = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T,$$

so that the squared singular values (resp. left singular vectors) are the eigenvalues (resp. eigenvectors) of $\mathbf{A}\mathbf{A}^T$. The right singular vectors can be retrieved by

$$\mathbf{V} = \mathbf{A}^T\mathbf{U}\mathbf{\Sigma}^{-T}. \tag{2.14}$$

If $n = m$ the two approaches are equivalent, while if $m > n$ the first approach should be preferred. Since $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{A}\mathbf{A}^T \in \mathbb{R}^{n\times n}$, in the first scenario the dimension of the resulting eigenvalue problem is smaller and, in general, computationally more efficient.
A powerful technique to compute eigenpairs is known as *Arnoldi method*. We look for approximations of eigenpairs $(\lambda, \mathbf{v})$ of a matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ in the Krylov subspace generated by $\mathbf{A}$ and a vector $\mathbf{u}_1$, denoted by $\mathcal{K}_k(\mathbf{A}, \mathbf{u}_1)$. We aim to construct a basis $\mathbf{U}_k$ of the subspace and a suitable Hessenberg matrix $\mathbf{H}_k$ via the *Arnoldi process*. Starting from an initial guess $\mathbf{u}_1$, we iteratively compute $\mathbf{w} = \mathbf{A}\mathbf{u}_j$, orthogonalize with respect to the first j vectors and normalize the resulting vector. Storing the projection coefficients $\mathbf{h}_j = \mathbf{U}_j^T\mathbf{w}$ and the norm $h_{j+1,j}$ of the projection $\mathbf{w} - \mathbf{U}_j\mathbf{h}_j$ we find the required matrices. The process is summarized by

$$\mathbf{A}\mathbf{U}_k = \mathbf{U}_k\mathbf{H}_k + h_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^T.$$

By imposing the Galerkin condition $\mathbf{A}\mathbf{v} - \lambda\mathbf{v} \perp \mathcal{K}_k(\mathbf{A}, \mathbf{u}_1),\ \mathbf{v} \in \mathcal{K}_k(\mathbf{A}, \mathbf{u}_1)$ and using orthogonality properties, the eigenpairs of $\mathbf{A}$ are given by $(\lambda, \mathbf{v} = \mathbf{U}_k\mathbf{z})$, where $(\lambda, \mathbf{z})$ is an eigenpair of $\mathbf{H}_k$. The eigenvalue problem for $\mathbf{H}_k$ can be solved using, e.g., the QR method, since in most cases its dimension k is relatively small. We refer to, e.g., [4, 5] for a detailed discussion on Arnoldi method. Since we deal with symmetric matrices, the scheme simplifies considerably, leading to the so-called *Lanczos method*.

- Consider an initial guess $\mathbf{u}_1$, which constitutes the first vector of the basis of the Krylov subspace.
- Compute $\mathbf{w} = \mathbf{A}\mathbf{u}_1$ and orthogonalize it with respect to $\mathbf{u}_1$ as follows. Compute $\mathbf{H}_{11} = \alpha_1 = \mathbf{u}_1^T\mathbf{w}$ and set $\tilde{\mathbf{u}}_2 = \mathbf{w} - \alpha_1\mathbf{u}_1$. Then, normalize the vector setting $\mathbf{H}_{21} = \beta_1 = \|\tilde{\mathbf{u}}_2\|$ and $\mathbf{u}_2 = \tilde{\mathbf{u}}_2/\beta_1$.
- At step $j = 2, \dots, k$, compute $\mathbf{w} = \mathbf{A}\mathbf{u}_j$. By symmetry of $\mathbf{A}$ and orthonormality of $\mathbf{U}_j$, the vector $\mathbf{w}$ is orthogonal to all $\mathbf{u}_i$ for $i = 1, \dots, j-2$. Thus, it is enough to compute $\mathbf{H}_{jj} = \alpha_j = \mathbf{u}_j^T\mathbf{w}$, while $\mathbf{H}_{j-1,j} = \beta_j$ is already available from the step $j-1$. Consequently, set $\tilde{\mathbf{u}}_{j+1} = \mathbf{w} - \alpha_j\mathbf{u}_j - \beta_j\mathbf{u}_{j-1}$, that represents the difference between $\mathbf{w}$ and its orthogonal projection onto the the space spanned by the columns of $\mathbf{U}_j$. The basis is updated by normalizing the resulting vector, setting $\beta_j = \|\tilde{\mathbf{u}}_{j+1}\|$ and $\mathbf{u}_{j+1} = \tilde{\mathbf{u}}_{j+1}/\beta_j$.

