2024 IEEE INTERNATIONAL WORKSHOP ON MACHINE LEARNING FOR SIGNAL PROCESSING, SEPT. 22–25, 2024, LONDON, UK

COMPONENT-WISE SQUARED FACTORIZATION

Joakim Lefebvre *Arnaud Vandaele* *Nicolas Gillis*

University of Mons

ABSTRACT

The component-wise squared factorization decomposes a matrix as the component-wise square of a low-rank matrix. It can be used to compute the so-called square root rank of a matrix, used in the compact representation of convex polytopes, but also to represent compactly nonnegative data including arithmetic circuits. This paper introduces a coordinate descent algorithm to solve this problem, along with an accelerated variant using extrapolation. We show that it compresses better dense and sparse nonnegative matrices than standard linear low-rank matrix factorization models such as the truncated singular value decomposition (TSVD) and nonnegative matrix factorization (NMF). Then, we explore its application on synthetic matrices generated for experimentation, and also on the slack matrices of regular n-gons and linear Euclidean distance matrices.

***Index Terms*—** square root rank, component-wise squared factorization, sparse matrices

1. INTRODUCTION

Low-rank matrix factorizations (LRMFs) have become standard tools in data analysis and signal processing. They allow us to reduce the dimensionality of data by representing it in a lower dimensional subspace spanned by some basis elements to be computed. Linear models are the simplest and most widely used; they allow to identify basis elements that are combined linearly with each other to approximate the original data. For example, given an input matrix $M \in \mathbb{R}^{m\times n}$ and a factorization rank r, the well-known truncated singular value decomposition (TSVD) finds matrices $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$ that minimizes the least squares error between M and its low-rank approximation UV, that is, that minimizes $\|M - UV\|_F^2$. Another closely related model is nonnegative matrix factorization (NMF) where U and V are constrained to be component-wise nonnegative, which makes sense for physical and probabilistic interpretations; see, e.g., [1].

However, some applications and real-world phenomena cannot be explained linearly, so nonlinear matrix decompositions (NMDs) [2, 3, 4] have recently been introduced to extend the capabilities of LRMFs. An important class of NMDs is the following: given a nonlinear function $f(\cdot)$, the matrix M is approximated by the matrix $f(UV)$. This paper focuses on the component-wise squared factorization (CSF) where $f(X) = X^{\cdot 2}$ is the component-wise square of matrix X. This choice is of particular importance since it leads to an unconstrained model designed to approximate nonnegative data. Let us define CSF more formally: given a matrix $M \in \mathbb{R}^{m\times n}$ and a natural number $r \leq \min(m, n)$, find $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$ such that

$$M \approx (UV)^{\cdot 2} = (UV) \circ (UV), \tag{1}$$

where $\circ$ is the component-wise product between two matrices. In other words, the CSF problem looks for two matrices, U and V, such that the input matrix M is close to the Hadamard product, or element-wise product, of the matrix UV with itself. In order to minimize the difference between M and $(UV)^{\cdot 2}$, we consider the least squares error, and our goal is to solve

$$\min_{U\in\mathbb{R}^{m\times r}, V\in\mathbb{R}^{r\times n}} \left\|M - (UV)^{\cdot 2}\right\|_F^2. \tag{2}$$

In addition to its application in compressing data (see Section 3), the CSF problem also plays a role in two other applications described in the following two paragraphs.

1) Circuits. Probabilistic circuits need to represent nonnegative probabilistic functions. The most popular models are linear (e.g., Gaussian mixture models) but they have limited expressiveness. In this context, CSF allows us to represent much more compactly such circuits; in fact it was proved they can be exponentially more compact [5, 4], that is, some circuits requires an exponential number of parameters for additive mixtures while squared substractive mixtures leads to a polynomial number.

