Università di Pisa

Computational Mathematics

A.Y. 2019/2020

Project Report

Authors:
Giulia Fois
Nicolas Manini

October 31, 2020

Contents

1 Structure of the report

The report is structured as follows:

- In Section 2 we give a general description of the problem and expose the algorithm to solve it;
- In Section 3 we provide a detailed analysis of properties of our problem and of our algorithm: we in fact treat the issues of convexity, compactness of the feasible set, differentiability of the function, complexity and stability of the algorithm and its convergence to a local optimum;
- In Section 4 we present the details of the algorithms we required in order to develop our algorithm.
- In Section 5 we present the datasets we are using in the experimental evaluations.
- In Section 6 we discuss the implementation choices we made when programming our algorithm.
- In Section 7 we conducted some preliminary experiments in order to better define the workflow and target some initial hypotheses.
- In Section 8 we present the main experimental results of our algorithm analyzing its behaviour under different kinds of inputs.
- In Section 9 we give some conclusion to our work and discuss some possible ways to further improve performances.
- Appendix A contains theorems and lemmas (with related proofs) that are necessary to our analyses and therefore cited throughout the sections of our report.

2 Description of the Problem

The problem we are solving is low-rank approximation of a matrix $A \in \mathbb{R}^{m \times n}$:

$$\min_{U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times n}} \|A - UV\|$$

The algorithm we are going to use to solve the problem is based on alternating optimization. It consists in finding, at each step, a better choice for alternatively U and V. Starting from an initial choice of a matrix V_0, we look for a U_1 that minimizes $\|A - U_1 V_0\|$, then in the same way we find the succeeding V_1 that in turn minimizes $\|A - U_1 V_1\|$, and proceeding this way we aim to converge to an optimal approximation (or to the best possible one). Each alternation step is characterized by the employment of QR factorization.

Since we look for a low-rank approximation of A we have $k < rk(A)$ and by **Lemma 10** we can look for U and V s.t. $k = rk(UV) < rk(A)$. By **Theorem 2** it follows that the optimal U and V have full rank: $rk(U) = rk(V) = k$.

First, let's analyze the two kind of steps of the algorithm:

Fix U, find V: Assuming $x \geqslant 1$, let us consider the steps in which we are given a matrix U_x from the previous iteration and need to find the solution V_x to the sub-problem:

$$V_x = \arg\min_V \|A - U_x V\|$$

By means of QR factorization of $U_x = Q_{U_x} R_{U_x}$ we can re-state the problem as:

$$V_x = \arg\min_V \|A - U_x V\| = \arg\min_V \|A - Q_{U_x} R_{U_x} V\|$$

and then exploiting multiplication by orthogonal matrices, which preserves the norm:

$$\arg\min_V \|A - Q_{U_x} R_{U_x} V\| = \arg\min_V \left\| {Q_{U_x}}^\top A - {Q_{U_x}}^\top Q_{U_x} R_{U_x} V \right\| = \arg\min_V \left\| {Q_{U_x}}^\top A - R_{U_x} V \right\|$$

Fix V, find U: Assuming $x \geqslant 1$ we now consider the steps in which we need to find a matrix U_x given V_{x-1} from the previous iteration, that is, finding a solution for:

$$U_x = \arg\min_U \|A - U V_{x-1}\|$$

Applying **Theorem 6** we can re-state the problem as:

$$\arg\min_U \|A - U V_{x-1}\|' = \arg\min_U \left\| A^\top - (U V_{x-1})^\top \right\| = \arg\min_U \left\| A^\top - {V_{x-1}}^\top U^\top \right\|$$

and by conveniently renaming $B = A^\top$, $T = {V_{x-1}}^\top$ and $W = U^\top$ the problem reduces to:

$$ {U_x}^\top = \arg\min_W \|B - TW\|$$

where the two $B \in \mathbb{R}^{m \times n}$ and $T \in \mathbb{R}^{n \times k}$ are given, and we want to find an optimal $W \in \mathbb{R}^{k \times m}$. This kind of problem is in the same form of the one observed in the previous paragraph and thus it can be solved analogously. For brevity reasons we are going to study in detail the first case, aiming at finding an optimal V given U.

Applying **Lemma 8** we know $rk(A) \leqslant \min\{m, n\}$ and by transitivity over $k < rk(A)$ we know that both $k < m$ and $k < n$, meaning that U is thin and V is fat (and by consequence R_{U_x} is thin too, T is thin and W is fat). Given that we can rewrite the problem as:

$$\arg\min_V \left\| {Q_{U_x}}^\top A - R_{U_x} V \right\| = \arg\min_V \left\| \begin{pmatrix} {Q_{1U_x}}^\top \\ {Q_{2U_x}}^\top \end{pmatrix} A - \begin{pmatrix} R_{1U_x} \\ 0 \end{pmatrix} V \right\|$$

where:

- ${Q_{1U_x}}^\top \in \mathbb{R}^{k \times m}$ is the matrix composed of the first k rows of ${Q_{U_x}}^\top$;
- ${Q_{2U_x}}^\top \in \mathbb{R}^{(m-k) \times m}$ is the matrix composed of the last $(m-k)$ rows of ${Q_{U_x}}^\top$;
- $R_{1U_x} \in \mathbb{R}^{k \times k}$ is the upper-triangular matrix composed of the first k rows of R_{U_x};

and expanding the product we get:

$$\arg\min_V \left\| \begin{pmatrix} {Q_{1U_x}}^\top A \\ {Q_{2U_x}}^\top A \end{pmatrix} - \begin{pmatrix} R_{1U_x} V \\ 0 \end{pmatrix} \right\| = \arg\min_V \left\| \begin{pmatrix} {Q_{1U_x}}^\top A - R_{1U_x} V \\ {Q_{2U_x}}^\top A \end{pmatrix} \right\| \tag{1}$$

where it is clear that the term ${Q_{2U_x}}^\top A$ is independent from the matrix V and thus the problem reduces to minimizing the upper portion:

$$\arg\min_V \left\| {Q_{1U_x}}^\top A - R_{1U_x} V \right\|$$

Since **Theorem 4** guarantees that each optimal solution w.r.t. the Frobenius norm is also optimal for the spectral norm we now restrict to considering $\|\cdot\| = \|\cdot\|_F$ and thus we can apply **Theorem 7** reducing the problem to computing (let a_i be the i-th column of A):

$$v_i^* = \arg\min_v \left\| ({Q_{1U_x}}^\top A)_i - R_{1U_x} v \right\| = \arg\min_v \left\| {Q_{1U_x}}^\top a_i - R_{1U_x} v \right\| \qquad (i = 1, \ldots, n) \tag{2}$$

The considerations we just made give an easy solution to the problem only when our known matrix U_x or V_{x-1}, depending on the specific step, is full rank. The fact that a matrix $X = QR$ is full rank, in fact, assures us that the full rank property also holds for R (and consequently for R_1: if R doesn't have any zero elements on its diagonal, the same can trivially be stated about R_1). The matrix not being full rank implies it not being invertible, and this constitutes an issue for the resolution of the Least Squares problem we got to in (2). One possible solution could be to use the Moore-Penrose pseudoinverse, but its computation in possibly many sub-steps would be too computationally expensive.
On the other hand we cannot assume, nor we want to impose, that U_x or V_{x-1} will be full rank at each step. As long as our algorithm converges to the optimal solution having rank k, we aren't really concerned with necessarily achieving full rank at any step before that. We in fact reasonably expect that for some steps this might not happen, and thus need to treat this possibility accordingly. For this reason, we decide to adopt a slightly different version of the QR factorization, that exploits a permutation strategy that allows us to solve our sub-problems both in case our matrix is full rank and in case it isn't. Given a matrix $X \in \mathbb{R}^{m \times k}$, and assuming $rank(X) = k' \leqslant k$, this method allows us to achieve the following factorization:

$$X\Pi = Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix}$$

where:

- $\Pi \in \mathbb{R}^{k \times k}$ is a permutation matrix (and thus it is orthogonal);
- $R_{11} \in \mathbb{R}^{k' \times k'}$ is upper-triangular and has full rank ($rk(R_{11}) = k'$);
- $R_{12} \in \mathbb{R}^{k' \times (k-k')}$

We exploit this QR factorization, called QR with Column Pivoting (see chapter 5.4.2 of [1]), to obtain the invertible component that allows us to easily solve our problem even in

case of not having full rank. In **Section 4.1** we give a more thorough explanation of the algorithm to obtain the QR with Column Pivoting factorization.

Our minimization problem (1) can then be reformulated by means of this variant of QR factorization as

$$\arg\min_V \left\| {Q_{U_x}}^\top A - \begin{pmatrix} R_{11U_x} & R_{12U_x} \\ 0 & 0 \end{pmatrix} {\Pi_{U_x}}^\top V \right\|$$

and then for each column v_j^*, $(j = 1, \ldots, n)$ we reduce the problem as we did for (2):

$$v_j^* = \arg\min_v \left\| {Q_{U_x}}^\top a_j - \begin{pmatrix} R_{11U_x} & R_{12U_x} \\ 0 & 0 \end{pmatrix} {\Pi_{U_x}}^\top v \right\| = \arg\min_v \left\| \begin{pmatrix} R_{11U_x} & R_{12U_x} \\ 0 & 0 \end{pmatrix} {\Pi_{U_x}}^\top v - {Q_{U_x}}^\top a_j \right\|$$

For readability reasons, let ${Q_{U_x}}^\top a_j = \begin{pmatrix} b \\ c \end{pmatrix}$ and ${\Pi_{U_x}}^\top v = \begin{pmatrix} y \\ w \end{pmatrix}$,

where:

- $y \in \mathbb{R}^{k'}$ are the first k' entries in a permutation of the target vector v;
- $w \in \mathbb{R}^{k-k'}$ are the last $(k - k')$ entries in a permutation of the target vector v;
- $b \in \mathbb{R}^{k'}$ are the first k' entries in the j-th column of ${Q_{U_x}}^\top A$
- $c \in \mathbb{R}^{m-k'}$ are the last $(m - k')$ entries in the j-th column of ${Q_{U_x}}^\top A$

we have that:

$$\arg\min_v \left\| \begin{pmatrix} R_{11U_x} & R_{12U_x} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y \\ w \end{pmatrix} - \begin{pmatrix} b \\ c \end{pmatrix} \right\| = \arg\min_v \left\| \begin{pmatrix} R_{11U_x} y + R_{12U_x} w - b \\ -c \end{pmatrix} \right\|$$

Recalling that ${\Pi_{U_x}}^\top v = \begin{pmatrix} y \\ w \end{pmatrix}$, we can state that the term $-c$ introduces a component in the minimization target which is independent from the entries of v, thus our solution is determined through minimization of the upper part only. More precisely, we need to find y, w such that $\|R_{11U_x} y + R_{12U_x} w - b\|$ is minimized. Since we used R_{11U_x}, that is full-rank, it follows that we can always find a solution to the problem:

$$\|R_{11U_x} y + R_{12U_x} w - b\| = 0$$

In particular, for any choice of w, we can derive the subsequent y that minimizes the norm by applying backwards substitution (since R_{11U_x} is upper triangular), thus obtaining $y = R_{11U_x}^{-1}(-R_{12U_x} w + b)$. Let's now go back to v:

$${\Pi_{U_x}}^\top v = \begin{pmatrix} y \\ w \end{pmatrix} \iff v = \Pi_{U_x} \begin{pmatrix} y \\ w \end{pmatrix}$$

Our v_j^* is thus any vector in the form:

$$v_j^* = \Pi_{U_x} \begin{pmatrix} R_{11U_x}^{-1}(-R_{12U_x}w + b) \\ w \end{pmatrix}. \tag{3}$$

Concerning the choice of w we identified the following options:

1. Picking a fixed vector as w (e.g. the zero vector).
2. Generating a random w (eventually bounding its norm), this introduces an aleatory component in the algorithm, which impedes reproducibility of experiments but we think that might be helpful in converging to the optimum.
3. Computing w such that $\left\|v_j^*\right\|$ is minimized, this gives a canonical choice for w but the added cost might not lead to significant improvements.

Moreover, under the hypothesis of convergence of the algorithm, we will reach some iteration after which $rk(U) = rk(V) = k$, which implies that $k = k'$ and since w lives in $\mathbb{R}^{k-k'}$ we anticipate that (hopefully) the choice of w will be of smaller impact as the algorithm progresses, up to when it will not be needed (i.e. when ${\Pi_{U_x}}^\top v = y$).

3 Complexity, Stability and Convergence analysis

The following is a pseudo-code of our algorithm, where the main steps are outlined in order to make the complexity analysis clearer.

Algorithm 1: Alternating Low-rank Approximation Algorithm

In : A matrix $A \in \mathbb{R}^{m \times n}$, an integer $k < rk(A)$.
Out: Two matrices $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{k \times n}$ such that $\|A - UV\|$ is minimized.

$find \leftarrow \mathcal{U}'$ /* Current minimization step */
$V \leftarrow PickInitial(k, n)$ /* Initialize to a $k \times n$ matrix */
do
  if $find = \mathcal{U}'$ **then** /* Find the minimizing U */
    $U^\top \leftarrow MinSubTask(A^\top, V^\top)$
    $find \leftarrow \mathcal{V}'$
  else /* Find the minimizing V */
    $V \leftarrow MinSubTask(A, U)$
    $find \leftarrow \mathcal{U}'$
  end
until *some stop criterion is met*
return U, V

Function *MinSubTask(A, Y)*
  Out: A matrix Z that minimizes $\|A - YZ\|$
  $\langle k', \Pi, Q, R \rangle \leftarrow QRColumnPivoting(Y)$ /* $Y\Pi = QR$, $rk(Y) = k'$ */
  $\begin{pmatrix} R_{11} & R_{12} \\ 0 & 0 \end{pmatrix} \leftarrow R$
  for $j = 0$ **to** n **do** /* We compute z_j^*, the j-th column of Z */
    $w \leftarrow$ fix a vector with $k - k'$ entries
    $b \leftarrow$ first k' entries of $Q^\top a_j$
    $t \leftarrow -R_{12} w$
    $t \leftarrow t + b$
    $y \leftarrow BackwardsSubstitution(R_{11}, t)$ /* R_{11} fullrank, triangular */
    $z_j^* \leftarrow Permute\left(\Pi, \begin{pmatrix} y \\ w \end{pmatrix}\right)$
  end
  return $\begin{bmatrix} z_1^* & \cdots & z_n^* \end{bmatrix}$

3.1 Complexity Analysis

The complexity of our algorithm relies on three factors:

- Under the hypothesis of convergence, the number of steps s to converge to the optimal solution (or otherwise to reach our stopping condition).
  In Section 1.4 of [2] the solution of our problem is given by an algorithm that follows the exact same steps as ours: it is based on the alternating projections method that, starting from an initial guess for one of the two matrices, alternatively fixes one of the two variable matrices and finds the other one by solving a linear least squares problem. The authors report a linear convergence rate for this algorithm. In the same way, in Section 4.2 of [3], it is stated that the alternating least squares algorithm (that again is exactly our one) converges linearly.

- The computational cost of the QR factorization with column pivoting of U_x or $T = {V_{x-1}}^\top$ at each step $x \geqslant 1$

- The cost of computing U_x given V_{x-1} or V_x given U_x at each step $x \geqslant 1$

Let's now consider the complexity of computing the QR factorization with column pivoting of a matrix $X \in \mathbb{R}^{m \times k}$, $rk(X) = k'$. The algorithm is optimized as follows:

- The Householder matrix H is never computed; instead, we exploit the householder vector and the equality $HX = X - 2v(v^\top X)$.

- The norms of the columns of X are only computed once, at the beginning of the algorithm, when $R_{22} = X$. At each step j, the norms of the column vectors of R_{22}^j are not computed from scratch, but by simply subtracting a certain quantity by the ones of the previous step, by virtue of the fact that for any orthogonal matrix $O \in \mathbb{R}^{l \times l}$ (thus including the j-th householder matrix) it holds the following:

$$O^\top z = \begin{pmatrix} \alpha \\ w \end{pmatrix} \implies \|w\|_2^2 = \|z\|_2^2 - \alpha^2$$

  Where $z \in \mathbb{R}^l, w \in \mathbb{R}^{l-1}, \alpha \in \mathbb{R}$.

The factorization requires exactly k' steps to be computed, because the termination step is reached when the column vectors of R_{22}^j have all norm 0, and this happens when we consider the sub-matrix $R_{22}[k'+1:m, k'+1:k]$, at step $j = k'+1$. The only additional cost related to the column pivoting version is given by the initial computation of the column norms, that contributes with an overhead of $\Theta(mk)$ flops + $O(k^2)$ flops to update them. This last addend derives from the fact that for each step j, for each vector norm correspondent to the columns in the interval $[j+1, k]$ we have 2 flops (a product and a subtraction). Therefore, the cost of updating the norms is

$$\sum_{j=1}^{k'} 2(k-j) = 2\sum_{j=1}^{k'} k - 2\sum_{j=1}^{k'} j = 2k'k - 2\frac{k'(k'+1)}{2} = k'k + k'(k-k'-1)$$

And since $k > k' = rank(X)$ and we expect the rank of our U_i, V_i to increase quickly as the algorithm progresses (up to $k' = k$, that **Lemma 10** suggests should be reached within the first iterations), the cost can be approximated as $k^2 - k = O(k^2)$. The total cost of QR factorization with column pivoting of X is thus

$$T_{QR} = 2mk^2 - \frac{2}{3}k^3 + \Theta(mk) + O(k^2)$$

which dependently on the values of k and m the cost can be approximated as:

- If $m \approx k$ then: $2k^3 - \frac{2}{3}k^3 + \Theta(k^2) + O(k^2) \approx \frac{4}{3}k^3 = \Theta\left(k^3\right)$
- If $m \gg k$ then the cost scales as $2mk^2 = \Theta(mk^2)$

And thus the column pivoting QR factorization has a performance which is asymptotically comparable to that of the standard QR factorization. Since at each step we compute alternately either V or U, and thus the cost of the QR factorization depends on m in one iteration step (as shown above) and in the same manner on n in the next one (thus swapping each occurence of m with n and vice versa); we define those the two costs relative to the factorization respectively as T^m_{QR} and T^n_{QR} and by taking an average of the two $T^{avg}_{QR} = \frac{T^m_{QR} + T^n_{QR}}{2}$ since it would be the average cost of each iteration in the long run; moreover if $m \approx n$ we have $T^{avg}_{QR} \approx T^m_{QR} \approx T^n_{QR}$ and thus we simply refer to T^{avg}_{QR} as T_{QR} in such case or when it's clear from the context whether we are taling about T^m_{QR} or T^n_{QR}.

For what concerns the computation of V_x (all the considerations we will make can be similarly applied to the computation of U_x), it is reduced to the derivation of all its columns (as for 3). Let's break down the computation of a generic column v_j into the very basic operations of it, by simultaneously analyzing their cost:

- Choosing w, given that we follow one of the first two options we stated before, does not involve any additional cost.
- Computing b, that is the vector of the first k' entries of $Q^\top a_j$, has a cost of $O(mk')$ because it is a linear combination of the first k' entries of each row of Q.
- The matrix-vector product $-R_{12}w$, where $R_{12} \in \mathbb{R}^{k' \times (k-k')}$ and $w \in \mathbb{R}^{k-k'}$ has a cost of $O(k'(k-k'))$, but if we choose $w = 0$ we can skip this step.
- The sum $-R_{12}w + b$ costs $O(k')$, and yet again, if we pick $w = 0$ we can skip this step.
- Computing $R_{11}^{-1}(-R_{12}w + b)$ (where $R_{11} \in \mathbb{R}^{k' \times k'}$) through backwards substitution has a cost of $O(k'^2)$.
- Finally, the permutation $\Pi_{U_x} \begin{pmatrix} R_{11}^{-1}(-R_{12}w + b) \\ w \end{pmatrix}$ has a computational cost of $O(1)$ since it does not involve any floating-point operation.

The total cost of each step of the algorithm is therefore

$$T_{QR} + n(O(mk') + O(k'(k-k')) + O(k') + O(k'^2))$$

and by considering $k = k'$ we get:

$$T_{QR} + O(nmk) + O(nk^2)$$

and since both the last term and the two cases for the QR factorization ($\Theta(k^3)$ and $\Theta(mk^2)$ or $\Theta(nk^2)$) are dominated by $O(nmk)$ since $k < m, n$ this is asymptotically equivalent to $O(mnk)$.

3.2 Choice of the stopping condition

The stopping condition of our algorithm relies on the relative difference between the approximation error at the previous step $i-1$, which is $\epsilon_A((UV)_{i-1}) = \|A - (UV)_{i-1}\|$, and the one at the current step i, $\epsilon_A((UV)_i) = \|A - (UV)_i\|$. We define our stopping condition as

$$\epsilon_{Ri} \leqslant \varepsilon \quad \textit{where} \quad \epsilon_{Ri} = \frac{\epsilon_A((UV)_{i-1}) - \epsilon_A((UV)_i)}{\epsilon_A((UV)_{i-1})}$$

and ε is an arbitrary small number which represent the desired accuracy, and constitutes a parameter of the algorithm. Recomputing the exact approximation error ϵ_{Ri} at each iteration has an additional cost of $O(mkn)+mn = O(mkn)$ for the product $(UV)_i$ and the subtraction $A - (UV)_i$, plus another $O(2mn)$ to compute the Frobenius norm. This additional cost is evidently dominated by the computation of the product between U and V. By comparing this cost to the one of computing V at each step we notice that they are asymptotically equivalent, which means that we have no overhead at asymptotic level and we expect its impact not to have a severe effect on performance, given the fact that many factors of the complexity of computing V (or U, depending on the step) are hidden by the big-O notation. In any case, the research of another method of computing the approximation error that doesn't involve the product of the two matrices is a matter of interest (for which we gave some insights in **Section 3.2.1**), but since this has proved to be a complex task to carry over we intend to implement the exact check, which we think will not impede execution time.
In Section 3.7.1 we discuss the possibility of choosing as our stop condition the norm of the gradient of ϵ_A^2: we compute its complexity and evaluate if it may be worth it to compare it to the relative approximation error at empirical level.