At the end of the process, we obtain the basis $\mathbf{U}_k$ and a tridiagonal symmetric matrix $\mathbf{H}_k$. The eigenpairs of the original problems are retrieved as in the standard Arnoldi method via the eigendecomposition of $\mathbf{H}_k$. We remark that the number of steps k can be fixed a priori, but this is not practical. The inequality

$$\|\mathbf{A}\mathbf{v} - \lambda\mathbf{v}\| \leq |h_{k+1,k}| = \beta_j$$

leads to the heuristic stopping criterion $\beta_j < tol$. In finite precision arithmetic, we might experience the *loss of orthogonality*. Due to numerical errors, the vectors $\tilde{\mathbf{u}}_{j+1}$ and $\mathbf{u}_i$ $(i = 1, \dots, j-2)$ might not be orthogonal. A simple yet popular cure consists in a re-orthogonalization step, computing $\hat{\mathbf{h}}_j = \mathbf{U}_j^T\tilde{\mathbf{u}}_{j+1}$, and updating the coefficients $\alpha_j \leftarrow \alpha_j + \hat{h}_{jj}$ and the vector $\tilde{\mathbf{u}}_{j+1} \leftarrow \tilde{\mathbf{u}}_{j+1} - \mathbf{U}_j\hat{\mathbf{h}}_j$. In order to keep the computational cost controlled, this is usually done only when $\|\tilde{\mathbf{u}}_{j+1}\| \,/\, \|\mathbf{w}\|$ is smaller than a specified tolerance. Following [5], we set it equal to 0.7. The whole process is described in Algorithms 1 and 2.
Hence, to compute the SVD of the Hadamard product $\mathbf{C} = \mathbf{A} \star \mathbf{B}$, we can apply Lanczos method to $\mathbf{C}^T\mathbf{C}$ or $\mathbf{C}\mathbf{C}^T$. It is evident that an efficient way to compute the effect of the matrix onto a vector is needed by the algorithm. Therefore, we exploit the procedure described in Section 2.3, applying it to both $\mathbf{C}$ and its transpose in the correct order. To retrieve the set of left or right singular vectors not computed via Lanczos, the efficient matrix-vector product can still be exploited. Recalling (2.13), a matrix-matrix multiplication with $\mathbf{C}$ is required. The most efficient approach consists in computing $\mathbf{V}\boldsymbol{\Sigma}^{-1}$ first, and applying the procedure of Section 2.3 for all columns of the resulting product then. A similar reasoning applies to (2.14).

3 Numerical results

To test the performance of the proposed approach, we consider the product of bi-variate smooth functions. Assume that

$$\begin{aligned}
\mathbf{A}_{ij} &= f(x_i, y_j), \quad \text{with } f(x,y) = \frac{1}{x+y}, \\
\mathbf{B}_{ij} &= g(x_i, y_j), \quad \text{with } g(x,y) = \frac{1}{\sqrt{x^2+y^2}}, \\
\mathbf{C}_{ij} &= \mathbf{A}_{ij}\mathbf{B}_{ij},
\end{aligned}$$

Algorithm 1 Lanczos method.