2) Hadamard square root of a matrix. Given a nonnegative matrix M, a Hadamard square root of M, denoted $\sqrt{M}$, is a matrix of the same size as M whose (i, j)th entry is $\sqrt{M_{ij}}$ or $-\sqrt{M_{ij}}$. The minimum rank among all Hadamard square roots of M is called the square root rank and denoted $\operatorname{rank}_{\sqrt{}}(M)$. For example, the following matrix M has rank 3 and it has a Hadamard square root of rank 2:

$$M = \begin{pmatrix} 4 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \quad \sqrt{M} = \begin{pmatrix} -2 & 0 & 1 \\ -1 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}.$$

Authors acknowledge the support by the European Union (ERC consolidator, eLinoR, no 101085607).

The square root rank of a matrix has applications in representing polytopes compactly; see [6, 7] for details. For a matrix $M \in \mathbb{R}^{m\times n}$, there exist $2^{\text{nnz}(M)}$ possible Hadamard square root matrices, where $\text{nnz}(M)$ is the number of non-zero entries of M, which makes the computation of the square root rank via brute force intractable. This exponential complexity cannot be avoided, since it was shown in [7] that computing the square root rank of a matrix is NP-hard. Hence, it makes sense to develop alternative heuristic methods to estimate the square root rank of a matrix. This can be done by solving CSF: if there exists a solution to (2) with a zero objective, that is, $M = (UV)^{\cdot 2}$, then r is an upper bound on the square root rank of M.

Outline and contribution of the paper. The paper is organized as follows. In Section 2, we introduce a method for solving the CSF problem (2). To the best of our knowledge, this is the first time an optimization algorithm is specifically designed for this problem. In Section 3, we empirically show its effectiveness to compress dense and sparse matrices with various experiments on synthetic and real-world data sets. In Section 3.2, we show how to use our algorithm to identify the square root rank of matrices.

2. A COORDINATE DESCENT APPROACH

The CSF factorization problem (2) is nonconvex and, unlike the TSVD and NMF, the problem remains nonconvex even if U or V is fixed. However, we will draw inspiration from algorithms developed in the context of NMF [1], and adopt an alternating scheme by optimizing one factor at a time. Our problem is symmetric in the sense that if we have an algorithm to optimize V given U, then we have an algorithm to optimize U given V since $M \approx (UV)^{\cdot 2}$ is equivalent to $M^\top \approx (V^\top U^\top)^{\cdot 2}$. Moreover, the problem in V is separable by columns since

$$\left\|M - (UV)^{\cdot 2}\right\|_F^2 = \sum_{j=1}^{n} \left\|(UV(:,j))^{\cdot 2} - M(:,j)\right\|_2^2. \quad (3)$$

Therefore, in the remainder of this section, we will only elaborate on an algorithm for solving

$$\min_{V(:,j)} \left\|(UV(:,j))^{\cdot 2} - M(:,j)\right\|_2^2. \quad (4)$$

The problem (4) is a minimization problem to which we will refer as a *Component-wise Squared Least Squares* (CSLS) problem, whose general form is

$$\min_{x\in\mathbb{R}^r} \left\|(Ax)^{\cdot 2} - b\right\|_2^2, \quad (5)$$

with $A \in \mathbb{R}^{m\times r}$ and $b \in \mathbb{R}^m$. One iteration of the general alternating scheme corresponds to the update of all the columns $V(:,j)$, for all j, followed by the update of all the rows $U(i,:)$, for all i. The pseudo-code of the alternating scheme for the CSF problem (2) is detailed in Algorithm 1.

Algorithm 1: Alternating scheme for the CSF problem

Input: $M \in \mathbb{R}^{m\times n}$, initial matrices $U \in \mathbb{R}^{m\times r}$, $V \in \mathbb{R}^{r\times n}$.
Output: $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$ s.t. $M \approx (UV)^{\cdot 2}$

```
for k ← 1 to maxiter do
    for j ← 1 to n do
        V(:,j) = CSLS(M(:,j), U, V(:,j))            (Algo. 2)
    end for
    for i ← 1 to m do
        U(i,:) = CSLS(M(i,:)^⊤, V^⊤, U(i,:)^⊤)^⊤    (Algo. 2)
    end for
end for
```