3.2.1 Approximated stopping condition

Our aim would be to find a way to provide an easily computable upper bound to ϵ_{Ri} which avoids a too coarse approximation. To achieve so we would need to either:

1. Find some ways to approximate the norm of $(A-(UV)_i)$ without explicitly computing the matrix product (which might include identifying a new norm in which the computation can be conducted easily and resorting, for instance, to norm equivalence).
2. Find some bounds for $(A-(UV)_i)$ which can be defined through the norm of the three separated matrices only.

We followed the second intuition and tried to find some upper and lower bounds for ϵ_A (which we will identify as $\widehat{\epsilon_A}$ and $\check{\epsilon_A}$ respectively) so that we could define an upper bound to ϵ_{Ri} as follows:

$$\epsilon_{Ri} = \frac{\epsilon_A((UV)_{i-1}) - \epsilon_A((UV)_i)}{\epsilon_A((UV)_{i-1})} \leqslant \frac{\widehat{\epsilon_A}((UV)_{i-1}) - \check{\epsilon_A}((UV)_i)}{\check{\epsilon_A}((UV)_{i-1})} = \widehat{\epsilon_{Ri}}$$

A possible upper bound for our approximation error is the following:

$$\epsilon_A((UV)_i) = \|A - (UV)_i\| \leqslant \|A\| + \|(UV)_i\| \leqslant \|A\|_F + \|U_i\|_F \|V_i\|_F = \widehat{\epsilon_{A,1}} \tag{4}$$

where the first inequality is obtained by applying the triangle inequality and the second one is obtained through submultiplicativity of the Frobenius norm.
Another upper bound is given by **Lemma 13**:

$$\epsilon_A((UV)_i) = \|A - (UV)_i\| \leqslant \|A\| + \|(UV)_i\| \leqslant \|A\|_F + \|U_i\|_2 \|V_i\|_F = \widehat{\epsilon_{A,2}} \tag{5}$$

and **Corollary 4** ensures that (5) gives a stricter bound w.r.t. (4). Complexity-wise the bound given by $\widehat{\epsilon_{A,1}}$ requires the computation of the Frobenius norm of A, U_i and V_i which respectively cost $O(mn)+O(mk)+O(kn)$ while the complexity of computing $\widehat{\epsilon_{A,2}}$ differs in the cost of computing the spectral norm of U_i, leading to a cost of $O(mn)+T_{norm2}(m,k)+O(kn)$. We know that since the spectral norm is related to the largest singular value $T_{norm2} = O(T_{SVD})$ and thus $T_{norm2}(m,k) = O(mk^2)$; whether the bound $\widehat{\epsilon_{A,2}}$ is convenient w.r.t. $\widehat{\epsilon_{A,1}}$ depends on both the values of m and k and on how much stricter of a bound it is.

In both $\widehat{\epsilon_{A,1}}$ and $\widehat{\epsilon_{A,2}}$ we isolated the norm of A using the triangle inequality, and this exposed another risk we are facing when proposing this kind of approximations: suppose that we could find a lower bound that could be expressed by isolating the norm of the two matrix addends in ϵ_A, thus some $\check{\epsilon_A} = \|A\|_F + f(U_i, V_i)$ for some function f, by substitution we would get:

$$\widehat{\epsilon_{Ri}} = \frac{\|A\| + \|U_i\|\|V_i\| - \|A\| - f(U_i, V_i)}{\check{\epsilon_A}((UV)_{i-1})} = \frac{\|U_i\|\|V_i\| - f(U_i, V_i)}{\check{\epsilon_A}((UV)_{i-1})}$$

in which the numerator depends on the two matrices U_i and V_i only, and this constitutes a problem since through **Theorem 1** we proved that the sequence of values $\{\epsilon_A((UV)_i)\}$ converges to a minimum, but this does not guarantee that the algorithm produces a sequence of candidates $\{(UV)_i\}$ which converges to a unique solution (in particular since **Corollary 1** exhibits cases in which we have multiple optimal solutions of rank exactly k for the Frobenius norm case, let A_k^1, A_k^2 where $A_k^1 \neq A_k^2$ be such two solutions, we could have that the sequence $\{\epsilon_A((UV)_{2j})\}_{j\in\mathbb{N}} \to A_k^1$ and $\{\epsilon_A((UV)_{2j+1})\}_{j\in\mathbb{N}} \to A_k^2$) and so we could be left with a numerator which cannot get arbitrarily close to 0.
Moreover, finding non-trivial lower bounds is more difficult than finding upper bounds, as observed through **Lemma 14**.
Yet again, since we can't foresee how inexact the bounds will be w.r.t. the original norms, it could be the case that the exact computation leads us to a relative error smaller than ε, while the upper bound doesn't (even when we are not in the aforementioned case).
We imagine that, if adequate bounds were found, a possible option would be to compute

the approximated error by using the upper bound while resorting to the exact computation of ϵ_A every n_{check} iterations (parameter of the algorithm as well), this would cover cases in which the approximation process fails to give a good estimate, while amortizing the cost of checking the exact stopping criterion.

3.3 Stability Analysis

We now discuss the backwards stability of the direct sub-steps of our algorithm, i.e. those in which we exploit U_x or V_{x-1} and QR factorization with column pivoting to compute respectively V_x or U_x.
If we were to employ the standard QR factorization with householder reflectors algorithm, we could comfortably state that each of our direct sub-steps would be backwards stable, since it involves the solution of systems of linear equations by using QR factorization, and this is demonstrated to be backwards stable (see **Theorem 16.2** of [4]).
Since we are actually using a variant of this algorithm, the question our discussion is centered on is whether it is backwards stable as well. Once this is ensured, the proof of the backwards stability of solving a linear equations system by employing QR with column pivoting will follow the exact same steps of the proof of the theorem we just cited.
QR factorization with column pivoting only differs from basic QR factorization because of the column permutations at each step, and the fact that the householder matrices are defined on the permuted matrix. No additional floating point arithmetic operations are performed. For this reason, based on the fact that QR factorization is backwards stable, we can easily state that the algorithm that involves column pivoting is backwards stable as well. Consequently, we can state that the direct sub-steps of our algorithm are backwards stable too.

3.4 Convergence Analysis

Concerning convergence let us study the sequence of approximations $\{(UV)_i\}$ produced by algorithm 1. At a given step the approximation error is $\epsilon_A((UV)_i) = \|A - (UV)_i\|$, and we have that:

Theorem 1 (Convergence of the Alternating Low-rank Approximation Algorithm)**.** *Let $\{(UV)_i\}$ be the sequence of approximations produced at each iteration by an execution of algorithm 1, then we have that $\{\epsilon_A((UV)_i)\} \to \epsilon$ for some $\epsilon \geqslant 0$.*

Proof. The proof is trivial, considering the fact that our algorithm produces a sequence of approximations giving non-increasing values of ϵ_A at each iteration. In fact, at each iteration, in each step we fix a matrix and compute the other one, that is the solution to a minimization problem; thus

$$\epsilon_A((UV)_i) = \epsilon_A(U_j V_{j-1}) \geqslant \epsilon_A(U_j V_j) = \epsilon_A((UV)_{i+1})$$

the same argument holds if we are in a "Fix V find U" step.
Since $\epsilon_A((UV)_i) \geqslant 0$ by definition of norm (or, to give a stricter bound, we have that for all i, $\epsilon_A((UV)_i) \geqslant \epsilon_A(A_k)$ where A_k is the optimal truncated SVD given by the Eckart-Young theorem), the sequence of $\{\epsilon_A((UV)_i)\}$ has to converge to some limit $\epsilon \geqslant \epsilon_A(A_k) \geqslant 0$ since it is non-increasing and bounded from below. □

Notice that **Theorem 1** ensures that the approximation error converges to some value, but it does not guarantee that it reaches a global optimum (the sequence might converge to some $\epsilon \neq \epsilon_A(A_k)$), and thus it might (in principle) produce some locally-optimal solution which isn't globally optimal; more in detail, notice that our problem isn't convex in general, in fact, fixed the matrix A and the rank-value k, it can be formulated as:

$$\min\{\epsilon_A(X) \,|\, rk(X) \leqslant k\}$$

and it is enough to observe that our feasible region $\mathcal{X}^{m,n}_{\leqslant k} = \left\{X \in \mathbb{R}^{m\times n} \,|\, rk(X) \leqslant k\right\}$ is, in general, a nonconvex set (see **Section 3.6, Lemma 2**).
Theorem 1 states that our algorithm produces a series of non-decreasing approximation errors, until converging to a certain ϵ. A more accurate result can actually be achieved by looking at our algorithm as a specific application of the Block Gauss-Seidel method [5]. Let us reformulate our problem by expressing our minimization variables U and V by means of the concatenation of the vectorizations of the two:

$$\min_{U\in\mathbb{R}^{m\times k}, V\in\mathbb{R}^{k\times n}} \|A - UV\| \quad \iff \quad \min_{w\in\mathbb{R}^{mk+kn}} \left\|a - h^{mk}_{nk}(w)\right\|$$

Where:

- $a = vec(A)$;
- $w = (vec(U)) :: (vec(V))$;
- $h^{mk}_{nk}(x) : \mathbb{R}^{mk+nk} \to \mathbb{R}^{mn}$ is the function such that $h^{mk}_{nk}(x) = vec(vec^{-1}(u) \cdot vec^{-1}(v))$, where $\overset{\leftrightarrow}{::}{}^{mk}_{nk}(x) = \langle u, v\rangle$. Given a concatenation of two vectorizations as its input, it obtains the respective original matrices and returns the vectorization of the product between them. We simply refer to it as $h(x)$.

We can look at $w \in \mathbb{R}^{mk+nk}$ as partitioned in two component vectors, $u \in \mathbb{R}^{mk}$ and $v \in \mathbb{R}^{kn}$, that respectively correspond to the vectorizations of U and V. Moreover, we will refer to our vectorized minimization function as $g(w) = \|a - h(w)\|$.
Our algorithm generates a sequence of points $\{w^i\}$, where

$$w^i = \{u^i, v^i\}$$

Each iteration is characterized by the application of some operations on either the first or the second component vector of w. In particular, the operations are exactly the ones we described in Section 2 when we discussed the steps needed to find one of the two matrices when fixing the other one. Notice that we previously broke down this problem into either m or n sub-problems, where our target was each single column of V or U^T, in a way introducing vectorization already.
The operations we just mentioned can be expressed by referring to two possible mappings $T_1 : \mathbb{R}^{mk+kn} \to \mathbb{R}^{mk}$ and $T_2 : \mathbb{R}^{mk+kn} \to \mathbb{R}^{kn}$. Given a vector y^i belonging to a sequence $\{y^i\}$, such mappings associate y^i either to the updated u-component or to the updated v-one.
Since we are in the case where our minimization is tied to two components, we can refer to a specific version of the Block Gauss-Seidel method, which is the 2-Block Gauss-Seidel

algorithm. By means of the mapping notation we defined above, we can express our iteration in the context of the 2-Block GS method by exploiting the parallel mapping scheme (see **Section 3.2** of [5]), where

$$m(i,j) = \begin{cases} \{T_j(w^i), v^i\} & \text{if } j = 1 \\ \{u^i, v^i\} & \text{if } j = 2 \end{cases}$$

Our iteration is then defined as

$$w^{i+1} = (T_1(w^i), T_2(m(i,1))$$

The fact that we are in the setting of a 2-Block GS method lets us conclude some strong results, in particular we can obtain global convergence without any convexity requirement on the target function (this would have not been the case if we were working with the general Block GS method, with an arbitrary number of blocks).

We exploit the definition above to show that we are in the conditions of **Theorem 6.3** of [5], whose conclusions constitute a very important result to us. The hypothesis of such theorem requires only that the global minimization with respect to each component is well defined, that is, for each T_i:

- T_i is a function, which is ensured by the fact that it is the function (in the mathematical sense) computed by the deterministic algorithm we presented.
- T_i finds a global solution (if such solution exists), which is ensured by correctness of our algorithm.
- There is always at least one optimal global solution, which follows from observing that the problem of minimizing $\|A - UY\|$ (or $\|A - YV\|$) constitutes a projection of A over the convex, closed, set $\{UY \mid Y \in \mathbb{R}^{k \times n}\}$, which exists by the Hilbert projection theorem.

We are thus in the conditions for which the premise of the above cited theorem is satisfied, and thus we can assert that the conclusions of the theorem apply to our case, that is that the sequence $\{w^i\}$ generated by our 2-Block GS method is such that

1. Every limit point of our sequence $\{w^i\}$ is a stationary point of g;
2. If $\mathcal{L} = \{X \in \mathbb{R}^{m \times n} \mid \epsilon_A(X) \leqslant \epsilon_A(X_0)\}$ is compact, we have $lim_{i \to \infty} \nabla g(w^i) = 0$ and there exists at least one limit point that is a stationary point for g;
3. If g is pseudoconvex in R^{mk+nk}, every limit point is a global minimizer of g.

The premise of point 2 is satisfied by the folliwing lemma:

Lemma 1. *The set $\mathcal{L} = \{X \in \mathbb{R}^{m \times n} \mid \epsilon_A(X) \leqslant \epsilon_A(X_0)\}$ is compact.*

Proof. Let $r_0 = \epsilon_A(X_0)$, we have that, by applying the definition of ϵ_A:

$$\mathcal{L} = \{X \mid \|A - X\| \leqslant r_0\} = \mathcal{B}(A, r_0),$$

which is compact by **Lemma 17**. □

This is an important result, because it allows us to identify more precisely the point our algorithm converges to. This point can either be a local minimum of g or a saddle point, since we proved that our minimization function is not convex (see **Lemma 4**).
For what concerns point 3, we aren't able to prove its premise; moreover, if such condition held, we would also have proven global convergence.

3.5 Picking the initial V_0

We briefly discuss the logic behind the function $PickInitial(m, n)$ function as used in Algorithm 1. In principle we could choose the initial matrix V_0 to ensure some arbitrary properties beneficial to the algorithm (as long as the complexity for generating such matrix does not impact the overall execution cost).

We could, for example, define $PickInitial$ such that the generated matrix is full-rank, aiding the algorithm in converging toward a rank-k approximation of A (even though by the above observations we expect that the algorithm will produce full-rank U_i, V_i matrices after a few minimization steps).

Moreover, if we could identify some conditions on U, V under which the function ϵ_A is strictly convex (as we tried in section 3.6), and we could prove that each iteration of the algorithm preserves such conditions as long as such conditions hold on its input matrices, we could safely generate V_0 in a way to ensure such preconditions (thus obtaining that each (U_i, V_i) falls into the region in which ϵ_A is convex as an invariant) which ensures convergence to a global minima.

Another important factor we should keep in mind when defining the function $PickInitial$ is that of introducing some form of randomization in the algorithm (note that we could pick a random matrix from those satisfying some other conditions), this could be beneficial to the algorithm in some ways:

- It would make it harder to generate a specific input matrix A on which our algorithm fails to perform well.
- We think that by picking some "canonical" constant values for V_0 (such as the identity matrix for example) we could run into some problems arising when our target matrix A has some kind of regularity in the disposition of saddle points (since, in principle, our algorithm might converge to one of those, as proved in section 3.4).
- Since we don't expect our algorithm to converge to a global optimum, we could run the algorithm multiple times initializing each trial to a different matrix and picking the best approximation among the ones obtained as local optima, in a random restart way.

On a side note, introducing some stochastic behaviour in our algorithm impacts the reproducibility of our experiments, but we think that the behaviour should be "not too much stochastic" since our only random choice is done at initialization time, and each execution step is still completely deterministic.

We will address this problem by testing different initialization strategies (such as picking a full rank starting matrix and introducing a random initialization) in order to compare the behaviour of the different executions to see if such strategies improve the execution time (or the approximation capability) of the algorithm.

3.6 Convexity and Compactness Analysis

Lemma 2. *Given $n, m, k \in \mathbb{N}_+$ and assuming $n \leqslant m$ without loss of generality (the proof can also be constructed working by rows or by observing that column rank is equal to row rank), let $u = \min\{n, m\}(= n)$, it holds that:*

$$(k < u) \iff \mathcal{X}_{\leqslant k}^{m,n} \text{ nonconvex}$$

Proof ($\implies$). Let $h = \min\{k, (n-k)\} \leqslant k$ we define $X_1, X_2 \in \mathbb{R}^{m\times n}$ as:

$$X_1 = 2\begin{bmatrix} \vdots & & \vdots & \vdots & & \vdots \\ e_1 & \cdots & e_k & 0 & \cdots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \end{bmatrix}, \qquad X_2 = 2\begin{bmatrix} \vdots & & \vdots & \vdots & & \vdots \\ 0 & \cdots & 0 & e_{n-h} & \cdots & e_n \\ \vdots & & \vdots & \vdots & & \vdots \end{bmatrix}$$

where e_i are the column vectors of the canonical basis of $\mathbb{R}^m$. We have that the columns $\{2e_1, \ldots, 2e_k\}$ of X_1 are linearly independent and since $|\{2e_1, \ldots, 2e_k\}| = k$ then $rk(X_1) = k$ and thus $X_1 \in \mathcal{X}_{\leqslant k}^{m,n}$, similarly since $|\{2e_{n-h}, \ldots, 2e_n\}| = h \leqslant k$ also $X_2 \in \mathcal{X}_{\leqslant k}^{m,n}$.

We now take $Y = \frac{1}{2}X_1 + \frac{1}{2}X_2$ which has as set of columns $\{e_1, \ldots, e_k, \ldots, e_{n-h}, \ldots, e_n\}$ and by construction it contains $h' = \min\{2k, n\}$ linearly independent vectors; since by hypothesis $k > 0$ then $2k > k$ and since also $k < u = n$ we have that $h' > k$ and thus $Y \notin \mathcal{X}_{\leqslant k}^{m,n}$, proving that $\mathcal{X}_{\leqslant k}^{m,n}$ is nonconvex since $Y \in conv(X_1, X_2)$. □

Proof ($\impliedby$). Suppose there exists a value k such that $\mathcal{X}_{\leqslant k}^{m,n}$ is nonconvex and $k \geqslant u$, observe that since $\mathcal{X}_{\leqslant k}^{m,n} \subseteq \mathbb{R}^{m\times n}$ by applying Lemma 8 we have that each $X \in \mathcal{X}_{\leqslant k}^{m,n}$ has $rk(X) \leqslant u$ and thus it holds that $\mathcal{X}_{\leqslant k}^{m,n} = \mathcal{X}_{\leqslant u}^{m,n}$. Since we are assuming that $\mathcal{X}_{\leqslant u}^{m,n}$ is nonconvex there exists $X_1, X_2 \in \mathcal{X}_{\leqslant u}^{m,n}$ such that $Y = \alpha X_1 + (1-\alpha)X_2 \notin \mathcal{X}_{\leqslant u}^{m,n}$ for some $\alpha \in [0, 1]$, but since $X_1, X_2 \in \mathcal{X}_{\leqslant u}^{m,n} \subseteq \mathbb{R}^{m\times n}$ it follows that $Y \in \mathbb{R}^{m\times n}$ and by applying Lemma 8 again it follows that $rk(Y) \leqslant u \implies Y \in \mathcal{X}_{\leqslant u}^{m,n} = \mathcal{X}_{\leqslant k}^{m,n}$, which gives a contradiction.

□

And since we just proved that our feasible region is nonconvex in general (and in particular, for all non trivial values of k), we expect that our algorithm might produce some locally (but not globally) optimal solutions.