Input: Symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, vector $\mathbf{x} \in \mathbb{R}^n$, maximum iterations *maxit*, tolerance *tol*.
Output: Ortonormal basis $\mathbf{U} = [\mathbf{u}_1, \dots, \mathbf{u}_k]$ of the Krylov subspace, symmetric tridiagonal matrix $\mathbf{H} = \text{tridiag}(\beta, \alpha, \beta)$.

$\mathbf{u}_1 = \mathbf{x}/\|\mathbf{x}\|$, $\mathbf{U} = [\mathbf{u}_1]$.
for $j = 1, 2, \dots, maxit$ **do**
  $\mathbf{w} = \mathbf{A}\mathbf{u}_j$.
  $\alpha_j = \mathbf{u}_j^T \mathbf{w}$.
  $\tilde{\mathbf{u}}_{j+1} = \mathbf{w} - \alpha_j \mathbf{u}_j$
  if $j > 1$ **then**
    $\tilde{\mathbf{u}}_{j+1} \leftarrow \tilde{\mathbf{u}}_{j+1} - \beta_j \mathbf{u}_{j-1}$
  end if
  if $\|\tilde{\mathbf{u}}_{j+1}\| \leq 0.7\|\mathbf{w}\|$ **then**
    $\hat{\mathbf{h}}_j = \mathbf{U}_j^T \tilde{\mathbf{u}}_{j+1}$.
    $\alpha_j \leftarrow \alpha_j + \hat{h}_{jj}$.
    $\tilde{\mathbf{u}}_{j+1} \leftarrow \tilde{\mathbf{u}}_{j+1} - \mathbf{U}_j \hat{\mathbf{h}}_j$.
  end if
  $\beta_j = \|\tilde{\mathbf{u}}_{j+1}\|$.
  if $\beta_j < tol$ **then**
    break.
  end if
  $\mathbf{u}_{j+1} = \tilde{\mathbf{u}}_{j+1}/\beta_j$, $\mathbf{U} \leftarrow [\mathbf{U}\ \mathbf{u}_{j+1}]$.
end for
if $j == maxit$ **and** $\beta_j \geq tol$ **then**
  $k = maxit - 1$.
else
  $k = j$.
end if
$\mathbf{H} = \text{tridiag}(\{\beta_1, \dots, \beta_{k-1}\}, \{\alpha_1, \dots, \alpha_k\}, \{\beta_1, \dots, \beta_{k-1}\})$.

Algorithm 2 Compute eigendecomposition.

Input: Symmetric matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$, maximum iterations *maxit*, tolerance *tol*.
Output: Matrix $\mathbf{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_k)$ containing a subset of the eigenvalues, matrix $\mathbf{V}$ of the corresponding eigenvectors.

Run Algorithm 1 with a random starting vector $\mathbf{x}$, *maxit*, *tol* to get $\mathbf{U}$,$\mathbf{H}$.
Compute eigendecomposition $(\mathbf{\Lambda}, \mathbf{Z})$ of $\mathbf{H}$ using, e.g., QR method.
Retrieve eigendecomposition $(\mathbf{\Lambda}, \mathbf{V}) = (\mathbf{\Lambda}, \mathbf{UZ})$ of $\mathbf{A}$.

where (x_i, y_j) are the points of the grid $\{0.1, 0.2, \dots, M\} \times \{0.1, 0.2, \dots, N\}$. The values of M, N are specified in the following.
Due to the regularity of the functions in the selected domain, we expect that all matrices admit low-rank representations. To check this, we simply look at the decay of the singular values, shown in Figure 3.1 for $M = N = 5$. An exponential decay is present, with a similar trend for all matrices. Thus, we expect that $\mathbf{A}, \mathbf{B}, \mathbf{C}$ can be efficiently compressed in a low-rank format, with ranks of similar order of magnitude.