Since coordinate descent (CD) has been successfully applied to many LRMF problems, e.g., [8, 9, 10, 11, 12], we use this approach to solve CSLS (5). It consists in updating one variable at a time while the others are fixed. In our case, if we fix all the entries of x except the pth one, the one-variable problem is

$$\min_{x_p\in\mathbb{R}} \left\|(A(:,p)x_p + d)^{\cdot 2} - b\right\|_2^2, \quad (6)$$

where $d = \sum_{k=1,k\neq p}^{r} A(:,k)x_k$. After expanding the different terms, we observe that solving (6) amounts to minimize a fourth-degree polynomial. It can be achieved by identifying the roots of its first derivative, that is, by solving

$$c_3x_p^3 + c_2x_p^2 + c_1x_p + c_0 = 0, \quad (7)$$

where

- $c_3 = 4\sum_{i=1}^{m} A_{ip}^4$,
- $c_2 = 12\sum_{i=1}^{m} A_{ip}^3 d_i$,
- $c_1 = 4\sum_{i=1}^{m} 3A_{ip}^2 d_i^2 - A_{ip}^2 b_i$, and
- $c_0 = 4\sum_{i=1}^{m} A_{ip} d_i^3 - A_{ip} d_i b_i$.

Computing the different roots of a third degree polynomial can be done with Cardano's method [13] in $\mathcal{O}(1)$ operations. At most three real roots will be identified and we must find the one minimizing the objective function (5).

The computation of the four coefficients in (7) can be done in $\mathcal{O}(m)$ operations when d is available. To do so without increasing the computational complexity, we can precompute $d = Ax$ and modify it carefully when updating the pth entry in order to have the correct residual vector. The pseudo-code of the CD scheme for (5) is given in Algorithm 2

With the two loops in Algorithm 2, updating once every entry of one column of V can be done in $\mathcal{O}(mr)$ operations and the update of every entry of one row of U can be done in $\mathcal{O}(nr)$. Overall, the algorithmic complexity of one iteration of our algorithm is $\mathcal{O}(mnr)$, as for most LRMFs.

Algorithm 2: CD scheme for the CSLS problem (5)

Input: $A \in \mathbb{R}^{m\times r}, b \in \mathbb{R}^m, x \in \mathbb{R}^r$
Output: $x \in \mathbb{R}^r$ solving (5)

$d = Ax$
for $p \leftarrow 1$ **to** r **do**
  $d = d - A(:,p)x_p$
  $c_0, c_1, c_2, c_3 = 0, 0, 0, 0$
  for $i \leftarrow 1$ **to** m **do**
    $c_3 = c_3 + 4A_{ip}^4$
    $c_2 = c_2 + 12A_{ip}^3 d_i$
    $c_1 = c_1 + 4(3A_{ip}^2 d_i^2 - A_{ip}^2 b_i)$
    $c_0 = c_0 + 4(A_{ip} d_i^3 - A_{ip} d_i b_i)$
  end for
  $x_p \leftarrow \texttt{cardanomethod}(c_3, c_2, c_1, c_0)$
  $d = d + A(:,p)x_p$
end for

Initialization We use two initializations for U and V:
1) Random initialization: each entry of U and V are drawn from the Gaussian distribution $N(0,1)$. When using this initialization, the initial approximation may be very different from M itself. To prevent this, we scale our initial U and V compared to M as follows: first compute the optimal scaling

$$\lambda = \underset{\lambda}{\operatorname{argmin}} \|M - \lambda (UV)^{\cdot 2}\|_F^2 = \frac{\langle (UV)^{\cdot 2}, M\rangle}{\langle (UV)^{\cdot 2}, (UV)^{\cdot 2}\rangle}, \quad (8)$$

where $\langle A, M\rangle = \sum_{i,j} A_{i,j} M_{i,j}$ is the inner product between two matrices. Then we scale: $U \leftarrow \lambda^{1/4} U$ and $V \leftarrow \lambda^{1/4} V$.
2) TSVD-based initialization: Given the TSVD $M \approx U\Sigma V^\top$ with $U \in \mathbb{R}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$ and $V \in \mathbb{R}^{r\times n}$, we let $U = U\sqrt{\Sigma}$ and $V = \sqrt{\Sigma} V$.

