Comput Optim Appl
https://doi.org/10.1007/s10589-018-9998-x

Algorithms for positive semidefinite factorization

Arnaud Vandaele[1] · **François Glineur**[2,3] · **Nicolas Gillis**[1]

Received: 26 July 2017

Abstract This paper considers the problem of positive semidefinite factorization (PSD factorization), a generalization of exact nonnegative matrix factorization. Given an m-by-n nonnegative matrix X and an integer k, the PSD factorization problem consists in finding, if possible, symmetric k-by-k positive semidefinite matrices $\{A^1, \ldots, A^m\}$ and $\{B^1, \ldots, B^n\}$ such that $X_{i,j} = \text{trace}(A^i B^j)$ for $i = 1, \ldots, m$, and $j = 1, \ldots, n$. PSD factorization is NP-hard. In this work, we introduce several local optimization schemes to tackle this problem: a fast projected gradient method and two algorithms based on the coordinate descent framework. The main application of PSD factorization is the computation of semidefinite extensions, that is, the representations of polyhedrons as projections of spectrahedra, for which the matrix to be factorized is the slack matrix of the polyhedron. We compare the performance of our

Nicolas Gillis acknowledges the support by the F.R.S.-FNRS (incentive Grant for scientific research No. F.4501.16) and by the ERC (starting Grant No. 679515). This paper presents research results of the Concerted Research Action (ARC) programme supported by the Federation Wallonia-Brussels (contract ARC 14/19-060).

✉ Arnaud Vandaele
arnaud.vandaele@umons.ac.be

François Glineur
francois.glineur@uclouvain.be

Nicolas Gillis
nicolas.gillis@umons.ac.be

[1] Department of Mathematics and Operational Research, Faculté Polytechnique, Université de Mons, Rue de Houdain 9, 7000 Mons, Belgium

[2] Center for Operations Research and Econometrics, Université Catholique de Louvain, Voie du Roman Pays, 34, 1348 Louvain-La-Neuve, Belgium

[3] ICTEAM Institute, Université Catholique de Louvain, 1348 Louvain-La-Neuve, Belgium

Published online: 21 March 2018

algorithms on this class of problems. In particular, we compute the PSD extensions of size $k = 1 + \lceil \log_2(n) \rceil$ for the regular n-gons when $n = 5$, 8 and 10. We also show how to generalize our algorithms to compute the square root rank (which is the size of the factors in a PSD factorization where all factor matrices A^i and B^j have rank one) and completely PSD factorizations (which is the special case where the input matrix is symmetric and equality $A^i = B^i$ is required for all i).

Keywords Positive semidefinite factorization · Extended formulations · Fast gradient method · Coordinate descent method

1 Introduction

Given an m-by-n nonnegative matrix X and an integer $k < \min\{m, n\}$, the standard nonnegative matrix factorization (NMF) problem seeks a matrix $\tilde{X}$ such that the (i, j)th entry of $\tilde{X}$ is equal to the inner product of two size-k nonnegative vectors w_i and h_j, and which is as close to X as possible. For all $i = 1, \ldots, m$ and $j = 1, \ldots, n$, we have

$$X_{ij} \approx \tilde{X}_{ij} = \langle w_i, h_j \rangle = w_i^T h_j \text{ with } w_i \text{ and } h_j \in \mathbb{R}^k_+. \tag{1}$$

The problem of PSD factorization addressed in this paper is a recently introduced generalization of NMF [13]. In a PSD factorization problem, the cone of positive semidefinite matrices replaces the nonnegative orthant of NMF. More precisely, the inputs of a PSD factorization problem are the same as for NMF, namely, a m-by-n nonnegative matrix X and an integer $k < \min\{m, n\}$. However, instead of using the inner product between two vectors of size k, the (i, j)th entry of the approximating matrix $\tilde{X}$ is given by the inner product between two symmetric k-by-k positive semidefinite matrices, A^i and B^j. The inner product of two matrices is a generalization of the dot product of two vectors, and is equal to the trace of the product of the two matrices. Hence, we have for $i = 1, \ldots, m$ and $j = 1, \ldots, n$,

$$X_{ij} \approx \tilde{X}_{ij} = \langle A^i, B^j \rangle = \operatorname{tr}(A^i B^j) \text{ with } A^i \text{ and } B^j \in \mathcal{S}^k_+.$$

The optimization problem corresponding to PSD factorization consists in minimizing the quantity $\|X - \tilde{X}\|_F^2$. It can be expressed by the following non-convex and NP-hard problem [26] where the variables are the two sets of matrices $\{A^1, \ldots, A^m\}$ and $\{B^1, \ldots, B^n\}$ belonging to the positive semidefinite cone $\mathcal{S}^k_+$:

$$\min_{\substack{A^i, B^j \in \mathcal{S}^k_+ \\ i=1,\ldots,m \\ j=1,\ldots,n}} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(X_{ij} - \left\langle A^i, B^j \right\rangle \right)^2. \tag{2}$$

In this work, we propose several algorithms for solving (2) numerically. To our knowledge, no algorithm has been proposed in the literature to solve this problem. The motivation comes from the computation of extended formulations, as explained in details in Sects. 2 and 5. As far as we know, there are currently no other application for solving (2), as opposed to NMF that has been extensively used in data analysis applications such as image processing, text mining and hyperspectral imaging; see, e.g., [10] and the references therein. It would be a particularly interesting direction for further research to explore other applications of PSD factorizations.

The paper is organized as follows. In Sect. 2, we introduce the PSD factorization problem in more details, highlighting its connection with extended formulations. In Sect. 3, we propose several algorithms to compute PSD factorizations (namely, a fast projected gradient method and two algorithms based on the coordinate descent framework). In Sect. 4, we compare the efficiency of the presented methods on a benchmark of nonnegative matrices. In Sect. 5, we show how to use our algorithms to compute (i) PSD factorizations of the slack matrices of regular n-gons, (ii) symmetric PSD factorizations related to completely PSD matrices, and (iii) the square root rank of nonnegative matrices.

2 Linear and semidefinite extensions, and factorizations

In the context of the NMF of an m-by-n nonnegative matrix X, the minimum value of the inner dimension k for which it is possible to find $W \in \mathbb{R}_+^{m\times k}$ and $H \in \mathbb{R}_+^{k\times n}$ such that $X = WH$ is called the nonnegative rank of X, and is denoted $\text{rank}_+(X)$. The search for such an exact factorization has tight connections with the study of linear extensions of polyhedrons. Let P be a polyhedron described by a system of linear inequalities. A linear extension of P is another polyhedron Q of higher dimension which projects linearly onto P, that is, for which there exists a linear map π such that $\pi(Q) = P$. Such linear extensions are particularly useful when the size (measured by the number of facets) of a linear extension is (much) smaller than the size of the initial polyhedron. For example, the left picture of Fig. 1 illustrates a linear extension of an irregular (planar) heptagon, which is three-dimensional but features only six facets. Among all possible linear extensions of a polyhedron P, the size of the smallest one is the *linear extension complexity* of P and is denoted by $\text{xc}(P)$.

An outstanding result of Yannakakis establishes a strong link between NMF and linear extensions [32]: the linear extension complexity of a polyhedron P is equal to the nonnegative rank of a particular matrix related to P, called the *slack matrix* $\mathcal{S}_P$:

$$\text{xc}(P) = \text{rank}_+(\mathcal{S}_P). \tag{3}$$

For a polyhedron P featuring f facets and v vertices, the slack matrix $\mathcal{S}_P$ is a f-by-v nonnegative matrix whose (i, j)th entry is the slack between the ith facet and the jth vertex (the slack between a facet defined by the inequality $a^T x \leq b$ and a vertex $v \in P$ is given by $b - a^T v \geq 0$). Furthermore, Yannakakis showeed that any rank-k nonnegative factorization of $\mathcal{S}_P$ (implicitly) provides a size-k linear extension of P. This result connecting the two fields has been at the core of many recent developments; see, e.g., [19] and the references therein.