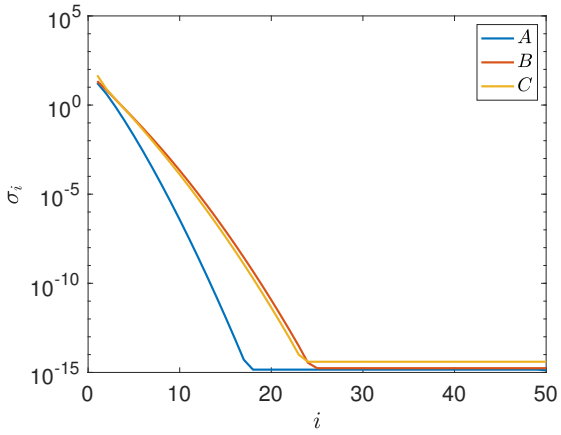

Figure 3.1: Decay of the singular values σ_i for increasing i using $M = N = 5$.

We can now check the correctness of our method. We consider first $M = 1, N = 2$ to keep the sizes controlled. The low-rank approximations of $\mathbf{A}, \mathbf{B}$ are obtained from their SVDs after truncating the singular values lower than $\epsilon = 10^{-4}$. For the Hadamard product $\mathbf{C} = \mathbf{A} \star \mathbf{B}$, we run Lanczos and MATLAB `svd` routine to get the low-rank approximations we need. We select a tolerance of ϵ^2, a maximum number of iteration of 10^3 and the matrix $\mathbf{C}\mathbf{C}^T$. A visual comparison of the singular values gets

$$\text{diag}(\mathbf{\Sigma}_{Lan}) = (4.65e+01,\ 7.36e+00,\ 1.34e+00,\ 2.08e-01,\ 2.75e-02,\ 3.08e-03,\ 2.90e-04,\ 1.93e-06)^T,$$

$$\text{diag}(\mathbf{\Sigma}_{svd}) = (4.65e+01,\ 7.36e+00,\ 1.34e+00,\ 2.08e-01,\ 2.75e-02,\ 3.12e-03,\ 2.97e-04)^T.$$

and qualitatively validates our method. A similar comparison holds for the singular vectors. A more quantitative analysis looks at the Frobenius norms of the resulting matrices, and gives

$$\begin{aligned}
\left\|(\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^T) \star (\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^T) - \mathbf{U}_{Lan}\mathbf{\Sigma}_{Lan}\mathbf{V}_{Lan}^T\right\|_F &= 2.9135e-06,\\
\left\|\mathbf{A} \star \mathbf{B} - \mathbf{U}_{Lan}\mathbf{\Sigma}_{Lan}\mathbf{V}_{Lan}^T\right\|_F &= 6.8986e-05,\\
\left\|(\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^T) \star (\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^T) - \mathbf{U}_{svd}\mathbf{\Sigma}_{svd}\mathbf{V}_{svd}^T\right\|_F &= 6.7492e-05,\\
\left\|\mathbf{A} \star \mathbf{B} - \mathbf{U}_{svd}\mathbf{\Sigma}_{svd}\mathbf{V}_{svd}^T\right\|_F &= 2.2818e-05,\\
\left\|\mathbf{U}_{Lan}\mathbf{\Sigma}_{Lan}\mathbf{V}_{Lan}^T - \mathbf{U}_{svd}\mathbf{\Sigma}_{svd}\mathbf{V}_{svd}^T\right\|_F &= 6.7434e-05.
\end{aligned}$$

All errors are relatively small, with no significant differences between our implementation and the `svd` routine. It is also interesting to compare the size $k_A k_B$ of $\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B$ and the ones of $\mathbf{\Sigma}_{Lan}$, $\mathbf{\Sigma}_{svd}$, denoted by $k = k_{Lan}$ and k_{svd}. We get