Acceleration via extrapolation To speed up our algorithm, we use an extrapolated coordinate descent; see, e.g., [14, 15] where it was successfully applied to NMF. This requires a simple modification of the algorithm as follows:

$$V(:,j)^{k+1} = \texttt{CSLS}\left(M(:,j), U, Z(:,j)^k\right) \text{ where}$$
$$Z(:,j)^k = V(:,j)^k + \beta^k (V(:,j)^k - V(:,j)^{k-1}),$$

and $\beta^k \in [0,1)$ is an extrapolation parameter. We update the rows of U in the same way. To choose the values of β^k, we draw inspiration from [14] and dynamically update it as described in Algorithm 3. The idea is to increase β^k as long as the objective function decreases and reduce it otherwise.

To determine good values for the parameters $\beta_1, \hat{\gamma}, \gamma$, and η in Algorithm 3, we conduct preliminary experiments with various sets of values. The best values we found for our extrapolation more or less agree with those from [14], except that a smaller β yielded better results in our case. Therefore, we chose the parameters $(\gamma, \hat{\gamma}) = (1.05, 1.01)$, $\beta_1 = 0.3$, and $\eta = 1.5$. These values are the best on average on several matrices of different sizes but could be fine-tuned when using a particular data set. Fig. 1 illustrates the acceleration effect of the extrapolation on randomly generated matrices.

Algorithm 3: Update of β^k

Input: $\beta_1 \in (0,1), 1 < \hat{\gamma} < \gamma < \eta$.
$\hat{\beta} = 1$.
if the error (9) decreases at iteration k **then**
  $\beta_{k+1} \leftarrow \min(\hat{\beta}, \gamma\beta_k)$.
  $\hat{\beta} \leftarrow \min(1, \hat{\gamma}\hat{\beta})$.
else
  $\beta_{k+1} \leftarrow \frac{\beta_k}{\eta}$; $\hat{\beta} \leftarrow \beta_{k-1}$.
end if

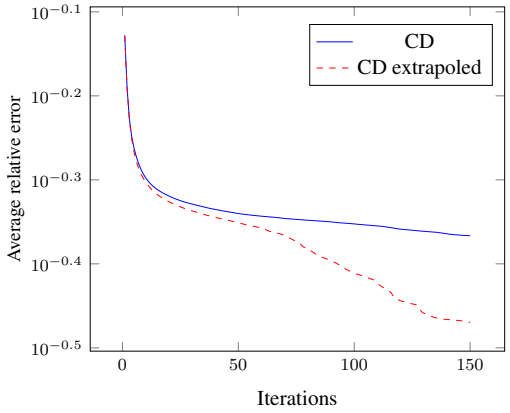

Fig. 1: Average relative errors $\|M - (UV)^{\cdot 2}\|_F / \|M\|_F$ of CD and its extrapolation, for 10 randomly generated matrices $M \in \mathbb{R}^{200\times 200}$ with $r = 10$; see Section 3.2.1 for details.

We have also compared the execution times in the context of exact factorization (see Section 3.2 for details):
1. To factorize 8 slack matrices of n-gons, the extrapolated CD took 165 seconds compared to 989 for CD.
2. To factorize 8 linear Euclidean distance matrices, the extrapolated CD took 313 seconds compared to 713 for CD.

Therefore, we will only use our extrapolated algorithm to perform the tests in Section 3.

3. NUMERICAL EXPERIMENTS

The codes are available online from `https://github.com/Riipou/CSF` and implemented in Julia. We will make the code available on github upon acceptance of the paper. All these tests are conducted on a Apple 11-core M3 Pro 18 GB RAM.