Recently, the work of Yannakakis was generalized to allow for arbitrary closed convex cones K instead of the nonnegative orthant [13]. From the point of view of extensions, linear extensions (for which $K = \mathbb{R}_+^k$) are replaced by conic extensions,

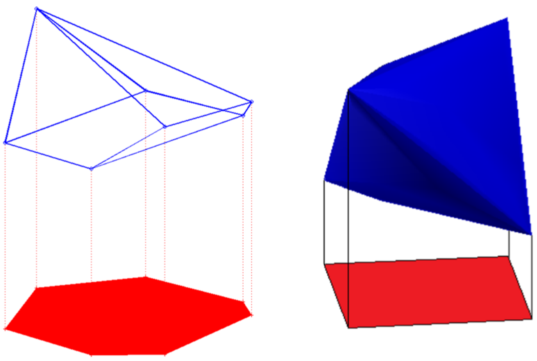


Fig. 1 Left: linear extension of size 6 of an irregular hexagon. Right: psd-lift of size 3 of the square

that is, representations as projections under a linear map of an affine slice of a cone K (an affine slice of a set K is its intersection with a set defined by some equalities $Ax = b$). These generalized extensions are also called *K-lifts*. In this paper, we focus on the case where $K = \mathcal{S}_+^k$, the cone of positive semidefinite matrices. In that case, given a polyhedron P, we are looking for a *spectrahedron* (an affine slice of a positive semidefinite cone) which projects onto P under a linear map. Moreover, we are trying to find such a *semidefinite extension* whose size (as measured by the dimension of the positive semidefinite cone) is as small as possible. This minimal size is called the *semidefinite extension complexity* of P, and is denoted $\mathrm{xc}_{\mathrm{psd}}(P)$.

The semidefinite extension complexity never exceeds the linear extension complexity, but can be strictly lower. For example, the linear extension complexity of the square is 4, but there exists a spectrahedron of size 3 which projects linearly onto the square (see the picture on the right of Fig. 1). Yannakakis' result (3) can be generalized in the following way, which uses the *positive semidefinite rank* (abbreviated psd-rank or $\mathrm{rank}_{\mathrm{psd}}$) to a special rank of the slack matrix of P [13, Theorem 3.3]:

$$\mathrm{xc}_{\mathrm{psd}}(P) = \mathrm{rank}_{\mathrm{psd}}(\mathcal{S}_P).$$

The positive semidefinite rank is related to the PSD factorization problem (2) in the same way than the nonnegative rank is connected to NMF. Formally, the psd-rank of a m-by-n nonnegative matrix X is the smallest integer k for which there exist two sets of k-by-k positive semidefinite matrices $\{A^1, \ldots, A^m\}$ and $\{B^1, \ldots, B^n\}$ such that $X_{ij} = \langle A^i, B^j \rangle$ holds for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$. We refer the reader to the survey [6] for further informations on the psd-rank.

Example 2.1 In order to illustrate the concept of the size of a PSD factorization, let the following 4-by-4 matrix be a slack matrix of the square,

$$S_4 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

Already highlighted by the picture on the right of Fig. 1, it is possible to find a $\mathcal{S}_+^3$ factorization of S_4, for example with the following factors:

$$A^1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, A^2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, A^3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, A^4 = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \end{pmatrix},$$

$$B^1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}, B^2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, B^3 = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}, B^4 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Designing algorithms for solving (2) is therefore of great interest in the search of psd-lifts based on the factorization of the corresponding slack matrices, and is the main objective of this paper.

3 Algorithms for PSD factorization

The PSD factorization problem (2) is nonconvex. However, when one of the two sets of matrix variables $\{A^1, \dots, A^m\}$ or $\{B^1, \dots, B^n\}$ is fixed, optimizing over the other set reduces to a convex problem. For this reason, we develop in this work algorithms using an alternating strategy for solving (2), by optimizing alternately over the sets $\{A^1, \dots, A^m\}$ and $\{B^1, \dots, B^n\}$. The same approach is used by nearly all NMF algorithms, which is also a nonconvex problem that becomes convex when one of the two factors is fixed. The pseudo-code of the general alternating scheme for PSD factorization is detailed in Algorithm 1.

Algorithm 1 Alternating Strategy for PSD Factorization

```
INPUT: $X \in \mathbb{R}_+^{m \times n}$ and initial iterates $\{A^1, \dots, A^m\}$ and $\{B^1, \dots, B^n\}$.
OUTPUT: $\{A^1, \dots, A^m\}$ and $\{B^1, \dots, B^n\}$.
while stopping criterion not satisfied do
    $\{A^1, \dots, A^m\} \leftarrow$ optimize subproblem($X, \{B^1, \dots, B^n\}$),
    $\{B^1, \dots, B^n\} \leftarrow$ optimize subproblem($X^T, \{A^1, \dots, A^m\}$).
end while
```

Since the subproblems are symmetric, we can assume without loss of generality for the presentation of the algorithms that the set $\{B^1, \dots, B^n\}$ is fixed and that we want to optimize over the A^i's. The corresponding problem can be written formally as:

$$\min_{\substack{A^i \in \mathcal{S}_+^k \\ i=1,\dots,m}} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(X_{ij} - \left\langle A^i, B^j \right\rangle \right)^2. \tag{4}$$

Matrices A^i do not influence each other in (4), that is, the problem is separable, hence it reduces to m independent convex problems, each corresponding to the optimization over a single factor A^i (corresponding to a single row of X). Hence, our first idea consists in solving each of these problems to optimality as described by Algorithm 2, which is an instance of a semidefinite program. The combination of Algorithms 1 and 2 leads to an exact two-block coordinate descent scheme. Since each block of variables belong to a closed convex set and the objective function is continuously differentiable, a stationary point of (2) is obtained in the limit [16].

Algorithm 2 Optimize subproblem (exact)

```
INPUT: $X \in \mathbb{R}_+^{m \times n}$ and $B^j \in \mathcal{S}_+^k$ with $j = 1, \ldots, n$.
OUTPUT: $\{A^1, \ldots, A^m\}$
for $i = 1$ to $m$ do
    $A^i \leftarrow \arg\min_{A^i \in \mathcal{S}_+^k} \sum_{j=1}^n \left( X_{ij} - \left\langle A^i, B^j \right\rangle \right)^2$
end for
```

We implemented Algorithm 2 with the general convex solver YALMIP [23] that uses SeDuMi [27] to solve the subproblems (an interior-point method). However, this approach has proven to be far too slow in comparison with the other methods developed hereafter; see Fig. 2 for an example on the regular 16-gon. As is also the case in the context of NMF, the reason for the poor performance is that it is not worth solving the subproblems (4) to optimality at each iteration. Once the objective function has decreased by some amount, it is preferable to move quickly to the other set of variables rather than performing extra work to refine the subproblem solution to optimality. The reason is twofold: (i) optimizing to optimality would require to spend time decreasing the objective function by a small amount during the last iterations while switching over the other block of variables would allow a significantly larger decrease, and (ii) since the block of variables are modified at each step, it does not make sense to optimize exactly over one block since the other block will be modified anyway (this is especially true at the initial steps of the algorithm where the initial variables are typically chosen randomly). Based on that observation, we propose in the following two iterative methods for solving (4): an algorithm based on the (accelerated) gradient method described in Sect. 3.1, and implementations of coordinate descent methods introduced in Sect. 3.2.