We also exhibit some other results over the entities we are working with in our problem, first we observe that our minimization target is convex:

Lemma 3. *The function $\epsilon_A(X)$ is convex.*

Proof. Recalling the definition $\epsilon_A(X) = \|A - X\|$ we observe that for any two $X, Y \in \mathbb{R}^{m\times n}$ we call $X' = (A - X), Y' = (A - Y)$ and then for any $\alpha \in [0, 1]$ we have:

$$\alpha\epsilon_A(X) + (1-\alpha)\epsilon_A(Y) = \alpha\|A - X\| + (1-\alpha)\|A - Y\| = \alpha\left\|X'\right\| + (1-\alpha)\left\|Y'\right\|$$

and by applying Lemma 11:

$$\alpha\left\|X'\right\| + (1-\alpha)\left\|Y'\right\| \geqslant \left\|\alpha X' + (1-\alpha)Y'\right\|$$

and since $\alpha X' + (1-\alpha)Y' = \alpha A + (1-\alpha)A - \alpha X - (1-\alpha)Y = (A - (\alpha X + (1-\alpha)Y))$ we have that:

$$\alpha\epsilon_A(X) + (1-\alpha)\epsilon_A(Y) \geqslant \|(A - (\alpha X + (1-\alpha)Y))\| = \epsilon_A(\alpha X + (1-\alpha)Y)$$

which completes the proof. □

Moreover, we check the convexity of the function when we consider U, V as two distinct parameters, and we get that:

Lemma 4. *The function $\epsilon_A(U, V)$ is nonconvex.*

Proof. Let us consider $A = 0$ for brevity reasons and without loss of generality (we can think of ϵ_A as the function ϵ_0 in a space where we apply a translation to the origin into A. We take two pairs (U_1, V_1) and (U_2, V_2) and exhibit a counterexample to

$$\epsilon_0(\alpha U_1 + (1-\alpha)U_2, \alpha V_1 + (1-\alpha)V_2) \leqslant \alpha\epsilon_0(U_1, V_1) + (1-\alpha)\epsilon_0(U_2, V_2)$$

with $\alpha \in [0, 1]$, this approach follows from observing that studying the convexity of $\epsilon_A : \mathbb{R}^{m\times k} \times \mathbb{R}^{k\times n} \to \mathbb{R}$ is the same as studying the convexity of $\bar{\epsilon_A} : \mathbb{R}^{mk+kn} \to \mathbb{R}$ defined over the concatenation of the vectorizations of U and V respectively (as $\|h(x)\|$ when considering h as defined in Section 3.4) and observing that:

$$(\alpha vec(U_1) + \beta vec(U_2)) :: (\alpha vec(V_1) + \beta vec(V_2)) = \alpha(vec(U_1) :: vec(V_1)) + \beta(vec(U_2) :: vec(V_2))$$

We fix $\alpha = 1/2$, and we get by definition of ϵ_0 that:

$$\left\| \left(\frac{1}{2}U_1 + \frac{1}{2}U_2\right) \left(\frac{1}{2}V_1 + \frac{1}{2}V_2\right) \right\| = \left\| \frac{1}{2}\left(U_1 + U_2\right) \frac{1}{2}\left(V_1 + V_2\right) \right\| == \frac{1}{4}\|\left(U_1 + U_2\right)\left(V_1 + V_2\right)\|$$

which by expanding the product becomes $\frac{1}{4}\|U_1V_1 + U_1V_2 + U_2V_1 + U_2V_2\|$, and also that:

$$\frac{1}{2}\|U_1V_1\| + \frac{1}{2}\|U_2V_2\| = \frac{1}{2}\left(\|U_1V_1\| + \|U_2V_2\|\right)$$

putting the two together gives:

$$\frac{1}{4}\|U_1V_1 + U_1V_2 + U_2V_1 + U_2V_2\| \leqslant \frac{1}{2}\left(\|U_1V_1\| + \|U_2V_2\|\right)$$

$$\iff$$

$$\frac{1}{2}\|U_1V_1 + U_1V_2 + U_2V_1 + U_2V_2\| \leqslant \|U_1V_1\| + \|U_2V_2\|$$

Now we can choose the two pairs (U_1, V_1) and (U_2, V_2) such that $U_1V_1 = U_2V_2 = 0$ and $U_1V_2 = U_2V_1 \neq 0$ (for example by considering all entries zero except for U_1 and V_2 having the first vector of the canonical basis as their first row and column respectively and U_2 and V_1 having the second vector of the canonical basis as their first row and column respectively, too), thus leading to:

$$0 < \frac{1}{2}\|U_1V_2 + U_2V_1\| \nleqslant \|U_1V_1\| + \|U_2V_2\| = 0$$

which constitutes a counterexample, proving nonconvexity. □

And also we exhibit a proof of noncompactness of our feasible region.

Lemma 5. *Given $n, m, k \in \mathbb{N}_+$ the set $\mathcal{X}_{\leqslant k}^{m,n}$ is noncompact.*

Proof. We prove $\mathcal{X}_{\leqslant k}^{m,n}$ isn't bounded, to do so observe that for any matrix X it holds $rk(X) = rk(\alpha X)$ for any $\alpha \geqslant 0$ (since linear independence of the columns is preserved by multiplying each by a scalar). Now suppose that $\mathcal{X}_{\leqslant k}^{m,n}$ is bounded, thus there exists $r > 0$ such that $\mathcal{X}_{\leqslant k}^{m,n} \subseteq \mathcal{B}(0, r)$, that is $\forall Y \in \mathcal{X}_{\leqslant k}^{m,n} \implies \|Y\| \leqslant r$ we then take any $X \in \mathcal{X}_{\leqslant k}^{m,n}$ such that $X \neq 0$ (which is always possible, since $k, n, m > 0$) and take any $|\alpha| > r/\|X\|$; it holds, since norms are homogeneous, that:

$$\|\alpha X\| = |\alpha|\|X\| > \frac{r\|X\|}{\|X\|} = r$$

and by the above observation $\alpha X \in \mathcal{X}_{\leqslant k}^{m,n}$ which gives a contradiction, since $\alpha X \notin \mathcal{B}(0, r)$. Since a set has to be both bounded and closed in order to be compact, the thesis follows. □

While in principle the noncompactness of our feasible region could have been a problem when studying the convergence of our algorithm, we obtained convergence results by other means (see **Section 3.4**).

3.7 Convexity analysis through the Hessian

We proved that our function $\epsilon_A(U, V)$ is not convex in Lemma 4, by showing a counterexample. In this subsection we study this matter in a more detailed way, by trying to characterize the properties that the matrix U and V (or some of their blocks/elements) should have to ensure convexity in some areas of the space. To do so, we study the positive definiteness of the Hessian matrix of $\epsilon_A(U, V)$. The computation is made element-wise with respect to the entries of U and V (thus applying vectorization to the two parameters). Let us recall that each element $x_{i,j}$ of the product UV is obtained by the dot product of the i-th row of U and the j-th column of V, so we consider the following decompositions (a row decomposition for U and a column one for V):

$$U = \begin{bmatrix} \cdots & u_1^\top & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & u_m^\top & \cdots \end{bmatrix} \qquad V = \begin{bmatrix} \vdots & \vdots & \vdots \\ v_1 & \vdots & v_n \\ \vdots & \vdots & \vdots \end{bmatrix}$$

We can thus rewrite our function as:

$$\epsilon_A(U, V) = \|A - UV\| = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{i,j} - x_{i,j})^2} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}(a_{i,j} - u_i^\top v_j)^2}$$

In the following sections we decided to examine the hessian of the square of the target function $(\epsilon_A(U, V))^2$ since its behaviour is the same w.r.t. a minimization problem and it simplifies a lot of calculations.

3.7.1 Gradient computation

We first compute the gradient. We start by computing the partial derivatives of $\epsilon_A(U, V)$, since they are useful and their computation is similar to that of their squared forms. We start by deriving on the generic elements of U $u_{a,b}$ (that is the b-th element of u_a):

$$\frac{\partial \epsilon_A(U,V)}{\partial u_{a,b}} = \frac{-1}{\epsilon_A(U,V)} \cdot \left[\sum_{j=1}^{n} (a_{a,j} - u_a^\top v_j) \cdot v_{b,j} \right]$$

In the same way, we derive on the elements of V $v_{a,b}$ (that is the a-th element of v_b):

$$\frac{\partial \epsilon_A(U,V)}{\partial v_{a,b}} = \frac{-1}{\epsilon_A(U,V)} \cdot \left[\sum_{i=1}^{m} (a_{i,b} - u_i^\top v_b) \cdot u_{i,a} \right]$$

The partial derivatives with respect to the squared norm are thus the following:

$$\frac{\partial \epsilon_A^2(U,V)}{\partial u_{a,b}} = -2 \sum_{j=1}^{n} (a_{a,j} - x_{a,j}) \cdot v_{b,j}$$
$$\frac{\partial \epsilon_A^2(U,V)}{\partial v_{a,b}} = -2 \sum_{i=1}^{m} (a_{i,b} - x_{i,b}) \cdot u_{i,a}$$

Our gradient $\nabla \epsilon_A^2(U, V) \in \mathbb{R}^{mk+kn}$ is the following vector:

$$\nabla \epsilon_A^2(U,V) = \begin{pmatrix} \frac{\partial \epsilon_A^2(U,V)}{\partial u_{a,b}} \\ \frac{\partial \epsilon_A^2(U,V)}{\partial v_{a,b}} \end{pmatrix} = \begin{pmatrix} -2 \sum_{j=1}^{n} (a_{a,j} - x_{a,j}) \cdot v_{b,j} \\ -2 \sum_{i=1}^{m} (a_{i,b} - x_{i,b}) \cdot u_{i,a} \end{pmatrix}$$

We now discuss if the gradient can be adopted as a possible stopping condition for our algorithm, and if doing so would be computationally more advantageous with respect to using the relative approximation error (as we explained in Section 3.2.1).
Our algorithm does not implement a gradient descent method, since the direction that is chosen at each step depends on the U or V matrix found by minimizing our function ϵ_A. Since we proved convergence to a stationary point, though, we are sure that the norm of our gradient can get arbitrarily close to 0. For this reason, a possible stopping condition would be to check if $\left\| \nabla \epsilon_A^2(U,V) \right\| = 0$ (that in practical terms would be translated in checking if it is smaller than some positive precision parameter of the algorithm).
Let us first notice that the gradient depends on every entry $x_{i,j}$ of the product UV, so we have to compute such product in $\mathcal{O}(mkn)$ flops, let us then evaluate how much is the cost of computing $\left\| \nabla \epsilon_A^2(U,V) \right\|$ at each iteration and therefore which would be its impact on the general complexity of the algorithm. For each element $u_{a,b}$ of U, the gradient contains an entry that can be computed as $-2 \sum_{j=1}^{n} (a_{a,j} - x_{a,j}) \cdot v_{b,j}$. Assuming the we already computed

UV each of this computations takes $2n+1$ flops, so the total cost of computing the entries of the gradient related to U is $mk[2n+1] = 2mkn+mk$. In the same way we can derive the total cost of computing the V-related entries of the gradient, which is $kn[2m+1] = 2mkn + kn$. The total cost of computing the gradient is therefore $4mkn + k(m+n) + \mathcal{O}(mkn)$. To then compute the Frobenius norm of the gradient we need $2(mk+nk)+1$ additional flops. Asymptotically, we would have a cost of $\mathcal{O}(4mkn)$, which doesn't give any asymptotical improvement w.r.t. computing the approximation error difference, that we discussed in section 3.2. We thought that it still may be worth considering it as stop condition: we will empirically compare it to the relative approximation error, for example by discussing its possible influence on the behaviour of the algorithm under some particular initializations.

3.7.2 Hessian computation

We now proceed by showing all the partial derivatives obtained by deriving the two entries of the gradient by every possible entry of the two matrices U and V:

$$\frac{\partial \epsilon_A^2(U,V)}{\partial^2 u_{a,b}} = 2\sum_{j=1}^{n} v_{b,j}^2 \qquad \frac{\partial \epsilon_A^2(U,V)}{\partial^2 v_{a,b}} = 2\sum_{i=1}^{m} u_{i,a}^2$$

$$\frac{\partial \epsilon_A^2(U,V)}{\partial u_{a,d}\partial u_{a,b}} = 2\sum_{j=1}^{n} v_{b,j}v_{d,j} \qquad \frac{\partial \epsilon_A^2(U,V)}{\partial v_{c,b}\partial v_{a,b}} = 2\sum_{i=1}^{m} u_{i,a}u_{i,c}$$

$$\frac{\partial \epsilon_A^2(U,V)}{\partial u_{c,d}\partial u_{a,b}} = 0 \qquad \frac{\partial \epsilon_A^2(U,V)}{\partial v_{c,d}\partial v_{a,b}} = 0$$

$$\frac{\partial \epsilon_A^2(U,V)}{\partial v_{b,d}\partial u_{a,b}} = -2\left[a_{a,d} - u_{a,b}v_{b,d} - x_{a,d}\right] \qquad \frac{\partial \epsilon_A^2(U,V)}{\partial u_{c,a}\partial v_{a,b}} = -2\left[a_{c,b} - u_{c,a}v_{a,b} - x_{c,b}\right]$$

$$\frac{\partial \epsilon_A^2(U,V)}{\partial v_{c,d}\partial u_{a,b}} = 0 \qquad \frac{\partial \epsilon_A^2(U,V)}{\partial u_{c,d}\partial v_{a,b}} = 0$$

3.7.3 Positive definiteness of the Hessian

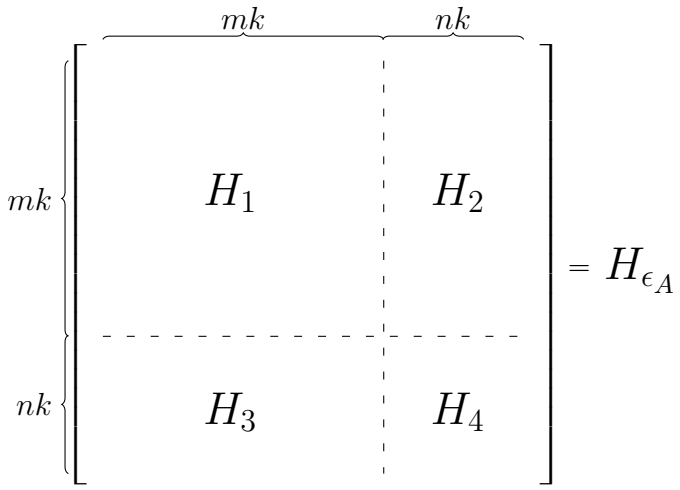
In order to highlight the structure of the Hessian matrix we computed we are now going to consider the vectorization we use in order to build our function $\bar{\epsilon_A} : \mathbb{R}^{mk+nk} \to \mathbb{R}$ as permuting (see **Definition 2**) the two input matrices:

$$\epsilon_A(U,V) = \bar{\epsilon_A}(vec^{\pi_1}(U) :: vec^{\pi_2}(V))$$

choosing π_1, π_2 such that we obtain the following:

$$vec^{\pi_1}(U) = (u_{1,1}, u_{1,2}, \ldots, u_{1,k}, u_{2,1}, \ldots, u_{m,k})$$
$$vec^{\pi_2}(V) = (v_{1,1}, v_{2,1}, \ldots, v_{k,1}, v_{1,2}, \ldots, v_{k,n})$$

that is, we vectorize U "by rows" and V "by columns". The resulting Hessian matrix H_{ϵ_A} is in the form:



Let us recall the column and row decompositions of U and V, this time decomposing each of them in both directions:

$$U = \begin{bmatrix} \cdots & u_1^{r\top} & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & u_m^{r\top} & \cdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ u_1^c & \vdots & u_k^c \\ \vdots & \vdots & \vdots \end{bmatrix}, \qquad V = \begin{bmatrix} \cdots & v_1^{r\top} & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & v_k^{r\top} & \cdots \end{bmatrix} = \begin{bmatrix} \vdots & \vdots & \vdots \\ v_1^c & \vdots & v_n^c \\ \vdots & \vdots & \vdots \end{bmatrix}$$

And then we can characterize H_1 as the block-diagonal matrix having m identical $k \times k$ matrices on its main diagonal, where each of them equal to:

$$2 \begin{bmatrix} (v_1^{r\top} v_1^r) & (v_1^{r\top} v_2^r) & \cdots & (v_1^{r\top} v_k^r) \\ (v_1^{r\top} v_2^r) & (v_2^{r\top} v_2^r) & \cdots & (v_2^{r\top} v_k^r) \\ \vdots & \vdots & \ddots & \vdots \\ (v_1^{r\top} v_k^r) & (v_2^{r\top} v_k^r) & \cdots & (v_k^{r\top} v_k^r) \end{bmatrix} = 2VV^\top$$

Similarly we have that H_4 is the block-diagonal matrix having n identical $k \times k$ matrices on its main diagonal, where each of them equal to:

$$2 \begin{bmatrix} (u_1^{c\top} u_1^c) & (u_1^{c\top} u_2^c) & \cdots & (u_1^{c\top} u_k^c) \\ (u_1^{c\top} u_2^c) & (u_2^{c\top} u_2^c) & \cdots & (u_2^{c\top} u_k^c) \\ \vdots & \vdots & \ddots & \vdots \\ (u_1^{c\top} u_k^c) & (u_2^{c\top} u_k^c) & \cdots & (u_k^{c\top} u_k^c) \end{bmatrix} = 2U^\top U$$

And finally we have $H_3 = H_2^\top = 2H_\Delta{}^\top$ and H_3 is structured as:

$$H_3 = 2 \overbrace{\left[\begin{array}{c:c:c:c} \overbrace{H^{\Delta}_{1,1}}^{k} & H^{\Delta}_{2,1} & \cdots & H^{\Delta}_{m,1} \\ \hdashline H^{\Delta}_{1,2} & H^{\Delta}_{2,2} & \cdots & H^{\Delta}_{m,2} \\ \hdashline \vdots & \vdots & \ddots & \vdots \\ \hdashline H^{\Delta}_{1,n} & H^{\Delta}_{2,n} & \cdots & H^{\Delta}_{m,n} \end{array}\right]}^{mk} \left.\vphantom{\begin{array}{c} a \\ b \\ c \\ d \end{array}}\right\} nk$$

where each $H^{\Delta}_{i,j}$ is defined as follows, let $\delta_{i,j} = a_{i,j} - x_{i,j}$:

$$H^{\Delta}_{i,j} = \begin{bmatrix} (u_{i,1}v_{1,j} - \delta_{i,j}) & 0 & \cdots & 0 \\ 0 & (u_{i,2}v_{2,j} - \delta_{i,j}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & (u_{i,k}v_{k,j} - \delta_{i,j}) \end{bmatrix}$$

And putting it all together (let $\otimes$ be the Kronecker product) we get:

$$H_{\epsilon_A} = 2 \left[\begin{array}{c:c} I_m \otimes (VV^{\top}) & H_{\Delta} \\ \hdashline H_{\Delta}{}^{\top} & I_n \otimes (U^{\top}U) \end{array}\right]$$

Since both $VV^{\top}$ and $U^{\top}U$ are symmetric, we get that the whole H_{ϵ_A} is symmetric, moreover, we have that both $VV^{\top}$ and $U^{\top}U$ are also positive semidefinite, thus H_1 and H_4 are also positive semidefinite.

Through application of **Lemma 18** we get that $H_{\epsilon_A} \succ 0 \iff \frac{1}{2}H_{\epsilon_A} \succ 0$ and so we can omit the multiplicative constant 2 when studying positive (semi)definiteness.

By application of **Lemma 21** we have that $VV^{\top}$ and $U^{\top}U$ are positive definite when $V^{\top}$ and U are tall-thin (which is always the case) and when they are rank k (which we assume to happen after few iterations), under the same assumptions, by means of **Lemma 20** we conclude that the two products are also invertible.

Under the aforementioned assumptions on the ranks of the parameters, let $W = (U^{\top}U)^{-1}$, since H_4 is block diagonal we have $(I_n \otimes (U^{\top}U))^{-1} = I_n \otimes W$.