3.1. Compression of synthetic and real data

We test our algorithm on three datasets, see Table 2, as well as randomly generated matrices. In the following, we compare

		CSF (rdm init.)	CSF (SVD init.)	NMF (rdm init.)	NMF (SVD init.)	TSVD
CBCL	$r = 10$	14.8 ± 0.0	$\mathbf{14.8}$	15.2 ± 0.0	15.2	14.9
	$r = 20$	11.6 ± 0.0	11.5	12.2 ± 0.0	12.2	$\mathbf{11.3}$
	$r = 49$	7.7 ± 0.0	$\mathbf{7.3}$	8.0 ± 0.0	8.0	7.4
CBCL facial features	$r = 10$	69.0 ± 0.0	$\mathbf{68.9}$	89.6 ± 0.0	89.7	89.3
	$r = 20$	39.5 ± 0.1	$\mathbf{39.2}$	81.0 ± 0.1	81.2	80.2
TDT2	$r = 10$	$\mathbf{56.2 \pm 0.5}$	56.2	70.6 ± 0.1	70.7	69.9
	$r = 20$	$\mathbf{40.6 \pm 0.4}$	42.5	59.0 ± 0.2	58.9	58.0
Sparse Matrix	$r = 10$	74.0 ± 0.4	$\mathbf{74.0 \pm 0.3}$	89.7 ± 0.2	89.7 ± 0.2	88.7 ± 0.2
	$r = 20$	50.8 ± 0.5	$\mathbf{50.7 \pm 0.5}$	82.8 ± 0.2	82.8 ± 0.3	80.0 ± 0.2

Table 1: Relative error (in percent) of NMF and CSF on CBCL, CBCL facial features, TDT2, and ten distinct sparse 200×200 matrices generated randomly using the Julia function `sprand`.

	Size	#nnz	Sparsity
CBCL	361×2429	876869	0%
CBCL facial features	361×100	5735	84.11%
TDT2	299×909	36835	86.45%

Table 2: Datasets

the relative error of our method,

$$\text{relative error} = \frac{\|M - (UV)^{.2}\|_F}{\|M\|_F}, \tag{9}$$

with that of NMF and TSVD. For NMF and CSF methods, we compute the average relative error over 10 different random initializations, as well as the error for an SVD initialization for different values of r. In this section, we stop NMF and CSF algorithms after one minute.

The results of our tests are available in Table 1. On the dense data set, CBCL, CSF yields slightly better results, although the errors of all methods are comparable. For sparse data sets, CSF has a relative error at least 13% smaller than NMF and TSVD. In some cases, the improvement is significant, e.g., for the CBCL facial features with $r = 20$, from an error of 81% for NMF and 80% for TSVD to 39% for CSF.

Let us perform two additional experiments. The CBCL facial features dataset contains the 85%-sparse facial features of CBCL obtained via NMF with $r = 100$. Fig. 2 compares the average relative error for $r \in \{10, 20, 30, 40\}$. We observe that CSF is significantly more effective for compression than NMF and TSVD.

Finally, we test our algorithm on randomly generated sparse matrices. For each test, we generate 10 sparse matrices randomly using Julia function `sprand`. We compare the average errors of the three methods with different levels of sparsity (see Fig 3). Again, CSF is significantly more efficient to compress sparse matrices than NMF and TSVD.

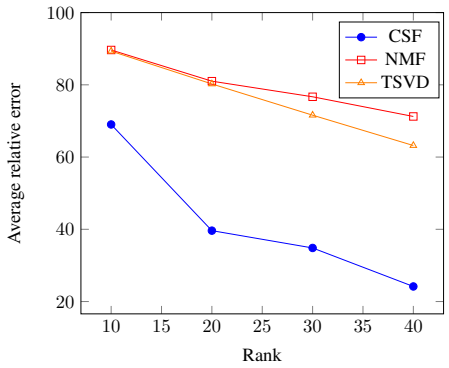

Fig. 2: Comparing the average relative error on the CBCL facial features using random initialization.