3.1 A fast projected gradient method

One of the most widely used method in continuous optimization is the *gradient method*. From a given starting point x_0, a sequence of points $\{x_t\}$ is built by taking a step in the direction $-\nabla f(x_{t-1})$ for each iterate $t = 1, 2, \ldots$. The next point is then computed as $x_t = x_{t-1} - \alpha_{t-1} \nabla f(x_{t-1})$, where quantity α_{t-1} is the step size along the steepest descent direction. The gradient method admits accelerated schemes, which were first introduced in [24]. The scheme used in this work is described as Algorithm 3 for the general problem $\min_{x \in Q} f(x)$ with Q a closed convex set.

The accelerated gradient method presented as Algorithm 3 has roughly the same computational cost as the usual gradient method. The difference lies in the fact that the gradient step (line 6) is made at an extrapolation point y_t (computed in line 5)

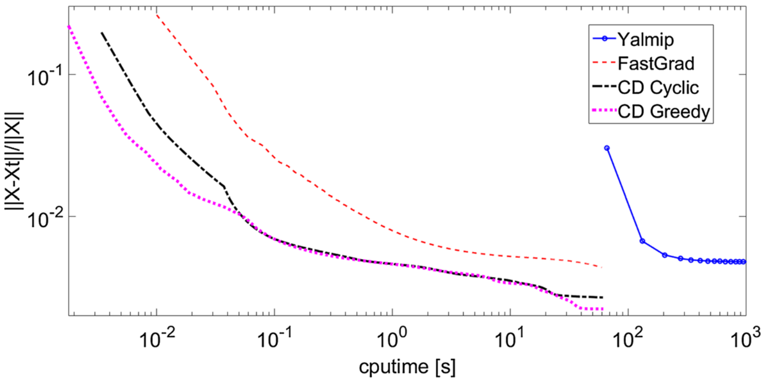


Fig. 2 Evolution of the relative error for different algorithms applied on the slack matrix of the regular 16-gon, using the same initial point. During the time YALMIP took to update once each variable (using SeDuMi), the fast gradient method (FastGrad) could perform about 4700 iterations, the cyclic coordinate descent method (CD Cyclic) about 85,000 iterations, and the greedy coordinate descent method (CD Greedy) about 140,000 iterations. (These algorithms are described in the remainder of this section. We used here their default parameters as described in Sect. 4)

Algorithm 3 Nesterov's accelerated gradient method

```
INPUT: $x_0 \in Q$
OUTPUT: $x_{\text{maxiter}}$
Set $x_{-1} = x_0$
for $t = 1$ : maxiter do
    $y_t = x_{t-1} + \frac{t-2}{t+1}(x_{t-1} - x_{t-2})$
    $x_t = \text{Proj}_Q(y_t - \frac{1}{L}\nabla f(y_t))$
end for
```

instead of the previous iterate x_{t-1}. When using a step size equal to $\frac{1}{L}$ (with L the Lipschitz constant of the objective function's gradient ∇f), the accelerated gradient method exhibits a convergence rate of $O(1/t^2)$, with t the number of iterations (see [24] for more details). In order to apply the accelerated scheme of Algorithm 3 to the PSD factorization problem (4), several issues must first be addressed.

- *Computing the gradient* Let f denote the quantity to minimize in (4). Using the Frobenius norm, f can be written as follows,

$$f = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(X_{ij} - \left\langle A^i, B^j \right\rangle \right)^2 = \left\| X - \mathcal{A}^T \mathcal{B} \right\|_F^2, \qquad (5)$$

with $\mathcal{A} = \left(vec(A^1) \ \ldots \ vec(A^m) \right)$ and $\mathcal{B} = \left(vec(B^1) \ \ldots \ vec(B^n) \right)$ being k^2-by-m and k^2-by-n matrices respectively. Using this notation, the gradient of f with respect to the variable $\mathcal{A}$ is:

$$\nabla f = -2 \left(X - \mathcal{A}^T \mathcal{B} \right) \mathcal{B}^T.$$

From (5), we can also derive the Lipschitz constant L needed in Algorithm 3, which will be equal to the largest eigenvalue of the Hessian $\nabla^2 f$, hence $L = 2\lambda_{\max}\left(\mathcal{B}\mathcal{B}^T\right)$.

- *Projecting on $\mathcal{S}_+^k$*. For our problem, the closed convex set Q that we need to project onto (see line 6 of Algorithm 3) is the cone of symmetric and positive semidefinite matrices, that is, $Q = \mathcal{S}_+^k$. For every k-by-k real symmetric matrix C, we have $C = U \Lambda U^T$ where U is an orthogonal matrix, and Λ is a diagonal matrix whose entries are the eigenvalues of C. Defining $\Lambda_+ = \text{diag}\left(\max(0, \lambda_1), \ldots, \max(0, \lambda_k)\right)$, we have the following closed-form formula for the projection

$$\text{Proj}_{\mathcal{S}_+^k}(C) = \arg \min_{X \in \mathcal{S}_+^k} \|X - C\| = U \Lambda_+ U^T. \tag{6}$$

The main computational cost of the projection (6) is spent computing the spectral decomposition of C.

The pseudo-code of the accelerated gradient method for PSD factorization is presented as Algorithm 4 and denoted FPGM (for Fast Projected Gradient Method). Recall that Algorithm 4 is used for solving the subproblems of the general alternating scheme of Algorithm 1. We choose to perform a (fixed) number of accelerated gradient steps proportional to the size of the factors, equal to $k\Delta$ where Δ is a parameter (line 7 of Algorithm 4). In Sect. 4, performance of the algorithm is compared for different values of Δ.

Algorithm 4 Optimize subproblem (5) with the fast projected gradient method (FPGM)

```
1: INPUT: $X \in \mathbb{R}_+^{m \times n}$ and $B^j \in \mathcal{S}_+^k$ with $j = 1, \ldots, n$, parameter $\Delta$.
2: OUTPUT: $\{A^1, \ldots, A^m\}$
3: $\{A^1, \ldots, A^m\} \leftarrow$ Initialization
4: Construct $\mathcal{A}_0$ from $\{A^1, \ldots, A^m\}$ and $\mathcal{B}$ from $\{B^1, \ldots, B^n\}$.
5: $L \leftarrow \lambda_{\max}\left(\mathcal{B}\mathcal{B}^T\right)$
6: Set $\mathcal{A}_{-1} = \mathcal{A}_0$
7: for $t = 1 : k\Delta$ do
8:     $\mathcal{Y}_t = \mathcal{A}_{t-1} + \frac{t-2}{t+1}(\mathcal{A}_{t-1} - \mathcal{A}_{t-2})$
9:     $\mathcal{A}_t = \text{Proj}_{\mathcal{S}_+^k}\left(\mathcal{Y}_t + \frac{1}{L}(X\mathcal{B}^T - \mathcal{Y}_t^T \mathcal{B}\mathcal{B}^T)\right)$
10: end for
11: Extract $\{A^1, \ldots, A^m\}$ from $\mathcal{A}_{k\Delta}$.
```

Algorithm 4 has two drawbacks. First, in the current form of the algorithm, it is not possible to adjust easily the rank of the A^i's and the B^j's while it is interesting to obtain low -rank factors (observe that the factors of Example 2.1 are all rank one); see the discussion in Sect. 3.2.5. Second, if we want to fix the values of some entries of the A^i's and the B^j's which is particularly helpful to compute exact PSD factorizations (see Sect. 5), it is not straightforward to keep them constant during the iterations of the algorithm (the projection step would become even more computationally expensive, as a linearly constrained semidefinite program would have to be solved). In the next section, we present coordinate-descent algorithms overcoming these limitations.