$$k_A = 5, \quad k_B = 7, \quad k = k_{Lan} = 8, \quad k_{svd} = 7.$$

Firstly, the result is consistent with the behavior of the singular vectors observed in Figure 3.1. Secondly, the difference with the standard `svd` implementation is of just one singular value, coherently with the similar error norms we computed. Thirdly, it confirms the fact that (2.7) does not encode the low-rank properties of the Hadamard product. Indeed, the resulting size of $\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B$ is $k_A k_B = 35$, which is much larger compared to $k = 8$. In terms of order of magnitude, we have that $k_A k_B = \mathcal{O}(k^2)$, so that product of low-rank approximations drastically increases the ranks, and the good compression properties cannot be captured.
In order to test the computational advantages of our method, we let $M = N$ vary from 50 to 300. The other

parameters are not changed from the previous simulation. Since we deal with square matrices, we expect no significant differences if $\mathbf{C}\mathbf{C}^T$ or $\mathbf{C}^T\mathbf{C}$ is used. However, we decide to verify it, with the two approaches named *Lanczos 1* and *Lanczos 2* respectively. We check the correctness of our results by computing the error norms. Figure 3.2 reports $\left\|(\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^T)\star(\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^T)-\mathbf{U}_{Lan}\mathbf{\Sigma}_{Lan}\mathbf{V}_{Lan}^T\right\|_F$ for different values of N, comparing them with the `svd` routine. The order of magnitude is the same, with a minor preference for the first version of the Lanczos method. We also measure the execution time, with the results reported in Figure 3.3. For small matrices,

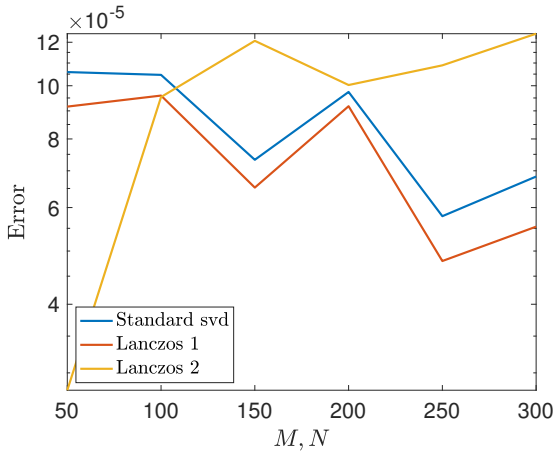

Figure 3.2: Error norms $\left\|(\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^T)\star(\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^T)-\mathbf{U}_{Lan}\mathbf{\Sigma}_{Lan}\mathbf{V}_{Lan}^T\right\|_F$ with Lanczos method and $\left\|(\mathbf{U}_A\mathbf{\Sigma}_A\mathbf{V}_A^T)\star(\mathbf{U}_B\mathbf{\Sigma}_B\mathbf{V}_B^T)-\mathbf{U}_{svd}\mathbf{\Sigma}_{svd}\mathbf{V}_{svd}^T\right\|_F$ with the `svd` routine as a function of $N=M$.

`svd` is preferred over our approach. This is possibly due to MATLAB optimizations triggered for small sizes. The in-built algorithm is probably different from a simple implementation of Lanczos method, and outperforms our approach, at least in terms of execution times. We also recall that the matrix-vector multiplication we proposed in Section 2.3 becomes effective when $k_A^2/n \simeq k_B^2/n < 1$ and n is sufficiently large. Since for small matrices the ratios are not significantly less than one, the benefits of our approach are not evident. Increasing the sizes, important differences appear. The speedup factor we get is almost 10 compared to the standard `svd` implementation, and our approach should be preferred. Finally, we analyze the compression properties of the Hadamard product. Using

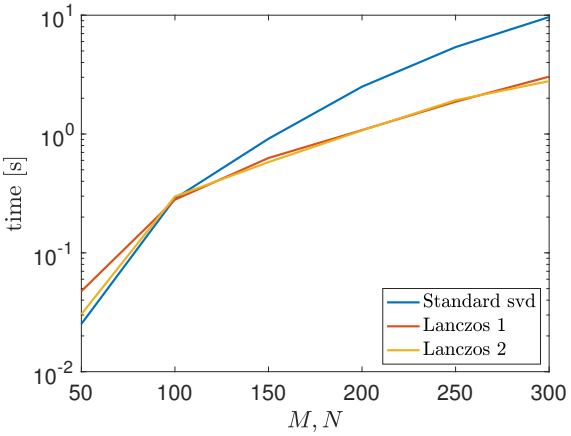

Figure 3.3: Execution time of the Lanczos method and MATLAB routine `svd` to extract the low-rank approximation of $\mathbf{C}$ as a function of $N=M$.