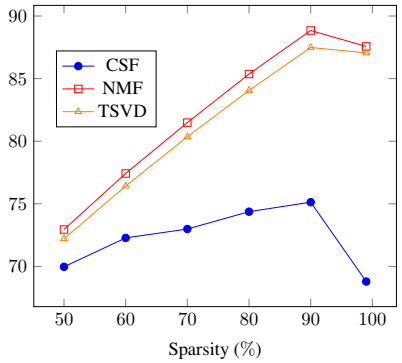

Fig. 3: Average relative error: NMF vs. CSF vs. TSVD on 200×200 sparse matrices, using random initialization, on a rank-10 factorization.

3.2. Exact Factorizations

In this section, we perform numerical experiments to identify exact factorizations $M = (UV)^{\cdot 2}$. Numerically, we report that an exact factorization is found when the relative error (9) is less than 10^{-3}. To do so, we run the algorithm as long

as the relative error decreases by at least a predefined factor $\alpha \in (0, 1)$ every ten iterations.

3.2.1. Tests on synthetic data

First, we generate synthetic matrices $M = (UV)^{\cdot 2}$ using randomly generated $U \in \mathbb{R}^{m\times 2}$ and $V \in \mathbb{R}^{2\times n}$. To randomly generate U and V with the Julia function `randn`. We use $m = n = [5, 10, 50, 100]$, maxiter $= 10^4$ and $\alpha = 0.99$. We repeat this test 100 times for each matrix size; see Table 3.

Matrix size	Success Rate (random initialization)	Success rate (SVD initialization)
$n = 5$	48%	50%
$n = 10$	57%	50%
$n = 50$	83%	73%
$n = 100$	85%	77%

Table 3: Comparison of CD success rates: Random vs SVD initialization for rank-2 CSF on $M \in \mathbb{R}^{n\times n}$.

We notice that the random initialization appears to work better, but overall, regardless of the initialization, our algorithm manages to find a factorization for $r = 2$ for all tested matrix sizes.

We conduct another test on synthetic data, generating a matrix M randomly for various sizes, similar to the previous experiment. For each matrix, we execute the algorithm 10 times using random initialization and count the number of times the algorithm achieves a successful factorization. We realize this test on matrix $M \in \mathbb{R}^{n\times n}$ for $n = [5, 10, 50, 100]$, using $\alpha = 0.9999$ and maxiter = 10^4. Results are available in Table 4. We observe that the algorithm manages to find

Matrix size	$n = 5$	$n = 10$	$n = 50$	$n = 100$
Success Rate $\alpha = \mathbf{0.9999}$	60%	70%	100%	90%

Table 4: Success rates of CD with random initialization across 10 executions for rank-2 CSF on $M \in \mathbb{R}^{n\times n}$.

at least one good factorization. However, for the same rank, larger matrices lead to a larger success rate; this can be explained because a larger matrix contains more samples.

3.2.2. Square root rank of slack matrices

As defined in Section 1, the square root rank of a matrix M is the smallest $r \in \mathbb{N}$ such that an exact factorization $M = (UV)^{\cdot 2}$ exists with $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{r\times n}$. This quantity is important in the field of extended polytope formulations, as the square root rank is an upper bound on the *psd-rank* of a matrix [7, 12].

Slack matrices of n-gons An important category of polytopes are the regular polygons for which we use our algorithm to identify the square root rank of their slack matrix. On slack matrices of size $n = 3, ..., 10$ and for different values of r, we performed 10^4 runs of our method using random initialization and reported the lowest relative error in Table 5. These results, together with the fact that $\text{rank}_{\text{psd}}(M) \leq \text{rank}_{\sqrt{}}(M)$, allowed us to propose a conjecture on the square root rank of the different slack matrices, see Table 6.