3.2 Coordinate descent algorithms

Although known for many years, coordinate descent (CD) methods have recently received a new lease of life [31]. This increase of interest is mainly due to the increasing number of large-scale optimization problems in data mining and machine learning applications for which the simplicity of the CD methodology allows efficient and competitive implementations (while high solution accuracy is usually not needed since data is typically rather noisy). In many of these applications, fixing all variables except one leads to an univariate optimization problem for which computation of a minimizer is cheap. For example, in the case of the NMF problem, the corresponding univariate optimization problem is quadratic, and its optimal solution can therefore be written in closed form. First introduced in [5] under the name HALS (for Hierarchical Alternating Least Square), methods based on the CD scheme have proven to be among the most effective ones for the NMF problem [4,11,18].

3.2.1 Change of variables

If we want to successfully apply the CD scheme to the PSD factorization problem, it is crucial that the update of one variable is computationally cheap and easy to implement. However, it is not straightforward to update the entries of the factors A^i and B^j: unlike NMF where nonnegativity of the variables had to be taken into account, which can be ensured separately in each variable (a separable constraint), matrices A^i and B^j are required to remain positive semidefinite, which is no longer separable. Hence, in order to adapt the problem (4) to the application of the CD scheme, we perform a simple change of variables popularized by the works of Burer and Monteiro on semidefinite programming [2]. Since every symmetric positive semidefinite matrix L can be written in the form $L = HH^T$, we introduce new (matrix) variables $a^i \in \mathbb{R}^{k \times r_i}$ for $i = 1, \ldots, m$ and $b^j \in \mathbb{R}^{k \times r_j}$ for $j = 1, \ldots, n$ linked to the original factors A^i and B^j as follows:

$$A^i = a^i {a^i}^T \text{ and } B^j = b^j {b^j}^T.$$

With this reformulation, entries of the new variables a^i and b^j are unconstrained, and positive semidefiniteness of the A^i's and B^j's is automatically guaranteed. Another benefit is the ability to easily adjust the inner rank of the factors A^i and B^j by choosing the number of columns r of the new variables, as the rank will be at most equal to this number. Moreover, if some entries of a^i or b^j are known and fixed, they can simply be ignored in the CD scheme. Since we have $\left\langle A^i, B^j \right\rangle = \sum_{h=1}^{r_i} \sum_{l=1}^{r_j} \left({a^i_{:,h}}^T b^j_{:,l} \right)^2$, the optimization problem (4) is now written as follows with the new variables:

$$\min_{a^i \in \mathbb{R}^{k \times r_i}, i=1,\ldots,m} f = \sum_{i=1}^{m} \sum_{j=1}^{n} \left(X_{i,j} - \sum_{h=1}^{r_i} \sum_{l=1}^{r_j} \left({a^i_{:,h}}^T b^j_{:,l} \right)^2 \right)^2. \tag{7}$$

Since we use an alternating scheme, we assume in the remainder of the section that the b^j's are given and that only the a^i's must be optimized. Note that the matrix $a^i \in \mathbb{R}^{k \times r_i}$ is made of kr_i entries, so that the number of variables of the problem (7) is $k \sum_{i=1}^m r_i$, and mk^2 in the full-rank case ($r_i = k$ for all i).

3.2.2 Update of one variable

In order to apply the CD scheme to (7), we need to derive the expression of the univariate function to minimize when all the variables of (7) are fixed but one, say the entry (p, q) of the factor a^i denoted $a^i_{p,q}$. Only the ith factor is impacted when the entry $a^i_{p,q}$ is updated since the factors A^i's are independent from one another. By highlighting $a^i_{p,q}$, the part of the objective function influenced by the variable is

$$\sum_{j=1}^{n} \left(X_{i,j} - \sum_{\substack{h=1\\h\neq q}}^{r} \sum_{l=1}^{r} \left({a^i_{:,h}}^T b^j_{:,l} \right)^2 - \sum_{l=1}^{r} \left({a^i_{\bar{p},q}}^T b^j_{\bar{p},l} \right)^2 \right. \\ \left. - \boxed{a^i_{p,q}} \left(2{a^i_{\bar{p},q}}^T \left(\sum_{l=1}^{r} b^j_{p,l} b^j_{\bar{p},l} \right) \right) \Bigg/ - \boxed{{a^i_{p,q}}^2} \|b^j_{p,:}\|^2 \right)^2, \tag{8}$$

where $\bar{p} = \{1, \ldots, k\} \backslash \{p\}$. We observe that the function to minimize is a fourth degree polynomial in $a^i_{p,q}$ and its gradient has therefore the form of a cubic polynomial,

$$\nabla_{a^i_{p,q}} f = c_3 {a^i_{p,q}}^3 + c_2 {a^i_{p,q}}^2 + c_1 a^i_{p,q} + c_0, \tag{9}$$

where

$$\begin{aligned} c_3 &= 4 \sum_{j=1}^{n} \|b^j_{p,:}\|^4, \\ c_2 &= 12 {a^i_{\bar{p},q}}^T \sum_{j=1}^{n} \left(\|b^j_{p,:}\|^2 \sum_{l=1}^{r_j} b^j_{p,l} b^j_{\bar{p},l} \right) \\ &= 12 {a^i_{:,q}}^T \sum_{j=1}^{n} \left(\|b^j_{p,:}\|^2 \sum_{l=1}^{r_j} b^j_{p,l} b^j_{:,l} \right) - 3 c_3 a^i_{p,q}, \\ c_1 &= 4 \sum_{j=1}^{n} \left(\|b^j_{p,:}\|^2 \left(\sum_{\substack{h=1\\k\neq q}}^{r_i} \sum_{l=1}^{r_j} \left({a^i_{:,h}}^T b^j_{:,l} \right)^2 + \sum_{l=1}^{r_j} \left({a^i_{\bar{p},q}}^T b^j_{\bar{p},l} \right)^2 - X_{i,j} \right) \right) \\ &\quad + 8 \sum_{j=1}^{n} \left({a^i_{\bar{p},q}}^T \sum_{l=1}^{r_j} b^j_{p,l} b^j_{\bar{p},l} \right)^2 \end{aligned}$$

$$
\begin{aligned}
&= 4\sum_{j=1}^{n}\left(\|b^j_{p,:}\|^2\left(\sum_{h=1}^{r_i}\sum_{l=1}^{r_j}\left({a^i_{:,h}}^T b^j_{:,l}\right)^2 - X_{i,j}\right)\right) + 8\sum_{j=1}^{n}\left({a^i_{:,q}}^T\sum_{l=1}^{r_j} b^j_{p,l} b^j_{:,l}\right)^2\\
&\quad - 2c_2 a^i_{p,q} - 3c_3 {a^i_{p,q}}^2,\\
c_0 &= 4{a^i_{:,q}}^T\sum_{j=1}^{n}\left(\left(\sum_{h=1}^{r_i}\sum_{l=1}^{r_j}\left({a^i_{:,h}}^T b^j_{:,l}\right)^2 - X_{i,j}\right)\sum_{l=1}^{r_j} b^j_{p,l} b^j_{:,l}\right)\\
&\quad - c_1 a^i_{p,q} - c_2 {a^i_{p,q}}^2 - c_3 {a^i_{p,q}}^3.
\end{aligned}
$$

For every entry $a^i_{p,q}$, we need to compute the different coefficients and find the root of (9) which minimizes the objective function (8). Note that such a real root always exists since the polynomial (9) is the derivative of a polynomial of the form (7) which goes to infinity as $a^i_{p,q}$ goes to $\pm\infty$ (unless it is identically zero). Computing the roots of a third degree polynomial can be done in $O(1)$ operations with Cardano's method (see for example [3]).