Since $\frac{1}{2}H_{\epsilon_A}$ is symmetric and under those hypothesis $H_4 \succ 0$ we can apply the characterization for positive definitness given in **Proposition 16.1 of [6]** and conclude that:

$$H_{\epsilon_A} \succ 0 \iff \left[I_m \otimes (VV^\top) - H_\Delta (I_n \otimes W) {H_\Delta}^\top\right] \succ 0$$

and by expanding the right term we get:

$$H_\Delta (I_n \otimes W) {H_\Delta}^\top = H_\Delta \begin{bmatrix} W & 0 & \cdots & 0 \\ 0 & W & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & W \end{bmatrix} \begin{bmatrix} H^\Delta_{1,1} & H^\Delta_{2,1} & \cdots & H^\Delta_{m,1} \\ H^\Delta_{1,2} & H^\Delta_{2,2} & \cdots & H^\Delta_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ H^\Delta_{1,n} & H^\Delta_{2,n} & \cdots & H^\Delta_{m,n} \end{bmatrix} =$$

$$\begin{bmatrix} H^\Delta_{1,1} & H^\Delta_{1,2} & \cdots & H^\Delta_{1,n} \\ H^\Delta_{2,1} & H^\Delta_{2,2} & \cdots & H^\Delta_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ H^\Delta_{m,1} & H^\Delta_{m,2} & \cdots & H^\Delta_{m,n} \end{bmatrix} \begin{bmatrix} WH^\Delta_{1,1} & WH^\Delta_{2,1} & \cdots & WH^\Delta_{m,1} \\ WH^\Delta_{1,2} & WH^\Delta_{2,2} & \cdots & WH^\Delta_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ WH^\Delta_{1,n} & WH^\Delta_{2,n} & \cdots & WH^\Delta_{m,n} \end{bmatrix} =$$

$$\begin{bmatrix} J_{1,1} & J_{1,2} & \cdots & J_{1,m} \\ J_{2,1} & J_{2,2} & \cdots & J_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ J_{m,1} & J_{m,2} & \cdots & J_{m,m} \end{bmatrix} \qquad \textit{where} \quad J_{i,j} = \sum_{h=1}^{n} (H^\Delta_{i,h} W H^\Delta_{j,h})$$

And thus by substituting in $H^*_{\epsilon_A} = I_m \otimes (VV^\top) - H_\Delta (I_n \otimes W) {H_\Delta}^\top$ we get:

$$H_{\epsilon_A} \succ 0 \quad \iff \quad -\begin{bmatrix} (J_{1,1} - VV^\top) & J_{1,2} & \cdots & J_{1,m} \\ J_{2,1} & (J_{2,2} - VV^\top) & \cdots & J_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ J_{m,1} & J_{m,2} & \cdots & (J_{m,m} - VV^\top) \end{bmatrix} \succ 0$$

we first we have to check the symmetry of $H^*_{\epsilon_A}$:

Lemma 6. *The matrix $H^*_{\epsilon_A}$ is symmetric for any choice of U, V (satisfying the conditions for which $H^*_{\epsilon_A}$ is defined).*

Proof. We first show that $J_{i,j} = {J_{j,i}}^\top$, in order to do so we notice that:

$${J_{j,i}}^\top = \left[\sum_{h=1}^{n} (H^\Delta_{j,h} W H^\Delta_{i,h})\right]^\top = \sum_{h=1}^{n} (H^\Delta_{j,h} W H^\Delta_{i,h})^\top = \sum_{h=1}^{n} ({H^\Delta_{i,h}}^\top W^\top {H^\Delta_{j,h}}^\top)$$

and since each $H^\Delta_{i,h}$ is diagonal it is also symmetric, and applying Lemma 22 on $W = (U^\top U)^{-1}$ by knowing that $UU^\top$ is symmetric we get:

$${J_{j,i}}^\top = \sum_{h=1}^{n} ({H^\Delta_{i,h}}^\top W^\top {H^\Delta_{j,h}}^\top) = \sum_{h=1}^{n} (H^\Delta_{i,h} W H^\Delta_{j,h}) = J_{i,j}$$

which proves that every block (except for those on the diagonal) is aligned with its transpose, we are left to prove that every block on the diagonal is symmetric, which follows since $J_{i,i}$ is symmetric by what we just proved and since we know that $VV^\top$ is also symmetric. □

In order to check positive definitness we consider a generic vector $z \in \mathbb{R}^{mk}$ which we decompose as:

$$z = \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix} \qquad where \quad z_i \in \mathbb{R}^k \quad for\ i = 1 \dots m$$

and check for which values of U, V we get $z^\top H^*_{\epsilon_A} z > 0$, thus:

$$z^\top \left[I_m \otimes (VV^\top) - H_\Delta (I_n \otimes W) H_\Delta{}^\top \right] z = z^\top \left[I_m \otimes (VV^\top) \right] z - z^\top \left[H_\Delta (I_n \otimes W) H_\Delta{}^\top \right] z$$

expanding the first term we get:

$$\begin{bmatrix} z_1{}^\top & z_2{}^\top & \cdots & z_m{}^\top \end{bmatrix} \begin{bmatrix} VV^\top & 0 & \cdots & 0 \\ 0 & VV^\top & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & VV^\top \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} =$$

$$\begin{bmatrix} z_1{}^\top VV^\top & z_2{}^\top VV^\top & \cdots & z_m{}^\top VV^\top \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

which gives

$$z^\top \left[I_m \otimes (VV^\top) \right] z = \sum_{h=1}^{m} \left(z_h{}^\top (VV^\top) z_h \right) \tag{6}$$

while expanding the second term gives:

$$\begin{bmatrix} z_1{}^\top & z_2{}^\top & \cdots & z_m{}^\top \end{bmatrix} \begin{bmatrix} J_{1,1} & J_{1,2} & \cdots & J_{1,m} \\ J_{2,1} & J_{2,2} & \cdots & J_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ J_{m,1} & J_{m,2} & \cdots & J_{m,m} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} \left(\sum_{l=1}^m z_l{}^\top J_{l,1} \right)^\top \\ \left(\sum_{l=1}^m z_l{}^\top J_{l,2} \right)^\top \\ \vdots \\ \left(\sum_{l=1}^m z_l{}^\top J_{l,m} \right)^\top \end{bmatrix}^\top \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix}$$

which gives

$$z^\top \left[H_\Delta (I_n \otimes W) H_\Delta{}^\top \right] z = \sum_{h=1}^{m} \sum_{l=1}^{m} \left(z_l{}^\top J_{l,h} z_h \right) \tag{7}$$

and substituting equations 6, 7 in $z^\top H^*_{\epsilon_A} z$ we get

$$\sum_{h=1}^{m} \left(z_h{}^\top (VV^\top) z_h \right) - \sum_{h=1}^{m} \sum_{l=1}^{m} \left(z_l{}^\top J_{l,h} z_h \right) = \sum_{h=1}^{m} \left[z_h{}^\top (VV^\top) z_h - \sum_{l=1}^{m} \left(z_l{}^\top J_{l,h} z_h \right) \right] \tag{8}$$

Since we are under the hypotesis for which $VV^\top > 0$, every $z_h{}^\top (VV^\top) z_h$ is for sure strictly positive, and therefore so is the term given from equation 6. In order for 8 to be strictly positive as well we need the second term, that is the one given from equation 7, to be either:

1. Negative (or zero).
2. Positive while satisfying $\sum_{h=1}^{m}(z_h^\top(VV^\top)z_h) > \sum_{h=1}^{m}\sum_{l=1}^{m}\left(z_l^\top J_{l,h} z_h\right)$

Let us now study the second term by expanding a generic $J_{i,j}$:

$$z_i^\top J_{i,j} z_j = z_i^\top \left[\sum_{h=1}^{n} H_{i,h}^{\Delta} W H_{j,h}^{\Delta}\right] z_j = \sum_{h=1}^{n}\left(z_i^\top H_{i,h}^{\Delta} W H_{j,h}^{\Delta} z_j\right)$$

Since we are under our original hypothesis for which $U^\top U \succ 0$ we can apply **Lemma 25** and conclude $W \succ 0$. A sufficient condition for the second term to be nonpositive would be to have $J_{i,j} \preceq 0$ for every combination of i, j, but we show that this is not possible, since:

Lemma 7. *Under the hypothesis for which $H_{\epsilon_A}^*$ is defined we have $J_{i,i} \succ 0$ for every choice of $i = 1, \ldots, m$.*

Proof. We first recall that $J_{i,i}$ is symmetric, as shown in the proof of Lemma 6, and by recalling the definition of $J_{i,i}$ we get:

$$z^\top J_{i,i} z = z^\top \left[\sum_{h=1}^{n} H_{i,h}^{\Delta} W H_{i,h}^{\Delta}\right] z = \sum_{h=1}^{n}(z^\top H_{i,h}^{\Delta}) W (H_{i,h}^{\Delta} z)$$

and since every $H_{i,h}^{\Delta}$ is diagonal, it is also symmetric, thus:

$$z^\top J_{i,i} z = \sum_{h=1}^{n}(z^\top H_{i,h}^{\Delta^\top}) W (H_{i,h}^{\Delta} z) = \sum_{h=1}^{n}(H_{i,h}^{\Delta} z)^\top W (H_{i,h}^{\Delta} z)$$

and since by Lemma 25 we know that $W \succ 0$ we can conclude that $z^\top J_{i,i} z > 0$, which completes the proof. □

Given the multitude of possible combinations for each term of the sum in equation 8, where each $J_{l,h}$ is in turn a sum that depends on the entries of U and V, it is very hard to characterize these matrices with the tools we have available.

3.8 Differentiability of our target function

We now discuss differentiability of the function $\epsilon_A(X)$, we recall its definition and compute the partial derivative w.r.t. each entry $x_{i,j}$ of X, by considering that its domain (the matrix vector space $\mathbb{R}^{m\times n}$) is isomorphic to the vector space $\mathbb{R}^{mn}$ and considering the associated $\epsilon_A : \mathbb{R}^{mn} \to \mathbb{R}$ (defined over the vectorized matrices, which uses $\|\cdot\|_2$ by isometry):

$$\frac{\partial \epsilon_A}{\partial x_{i,j}} = \frac{\partial \|A - X\|}{\partial x_{i,j}} = \frac{\partial\left[\sqrt{\sum_{k,l=1}^{\leqslant m,n}(a_{k,l} - x_{k,l})^2}\right]}{\partial x_{i,j}}$$

and by applying the chain rule and decomposing the sum:

$$\frac{\partial\left[\sqrt{\sum_{k,l=1}^{\leqslant m,n}(a_{k,l}-x_{k,l})^2}\right]}{\partial\left[\sum_{k,l=1}^{\leqslant m,n}(a_{k,l}-x_{k,l})^2\right]}\cdot\frac{\partial\left[\sum_{k,l=1}^{\leqslant m,n}(a_{k,l}-x_{k,l})^2\right]}{\partial x_{i,j}}=$$

$$=\frac{1}{2\|A-X\|}\cdot\left(\frac{\partial\left[\sum_{k,l=1}^{\neq i,j}(a_{k,l}-x_{k,l})^2\right]}{\partial x_{i,j}}+\frac{\partial(a_{i,j}-x_{i,j})^2}{\partial x_{i,j}}\right)=$$

$$=\frac{1}{2\|A-X\|}\cdot\frac{\partial(a_{i,j}-x_{i,j})^2}{\partial(a_{i,j}-x_{i,j})}\cdot\frac{\partial(a_{i,j}-x_{i,j})}{\partial x_{i,j}}=$$

$$=\frac{1}{2\|A-X\|}\cdot 2(a_{i,j}-x_{i,j})\cdot(-1)=-\frac{a_{i,j}-x_{i,j}}{\|A-X\|}$$

And thus our gradient (let us consider X and A to be the corresponding vectors mapped in $\mathbb{R}^{mn}$) is $\nabla\epsilon_A(X)=\dfrac{X-A}{\epsilon_A(X)}$. This function is well defined over the entire domain except in $X=A$ where the denominator is null, moreover, since both the numerator and the denominator are continuous functions (as every norm is continuous) it follows that ϵ_A is differentiable on all $\mathbb{R}^{m\times n}\backslash\{A\}$. To study the second order partial derivatives we first check $\dfrac{\partial^2\epsilon_A}{\partial^2 x_{i,j}}$ and then $\dfrac{\partial^2\epsilon_A}{\partial x_{k,l}\partial x_{i,j}}$, staring from the gradient we just obtained:

$$\frac{\partial^2\epsilon_A}{\partial^2 x_{i,j}}=\frac{\partial\left[-\dfrac{(a_{i,j}-x_{i,j})}{\epsilon_A(X)}\right]}{\partial x_{i,j}}$$

and by deriving the fraction, and substituting the previous result:

$$-\frac{\dfrac{\partial(a_{i,j}-x_{i,j})}{\partial x_{i,j}}\epsilon_A(X)-\dfrac{\partial\epsilon_A(X)}{\partial x_{i,j}}(a_{i,j}-x_{i,j})}{\left(\epsilon_A(X)\right)^2}=$$

$$=-\frac{-\epsilon_A(X)+\dfrac{(a_{i,j}-x_{i,j})}{\epsilon_A(X)}(a_{i,j}-x_{i,j})}{\left(\epsilon_A(X)\right)^2}=$$

$$=\frac{\epsilon_A(X)-\dfrac{(a_{i,j}-x_{i,j})^2}{\epsilon_A(X)}}{\left(\epsilon_A(X)\right)^2}=\frac{\epsilon_A(X)^2-(a_{i,j}-x_{i,j})^2}{\left(\epsilon_A(X)\right)^3}$$

analogously, we compute $\frac{\partial^2 \epsilon_A}{\partial x_{k,l} \partial x_{i,j}}$ as:

$$\frac{\partial \left[-\frac{(a_{i,j} - x_{i,j})}{\epsilon_A(X)} \right]}{\partial x_{k,l}} = -\frac{\frac{\partial (a_{i,j} - x_{i,j})}{\partial x_{k,l}} \epsilon_A(X) - \frac{\partial \epsilon_A(X)}{\partial x_{k,l}} (a_{i,j} - x_{i,j})}{\left(\epsilon_A(X)\right)^2} =$$

$$= -\frac{0 \cdot \epsilon_A(X) + \frac{(a_{i,j} - x_{k,l})}{\epsilon_A(X)} (a_{i,j} - x_{i,j})}{\left(\epsilon_A(X)\right)^2} = -\frac{(a_{i,j} - x_{k,l})(a_{i,j} - x_{i,j})}{\left(\epsilon_A(X)\right)^3}$$

And as for the first order partial derivatives, they are composed of continuous functions only and thus they are continuous on all the domain (except for A in which they are not defined), thus ϵ_A is differentiable twice on all $\mathbb{R}^{m \times n} \backslash \{A\}$.

4 Algorithms

4.1 QR with Column Pivoting

In this subsection we give a brief explanation of the algorithm we use to obtain the QR with Column Pivoting factorization (see chapter 5.4.2 of [1]).

Let $X \in \mathbb{R}^{m\times n}$, $rk(X) = r$. The QR with Column Pivoting algorithm works as follows. imagine to have completed the (k-1)-th step, thus to have computed the first k-1 householder matrices and the first k-1 permutation matrices.

$$H_{k-1}H_{k-2}...H_1X\Pi_1\Pi_2...\Pi_{k-1} = \begin{pmatrix} {R_{11}}^{k-1} & {R_{12}}^{k-1} \\ 0 & {R_{22}}^{k-1} \end{pmatrix}$$

Where

- ${R_{11}}^{k-1} \in \mathbb{R}^{(k-1)\times(k-1)}$ is non-singular and upper-triangular;
- ${R_{22}}^{k-1} \in \mathbb{R}^{(m-k+1)\times(n-k+1)}$

Let's now consider a column partitioning of ${R_{22}}^{k-1}$ as

$${R_{22}}^{k-1} = \left[{z_k}^{k-1} \mid \cdots \mid {z_n}^{k-1} \right]$$

And let $p \in [k, n]$ be the smallest index such that ${z_p}^{k-1}$ has maximum norm with respect to the other column vectors of ${R_{22}}^{k-1}$:

$$\left\| {z_p}^{k-1} \right\| = \max \left\{ \left\| {z_k}^{k-1} \right\|, \ldots, \left\| {z_n}^{k-1} \right\| \right\}$$

If $rk(X) = r = k-1$ then $\forall i \in [k, n]$ it holds that $\left\| {z_i}^{k-1} \right\| = 0$, because we are considering the sub-matrix $R^{k-1}[r+1 : m, r+1 : n]$ that by definition, for this factorization, is a zero matrix. This is thus our termination step.
If $k - 1 < r$, instead, we define Π_k as the identity matrix with columns p and k exchanged, and H_k as the householder matrix that zeroes all the elements of $(R^{k-1}\Pi^k)[k+1 : m, k]$, namely the subdiagonal entries of column k. R^k is then defined as

$$R^k = H_k R^{k-1} \Pi_k$$

The complete factorization is thus given by

$$H_r H_{r-1}...H_1 X\Pi_1\Pi_2...\Pi_r = R \equiv X\Pi = QR \equiv X = QR\Pi^\top$$

5 Choice of our input data

The input data of our algorithm consists of two different datasets of matrices:

- The first dataset is composed by randomly genarated matrices using MATLAB's library functions. We generate matrices having different shapes (square, tall-thin or short-fat) and of different dimensions, to test how well our algorithm scales, both full-rank and rank-deficient. For each matrix we try out different values of k in order to evaluate how our approximation deviates from the optimal one (that is the one given by the truncated SVD). In order to generate a random matrix with a given rank we picked two random orthogonal matrices and a random diagonal matrix using MATLAB's library functions and proceeded by truncating the obtained SVD.

- The second dataset contains pictures imported as matrices with the MATLAB function `imread`. The pictures belong to CDnet 2014 [7], a dataset made up of frames captured by indoor and outdoor cameras. Each frame is a *jpg* image that is imported in MATLAB as a 3-dimensional array, that corresponds to the RGB model; since we work with bi-dimensional arrays (that is matrices), the images have been converted to grayscale with the function `rgb2gray`. There frame are gathered from a total of 23 different cameras, grouped in 11 categories.
  We aim to use such pictures as a benchmark for our algorithm, to understand in more of a "graphic way" how accurately our approximation depicts the original image with respect to the optimal approximation, and to investigate whether our algorithm would behave differently with respect to it being executed on random matrices.

6 Implementation details

The algorithm was implemented in MATLAB. The code is composed by two main functions, `MyQRP` (that is the implementation of the QR factorization with column pivoting as described in Section 4.1) and `LowRankAlgo` (that implements our overall main iterative algorithm to find the low rank approximation). `MyQRP` has the following signature:

```
[Q,R,P,r]= MyQRP(Y)
```

Where:

- `Y` is the matrix for which the QR factorization with pivoting has to be computed;

- `Q`, `R` and `P` are the three matrices that the algorithm computes, being respectively the Q and R of the QR factorization and the permutation matrix Π;

- `r` is the rank of R, that is the number of rows of columns of its invertible square sub-matrix, namely the R_{11} we mentioned before. This is possible because the QR factorization with column pivoting is a rank revealing factorization.

`LowRankAlgo` has the following signature:

```
[U, V, it, errs, gradnorms] = LowRankAlgo(A, k, max_it, err_eps, grad_eps,
                stop_c_type, init_t, init_w, printsteps)
```

Where:

- `A` is the target matrix and `k` the target rank to which we want to approximate `A`;
- `max_it` is the maximum number of iterations (the value default for it is 1000);
- `err_eps` and `grad_eps` are respectively is the relative approximation error threshold and the gradient norm threshold; at each iteration, depending on the type of the stop condition chosen (either the difference in realtive error or the gradient norm) they are compared with the respective values in order to check for termination;
- `stop_c_type` is the stop condition type. It can take the following values:

  approxerror to compare the difference in relative error with its threshold as stop condition;

  gradientnorm to compare the gradient norm with its threshold as stop condition;

  both for which both the above mentioned stop conditions have to be satisfied;

  any for which any of the above mentioned stop conditions have to be satisfied.

- `init_t` is indicates the logic behind the choice of the initial matrix V_0. It can take the following values:

  random to initialize V_0 as a random matrix;

  randfull to initialize V_0 as a random matrix, with the additional constraint for it to be full-rank;

  eye to initialize V_0 as the identity matrix;

  randeye to initialize V_0 as a diagonal matrix with random entries on the diagonal;

  eyedistinct to initialize V_0 as an "identity matrix" with no repeating columns (that is precisely the identity matrix when V_0 is square, or a concatenation of identity matrices multiplied each time by a different scalar when V_0 is rectangular);

  eyeextended as for `eyedistinct` but we only concatenate identity matrices, without multiplying each of them;

- `item_w` determines how the w vector has to be initialized in the minimization sub-steps. It can take the following values:

  zeros to initialize w as a zero vector;

  ones to initialize w as a vector of ones;

  rand to initialize w as a random vector;

- `printsteps` determines how detailed will be the information printed during the execution of the algorithm. It can take the following values:

0 to print no information at all;

1 to print a summary log at the end of the algorithm execution;

2 to print detailed information at each iteration of the algorithm.

We decided to implement various initialization strategies for V_0 in order to observe the impact of ensuring different structural properties on said matrix, starting from `random` (which offers no particular structure) we have that:

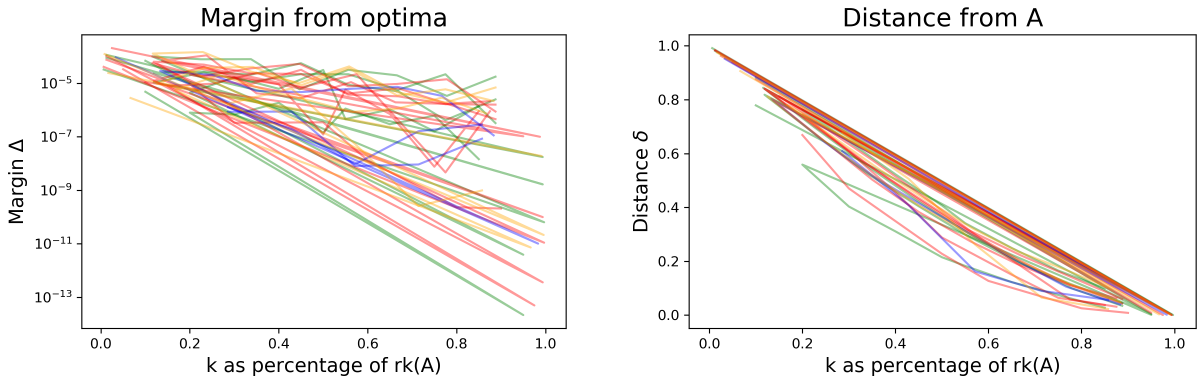
randfull	ensures that the matrix is full-rank;
eye	ensures that the matrix is both full-rank and diagonal;
randeye	ensures that the matrix is diagonal;
eyeextended	ensures that the matrix is full-rank and has no zero columns;
eyedistinct	ensures that the matrix is full-rank and has no zero or duplicated columns;

According to some preliminary tests, we didn't notice any particular differences between using one of the non-random initializations for V_0 both on the matter of the number of steps required for convergence and on the one of the difference from the optimal solution when approximating matrices which do not exhibit a particular structure. For this reason, we decided to adopt **eyedistinct** initialization as the default one over `eye`, since by putting $V_0 = I$ it is easy to observe that the first minimization problem becomes $\|A - UI\|$, and thus U_0 is the matrix composed of the first k columns of A, which leads to a bad behaviour if, for example, A has all zero entries in said columns. We also conducted some test about the initialization of w observing no particular differences: we thus decided to choose the zero vector as default.

7 Preliminary experiments

For each input task (that is, a pair $\langle A, k \rangle$ asking to approximate the matrix A to rank k) we examined some metrics such as:

- The computation time t of our algorithm, producing $X = UV$ as our approximation;
- The number of iterations n_{it} of our algorithm;
- The average time for iteration $t_{it} = t/n_{it}$ of our algorithm;
- The computation time t^* of MATLAB's `svd` used to compute the optimal solution A^*;
- The distance between X and A, as $\delta = \|A - X\| / \|A\|$;
- The distance between A^* and A, as $\delta^* = \|A - A^*\| / \|A\|$;
- The margin $\Delta = (\delta - \delta^*)$ between the two distances.