Matrix size	r	Success rate (random initialization)	Lowest error (9)
$m = n = 10$	9	49.6%	$\mathbf{1.00 \cdot 10^{-6}}$
	8	1.2%	$\mathbf{1.19 \cdot 10^{-4}}$
	7	0.0%	$3.93 \cdot 10^{-3}$
$m = n = 9$	8	41.1%	$\mathbf{9.27 \cdot 10^{-7}}$
	7	0.3%	$\mathbf{2.97 \cdot 10^{-4}}$
	6	0.0%	$8.53 \cdot 10^{-3}$
$m = n = 8$	7	36.8%	$\mathbf{1.60 \cdot 10^{-7}}$
	6	0.1%	$\mathbf{3.62 \cdot 10^{-6}}$
	5	0.0%	$1.34 \cdot 10^{-2}$
$m = n = 7$	6	28.3%	$\mathbf{1.55 \cdot 10^{-7}}$
	5	0.0%	$1.55 \cdot 10^{-3}$
	4	0.0%	$2.44 \cdot 10^{-2}$
$m = n = 6$	5	27.9%	$\mathbf{1.27 \cdot 10^{-7}}$
	4	0.1%	$\mathbf{4.82 \cdot 10^{-6}}$
	3	0.0%	$5.02 \cdot 10^{-2}$
$m = n = 5$	4	0.0%	$1.09 \cdot 10^{-3}$
	3	0.0%	$6.841 \cdot 10^{-2}$
$m = n = 4$	3	59.4%	$\mathbf{8.24 \cdot 10^{-9}}$
	2	0.0%	$1.69 \cdot 10^{-1}$
$m = n = 3$	2	0.0%	$3.333 \cdot 10^{-1}$

Table 5: Tests on slack matrix of the regular n-gon with $\alpha = 0.9999$.

n	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
psd-rank [12]	3	3	4	4	4 or 5	≤ 4	?	≤ 5
square root rank (conjecture)	3	3	5	4	≤ 6	≤ 6	≤ 7	≤ 8

Table 6: Conjecture on the value of the square root rank for the slack matrices of the regular n-gon for $n = 3$ to $n = 10$.

Linear Euclidean Distance Matrices (LEDMs) The LEDM of size n is a n-by-n matrix defined by:

$$M(i, j) = (i - j)^2, \text{ for } i, j = 1, 2, \ldots, n. \qquad (10)$$

Unlike the slack matrices of the n-gons for which the true square root rank value is not known, the square root rank of the LEDM of any dimension is 2, see [7]. We used our method on LEDMs of size $n = 3, ..., 10$ with $r = 2$ and $r = 3$ and reported the results in Table 7. These results are consistent with

the theory and illustrate that our method is able to provide insights into the value of the square root rank of matrices.

Matrix size	r	Success rate (Random initialization)	Lowest error (9)
10×10	3	36.1%	$\mathbf{7.84 \cdot 10^{-16}}$
	2	35.6%	$\mathbf{3.88 \cdot 10^{-16}}$
	1	0.0%	0.66
9×9	3	38.8%	$\mathbf{4.43 \cdot 10^{-16}}$
	2	58.1%	$\mathbf{2.83 \cdot 10^{-16}}$
	1	0.0%	0.66
8×8	3	43.0%	$\mathbf{3.40 \cdot 10^{-16}}$
	2	38.9%	$\mathbf{2.42 \cdot 10^{-16}}$
	1	0.0%	0.66
7×7	3	49.0%	$\mathbf{6.10 \cdot 10^{-16}}$
	2	63.0%	$\mathbf{3.53 \cdot 10^{-16}}$
	1	0.0%	0.66
6×6	3	56.4%	$\mathbf{3.26 \cdot 10^{-16}}$
	2	43.8%	$\mathbf{1.75 \cdot 10^{-16}}$
	1	0.0%	0.66
5×5	3	62.8%	$\mathbf{4.13 \cdot 10^{-16}}$
	2	68.9%	$\mathbf{1.73 \cdot 10^{-16}}$
	1	0.0%	0.66
4×4	3	77.2%	$\mathbf{1.43 \cdot 10^{-8}}$
	2	74.5%	$\mathbf{4.23 \cdot 10^{-11}}$
	1	0.0%	0.66
3×3	3	100.0%	$\mathbf{3.41 \cdot 10^{-9}}$
	2	85.4%	$\mathbf{6.45 \cdot 10^{-8}}$
	1	0.0%	0.67

Table 7: Tests on linear Euclidean distance matrices.