3.2.3 Computational complexity of the updates

The computation of the coefficients c_i's must be implemented very carefully in order to avoid a high computational cost during the updates of the variables one after the other. For example, we notice that the computation of the coefficient c_0 from scratch needs $O(nk^3)$ operations for a specific triplet (i, p, q). Updating once the mk^2 entries would therefore cost $O(mnk^5)$. In the following, we explain how to reach a computational cost of $O(mk^5)$ for one pass over the mk^2 entries of the problem. A loop over the 'large' dimension (n) can be avoided with the precomputation of some quantities independent of $a^i_{p,q}$ and used during all the iterations. For example, the term $\sum_{j=1}^{n}\|b^j_{p,:}\|^4$ can be precomputed and the computation of c_3 takes only $O(1)$ operations. However, the situation is more complicated for some other terms, especially c_1 and c_0. We describe below how to handle efficiently these computations and which quantities need to be precomputed.

Computing c_0 In order to compute the coefficient c_0, the value of the gradient is precomputed and maintained for all the variables during the iterations. For the purpose of clarity, we denote the quantity $\nabla_{a^i_{p,q}} f$ as $g^i_{p,q}$. From the expression of the coefficients of (9), we have:

$$
\begin{aligned}
g^i_{p,q} &= 4{a^i_{:,q}}^T\sum_{j=1}^{n}\left(\left(\sum_{h=1}^{r_i}\sum_{l=1}^{r_j}\left({a^i_{:,h}}^T b^j_{:,l}\right)^2 - X_{i,j}\right)\sum_{l=1}^{r_j} b^j_{p,l} b^j_{:,l}\right),\\
&= 4{a^i_{:,q}}^T\sum_{j=1}^{n}\left(\left\langle A^i, B^j\right\rangle - X_{i,j}\right) B^j_{:,p} = 4{a^i_{:,q}}^T C^i_{:,p},
\end{aligned}
$$

where the different matrices $C^i = \sum_{j=1}^{n}\left(\left\langle A^i, B^j\right\rangle - X_{i,j}\right) B^j$, $i = 1, \ldots, m$, can be precomputed for a total of $O(mnk^2)$ operations. With the C^i matrices available, it is possible to compute $g^i_{p,q}$ in $O(mk^3)$ operations for any triplets (i, p, q). However, C^i depends on the variable $a^i_{p,q}$ and once $a^i_{p,q}$ has been assigned to its optimal value, all the entries of C^i must be updated. The entry (u, v) of C^i can be updated in the following way,

$$\begin{aligned} {C^i_{u,v}}^{new} &\leftarrow {C^i_{u,v}}^{old} - \sum_{j=1}^{n} \left\langle {A^i}^{old}, B^j \right\rangle B^j_{u,v} + \sum_{j=1}^{n} \left\langle {A^i}^{new}, B^j \right\rangle B^j_{u,v} \\ &= {C^i_{u,v}}^{old} - \left\langle {A^i}^{old} - {A^i}^{new}, \sum_{j=1}^{n} B^j_{u,v} B^j \right\rangle, \end{aligned} \tag{10}$$

and since ${A^i}^{old} - {A^i}^{new}$ is a matrix with only one non-zero row and column, the update of the (u, v)th entry can be done in $O(k)$ operations if the quantity $\sum_{j=1}^{n} B^j_{u,v} B^j$ is available. To this end, we precompute $D_{u,v,:,:} = \sum_{j=1}^{n} B^j_{u,v} B^j$ for all u and v. To sum up, if g^i is available, the coefficient c_0 can be computed in $O(1)$ operations. However, after the optimization of the variable $a^i_{p,q}$, all the entries of g^i and C^i must be updated and it can be done in at total of $O(k^3)$ operations, which does not depend on n.

Computing c_1 The second issue is the term $\sum_{j=1}^{n} \left({a^i_{:,q}}^T \sum_{l=1}^{r_j} b^j_{p,l} b^j_{:,l} \right)^2$ appearing in the computation of c_1. The loop over the dimension n can be avoided since we have

$$E^i_{p,q} = \sum_{j=1}^{n} \left({a^i_{:,q}}^T \sum_{l=1}^{r_j} b^j_{p,l} b^j_{:,l} \right)^2 = \left\langle a^i_{:,q} {a^i_{:,q}}^T, \sum_{j=1}^{n} B^j_{:,p} {B^j_{:,p}}^T \right\rangle.$$

In fact, if the quantity $\sum_{j=1}^{n} B^j_{:,p} {B^j_{:,p}}^T$ is available (and it is the case via the precomputed tensor D), we can maintain and update the column q of E^i in $O(k^2)$ operations:

$${E^i_{l,q}}^{new} \leftarrow {E^i_{l,q}}^{old} + 2({a^i_{p,q}}^{new} - {a^i_{p,q}}^{old}) {a^i_{:,q}}^T D_{:,l,p,l}.$$

Table 1 gathers the different quantities to precompute before the start of the iterations. Assuming that m and n are of the same order of magnitude, the overall computational complexity of the precomputations is $O(mk^2 \max(n, k^2))$. In the point of view of the space complexity, we observe that given the A^i's and the B^j's, the approximation matrix $\tilde{X}$ or the residual $X - \tilde{X}$ are never computed. In this way, the storage of a dense m-by-n matrix is avoided (which could be impractical with a large and sparse matrix X).

3.2.4 Variables selection: cyclic or greedy

Algorithm 5 illustrates a cyclic run of a CD scheme over all the variables. After the computation of the optimal value of one of the mk^2 entries of the problem, the updates of C^i and g^i in $O(k^3)$ operations are the bottleneck of the method causing the overall $O(mk^5)$ complexity.

Table 1 List of precomputations for the CD methods. (CC = computational complexity, SC = space complexity, $[x] = \{1, \ldots, x\}$)

	CC	SC
$A^i \leftarrow a^i {a^i}^T$ for all $i \in [m]$	$O(mk^3)$	$O(mk^2)$
$B^j \leftarrow b^j {b^j}^T$ for all $j \in [n]$	$O(nk^3)$	$O(nk^2)$
$C^i \leftarrow \sum_{j=1}^n \left(\left\langle A^i, B^j \right\rangle - X_{i,j} \right) B^j$ for all $i \in [m]$,	$O(mnk^2)$	$O(mk^2)$
$g^i \leftarrow 4{a^i}^T C^i$ for all $i \in [m]$	$O(mk^3)$	$O(mk^2)$
$D_{u,v,:,:} \leftarrow \sum_{j=1}^n B^j_{u,v} B^j$ for all $u, v \in [k]$	$O(nk^4)$	$O(k^4)$
$E^i_{p,q} \leftarrow \left\langle a^i_{:,q} {a^i_{:,q}}^T, \sum_{j=1}^n B^j_{:,p} {B^j_{:,p}}^T \right\rangle$ for all $i \in [m]$, $p, q \in [k]$	$O(mk^4)$	$O(mk^2)$

Algorithm 5 Optimize subproblem (7) with cyclic coordinate descent

```
INPUT: $X \in \mathbb{R}_+^{m \times n}$, $\{b^1, \ldots, b^n\} \in \mathbb{R}^{k \times r}$.
OUTPUT: $\{a^1, \ldots, a^m\} \in \mathbb{R}^{k \times r}$.
$\{a^1, \ldots, a^m\} \leftarrow$ Initialization
$[C, D, E, g] \leftarrow$Precomputation$(X, \{a^1, \ldots, a^m\}, \{b^1, \ldots, b^n\})$
for $i = 1 : m$ do
    for $p = 1 : k$ do
        for $q = 1 : r$ do
            $x \leftarrow a^i_{p,q}$
            $c_3 \leftarrow 4D_{p,p,p,p}$
            $c_2 \leftarrow 12{a^i_{:,q}}^T D_{p,p,p,:} - 3c_3x$
            $c_1 \leftarrow 4C^i_{p,p} + 8E^i_{p,q} - 2c_2x - 3c_3x^2$
            $c_0 \leftarrow 4g^i_{p,q} - c_1x - c_2x^2 - c_3x^3$
            $a^i_{p,q} \leftarrow CardanoMethod(c_3, c_2, c_1, c_0)$
            Update $C^i$, $g^i$, and $E^i_{:,q}$
        end for
    end for
end for
```

As explained above, the gradient of any variable is always available in Algorithm 5. In order to improve the efficiency of the algorithm, we propose to use the information given by the gradient for selecting first the coordinates in a greedy way instead of processing them cyclically. This is called the Gauss-Southwell rule: at each iteration, the variable with the largest gradient is updated. It allows to guide the CD scheme towards the coordinates that will potentially decrease the objective function the most. Algorithm 6 describes the implementation of the Gauss-Southwell strategy for PSD factorization. The main difference with Algorithm 5 lies in the selection of the variables to optimize.