(a) Plot of the difference Δ among different matrices when varying the value of k.

(b) Plot of the value δ among different matrices when varying the value of k.

Figure 1: Plots comparing behaviours when varying input structure: different colors indicate different characteristics (red: square/fullrank; orange: square/singular; green: rectangular/fullrank; blue: rectangular/singular).

7.1 Impact of inputs' shape and rank

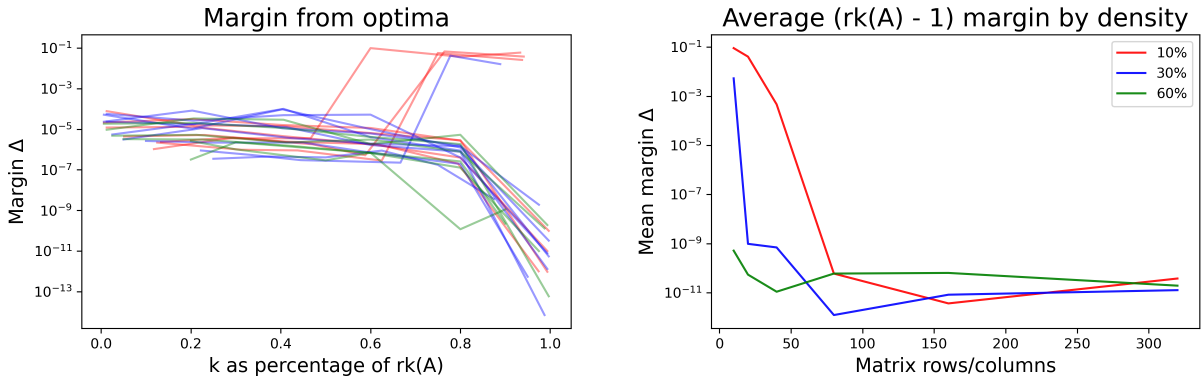
We first assessed the problem of determining how the structure of A impacts the quality of the resulting approximation (in terms of square, rectangular, fullrank and singular inputs); in order to do so we tested on a dataset of random matrices varying size, shape and rank and tried approximating each matrix for different values of k, picking 10 uniformly distributed values ranging from $(rk(A) - 1)$ down to 2, the dataset for this task was composed by 3 randomly generated matrices for each of the following:

- Square Fullrank matrices of sizes $M \times M$;
- Square Singular matrices of sizes $M \times M$, each having rank equal to $\frac{3}{4}M$;
- Rectangular Fullrank matrices of sizes $M \times 2M$;
- Rectangular Singular matrices of sizes $M \times 2M$, each having rank equal to $\frac{3}{4}M$;

for M taking values in $\{10, 20, 40, \ldots, 320\}$, giving a total of 72 matrices, each of which we approximated using our algorithm for values of k uniformly picked in the range $\big[(rk(A) - 1), 2\big]$ (notice that for matrices having $M = 10$ we picked less values of k), giving a total of 666 approximation tasks we considered.

In order to assess the quality of approximation among our tasks $T_i = (A_i, k_i)$ we decided to consider the sequences $T_{j_1}, T_{j_2}, \ldots, T_{j_{10}}$ of tasks having $A = A_{j_1} = A_{j_2} = \cdots = A_{j_{10}}$ (that is, successive approximations of the same target matrix when varying k) and plotted the two metrics Δ and δ as a function of the ratio $k_i/rk(A)$ (i.e. the percentage of $rk(A)$ at which we approximate), as shown in Figure 1.

We didn't observe any significant change in the quality of approximation when varying the structure between rectangular/square and fullrank/singular, and thus we decided to proceed by working on fullrank-square matrices from here on. The two plots in Figure 1 give us some interesting insights on the algorithm:



(a) Plot of the difference Δ among different matrices when varying the value of k.

(b) Plot of the mean margin among inputs when $k = rk(A) - 1$, by varying the size of A.

Figure 2: Plots comparing behaviours when varying input sparsity: different colors indicate different densities (red: 10%; blue: 30%; green: 60%).

- Figure 1a shows a decrease in the margin Δ as k approaches $rk(A)$, suggesting that our algorithm gives better approximations as k gets bigger.
- Figure 1b shows a linear decrease in the distance δ which is coherent with the fact that our matrices were randomly generated, and thus we expect the singular values of A to decrease uniformly.

Note that the executions above were obtained by using default values for the algorithm's parameters, which means using `err_eps=1e-6` and `stop_c_type='approxerror'`, this causes the values of δ we obtained to be precise up to about the 6th decimal digit w.r.t. that of the optimal solution to which our algorithm converged; this choice of parameters relates to the fact that our margins are of the order of 10^{-6}, since a margin of such order would be likely if our algorithm converged to a global optimum.

7.2 Impact of input sparseness

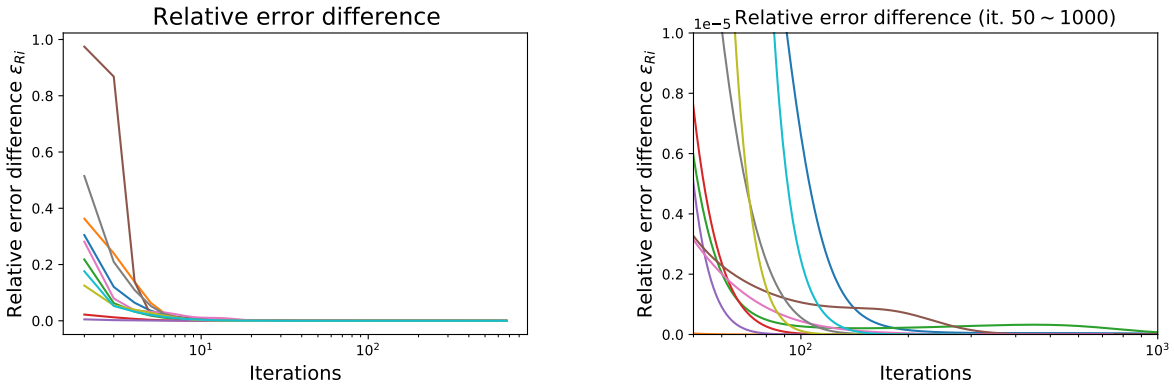
We also assessed the problem of determining whether the sparsity of A impacts the quality of the resulting approximation, in order to do so we generated a dataset of random matrices consisting of randomly generated square matrices of increasing size and density (having row count $10, 20, 40, \ldots, 320$, and density $0.1, 0.3$ and 0.6) for a total of 3 matrices for each size/density combination. We then proceeded by approximating each matrix A to 6 values of k equally spaced in $[2, rk(A)]$ (for smaller inputs we had to pick fewer values), for a total of 317 Tasks, and compared the resulting approximations with the optimal ones as we did before.

Figure 2a shows no particular trend with smaller values of k, while for values close to $rk(A)$ we noticed that smaller matrices (having row count $10, 20$) tend to produce worse approximations, while for bigger matrices (having row count $\geqslant 80$) we observed margins several orders of magnitude smaller than what we expected (down to $1e - 18$); we also noticed that this behaviour is increased as the sparsity of the input increases, as shown in

Figure 2b. Overall we think that this behaviour should not be a problem for our algorithm, as for smaller inputs we could easily resort to computing the truncated SVD, while we expect our algorithm to be convenient when applied to bigger inputs, on which we observe no decrease in the quality of approximation.

7.3 Setting err_eps and grad_eps

We also exploited preliminary experiments to analyze the behaviour of our algorithm in terms of relative error difference ϵ_{R_i} decrease and gradient norm trend. In particular, we wanted to understand whether we could set the two parameters `err_eps` and `grad_norm` in a way that allowed us to reach, in subsequent experiments, a solid trade-off between accuracy of the solution and length of the execution (in temporal terms).
We now discuss the choice of the default value for `err_eps` based on some observations about its trend. The experiments we ran in this phase were 74, chosen uniformly at random among the 666 tasks described in section 7.1. We chose to use *approxerror* as `stop_c_type`, but we anyway computed and stored the values of the gradient norm to further use it for the analyses related to it. We set `err_eps` to `1e-12` and `max_it` to 10000.



(a) Trend of the relative approximation error difference

(b) Closeup of the relative approximation error difference trend, for iterations from 50 to 1000

Figure 3: Trend of the relative approximation error difference for 10 tasks. The left one shows the general trend, while the right one is a closeup on a subset of the iterations

Figure 3a shows the trend of ϵ_{R_i} with respect to the iterations of the algorithm. The x-axis is in log-scale to achieve a better readability of the plot. For the same reason, we only plotted 10 lines out of the 74 tasks. We can clearly see how the error tends to drop significantly and steeply during the first ~10 iterations. In order to understand what happens next, it is necessary to "zoom" on certain subsets of iterations, to catch parts of the algorithm where ϵ_{R_i} tends to be of a certain order of magnitude. By looking at **Figure 3b**, for example, we can see how it drops again of an order of magnitude after the first ~100 iterations. This kind of plots is certainly useful to get a grasp of the general trend of the relative approximation error difference during the execution of the algorithm, but it doesn't help in understanding the point where a good accuracy is reached after a not too big number of

iterations, and from which too many other iterations would be needed to reach a smaller order of magnitude.
Our purpose is better satisfied by the CDF plot. Through it, we aim to show the proportion of iterations where the value of ϵ_{R_i} is of a certain order of magnitude.

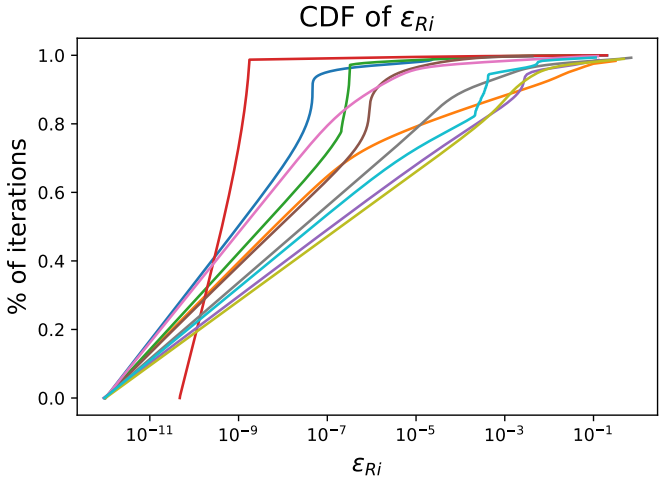
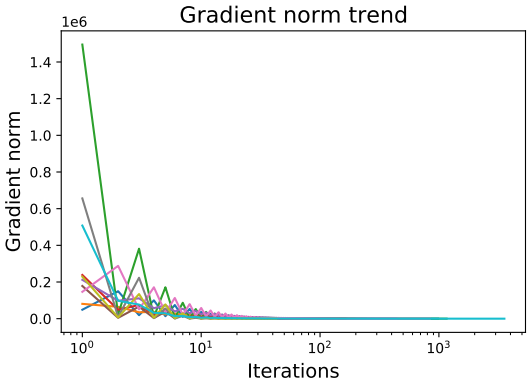
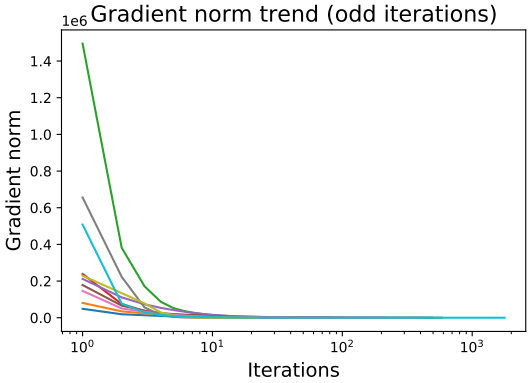


Figure 4: CDF plot of ϵ_{R_i} for 10 out of 74 tasks

Figure 4 is a comprehensive CDF plot of ϵ_{R_i} for the same 10 tasks of the previous discussion. For each task, which is represented by a colored curve, on the ordinates we have the percentage of iterations for which the relative approximation error difference is smaller or equal with respect to the correspondent value on the x-axis. As is clear from the plot, for half of the tasks it occurs that for 40%∼60% of the iterations the error is of the order of magnitude of 10^{-6} or smaller, while for the other tasks the same happens for 60%∼80% of the iterations, and for 5 tasks (respectively the ones represented by the red, blue, green, brown and pink lines) the percentage is even more than 80%. This means that our algorithm generally takes the smaller part of its time to get to a ϵ_{R_i} of the order of 10^{-6}, and the majority of time is taken for the aforementioned error to decrease. For all tasks, a solid 10% (or more) of the iterations is needed to make ϵ_{R_i} drop from 10^{-6} to 10^{-7}.
Based on these observations, we decided that for our further experiments 10^{-6} could be a good trade-off between achieving a satisfactorily accurate solution and not take too many iterations to compute the result.



(a) Trend of the gradient norm



(b) Trend of the gradient norm, only considering odd iterations

Figure 5: Trend of the gradient norm for 10 tasks. The left plot shows the general oscillatory behaviour, while the right one shows how, taking into consideration only odd iterations, the trend appears to be decreasing

Let us now analyze the gradient norm trend more in detail. All the considerations we make from now on are related to the gradient norm taken 2 iterations at a time, since as it can be seen in **Figure 5** the trend appears to be generally decreasing every other iteration. We first carried out some initial experiments based on the previously mentioned 10 task as basis, and by setting the gradient stop condition threshold to `1e-12`. We noticed that after a steep descent the gradient norm tended to flatten out and oscillate around values of the same order of magnitude. We thus decided to take a step further and analyze the full trend, by setting `grad_eps` to `1e-15`.

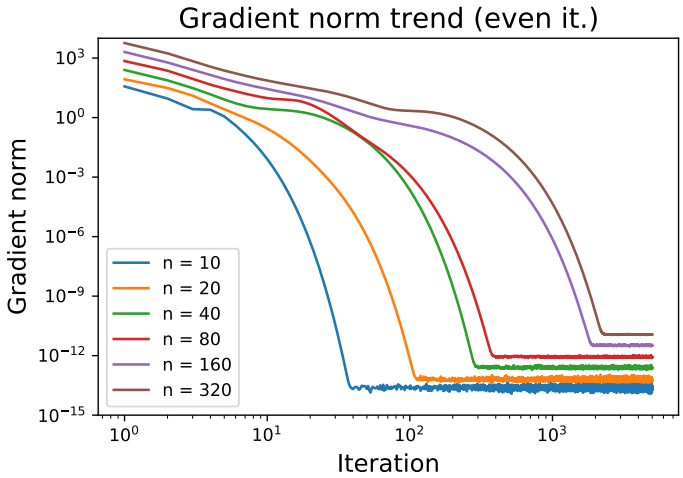


Figure 6: Trend of the gradient norm by considering even iterations and growing-size matrices

We executed some trials on 6 square random fullrank matrices of doubling size n in the interval [10, 320], with k taken as half their rank, to check whether the matrix size was in any way related to the order of magnitude the gradient norm oscillated around. The plot in

Figure 6, where both axes are in log-scale to be able to analyze the global behaviour, shows how the gradient norm tends to reach smaller orders of magnitude for smaller matrices. In particular, for the matrices having size 10×10, 20×20 and 40×40 the gradient norm reaches pretty quickly a satisfying value in the order of 10^{-14}, whereas as the size grows the point around which the gradient norm flattens and oscillates tends to get more and more further away from 0. To explain this, let us consider the fact that as the matrix size grows, so does the length of the gradient, the computation of which requires a considerable number of matrix-vector products. As the number of the possibly perturbed element grows, the norm itself of the gradient will grow as well. Since we didn't want the stop condition of our further experiments to rely on the size of the matrices, by taking into account these preliminary empirical results, we determined that ϵ_{R_i} was the best general choice as our stop condition.

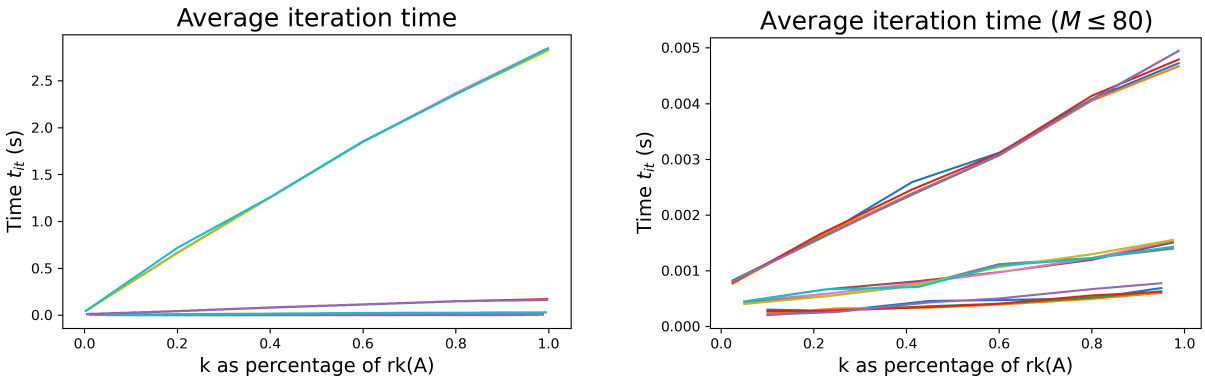
8 Results

8.1 Complexity evaluation

We checked how our algorithm scales w.r.t. both the dimensions of A and the value of k, in order to verify the theoretical behaviour we obtained in **Section 3.1**, that is, each iteration has an asymptotical time cost of $\mathcal{O}(mnk)$.

One first consideration is that our algorithm has a structure which alternates between the two dimensions m and n, since we proceed by alternatively working on one of the two matrices U and V; by considering the average time for one iteration t_{it} as the time required for the whole computation divided by the number of iterations (we are ignoring the overhead due to the initialization and final steps of the algorithm) we would, in principle obtain a cost which is an average of the costs of the two distinct kinds of minimization steps, but since our function mnk is invariant w.r.t. swapping m and n this should not impact our analysis.

We proceeded by working on square fullrank matrices and by considering an initial dataset of 180 tasks obtained by generating matrices having $20, 40, \ldots, 640$ rows and columns, 5 for each size, and approximating each one for 6 distinct values of k, uniformly picked in $[2, rk(A) - 1]$. We show in Figure 7 the measured execution time as we increase the value of k, from which it is evident that when m, n are fixed, then the cost scales linearly with k.



(a) Plot of the time t_{it} among different matrices when varying the value of k.

(b) Plot of the time t_{it} among different matrices having less than 80 rows and columns.

Figure 7: Plots comparing the average execution time for one iteration when varying k.

Another interesting result is that shown in Figure 8, where it is clear that the total execution time quickly decreases as k gets very close to $rk(A)$, this is due to the fact that our algorithm converges in few iterations for such values of k.

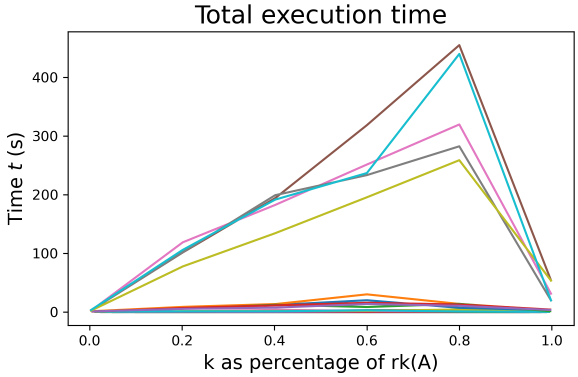


Figure 8: Plot of the total execution time when varying k.

We then proceeded to run our algorithm on another dataset composed of 105 tasks obtained by randomly generating square fullrank matrices having row count $150, 200, \ldots, 450$, three for each size, and approximating each for every value of k in $\{2, 10, 20, 50, 75\}$. We collected the execution times t_{it} and computed the average time $\bar{t_{it}}$ for each pair (m, k). In order for our hypothesis to be verified we expected $\bar{t_{it}}$ to grow quadratically w.r.t. to m (as we are working with square matrices) when the value of k is fixed; we thus show in Figure 9 the square root $\sqrt{\bar{t_{it}}}$ when varying m, for various values of k.

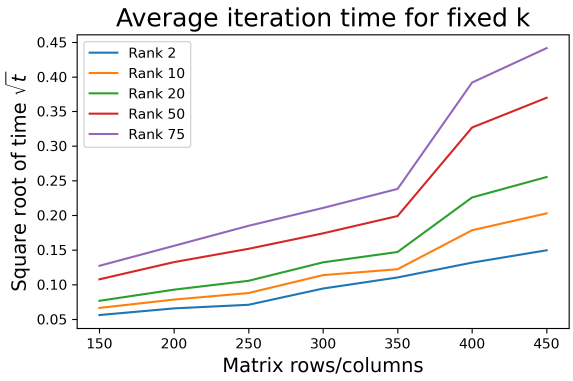


Figure 9: Plot of the square root of the average iteration time when varying m.

We notice that the growth appears linear up to $m = 350$, then has a steep increase, and then decreases again, we think that this behaviour is due to the fact that the execution time is impacted by memory size and the steep increase is related cache saturation; overall the growth appears to be linear, thus confirming the theoretical cost of $\mathcal{O}(mnk)$ we obtained in **Section 3.1**.