(2.7), we get a matrix with size k_Ak_B, while Lanczos methods keeps only k vectors. From Figure 3.4, it is evident that $\mathbf{\Sigma}_A\otimes\mathbf{\Sigma}_B$ does not efficiently encode the low-rank structure of the Hadamard product. In particular, a trend $k_Ak_B=\mathcal{O}(k^2)$ can be observed. The differences with standard MATLAB implementation are rather minimal.
As a side remark, we point out that the starting guess $\mathbf{u}_1$ for Lanczos method is selected randomly. Even though

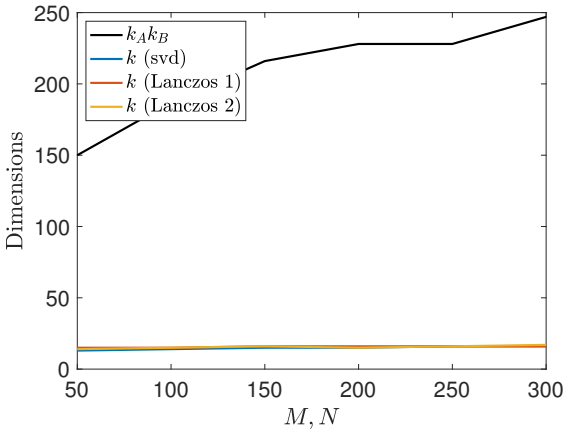

Figure 3.4: Size $k_A k_B$ of $\mathbf{\Sigma}_A \otimes \mathbf{\Sigma}_B$, size k of $\mathbf{\Sigma}_{Lan}$ obtained via Lanczos method and size of $\mathbf{\Sigma}_{svd}$ obtained via MATLAB routine `svd` as a function of $N = M$.

the results are quite robust with respect to such a variability, a slightly different computational performance can be expected at each run of the algorithm.
Finally, the selected tolerance ϵ^2 in Lanczos method can be further decreased to get better approximations, even though errors due to machine precision will eventually play a significant role.

4 Conclusion

In this project we studied the low-rank approximation properties of the element-wise product of two matrices. It is known that the Hadamard product generally multiplies ranks, so that compression properties can be lost, even though the starting matrices are compressed efficiently. However, exploiting a suitable representation, we presented a method able to efficiently compress the resulting matrix via its singular value decomposition. It uses the well-known Lanczos method to compute eigenpairs, taking advantage from a fast matrix-vector multiplication. As a practical application of the method, we considered the low-rank approximation of the product of bi-variate functions on a tensorial grid. We showed that our method gives an error similar to standard approaches, while obtaining a significant speedup for large matrices.

References

[1] R. Schneider, A. Uschmajew. Approximation rates for the hierarchical tensor format in periodic Sobolev spaces. *J. Complexity*, 2014.

[2] D. Kressner, L. Periša. Recompression of Hadamard Products of Tensor in Tucker Format. *SIAM Journal on Scientific Computing*, 2017.

[3] H. D. Simon, H. Zha. Low-rank Matrix Approximation using the Lanczos Bidiagonalization Process with Application. *SIAM Journal on Scientific Computing*, 2000.

[4] A. Quarteroni, R. Sacco, F. Saleri. Numerical Mathematics. *Springer*. 2007.

[5] S. Massei, L. Periša. Lecture notes for the EPFL course Computational Linear Algebra. 2019.