We propose to make a number of iterations on each factor i proportional to kr (the number of variables) using the parameter α. In Sect. 4, the performances of Algorithm 6 are compared for different values of α.

Algorithm 6 Optimize subproblem (7) with Gauss-Southwell coordinate descent

```
INPUT: $X \in \mathbb{R}_+^{m \times n}$, $\{b^1, \dots, b^n\} \in \mathbb{R}^{k \times r}$, $\alpha \in \mathbb{R}_+$.
OUTPUT: $\{a^1, \dots, a^m\} \in \mathbb{R}^{k \times r}$.
$\{a^1, \dots, a^m\} \leftarrow$ Initialization
$[C, D, E, g] \leftarrow$Precomputation($X$, $\{a^1, \dots, a^m\}$, $\{b^1, \dots, b^n\}$)
for $i = 1 : m$ do
    for $t = 1 : \lceil \alpha k r \rceil$ do
        $(p^*, q^*) = \arg\max_{p,q} |g^i_{p,q}|$
        $x \leftarrow a^i_{p*,q*}$
        $c_3 \leftarrow 4D_{p*,p*,p*,p*}$
        $c_2 \leftarrow 12{a^i_{:,q*}}^T D_{p*,p*,p*,:} - 3c_3 x$
        $c_1 \leftarrow 4C^i_{p*,p*} + 8E^i_{p*,q*} - 2c_2 x - 3c_3 x^2$
        $c_0 \leftarrow 4g^i_{p*,q*} - c_1 x - c_2 x^2 - c_3 x^3$
        $a^i_{p^*,q^*} \leftarrow CardanoMethod(c_3, c_2, c_1, c_0)$
        Update $C^i$, $g^i$, and $E^i_{:,q}$
    end for
end for
```

3.2.5 Inner rank of the factors

In many cases, the factors A^i's and the B^j's are rank deficient. For example, in the exact case ($X_{ij} = \langle A^i, B^j \rangle$ for all i, j), if $X_{ij} = 0$ and the ith row of X and jth column of X are not identically zero (implying $A^i \neq 0$ and $B^j \neq 0$), A^i and B^j cannot be full rank otherwise $\langle A^i, B^j \rangle > 0$. For slack matrices, there is at least one zero per row and per column in X, hence $r_i \leq k - 1$ for all i. In fact, this idea can be generalized [22] to improve the upper bound on the r_i's, and was used for example in [7].

With the CD methods previously presented, it is easy to allow different values for the rank of the A^i's by using initial factors a^i's with appropriate sizes. However, for the numerical experiments in Sect. 4, we will use $r_i = k$ for all i to have a fair comparison with FPGM and to check whether the coordinate descent algorithms are able to generate low-rank factors. Moreover, this possibility to handle rank deficient factors will allow us to focus on the problem of the square root rank where $r_i = 1$ for all factors; see Sect. 5.3.

4 Numerical experiments

The algorithms presented in the previous section are the first numerical methods developed for solving the optimization problem (2). It is therefore not possible to any make experimental comparisons with algorithms from the literature. However, this section has two main goals:

- In Algorithms 4 and 6, there are parameters that may influence the effectiveness of the methods, Δ and α respectively. Hence the first goal is to compare the performances of these two algorithms for different values of the parameters.

- Once the best values of the parameters are known, the second goal is to compare the Algorithms 4, 5 and 6. This will allow us to select the most effective algorithm to solve the PSD factorization problems discussed in Sect. 5.

4.1 Initialization and scaling

Algorithms 4, 5 and 6 are iterative and need starting points. In this paper, the entries of the a^i's and the b^j's are initialized using the normal distribution $\mathcal{N}(0, 1)$. Note that for Algorithm 4, we use $a^i a^{i^T}$ and $b^j b^{j^T}$ as random initial iterates so that all algorithms are initialized with the same values.

However, it may happen that with such random factors, we have an initial approximation matrix $\tilde{X}$ way larger or smaller than X. In order to avoid such situations, we scale the initial factors compared to X: given initial iterates a^i and b^j, we compute

$$\begin{aligned}\lambda^* &= \arg\min_{\lambda} \sum_{i=1}^{m} \sum_{j=1}^{n} \left(X_{i,j} - \lambda \left\langle A^i, B^j \right\rangle \right)^2 \\ &= \arg\min_{\lambda} \left\| X - \lambda \mathcal{A}^T \mathcal{B} \right\|_F^2 \\ &= \frac{\left\langle X\mathcal{B}^T, \mathcal{A} \right\rangle}{\left\langle \mathcal{B}\mathcal{B}^T, \mathcal{A}\mathcal{A}^T \right\rangle}, \end{aligned} \tag{11}$$

with $\mathcal{A} = \left(vec(A^1) \ \ldots \ vec(A^m) \right)$ and $\mathcal{B} = \left(vec(B^1) \ \ldots \ vec(B^n) \right)$. The initial error is therefore

$$e_0 = \left\| X - \tilde{X} \right\|_F = \sqrt{\|X\|_2^2 - \frac{\left\langle X\mathcal{B}^T, \mathcal{A} \right\rangle^2}{\left\langle \mathcal{B}\mathcal{B}^T, \mathcal{A}\mathcal{A}^T \right\rangle}} \leq \|X\|_F ,$$

with the appropriate scaling,

- $A^i \leftarrow \lambda^* A^i$ for $i = 1, \ldots, m$ for FPGM and,
- $a^i \leftarrow \sqrt{\lambda^*} a^i$ for $i = 1, \ldots, m$ for the CD methods.

4.2 Data sets

The matrices used for the numerical comparisons are slack matrices; see the discussion in Sect. 2. Table 2 summarizes the different matrices used in the tests. The factorization rank k used in the experiments is specified in the fourth column. Note that this is not necessarily the true value of the rank_{psd} which is used, but it is either a conjecture or an upper bound. The data set is composed of three types of matrices:

- The slack matrices of regular n-gons are n-by-n circulant matrices for which the (i, j)th entry is the slack between the ith facet and the jth vertex of the regular n-gon (see [28] for more details on the construction of such matrices). The values of the factorization rank k are given by the conjecture made on the rank_{psd} of regular n-gons in Sect. 5.1.