We also verified the results we cited about the algorithm having linear convergence by plotting the distance to the solution we are converging to at each iteration for square matrices of various size (up to 300 rows/columns), which we show in **Figure 10**, it is clear from the plot that the distance follows a straight line in a logarithmic scale, which shows that the convergence rate is, in fact, linear.

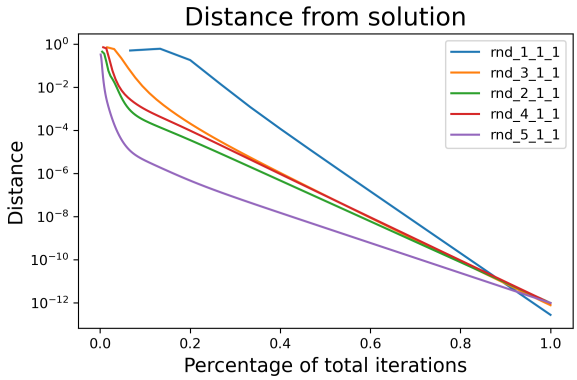


Figure 10: Log plot of the distance from the found solution at each iteration for different tasks.

8.2 Time impact of the m/n ratio

We analyzed how the execution time changes when it comes to rectangular matrices w.r.t. square ones; from a theoretical point of view our results shown in Section 3.1 suggest that from an asymptotic perspective we should observe no variation when dealing with rectangular matrices instead of square ones, as long as the product mn remains constant. We decided to test our algorithm by considering random matrices of various dimensions and their respective m/n ratios:

- 3 matrices of size 256×256: having $m/n = 1$;
- 3 matrices of size 512×128: having $m/n = 4$;
- 3 matrices of size 1024×64: having $m/n = 16$;
- 3 matrices of size 2048×32: having$m/n = 64$;

Each of which generated as full rank, and approximating every input to rank $k = 16$, in order to evaluate the average iteration time t_{it}. We computed the average result for each size and plotted the resulting values in **Figure 11**.

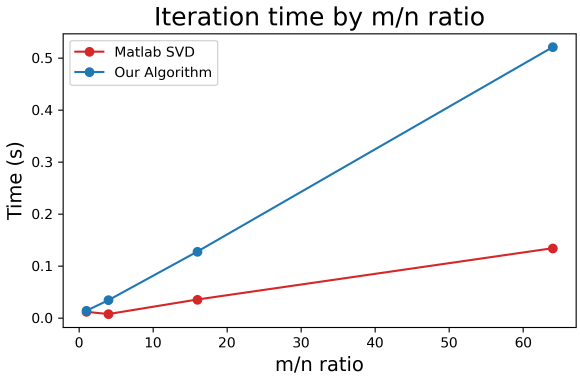


Figure 11: Plot of the average iteration time t_{it} when varying m/n.

The results do not match the theoretical complexity of $\mathcal{O}(mnk)$, which would cause the blue line to be flat, we gave some explanation on why this might happen:

Big-O notation The asymptotical bound $\mathcal{O}(mnk)$ ignores some lower-order terms which might cause the actual execution time to be significantly different when one of the two dimensions is significantly bigger than the other.

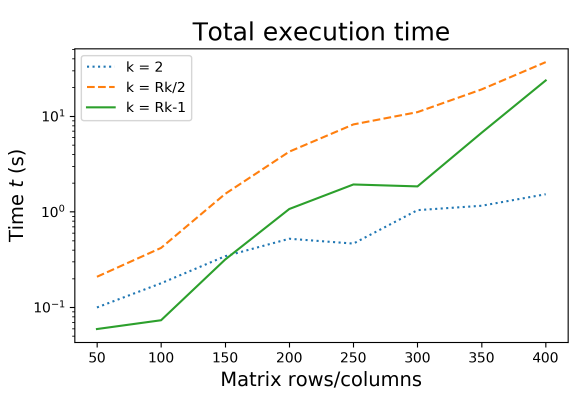
Memory impact If we consider matrices with a high m/n ratio such as when $m = 2048$ and $n = 32$ and a fixed value for k (in our case $k = 16$) our algorithm ends up alternating between two matrices U, V having a significantly different number of entries (in our example we would obtain a matrix U of size 2048×16 which has exactly half of the number of entries of A) and this might impact the execution time, since it requires a higher number of transfers between memory levels. We observe, in fact, that the number of memory transfers should be lower bounded by $b(\#V + \#U) \propto (mk + kn) \propto (m + n)$ where $\#V$ and $\#U$ are the memory occupancies of U and V respectively and b is some machine-dependent constant.

Since we are working with a constant $mn = T$, let $x = m/n$ we can obtain $\frac{T}{x} = \frac{mnn}{m} = n^2$ and $Tx = \frac{mnm}{n} = m^2$ and by substitution $(m + n) = (\sqrt{m^2} + \sqrt{n^2}) = \sqrt{Tx} + \sqrt{\frac{T}{x}}$ which is strictly increasing for $x > 1$ (and also for $x < 1$); which suggests us that the number of cache misses in our execution is, in fact, increasing as the matrices in our example get thinner.

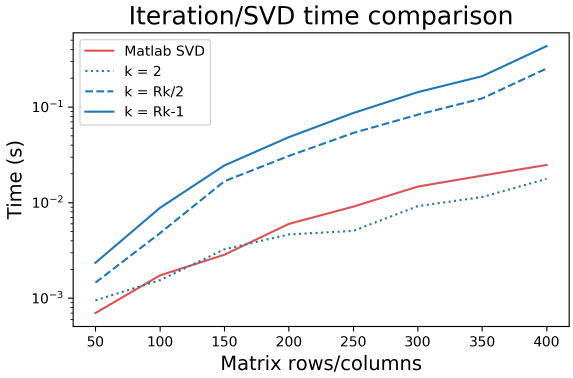
8.3 Comparison with MATLAB's SVD

We compared the performance of our algorithm with that of MATLAB's `svd` library function, since by means of that we can compute the globally optimal solution to our problem by truncation.

We thus generated a dataset of square fullrank matrices having $50, 100, 150, \ldots, 400$ rows, 3 for each size, and approximated each for $k = 2$, $rk(A)/2$ and $(rk(A) - 1)$, for a total of 72 tasks. We then proceeded by avarging the results for each (size, rank) combination and plotted the values for t and t_{it} as shown in **Figure 12**.



(a) Plot of the average execution time t among different matrices when varying the value of k and the input size.



(b) Plot of the average iteration time t_{it} among different matrices when varying the value of k and the input size, in comparison with MATLAB's svd function.

Figure 12: Plots comparing the average execution times when varying k and input size.

Once again, **Figure 12a** shows that when $k = rk(A) - 1$ our algorithm converges in few iterations (as for smaller matrices it outperforms the other values of k), moreover it is clear from **Figure 12b** that our algorithm has an iteration time comparable to that of MATLAB's `svd` for small values of k, and it scales better than `svd` when $k = 2$.

We think that these results are promising, since:

- Our algorithm's performances are slowed down by operations which could be avoided in a deployment phase (such as those logging the statistics at each iteration, computing both the stop condition metrics when we need only one, which we needed in order to analyze the trends).
- Our algorithm is being compared to MATLAB's native `svd`, a library function which is highly optimized. Better versions of our implementation could lead to better overall performances.
- Some sections of the algorithm could benefit from parallelization (or GPU acceleration), such as solving the sub-minimization problems "by columns".
- The algorithm uses our implementation of the QR factorization with column pivoting, and again, this could be optimized.

We think that our algorithm could be employed when the input consists of big matrices (preferably square, or close-to-be squared ones) to be approximated for smaller values of k; one advantage over using `svd` is that we could work under time constraints, as the iterative approach allows us to stop the execution as we reach a timeout, producing an approximated result (which should be acceptable, since the error trend as shown in Figure 3a shows that the error decreases quickly within the first iterations) which could be useful in some application domains, such as real-time systems.

8.4 Benchmark results

In this subsection we show some statistics and plots concerning the execution of the algorithm on the benchmark dataset.
First of all, we executed some initial general tests to analyze the results based on the category of images. We chose a total of 30 pictures for each category, and on each of them we executed the algorithm with `init_t` as the identity matrix, `err_eps` set to `1e-6` and `k` set as 30% of the rank of the correspondent matrix. The majority of the matrices, as we could predict, were not full rank, because of their regularity (given the fact that they come from real-world pictures).

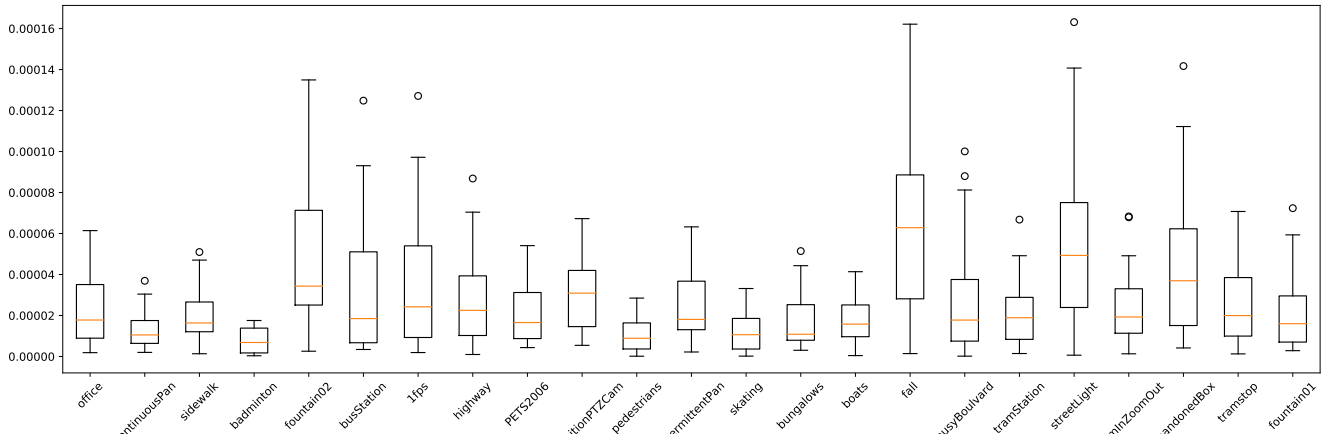


Figure 13: Box-plot of the margin obtained by executing the algorithm on 30 images for each of the 23 categories

Figure 13 shows a box-plot drawn over the margins (i.e. the difference between the error obtained by executing our algorithm and the optimal one related to the truncated SVD) we got for the selected frames, divided by category. Before drawing the plot we removed the outliers from each category, because being different of at least one order of magnitude they could have skewed the quartile values. As we can see, for four categories of frames (that are *fountain02*, *fall*, *streetLight* and *abandonedBox*), the "maximum" of the boxplot is of an order of magnitude bigger with respect to the other categories. In case of *fall* and *streetLight* we have that the median is also considerably higher with respect to all the other categories.

We now take a look to the *fall* and *abandonedBox* categories, for which we have respectively the 20% and the 23% of frames for which the normalized approximation error is bigger than 10^{-4}. Upon visual inspection, the frames for which the approximation error was higher than the threshold just mentioned didn't present any particularities with respect the other ones; let us recall that the frames have been extracted from camera footage that filmed the same scenario for hours, with at most few different elements. For this reason, we did not find any qualitative ways of explaining the worse results we got in those categories of frames. We thus decided to execute more experiments by employing the Random Restart technique, that is explained thoroughly in Section 8.7. Out of all the frames in the two categories mentioned above, we chose the frames on which the algorithm performed the worst, that is where the margin was above 10^{-4}; we then chose 6 frames in each of the two categories where the algorithm had its best performance, i.e. *pedestrians* and *fall*. For the two 'bad' categories and the two 'good' ones, we set the Random Restart to execute 10 experiments for each frame; we then extracted, for each frame, the mean margin over the 10 experiments, and then drew a general mean margin for each category, to see if we could obtain similar results among the four categories considered.

Frames category	**Mean margin**
abandonedBox	2.04239e-6
fall	3.66183e-6
badminton	5.02952e-7
pedestrians	4.85839e-7

Table 1: Mean margins for the four categories we mentioned above. This table shows how, by executing experiments with the Random Restart technique, the mean margins are comparable

Table 1 contains the mean margins computed for each of the categories. The first two rows' categories are the one for which we had highest values in the box plot, while the third and fourth rows contain the two categories for which we had the lowest values. As we can see, the mean margins obtained for the first two categories differ from the ones of the other two categories of just an order of magnitude. A margin of the order of 10^{-6} means that we reached an adequately good approximation; the fact that it is not that much different from the margin of the 'good' categories means that the results shown in the box plot were not due to some characteristics of the original frames, but rather to numeric factors (a possible reason could be the fact that we used the identity matrix as our initialization, which happened to be an 'unlucky' choice for some frames). To wrap up, experiments on this dataset, that contains regular matrices coming from real-life images, have shown that our algorithm produces good results in the majority of cases, while still being subject to some categories for which certain initialization matrices happen to misperform; we saw no significant problems arise when using a randomized initialization approach and thus think that it can be useful in order to avoid picking a constant initialization matrix which could turn out to be a bad choice for the category of images we want to approximate.

8.5 Approximating images: comparison with the truncated SVD

In this subsection we show how well our algorithm performs the task of images approximation with respect to the adoption of the truncated SVD.
For this particular task, we chose a simple picture with a cat, shown in **Figure 14**, as its subject, having a background with no significant irregularities, and we imported it as a matrix by converting it to gray-scale. Such matrix has 592 rows and 594 columns, and a rank of 541.

Figure 14: Image that has been used to test how good our algorithm's approximation is, when compared to the one obtained by truncating the SVD

We ran our algorithm by using the following configuration: we set the stop condition to be used as `approxerr` and the threshold for it to `1e-6`, that as we previously discussed represents a good trade-off between a short enough execution and the achievement of a good approximation; we set `max_it` to 2000 and initialized V as the identity matrix (extended) and w as the 0 vector. We tried the algorithm with multiple values of k, ranging from 0.5% of the original rank to 60% of the original rank.

(a) Approximation given by the truncated SVD with k as 60% of the rank

(b) Approximation given by the low rank approximation algorithm with k as 60% of the rank

(c) Approximation given by the truncated SVD with k as 20% of the rank

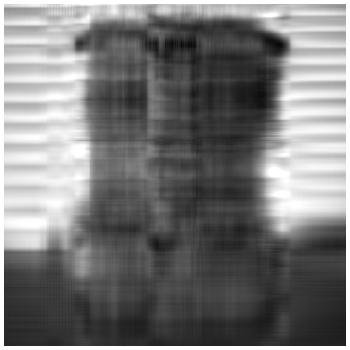
(d) Approximation given by the low rank approximation algorithm with k as 20% of the rank

(e) Approximation given by the truncated SVD with k as 5% of the rank

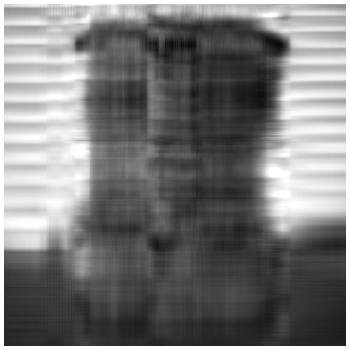
(f) Approximation given by the low rank approximation algorithm with k as 5% of the rank

(g) Approximation given by the truncated SVD with k as 2.5% of the rank

(h) Approximation given by the low rank approximation algorithm with k as 2.5% of the rank



(i) Approximation given by the truncated SVD with k as 1% of the rank



(j) Approximation given by the low rank approximation algorithm with k as 1% of the rank

Figure 15: Comparison between the approximation given by the truncated SVD and the one given by our Low rank approximation algorithm for different values of k

In **Figure 15** we show, for 5 different values of k, how the image is approximated both by the truncated SVD and by our Low rank approximation algorithm, down to the value for which the picture starts to get unrecognizable. We can notice how, for each pair of pictures, the difference is invisible to the naked eye; in **Table 2** we show the difference between the optimal approximation error and the one achieved by our algorithm, and we can see how even if the optimum isn't reached the result is still appreciable when it comes to the task of finding approximations for pictures.

k as percentage of rank	**Difference from the optimum**
60%	4.2096e-07
20%	1.0356e-06
5%	4.2692e-06
2.5%	3.4273e-07
1%	5.5134e-07

Table 2: Normalized different of between the error given by the optimal approximation and the one given by our Low Rank approximation for various values of k

8.6 Approximating images: stopping after few iterations

Even though by approximating a picture with our Low Rank approximation algorithm we obtain the same results to the naked eye as if we were using the truncated SVD, we aim to achieve such results in less time than by truncation. We thus decided to try approximating our cat image with a set of increasing values for the parameter `max_it`, to see if the result would still be appreciable, and to establish if the elapsed time of the computation would be comparable to the one of computing the SVD. For this experiment we decided to set k as 50% of the rank. The SVD computation time for this task was of 0.029 seconds.

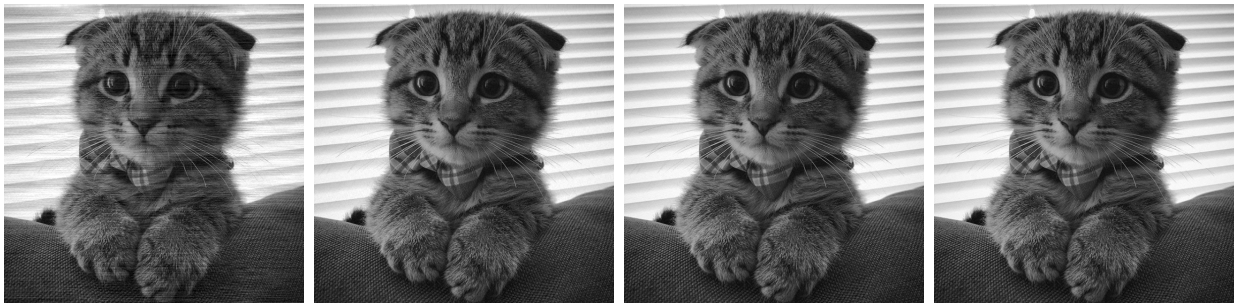


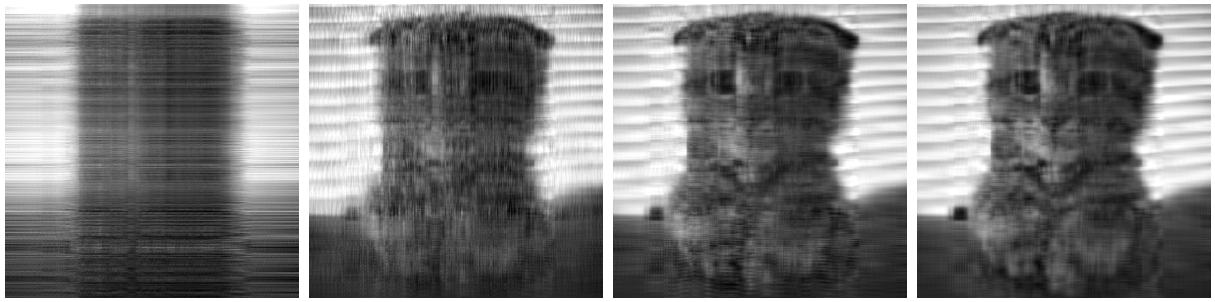
Figure 16: Approximation of the cat picture after respectively 1, 2, 3 and 10 iterations. To the naked eye, a very good approximation is already reached after 2 iterations.

Figure 16 shows 4 different approximations of our picture, obtained by stopping our algorithm after respectively 1, 2, 3 and 10 iterations. We can notice how the first one presents some major inaccuracies in the shape of horizontal blurry lines, that come from the fact that, in the pair $[U, V]$ that is returned, only U is actually picked by finding an optimization based on the original matrix, while V is still the initialization matrix. From

the second approximation onward, to the naked eye we reach an approximation that is really close to the original figure. In this case, therefore, we can state that 2 iterations are enough to reach a good half-rank approximation of our image. However, for such approximation, our algorithm took 1.259 seconds, that is still worse than the SVD.

We noticed that as we decrease the percentage of k the frames tend to be indistinguishable after a slightly higher number of iterations, but even for k equal to 2.5% of the original rank, the differences w.r.t. the truncated SVD are almost invisible to the naked eye after 4 or 5 iterations, as shown in figure 17.

(a) Approximation given by our Low Rank approximation algorithm after 1, 2, 3, and 5 (from left to right) iterations for $k = 5\%$ of the original matrix rank.



(b) Approximation given by our Low Rank approximation algorithm after 1, 2, 3, and 5 (from left to right) iterations for $k = 2.5\%$ of the original matrix rank.

Figure 17: Approximation of the cat picture for increasing numbers of iterations and smaller values of k. To the naked eye, the final approximation is always reached withing few iterations.

To execute 5 iterations, our algorithm took respectively 0.507 seconds (for k set as 2.5% of the rank) and 0.65 seconds (for k set as 5% of the rank). Despite being considerably higher with respect to the time needed to compute the SVD, we think that if optimized our algorithm could possibly be competitive to approximate matrices with low values (proportionally to the rank) of k.