4. CONCLUSION

In this paper, we studied the component-wise squared factorization (CSF) problem, a recent nonlinear matrix decomposition model to approximate nonnegative matrices. To solve this non-convex problem, we designed a coordinate descent scheme for which the one-variable minimization subproblem is a fourth-degree polynomial. In addition, we proposed an accelerated variant using extrapolation, which is faster and provides more accurate solutions. Based on numerical experiments on synthetic and real data sets, we observed that CSF is particularly effective for compressing sparse matrices compared to NMF and TSVD. We also empirically showed that our algorithm is able to factorize, up to machine precision, slack matrices, and hence provides an estimate of their square root rank.

5. REFERENCES

[1] N. Gillis, *Nonnegative matrix factorization*, SIAM, Philadelphia, 2020.

[2] N. Whiteley, A. Gray, and P. Rubin-Delanchy, "Matrix factorisation and the interpretation of geodesic distance," *NeurIPS*, vol. 34, pp. 24–38, 2021.

[3] L. K. Saul, "A nonlinear matrix decomposition for mining the zeros of sparse data," *SIAM J. Math. Data Sci.*, vol. 4, no. 2, pp. 431–463, 2022.

[4] L. Loconte, A. M. Sladek, S. Mengel, M. Trapp, A. Solin, N. Gillis, and A. Vergari, "Subtractive mixture models via squaring: Representation and learning," in *ICLR*, 2024.

[5] L. Loconte, N. Di Mauro, R. Peharz, and A. Vergari, "How to turn your knowledge graph embeddings into generative models," *NeurIPS*, 2023.

[6] T. Lee and Z. Wei, "The square root rank of the correlation polytope is exponential," *arXiv preprint arXiv:1411.6712*, 2014.

[7] H. Fawzi, J. Gouveia, P. A. Parrilo, R. Z. Robinson, and R. R. Thomas, "Positive semidefinite rank," *Math. Prog.*, vol. 153, no. 1, pp. 133–177, July 2015.

[8] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale nonnegative matrix and tensor factorizations," *IEICE Trans. Fundam. Electron. Comput. Sci.*, vol. 92, no. 3, pp. 708–721, 2009.

[9] N. Gillis and F. Glineur, "Accelerated multiplicative updates and hierarchical als algorithms for nonnegative matrix factorization," *Neural Comput.*, vol. 24, pp. 1085–1105, 2012.

[10] C.-J. Hsieh and I. S. Dhillon, "Fast coordinate descent methods with variable selection for non-negative matrix factorization," in *ACM SIGKDD*, 2011, pp. 1064–1072.

[11] A. Vandaele, N. Gillis, Q. Lei, K. Zhong, and I. Dhillon, "Efficient and non-convex coordinate descent for symmetric nonnegative matrix factorization," *IEEE Trans. Signal Process.*, vol. 64, no. 21, pp. 5571–5584, 2016.

[12] A. Vandaele, F. Glineur, and N. Gillis, "Algorithms for positive semidefinite factorization," *Computat. Optim. Appl.*, vol. 71, no. 1, pp. 193–219, 2018.

[13] G. Cardano, *Ars magna or the rules of algebra*, Dover Publications, 1968.

[14] A. M. S. Ang and N. Gillis, "Accelerating nonnegative matrix factorization algorithms using extrapolation," *Neural Comput.*, vol. 31, pp. 417–439, 2019.

[15] L. T. Hien, D. N. Phan, and N. Gillis, "An inertial block majorization minimization framework for nonsmooth nonconvex optimization," *Journal of Machine Learning Research*, vol. 24, no. 18, pp. 1–41, 2023.