- For a given positive integer n, let U_n (resp. V_n) be the $\{0,1\}^{\binom{n}{\lfloor \frac{n}{2} \rfloor}\times n}$ (resp. $\{0,1\}^{\binom{n}{\lceil \frac{n}{2} \rceil}\times n}$) matrix where the rows correspond to the subsets of $\{1,\ldots,n\}$ of size $\binom{n}{\lfloor \frac{n}{2} \rfloor}$ (resp. $\binom{n}{\lceil \frac{n}{2} \rceil}$). Let P_n be the $\binom{n}{\lfloor \frac{n}{2} \rfloor}$-by-$\binom{n}{\lceil \frac{n}{2} \rceil}$ matrix defined as

$$P_n = U_n V_n^T.$$

These matrices have an interpretation in terms of an inscribed polytope in the $(n-2)$-sphere (see Problems 9.1 and 9.2 in [6]). The exact value of $\text{rank}_{\text{psd}}(P_n)$ is not known but it is bounded as follows,

$$\left\lceil \frac{\sqrt{1+8n}-1}{2} \right\rceil \leq \text{rank}_{\text{psd}}(P_n) \leq 2\left\lceil \sqrt{n} \right\rceil,$$

except for $n=5$ for which $3 \leq \text{rank}_{\text{psd}}(P_5) \leq 4$. The values of the factorization rank k of the matrices P_n used in the tests are the upper bounds mentioned above.

- The correlation polytope is the convex hull of all n-by-n rank-one 0/1 matrices. Let COR_n be a submatrix of the slack matrix of the correlation polytope. The rows and columns of this 2^n-by-2^n matrix are indexed by vectors $u, v \in \{0,1\}^n$ such that

$$COR_n(u,v) = \left(1 - u^T v\right)^2.$$

Although the nonnegative rank of COR_n has been proved to be exponential in n, there exists an explicit PSD factorization such that $\text{rank}_{\text{psd}}(COR_n) = n+1$ [8]. These values are used for the factorization rank in the tests.

4.3 Comparisons for different values of the parameters

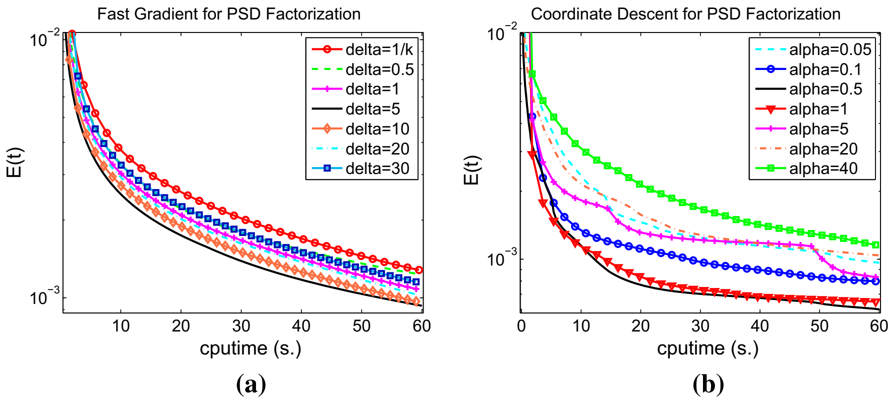
In order to compare the performances of the algorithms, we use the measure $E(t)$ defined by

$$E(t) = \frac{e(t)}{e_0} \tag{12}$$

where e_0 is the initial error (see Sect. 4.1), and $e(t)$ is the error $\|X - \tilde{X}\|_F$ achieved by an algorithm for a given initialization within t seconds. Since our algorithms are nonincreasing, we have $E(t) \in [0,1]$ for all t, with $E(0) = 1$ and $E(t) \rightarrow_{t\rightarrow\infty} 0$ if the corresponding algorithm converges towards an exact factorization. In order to illustrate the efficiency of a given algorithm, (12) has the advantage that it makes sense to take the average of $E(t)$ for several initializations and data sets and display a single curve. The algorithms were run 10 times with different initializations during 60 s for the following parameters values:

Table 2 Benchmark of nonnegative matrices used in the numerical comparisons

	m	n	k
Slack matrix of the 12-gon	12	12	5
Slack matrix of the 16-gon	16	16	5
Slack matrix of the 20-gon	20	20	6
Slack matrix of the 24-gon	24	24	6
Slack matrix of the 28-gon	28	28	6
Slack matrix of the 32-gon	32	32	6
P_5	10	10	4
P_6	20	20	6
P_7	35	35	6
COR_3	8	8	4
COR_4	16	16	5
COR_5	32	32	6



(a) **(b)**

Fig. 3 Evolution of the average measure $E(t)$ for different values of the parameters Δ and α on the data sets of Table 2

- $\Delta = \{\frac{1}{k}, 0.5, 1, 5, 10, 20, 30\}$ for Algorithm 4, and
- $\alpha = \{0.05, 0.1, 0.5, 1, 5, 20, 40\}$ for Algorithm 6.

FPGM was implemented with Matlab while the CD methods were developed in C with a Matlab interface using Mex files. The reason is that Matlab is not a well-suited language when one requires to perform many loops as in Algorithms 5 and 6. The codes are available at https://sites.google.com/site/exactnmf/. All tests were performed on a PC Intel CORE i5-4570 CPU @3.2GHz × 4, with 7.7G RAM.

The results are displayed on Fig. 3.

For FPGM, we observe that the number of inner steps does not influence the efficiency significantly. We observe that the best average performances are obtained around $\Delta = 5$. For the Gauss-Southwell algorithm, the best value of the parameter α is between 0.5 and 1. It means that the number of updated entries must be roughly the same as in the cyclic case. For the numerical tests that follow, we use the following algorithms:

- FPGM with $\Delta = 5$.
- The cyclic CD algorithm.
- The Gauss Southwell CD algorithm with $\alpha = 0.5$.

Note that this choice of parameters is by no means the best in all situations, although they perform best in average on the tested data sets. However, for simplicity, we will use them for the remainder of the paper. Of course, nothing prevents the user to fine tune these parameters for his own data sets.

In the remaining of the section, we compare the performances of these algorithms. Instead of ploting an average measure, for each matrix and each method, we display the curves of the error $\|X - \tilde{X}\|_F$ corresponding to five different initializations. It allows us to observe the behavior of the methods for different starting points. The data sets used are those described in Table 2. For each type of matrices, we present the results for two instances: the matrices with the smallest and the largest size.

From Fig. 4, we observe the following:

- There is a general trend emerging from these numerical tests: the Gauss-Southwell CD method outperforms the cyclic strategy, while this last method performs better than FPGM.
- Algorithm are very sensitive to initialization. For example, the solutions obtained with FPGM on the 12-gon after 60 s are rather different, illustrating the fact the local algorithms can get stuck in local minima.

  This is clear from the results obtained with the *COR*3 matrix where most of the runs get stuck in local minima.
- Although the results presented on the left of Fig. 4 are instances of small sizes, the final error $\|X - \tilde{X}\|_F$ remains relatively large even after 60 s. It contrasts with NMF where the convergence on small matrices is faster [29].

In conclusion, we recommend to use the Gauss-Southwell CD method which performs best in most cases. This algorithm will therefore be used in the next section for several applications where the rank_{psd} is sought.

5 Applications

In this section, we discuss the use of our numerical algorithms for the computation of the psd-rank of particular matrices. In this purpose, let us give the following (obvious) fact.