8.7 Randomized approaches

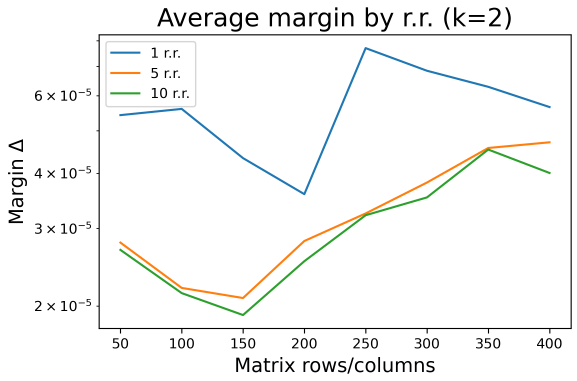
We also tested the behaviour of our algorithm when using a random matrix as initialization (therefore setting `init_t='random'`). We implemented a random restart approach by executing our algorithm multiple times using a random initial matrix and taking the one which minimizes $\|A - UV\|$ among the results of each execution, as shown in **Algorithm 2**.

Algorithm 2: Alternating Low-rank Approximation with Random Restart

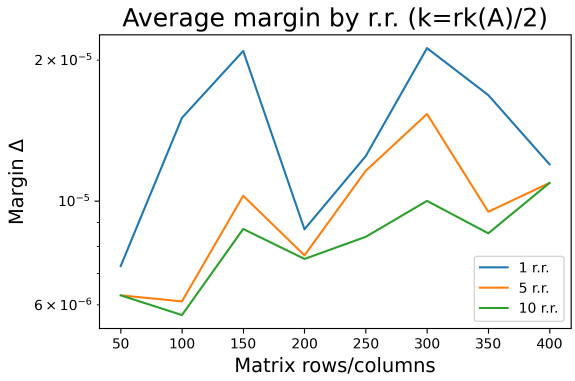
In : A matrix $A \in \mathbb{R}^{m \times n}$, an integer $k < rk(A)$ and the number of restarts $rr > 0$.
Out: Two matrices $U \in \mathbb{R}^{m \times k}$ and $V \in \mathbb{R}^{k \times n}$ such that $\|A - UV\|$ is minimized.

$best \leftarrow +\infty$
for $i \leftarrow 1, \ldots, rr$ **do**
  $\langle U_i, V_i \rangle \leftarrow LowRankApprox(A, k)$ `/* Random initialization */`
  if $\epsilon_A(U_i, V_i) < best$ **then**
    $\langle U, V \rangle \leftarrow \langle U_i, V_i \rangle$
    $best \leftarrow \epsilon_A(U_i, V_i)$
  end
end
return U, V

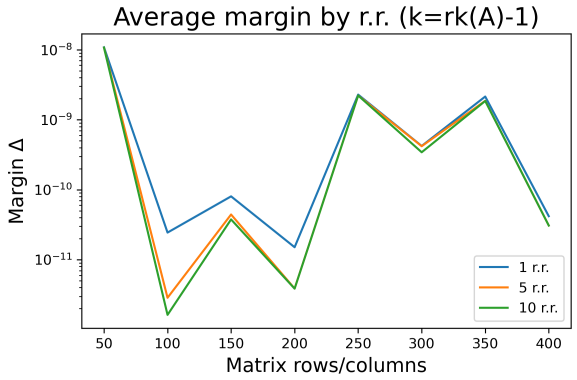
We noticed that the resulting performances when using random initialization were comparable to those of the previous experiments, and we proceeded by assessing the gain in terms of margin Δ obtained through the random restart approach. We show in Figure 18 the margin we got when varying input size, target rank k and the number of random restarts (r.r.), by testing this approach on a dataset having the same structure of that used in section 8.3.



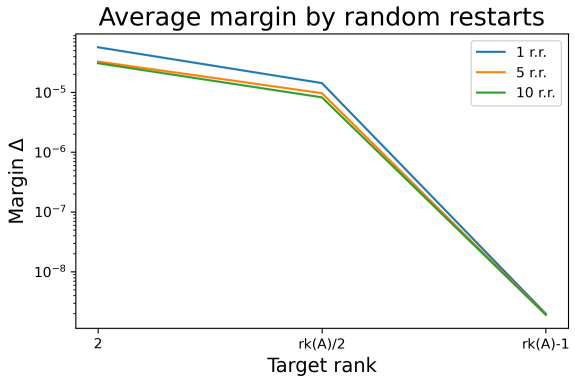
(a) Plot of the average margin when varying input size and r.r. count, fixing $k = 2$.



(b) Plot of the average margin when varying input size and r.r. count, fixing $k = rk(A)/2$.



(c) Plot of the average margin when varying input size and r.r. count, fixing $k = rk(A) - 1$.



(d) Plot of the average margin when varying the value k r.r. count, averaging among input sizes.

Figure 18: Plots showing margins when using a random restart approach.

The random restart approach led to better results in term of obtained margins, which were consistently of order close or smaller than 10^{-6} (which is ideal given our choice of `err_eps`), it is clear from the plots in Figure 18 that a low number of r.r. is sufficient to obtain a significant increase in performance (as the blue-orange lines' distance is bigger than the orange-green one, while the r.r. count increase is higher between the latter pair); we also noticed that the benefit from this approach is less relevant as k gets closer to $rk(A)$, as shown in Figures 18c and 18d. Overall we think that these results are interesting from a point of view of showing how the initial matrix's choice influences the convergence of our algorithm, nevertheless we think of this technique as hardly employable in some contexts for which the algorithm might be convenient (such as real-time systems, as stated before).

Environmental Setup

The experiments have been run on our personal computers. Those are three different machines, having the following specifics:

- The first machine mounts an Intel Core i5-6200U CPU, with 2 cores/4 threads and a frequency of 2.4GHz, and a 4GB RAM. The experiments have been run on MATLAB

(version 9.7.0.1261785, R2019b) installed on Windows 10 64-bit.

- The second machine mounts an Intel Xeon e3-1505M v5 CPU, with 4 cores/8 threads and a frequency of 2.8GHz, and a 16GB RAM. The experiments have been run on MATLAB (version 9.8.0.1451342, R2020a) installed on Windows 10 64-bit.
- The third machine mounts an Intel Core i7-8850H CPU, with 6 cores/12 threads and a frequency of 2.6GHz, and a 16GB RAM. The experiments have been run on MATLAB (version 9.9.0.1467703, R2020b) installed on macOS Catalina 64-bit.

9 Conclusions

The results we obtained through experimental evaluations are consistent with what we expected from the theoretical analysis, with some more behaviours which we explained by means of the impact of memory transfers on executions (a more detailed analysis could be conducted by analyzing the algorithm in an external memory model and dealing with optimizations on the number of memory accesses). We have also shown that the algorithm usually performs well on both randomly generated matrices and matrices coming from real-world problems, and we think that the algorithm might be suitable for certain tasks where we require a value of k significantly smaller than the original rank and only few iterations are enough in order to reach an acceptable result; moreover, we think that it could lead to better performances than computing an SVD in said cases with some more optimizations.

More ways in which we could gain better performances might include:

- Overall optimization of both the algorithm and the QR factorization employed;
- Parallelization of the algorithm, such as when solving the sub-problems "by columns", eventually employing GPU acceleration;
- Better initialization strategies, such as domain-dependent ones (e.g. when approximating subsequent frames we could pick the initial V_0 based on the matrix V given as output from the previous frame, shaving off even more iterations);

A Mathematical Background

Lemma 8. *Let $X \in \mathbb{R}^{m\times n}$, then X has rank at most the minimum between m and n, that is: $rk(X) \leqslant \min\{m, n\}$.*

Proof. Since the rank of a matrix is the dimension of the vector space spanned by its columns (or equivalently, by its rows), the thesis follows immediately observing that X has at most m linearly independent rows and at most n linearly independent columns. □

Lemma 9. *Let $X = AB$, where $A \in \mathbb{R}^{m\times k}, B \in \mathbb{R}^{k\times n}$, then $rk(X) \leqslant \min\{rk(A), rk(B)\}$.*

Proof. Recalling that we can see the columns of X as combinations of the columns of A, and dually the rows of X as a combination of the rows of B, we have that the vector space spanned by the columns of X is included in the vector space spanned by the columns of A, proving $rk(X) \leqslant rk(A)$. The same holds considering the vector space spanned by the rows of B and thus we conclude $rk(X) \leqslant rk(B)$, which completes the proof. □

Theorem 2. *Let $X = AB$, where $A \in \mathbb{R}^{m\times k}, B \in \mathbb{R}^{k\times n}$, if $rk(X) = k$, then $rk(A) = rk(B) = k$ and $m \geqslant k \leqslant n$.*

Proof. By lemma 8 we have that $rk(A) \leqslant \min\{m, k\}$ and $rk(B) \leqslant \min\{k, n\}$ and lemma 9 tells us that $k = rk(X) \leqslant \min\{rk(A), rk(B)\}$; by the first observation we notice that both $rk(A) \leqslant k$ and $rk(B) \leqslant k$. Supposing $rk(A) < k$, then by the second observation we have $k = rk(X) \leqslant rk(A) < k$ which is obviously a contradiction; the same argument holds for B, and thus $rk(A) \geqslant k$, $rk(B) \geqslant k$, which completes the proof of the first assertion.

To prove that $m \geqslant k \leqslant n$ suppose that $m < k$, then by lemma 8 we have $rk(A) \leqslant \min\{m, k\} = m < k$ but this contradicts the previous result; the same argument applies swapping m and A with n and B respectively, proving the second assertion. We then tested our □

Lemma 10. *Let $A \in \mathbb{R}^{n\times m}$, $k \leqslant rk(A)$ then (at least) for the Frobenius and the spectral norm the problem*

$$\arg\min_{rk(A_k)\leqslant k} \|A - A_k\|$$

1. *has a solution of rank exactly k;*
2. *there is no solution of rank strictly smaller than k when $\|\cdot\| = \|\cdot\|_F$.*

Proof. Point 1 follows directly from the Eckart-Young theorem, which gives a solution of rank k. Concerning point 2 note that (following the notation used in lectures), let $h = \min\{m, n\}$:

$$\|A - A_k\| = \left\| \sum_{i=1}^{h} u_i\sigma_i v_i^\top - \sum_{i=1}^{k} u_i\sigma_i v_i^\top \right\| = \left\| \sum_{i=k+1}^{h} u_i\sigma_i v_i^\top \right\| = \|D_k\|$$

Observe that D_k is already in SVD form, thus $D_k = U_k\Sigma_k V_k$ and $\|D_k\| = \|\Sigma_k\|$. Considering the Frobenius norm we have that:

$$\|A - A_k\|_F = \|D_k\|_F = \|\Sigma_k\|_F = \sum_{i=k+1}^{h} \sigma_i^2$$

Since we are under the hypothesis $k \leqslant rk(A)$ then we have $0 < \sigma_k \leqslant \sigma_{k-1} \leqslant \cdots \leqslant \sigma_1$ and thus, considering any optimal solution of rank $k' < k$:

$$\|A - A_{k'}\|_F = \sum_{i=k'+1}^{h} \sigma_i^2 = \sum_{i=k'+1}^{k} \sigma_i^2 + \sum_{i=k+1}^{h} \sigma_i^2 > \sum_{i=k+1}^{h} \sigma_i^2 = \|A - A_k\|_F$$

which completes the proof. □

Corollary 1. *Under the assumptions of the previous theorem, whenever the matrix A has (at least) two equal singular values, let them be $\sigma_h = \sigma_{h+1}$ then we have that:*

1. *If the problem asks for an approximation of rank $\leqslant h$ w.r.t. the spectral norm, then the problem admits at least two solutions of different rank.*
2. *If the problem asks for an approximation of rank $\leqslant h$ w.r.t. the Frobenius norm, then the problem admits at least two solutions of rank h.*

Proof. Concerning the first point it is enough to observe the equality between the value of the spectral norm of both the rank-h and the rank-$(h-1)$ optimal approximations given from the Eckart-Young theorem (namely A_h and A_{h-1}) following a similar argument to that of preceding theorem:

$$\|A - A_h\|_2 = \|D_h\|_2 = \|\Sigma_h\|_2 = \sigma_{h+1} = \sigma_h = \|\Sigma_{h-1}\|_2 = \|D_{h-1}\|_2 = \|A - A_{h-1}\|_2$$

Concerning the second point we observe that the nondecreasing ordering of the singular values is not unique, in fact we can order them as either $\sigma_1, \ldots, \sigma_h, \sigma_{h+1}, \ldots, \sigma_{\min\{m,n\}}$ or swapping the two equal values and the corresponding right and left singular vectors, leading to the ordering $\sigma_1, \ldots, \sigma_{h+1}, \sigma_h, \ldots, \sigma_{\min\{m,n\}}$. This gives two possible rank-h truncated SVDs which are optimal because of the Eckart-Young theorem, namely $A_{h1} = \left(\sum_{i=1}^{h-1} u_i \sigma_i {v_i}^\top + u_h \sigma_h {v_h}^\top\right)$ and $A_{h2} = \left(\sum_{i=1}^{h-1} u_i \sigma_i {v_i}^\top + u_{h+1} \sigma_{h+1} {v_{h+1}}^\top\right)$, and $A_{h1} \neq A_{h2}$ follows from orthogonality of the singular vectors. It is now sufficient to observe that:

$$\|A - A_{h1}\|_F = \sigma_{h+1}^2 + \sum_{i=h+2}^{\min\{m,n\}} \sigma_i^2 = \sigma_h^2 + \sum_{i=h+2}^{\min\{m,n\}} \sigma_i^2 = \|A - A_{h2}\|_F$$

which completes the proof. □

Theorem 3. *Let $X \in \mathbb{R}^{n \times m}$ then let $Y_F \in \mathbb{R}^{n \times m}$ be any optimal rank-k ($k \leqslant rk(X)$) approximation w.r.t. the Frobenius norm, we have that Y_F is at least "close to be optimal" w.r.t. the spectral norm too, where "close to be optimal" means that the approximation error of Y_F is bounded by a constant times the lowest achievable error, in the spectral norm.*

Proof. Observing that $\|\cdot\|_2$ and $\|\cdot\|_F$ are equivalent in any finite-dimensional real (or complex) vector space (as stated by applying corollary 5.4.6 in definition 5.4.7 of [8]) we have that there exist some constants m_F, M_F such that for any matrix A in said vector space:

$$m_F \|A\|_2 \leqslant \|A\|_F \leqslant M_F \|A\|_2 .$$

Now taking the optimal rank-k approximation of any matrix X given from Eckart-Young th. X_k, define the smallest achievable rank-k approximation errors as:

$$\epsilon_2(X_k) = \|X - X_k\|_2$$
$$\epsilon_F(X_k) = \|X - X_k\|_F$$

And by substitution in the previous inequality we have

$$m_F \epsilon_2(X_k) \leqslant \epsilon_F(X_k) \leqslant M_F \epsilon_2(X_k)$$

Now given that Y_F is optimal w.r.t. the Frobenius norm we have

$$\epsilon_F(Y_F) = \|X - Y_F\|_F = \epsilon_F(X_k)$$

and plugging this equality the previous line we get that $\epsilon_F(Y_F) \leqslant M_F \epsilon_2(X_k)$, which tells us that M_F is the constant mentioned in the theorem's statement. □

Corollary 2. *The bound found in the proof of the previous theorem can be specified to both*

$$\epsilon_F(Y_F) \leqslant \sqrt{\min\{m, n\}}\, \epsilon_2(X_k) \tag{9}$$
$$\epsilon_F(Y_F) \leqslant \sqrt{k + rk(X)}\, \epsilon_2(X_k) \tag{10}$$

Proof. Observing the result given in 5.6.P24 of [8] which states that $\|A\|_F \leqslant \sqrt{rk(A)}\|A\|_2$ we have that in the hypothesis of theorem 3 this translates to a better approximation of the inequality shown in the proof as:

$$\epsilon_F(Y_F) \leqslant \sqrt{rk(X - X_k)}\, \epsilon_2(X_k)$$

and by applying lemma 8 we get inequality (9).

Moreover, observing that $rk(A + B) \leqslant (rk(A) + rk(B))$ as stated in inequality 0.4.5.1 of [8] we can obtain

$$\epsilon_F(Y_F) \leqslant \sqrt{rk(X) + rk(X_k)}\, \epsilon_2(X_k)$$

and since by construction $rk(X_k) = k$ we get inequality (10). □

Theorem 4. *Let $A \in \mathbb{R}^{n \times m}$, $k \leqslant rk(A)$ then:*

$$X^* = \underset{rk(X) \leqslant k}{\arg\min} \|A - X\|_F \implies X^* = \underset{rk(X) \leqslant k}{\arg\min} \|A - X\|_2$$

that is, any low-rank approximation which is optimal w.r.t. the Frobenius norm is also optimal w.r.t. the spectral norm.

Proof. The proof proceeds by showing that for every $X^*, rk(X^*) = k$ which minimizes $\|A - X\|_F$ we have that $A - X^*$ has as singular values $\sigma_{k+1}(A), \ldots, \sigma_u(A)$ (where $u = \min\{m, n\}$) eventually followed by some zeroes.

Considering a generic matrix X having $rk(X) \leqslant k$ we have that:

$$\sigma_{k+i}(A) = \sigma_{k+i}(X + (A - X))$$

and then by applying inequality 7.3.13 of [8], putting $k = j - 1$ it follows:

$$\sigma_{j+i-1}(X + (A - X)) \leqslant \sigma_j(X) + \sigma_i(A - X) = \sigma_{k+1}(X) + \sigma_i(A - X)$$

and since $rk(X) \leqslant k$ we have $\sigma_{k+1}(X) = 0$, thus:

$$\sigma_{k+i}(A) \leqslant \sigma_i(A - X) \tag{11}$$

In order to be able to apply 7.3.13 of [8] we observe that the conditions hold:

- $1 \leqslant i \leqslant q$: since we are considering the singular values indexed in $[rk(X) + 1, \ldots, q]$;
- $1 \leqslant j \leqslant q \implies 0 \leqslant k \leqslant q - 1$: as k is the rank of the approximation, and $k < q$ since we are considering proper approximations;
- $i + j \leqslant q + 1 \implies k + 1 \leqslant q$: as for the first condition;

Now since we require that X^* is minimizing the norm we have that (let X_k be the truncated SVD, optimal by Eckart-Young):

$$\sum_{i=1}^{q}(\sigma_i(A - X^*))^2 = \|A - X^*\|_F^2 = \|A - X_k\|_F^2 = \sum_{i=k+1}^{q}(\sigma_i(A))^2$$

and since the above considerations we have that the first $(u - k)$ singular values of $(A - X^*)$ are lower bounded by the last $(u - k)$ singular values of A. It follows that the inequalities in (11) hold as equalities, proving that for any optimal matrix X^* w.r.t. Frobenius norm we have that the singular values of $(A - X^*)$ are the same as those of $(A - X_k)$ and since we have that the spectral norm is dependent on the largest singular value only, it follows that

$$\|A - X^*\|_2 = \|A - X_k\|_2$$

which completes the proof, since X_k is optimal w.r.t. the spectral norm too. □

Theorem 5. *Let $A \in \mathbb{R}^{n\times m}$, $B \in \mathbb{R}^{m\times n}$ with $n \leqslant m$. Then the m eigenvalues of BA are the n eigenvalues AB together with $n - m$ zeroes.*

Proof. Follows directly from theorem 1.3.22 of [8] □

Corollary 3. *Let $X \in \mathbb{R}^{m\times n}$ then $\lambda_{max}(X^\top X) = \lambda_{max}(XX^\top)$, i.e. $X^\top X$ and $XX^\top$ share the same maximal eigenvalue.*

Proof. Considering $A = X$ and $B = X^\top$ and applying theorem 5 we get that the sets of eigenvalues of $XX^\top$ and those of $X^\top X$ differ at most for the element 0. Observing that both matrices are positive semidefinite concludes the proof, as their eigenvalues (and in particular the maximum one) are all $\geqslant 0$. □

Theorem 6. *Given $\|\cdot\|$ we define its adjoint norm $\|\cdot\|'$ as the function $\|X\|' = \left\|X^\top\right\|$.*
Let $X \in \mathbb{R}^{m\times n}$, then $\|X\|_2 = \|X\|'_2$ and $\|X\|_F = \|X\|'_F$.

Proof. Concerning the spectral norm ($\|\cdot\|_2$) we have that $\|X\|_2$ is equal to the largest singular value of X, let it be σ_1. We observe that $\sigma_1 = \sqrt{\lambda_{max}(X^\top X)}$. By applying corollary 3:

$$\|X\|_2 = \sqrt{\lambda_{max}(X^\top X)} = \sqrt{\lambda_{max}(XX^\top)} = \sqrt{\lambda_{max}((X^\top)^\top (X^\top))} = \left\|X^\top\right\|_2 = \|X\|_2'$$

Concerning the Frobenius norm it is enough to apply commutativity of addition:

$$\|X\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}^2} = \sqrt{\sum_{j=1}^{n}\sum_{i=1}^{m} x_{ij}^2} = \left\|X^\top\right\|_F = \|X\|_F'$$

□

Note that the spectral norm is the only induced norm for which $\|X\| = \|X\|'$, this follows directly from theorem 5.6.35 of [8].

Theorem 7. *Given $A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{m\times h}$, let $X \in \mathbb{R}^{h\times n}$ and their respective columns be:*

$$A = \begin{bmatrix} \vdots & & \vdots \\ a_1 & \cdots & a_n \\ \vdots & & \vdots \end{bmatrix}, \qquad B = \begin{bmatrix} \vdots & & \vdots \\ b_1 & \cdots & b_h \\ \vdots & & \vdots \end{bmatrix}, \qquad X = \begin{bmatrix} \vdots & & \vdots \\ x_1 & \cdots & x_n \\ \vdots & & \vdots \end{bmatrix}$$

Then the solution to the minimization problem:

$$X^* = \arg\min_X \|A - BX\|_F$$

can be obtained by solving n sub-problems in the form $x_i^* = \arg\min_x \|a_i - Bx\|_2$ as:

$$X^* = \begin{bmatrix} \vdots & & \vdots \\ x_1^* & \cdots & x_n^* \\ \vdots & & \vdots \end{bmatrix}$$

Proof. Recalling the definition of $\|\cdot\|_F$ (and the fact that from a minimization point of view it's the same as its square) we have

$$\arg\min_X \|A - BX\|_F = \arg\min_X \sum_{i=1}^{m}\sum_{j=1}^{n} (A - BX)_{ij}^2 = \arg\min_X \sum_{j=1}^{n}\left(\sum_{i=1}^{m} (A - BX)_{ij}^2\right)$$

which is, rewriting the inner sum as a function of the columns of A and X and by definition of the vector norm $\|\cdot\|_2$:

$$\arg\min_X \sum_{j=1}^{n}\left(\sum_{i=1}^{m} (a_j - Bx_j)_i^2\right) = \arg\min_X \sum_{j=1}^{n}\left(\left\|a_j - Bx_j\right\|_2^2\right).$$

Now, observing that each term of the summation is dependent on one column of X only, and that the optimal solution X^* is composed by juxtaposition of the columns x_i^* we can conclude that each column can be addressed individually, effectively reducing the computation to solving n least-mean-square problems.

□

Lemma 11. *Let $\|\cdot\| : X \to [0, +\infty)$ be any norm, then $\|\cdot\|$ is convex.*

Proof. By definition of norm, for any $x, y \in X$ and any two scalars λ_x, λ_y:

$$\left\|\lambda_x x + \lambda_y y\right\| \leqslant \|\lambda_x x\| + \left\|\lambda_y y\right\| = \lambda_x \|x\| + \lambda_y \|y\|$$

and by taking any $\alpha \in [0, 1]$, letting $\lambda_x = \alpha$ and $\lambda_y = (1 - \alpha)$ completes the proof. □

Lemma 12. *Let $\|\cdot\|_F, \|\cdot\|_2$ be the Frobenius and the spectral norm respectively, then for any matrix X it holds $\|X\|_2 \leqslant \|X\|_F$.*

Proof. Let us consider any vector x, it holds (since the Frobenius norm on vectors equals norm 2, and then by submultiplicativity of the Frobenius norm):

$$\frac{\|Ax\|_2}{\|x\|_2} = \frac{\|Ax\|_F}{\|x\|_F} \leqslant \frac{\|A\|_F \|x\|_F}{\|x\|_F} = \|A\|_F$$

This bound holds for any x and in particular for the vector which occurs in the supremum in the definition of the spectral norm:

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} \leqslant \|A\|_F$$

which completes the proof. □

Lemma 13. *Let $\|\cdot\|_F, \|\cdot\|_2$ be the Frobenius and the spectral norm respectively, then it holds, for any $A \in \mathbb{R}^{n \times h}, B \in \mathbb{R}^{h \times m}$:*

$$\|AB\|_F \leqslant \|A\|_2 \|B\|_F$$

Proof. Let us consider the columns of $B = \begin{bmatrix} b_1 | \cdots | b_n \end{bmatrix}$ then we have $AB = \begin{bmatrix} Ab_1 | \cdots | Ab_n \end{bmatrix}$ and thus it follows that:

$$\|AB\|_F^2 = \sum_{i=1}^{n} \|Ab_i\|_2^2$$

and by the property of the induced spectral norm it follows:

$$\sum_{i=1}^{n} \|Ab_i\|_2^2 \leqslant \sum_{i=1}^{n} \|A\|_2^2 \|b_i\|_2^2 = \|A\|_2^2 \sum_{i=1}^{n} \|b_i\|_2^2 = \|A\|_2^2 \|B\|_F^2$$

and applying the square root completes the proof. □

Corollary 4. *The bound found in Lemma 13 is a stricter bound w.r.t. the one given from submultiplicativity of the Frobenius norm, i.e.:*

$$\|AB\|_F \leqslant \|A\|_2 \|B\|_F \leqslant \|A\|_F \|B\|_F$$

Proof. This follows immediately by observing that $\|X\|_2 \leqslant \|X\|_F$ by Lemma 12. □

Lemma 14. *Let $\|\cdot\| : X \to [0, +\infty)$ be any norm $A, B \in X$, it holds that there exists no non-trivial multiplicative constant c which gives a lower bound in either of the two forms:*

- $\|A+B\| \geqslant c(\|A\| + \|B\|)$
- $\|A+B\| \geqslant c\|A\|\|B\|$

Proof +. Suppose there exists some value c which satisfies the inequality for any pair A, B, then take any non-zero matrix A and take $B = -A$, it holds:

$$\|A - A\| = \|0\| = 0 \geqslant c(\|A\| + \|-A\|)$$

and since we took $A \neq 0$ it follows that $\|A\| = \|-A\| > 0$ by definition of norm, and thus the constant c must be a non-positive value in order to satisfy the inequality, making it trivial (since any norm is by definition greater than 0). □

Proof ×. As for the previous proof, take any non-zero matrix A and take $B = -A$, it holds:

$$\|A - A\| = \|0\| = 0 \geqslant c(\|A\|\|-A\|)$$

and since we took $A \neq 0$ it follows that $\|A\| = \|-A\| > 0$ by definition of norm, and thus the constant c must be a non-positive value making it trivial. □

Definition 1 (Vectorization)**.** *Given $m, n > 0$ we define $vec(\cdot) : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ as the function mapping each matrix $X \in \mathbb{R}^{m\times n}$ into the vector composed by putting each column of X on top of each other:*

$$X = \begin{bmatrix} \vdots & & \vdots \\ x_1 & \cdots & x_n \\ \vdots & & \vdots \end{bmatrix} \qquad \iff \qquad vec(X) = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

The function $vec(\cdot)$ constitutes a bijection by construction; moreover, since multiplying X by a scalar α is equivalent to multiplying each column αx_i and by construction this leads to

$$(\alpha X = Y) \iff (vec(Y) = \alpha vec(X)).$$

And since it also holds (by a similar argument over the columns) that:

$$(X + Y = Z) \iff (vec(Z) = vec(X) + vec(Y)),$$

then $vec(\cdot)$ is a linear map. Finally, since

$$\|X\|_F = \sqrt{\sum_{i,j} x_{i,j}^2} = \|vec(X)\|_2,$$

then $vec(\cdot)$ constitutes an isometry from the normed vector space $(\mathbb{R}^{m\times n}, \|\cdot\|_F)$ to $(\mathbb{R}^{mn}, \|\cdot\|_2)$ (and since it is bijective the reverse also holds, making it a global isometry).

Definition 2 (Permuting Vectorization)**.** *Given $m, n > 0$ and a permutation $\pi : \mathbb{R}^{mn} \to \mathbb{R}^{mn}$ we define the permuting vectorization:*

$$vec^{\pi}(X) = \pi(vec(X)) = \left[[vec(X)]^{\top} \Pi \right]^{\top}$$

where Π is the permutation matrix associated with π. Since π is a bijective linear map, all of the properties stated in Definition 1 also hold. We will omit the symbol π when it is clear from the context which permutation we are referring to.

Definition 3 (Vector Concatenation)**.** *Let $m, n > 0$, we define the (family of) operator(s) $(::_n^m) : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^{m+n}$ as the concatenation operator, which takes two vectors and returns the vector obtained by putting the first one on top of each other:*

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad \Longrightarrow \quad x ::_n^m y = \begin{pmatrix} x_1 \\ \vdots \\ x_m \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

When it's clear from the context we will often omit m, n in the operator notation and thus refer to it as $x :: y$ only, moreover we notice that every operator $::_n^m$ is bijective and we will denote its inverse as $\begin{pmatrix} \leftrightarrow \\ ::_n^m \end{pmatrix} : \mathbb{R}^{m+n} \to \mathbb{R}^m \times \mathbb{R}^n$ and we simply refer to it as $\overset{\leftrightarrow}{::}$ for simplicity.

Lemma 15. *Let $n > 0$, then $\mathbb{R}^n$ and $\varnothing$ are both open and closed sets in $\mathbb{R}^n$.*

Proof ($\mathbb{R}^n$ open). Take any $x \in \mathbb{R}^n$ and any $r > 0$, it holds that $\mathcal{B}(x, r) \subseteq \mathbb{R}^n$ by definition of ball and thus $x \in int(\mathbb{R}^n) = \mathbb{R}^n$, which proves that $\mathbb{R}^n$ is open. □

Proof ($\mathbb{R}^n$ closed). Suppose there exists x in the boundary of $\mathbb{R}^n$ ($x \in \delta(\mathbb{R}^n)$), then we have that for any $r > 0$ it holds $\exists y \in \mathcal{B}(x, r) \,.\, y \notin \mathbb{R}^n$, but since by definition of ball $\mathcal{B}(x, r) \subseteq \mathbb{R}^n$ this gives a contradiction, proving that $\delta(\mathbb{R}^n) = \varnothing$. Since by definition $cl(\mathbb{R}^n) = int(\mathbb{R}^n) \cup \delta(\mathbb{R}^n) = int(\mathbb{R}^n)$ and since we know that $\mathbb{R}^n$ is open, then $cl(\mathbb{R}^n) = \mathbb{R}^n$, thus proving that $\mathbb{R}^n$ is closed. □

Proof ($\varnothing$ open). Since $\varnothing = \mathbb{R}^n \backslash \mathbb{R}^n = (\mathbb{R}^n)^c$ and $\mathbb{R}^n$ closed, the thesis follows. □

Proof ($\varnothing$ closed). Since $\varnothing = \mathbb{R}^n \backslash \mathbb{R}^n = (\mathbb{R}^n)^c$ and $\mathbb{R}^n$ open, the thesis follows. □

Corollary 5. *Let $m, n > 0$ and considering the topolgy over $\mathbb{R}^{m \times n}$ induced by the Frobenius norm, we have that $\mathbb{R}^{m \times n}$ is both open and closed.*

Proof. It is enough to consider the vectorization $vec(X)$ of each matrix $X \in \mathbb{R}^{m \times n}$, since both m and n are positive then we can apply Lemma 15 over $\mathbb{R}^{nm}$ and conclude that $\mathbb{R}^{m \times n}$ is both open and closed. □

Lemma 16. *The set $\mathbb{R}$ is convex.*

Proof. This follows immediately since $\mathbb{R}$ is a field, closed w.r.t. addition and multiplication, thus for any two $x, y \in \mathbb{R}$ it holds $conv(x, y) \subseteq \mathbb{R}$. □

Lemma 17. *Every closed ball (that is, the ball as defined during the course: $\mathcal{B}(x, r) = \{y \,|\, \|x - y\| \leqslant r\}$ is compact.*

Proof (boundedness). Since we have $\mathcal{B}(x, r) \subseteq \mathcal{B}(x, r)$ then $\mathcal{B}(x, r)$ is bounded. □

Proof (closedness). We prove that $(\mathcal{B}(x,r))^c$ is open, to do so we show that for any $y \in (\mathcal{B}(x,r))^c$ we can find a ball centered in y such that it is fully contained in $(\mathcal{B}(x,r))^c$. First we observe that for every $y \in (\mathcal{B}(x,r))^c$ it holds $\|x-y\| > r$, for every such y we let $r' = \|x-y\|$ and thus $(r'-r) > 0$, we now pick any $0 < d < (r'-r)$ and consider any point $z \in \mathcal{B}(y,d)$, by the triangle inequality we have $\|x-y\| \leqslant \|x-z\| + \|z-y\|$ and by substitution $r' \leqslant \|x-y\| \leqslant \|x-z\| + \|z-y\| < \|x-z\| + (r'-r)$ and thus $\|x-z\| > r' + r - r' = r$ proving that $\mathcal{B}(y,d) \subseteq (\mathcal{B}(x,r))^c$. □

Lemma 18. *Let $A \in \mathbb{R}^{n\times n}$, then for any $\alpha > 0$ it holds:*

$$(A \succ 0) \iff (\alpha A \succ 0)$$

(the same holds for $\succeq$, substituting everywhere)

Proof. By definition of positive definitness we have that for every nonzero vector x it holds $x^\top Ax > 0$, we now check that $x^\top(\alpha A)x = \alpha x^\top Ax > x^\top Ax > 0$ which completes the proof. □

Lemma 19. *Let $A \in \mathbb{R}^{n\times n}$, then it holds:*

$$(A \succ 0) \iff (-A \prec 0)$$

Proof. From definition of positive definitness we have:

$$(A \succ 0) \iff z^\top Az > 0 \iff (-1)z^\top Az < 0 \iff z^\top(-A)z < 0$$

which completes the proof. □

Lemma 20. *Let $A \in \mathbb{R}^{m\times n}$ it holds that*

$$(A^\top A \textit{ invertible}) \iff (m \geqslant n) \wedge (A \textit{ fullrank})$$

Proof ($\implies$). First notice that by applying Lemma 9 and then Lemma 8 it follows

$$rk(A^\top A) \leqslant \min\{rk(A^\top), rk(A)\} = \min\{rk(A), rk(A)\} = rk(A) \leqslant \min\{m,n\}$$

We now prove the contrapositive $(m < n) \vee (A \textit{ not fullrank}) \implies (A^\top A \textit{ singular})$ by instantiating the above inequality:

- If $m < n$ we get $rk(A^\top A) \leqslant \min\{m,n\} = m$ and since $A^\top A \in \mathbb{R}^{n\times n}$ it is singular.
- If A isn't fullrank then $rk(A) < \min\{m,n\} \leqslant n$ and since $A^\top A \in \mathbb{R}^{n\times n}$ it is singular.

□

Proof ($\impliedby$). We prove that $ker(A^\top A) = ker(A)$, to do so notice that for any x:

$$\Big[Ax = 0 \implies A^\top(Ax) = 0\Big] \implies ker(A) \subseteq ker(A^\top A)$$

moreover, in order to prove $ker(A^\top A) \subseteq ker(A)$ we notice:

$$A^\top Ax = 0 \implies x^\top(A^\top Ax) = 0 \iff (Ax)^\top(Ax) = 0 \iff \|Ax\|^2 = 0 \iff Ax = 0$$

it is now enough to observe that $(m \geqslant n) \wedge (A \textit{ fullrank}) \implies ker(A) = \{0\}$ (intuitively, since the column rank and the row rank of A are equal, when solving the linear system $Ax = 0$ through Gaussian elimination we are left with an upper $n \times n$ portion of A which is both upper triangular and full-rank since $rk(A) = n$, which implies that the linear system admits at most one solution) and thus $ker(A^\top A) = \{0\}$ meaning that $A^\top A$ is fullrank, since it is squared. □

Lemma 21. *Let $A \in \mathbb{R}^{m \times n}$ it holds that*

$$(A^\top A \succ 0) \iff (m \geqslant n) \wedge (A \textit{ fullrank})$$

Proof ($\implies$). Since $A^\top A \succ 0$ we have that $(x^\top A^\top Ax = 0) \implies (x = 0)$; now suppose that $(m < n) \vee (A \textit{ not fullrank})$, we would have $ker(A) \neq \{0\}$ and thus $ker(A^\top A) \neq \{0\}$ (following the same arguments as shown in the second proof of Lemma 20), then we would have some $0 \neq y \in ker(A^\top A)$ and thus:

$$y^\top (A^\top Ay) = y^\top 0 = 0$$

which gives a contradiction, as $y \neq 0$. □

Proof ($\impliedby$). We know that $x^\top A^\top Ax = (Ax)^\top (Ax) = \|Ax\|^2 \geqslant 0$ and thus we are left to check that $(m \geqslant n) \wedge (A \textit{ fullrank}) \implies ((x^\top A^\top Ax = 0) \implies (x = 0))$.

Assuming the premise and since $x^\top A^\top Ax = \|Ax\|^2$, by definition of norm this is equivalent to proving that $(Ax = 0) \implies (x = 0)$, by applying a similar argument to that in the proof of Lemma 20 this follows from the fact that $ker(A) = \{0\}$, which completes the proof. □

Lemma 22. *Let $A \in \mathbb{R}^{n \times n}$ be a symmetric invertible matrix, then A^{-1} is symmetric.*

Proof. We have $AA^{-1} = I = I^\top = (AA^{-1})^\top = (A^{-1})^\top A^\top$ and since $AA^{-1} = I = A^{-1}A$ we get $A^{-1}A = (A^{-1})^\top A^\top$ and since $A = A^\top$ we have

$$A^{-1}A = (A^{-1})^\top A \implies A^{-1}AA^{-1} = (A^{-1})^\top AA^{-1} \implies A^{-1} = (A^{-1})^\top$$

which completes the proof. □

Lemma 23. *Let $A \in \mathbb{R}^{n \times n}$ be an invertible matrix with eigenvalues $\lambda_1, \ldots, \lambda_n$, then A^{-1} has as eigenvalues $\lambda_1^{-1}, \ldots, \lambda_n^{-1}$.*

Proof. by definition of eigenvalue:

$$Av = \lambda v \iff A^{-1}Av = \lambda A^{-1}v \iff A^{-1}v = \frac{1}{\lambda}v$$

which completes the proof. □

Lemma 24. *Let $A \in \mathbb{R}^{n \times n} \succ 0$, then A is invertible.*

Proof. Since every eigenvalue of A is strictly positive, then so it is the product of the eigenvalues, which corresponds to $det(A)$, implying that A is nonsingular. □

Lemma 25. *Let $A \in \mathbb{R}^{n \times n} \succ 0$, then $A^{-1} \succ 0$.*

Proof. Since $A \succ 0$, then it is symmetric and by Lemma 22 we know that A^{-1} is symmetric. Moreover, since $A \succ 0$ we have that each of its λ_i is positive, by means of Lemma 23 we conclude that each of the eigenvalues of A^{-1}, that is in the form λ_i^{-1} is positive and thus $A^{-1} \succ 0$. □

Lemma 26. *Let $A \in \mathbb{R}^{n\times n} \succ 0$, then there exists a unique square root $A^{1/2}$ such that $A^{1/2}A^{1/2} = A$ and such square root is symmetric.*

Proof. This follows directly from the proof of Theorem 7.2.6.a of [8], using $k = 2$ and recalling that our definition of positive definitness implies symmetry. □

Notice that the square root is also invertible, since by Lemma 24 we know that $det(A) \neq 0$, and since we know that $det(XY) = det(X)det(Y)$ for any two matrices, it follows that $det(A^{1/2}) \neq 0$ for the particular case $X = Y = A^{1/2}$.

Lemma 27. *Let $A, B \in \mathbb{R}^{n\times n} \succ 0$, then AB has no nonpositive eigenvalues.*

Proof. Let $B^{1/2}$ be the square root of B as of Lemma 26, then:

$$AB = IAB^{1/2}B^{1/2} = (B^{1/2})^{-1}\left[B^{1/2}AB^{1/2}\right](B^{1/2})$$

proving that AB is similar to $\left(B^{1/2}AB^{1/2}\right)$ (as for Definition 1.3.1 of [8]) and thus by Theorem 1.3.3 of [8] they have the same characteristic polynomial (and thus the same eigenvalues).

We now observe that for any nonzero vector z we have:

$$z^\top\left(B^{1/2}AB^{1/2}\right)z = \left(z^\top(B^{1/2})^\top\right)A\left(B^{1/2}z\right) = \left(B^{1/2}z\right)^\top A\left(B^{1/2}z\right)$$

And since $A \succ 0$ we get $z^\top\left(B^{1/2}AB^{1/2}\right)z > 0$, and since we can also prove symmetry as $\left(B^{1/2}AB^{1/2}\right)^\top = \left((B^{1/2})^\top A^\top (B^{1/2})^\top\right) = \left(B^{1/2}AB^{1/2}\right)$ we prove that $B^{1/2}AB^{1/2} \succ 0$, thus both $\left(B^{1/2}AB^{1/2}\right)$ and AB have only positive eigenvalues. □

Corollary 6. *Under the hypothesis of Lemma 27 it holds:*

$$AB \succ 0 \quad \iff \quad AB = (AB)^\top$$

Proof ($\Longrightarrow$). This follows directly from our definition of positive definitness, which requires symmetry. □

Proof ($\Longleftarrow$). From Lemma 27 we get that AB has only positive eigenvalues, that together with symmetry proves positive definitness. □

References

[1] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.

[2] I. Markovsky. *Low Rank Approximation: Algorithms, Implementation, Applications*. Springer, 2012.

[3] Mariya Ishteva, Konstantin Usevich, and Ivan Markovsky. Factorization approach to structured low-rank approximation with applications. *SIAM Journal on Matrix Analysis and Applications*, 35(3):1180–1204, 2014.

[4] Lloyd N. Trefethen and David Bau. *Numerical Linear Algebra*. SIAM, 1997.

[5] Luigi Grippo and Marco Sciandrone. Globally convergent block-coordinate techniques for unconstrained optimization. *Optimization Methods and Software*, 10(4):587–637, 1999.

[6] Jean Gallier. *Schur Complements and Applications*, pages 431–437. Springer New York, New York, NY, 2011.

[7] Y. Wang, P. Jodoin, F. Porikli, J. Konrad, Y. Benezeth, and P. Ishwar. Cdnet 2014: An expanded change detection benchmark dataset. In *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 393–400, 2014.

[8] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Matrix Analysis. Cambridge University Press, 2 edition, 2013.