Observation 1 *For a given matrix X with $\text{rank}_{\text{psd}}(X) = k$, let us denote $\tilde{X}_l^*$ the best approximation matrix with l-by-l PSD factors. By definition of rank_{psd}, we have*

$$\|X - \tilde{X}_l^*\|_F = 0 \textit{ for all } l \geq k, \quad \textit{and} \quad \|X - \tilde{X}_l^*\|_F > 0 \textit{ for all } l < k.$$

Given a matrix X and a target factorization rank k, our nonlinear local optimization methods provide no guarantee; we can only hope to identify good local minima of the

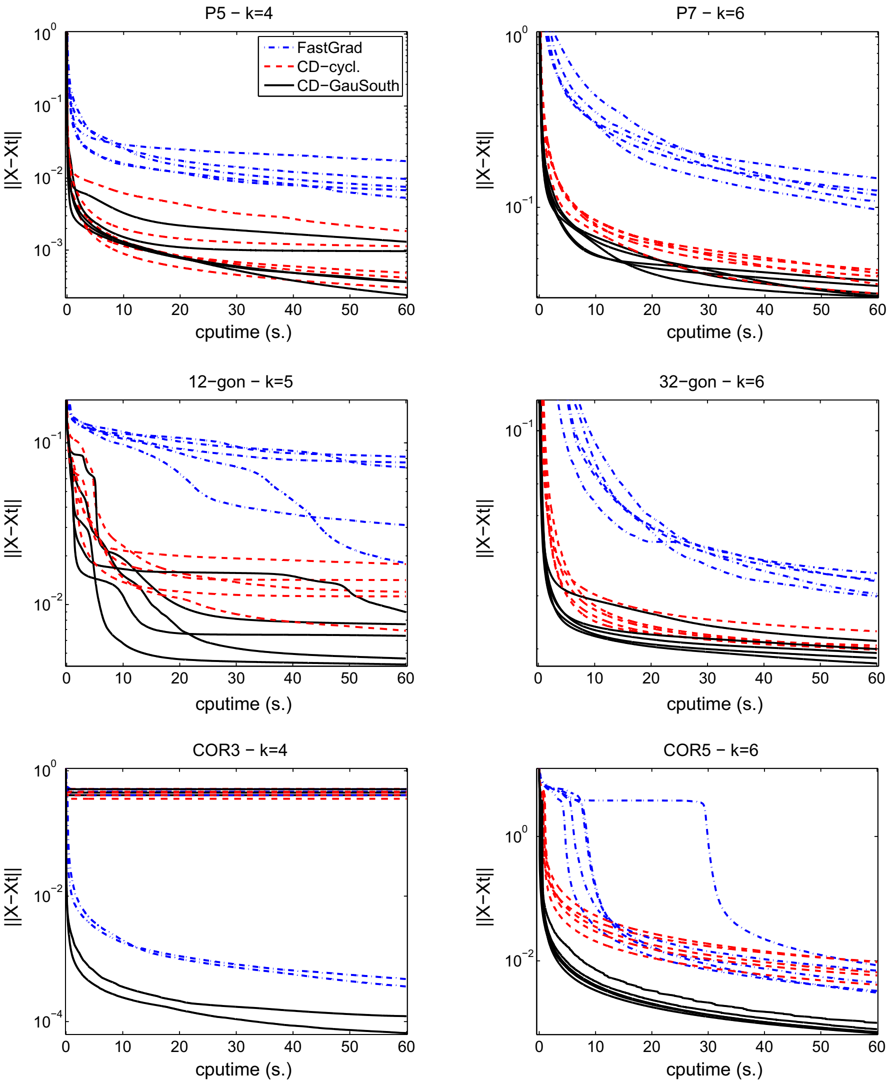


Fig. 4 Evolution of the error for the different algorithms on the dataset

nonconvex problem (2). However, as experimentally demonstrated in [29] for exact NMF, such algorithms can be used in multi-start strategies to detect if the error $\|X - \tilde{X}\|_F$ gets (close) to zero. Moreover, beside conjectures on the psd-rank, Algorithms 5 and 6 can be helpful to find exact factorizations by trial and error and by fixing manually some entries to specific values.

As an illustration, we discuss the value of the psd-rank of the regular polygons in Sect. 5.1. With the help of Algorithm 6, a conjecture is proposed which is confirmed showing exact factorizations, for the first time, for $n = 5$, $n = 8$ and $n = 10$. In Sects. 5.2 and 5.3, we show how to adapt our methods in order to deal with two related problems, the completely PSD Factorization problem and the problem of computing the square root rank.

5.1 Conjecture on the psd-rank of regular n-gons

Let S_n denote the slack matrix of the regular n-gon. We have that

$$\Omega\left(\frac{\log n}{\log\log n}\right) \leq \operatorname{rank}_{\text{psd}}(S_n) \leq 2\lceil\log_2(n)\rceil,$$

where the first inequality comes from quantifier elimination theory [13,14] and the second inequality uses the upper bound on $\operatorname{rank}_{+}(S_n)$ [9]. The exact value of $\operatorname{rank}_{\text{psd}}(S_n)$ is unknown for general n, and it is an important open question. However, it is known that (i) the psd-rank of the square is three, (ii) all pentagons and hexagons have psd-rank exactly four and (iii) the psd-rank of the heptagons is either four or five [14]. Moreover, to the best of our knowledge, an explicit factorization for regular n-gons is only known for $n = 3$, $n = 4$ and $n = 6$.

For different values of n and k, we run Algorithm 6 on S_n with the inner rank of the factors $r = k - 2$ (see Sect. 3.2.5). Table 3 reports the smallest relative error $\frac{\|X-\tilde{X}\|_F}{\|X\|_F}$ found after 100 runs of 10 s with different initializations.

In order to guess a value for the psd-rank of S_n, we have to look at the corresponding row of Table 3. If an exact factorization is possible for k, the error should be close to zero in the entry (n, k) and larger in the entry $(n, k-1)$. For the smallest regular n-gons ($n = 3, 4, 5,$ and 6), the obtained errors are consistent with the known values of the psd-

Table 3 Smallest relative error obtained over 100 different runs of 10 s

	$k = 3$	$k = 4$	$k = 5$	$k = 6$	$k = 7$
$n = 3$	**3.1e−7**				
$n = 4$	**1.3e−7**	6.6e−7			
$n = 5$	0.065	**2.4e−6**	3.8e−6		
$n = 6$	0.049	**5.5e−6**	5.4e−6		
$n = 7$	0.036	**3.9e−5**	1.3e−5		
$n = 8$	0.028	**1.9e−5**	3.4e−5		
$n = 9$	0.022	0.004	**8.5e−5**	3.7e−5	
$n = 10$	0.018	0.003	**8.1e−5**	4.9e−5	
$n = 11$	0.015	0.006	**1.4e−4**	5.4e−5	
$n = 12$	0.012	0.007	**2.7e−4**	1.2e−4	
$n = 13$	0.01	0.007	**5.5e−4**	1.6e−4	
$n = 14$	0.009	0.006	**6.9e−4**	2.8e−4	
$n = 15$	0.008	0.005	**8e−4**	4e−4	
$n = 16$	0.007	0.005	**0.001**	5e−4	
$n = 17$	0.006	0.004	0.002	**5.6e−4**	4.6e−4

rank. For $n = 7$ and $n = 8$, the results suggest[1] that $\mathrm{rank}_{\mathrm{psd}}(S_7) = \mathrm{rank}_{\mathrm{psd}}(S_8) = 4$. Actually, there is a pattern emerging for $n \geq 7$ leading to the following conjecture:

Conjecture 1 *The psd-rank of S_n, the slack matrix of the regular n-gon, is given by*

$$\mathrm{rank}_{\mathrm{psd}}(S_n) = 1 + \lceil \log_2(n) \rceil.$$

In Table 3, the entries corresponding to the conjecture are highlighted in bold. We have not pursued the computations beyond $n > 17$ because the results are less and less clear. The reason is that as n gets bigger, the regular n-gon get closer to the circle which has a psd-lift of size 2.

By trial and error and by fixing more and more entries manually in the factors, we were able to construct, for the first time, an exact PSD factorization of the 5-gon, the 8-gon and the 10-gon with respective sizes consistent with Conjecture 1; see the examples below.

Example 5.1 With $\phi = \frac{1+\sqrt{5}}{2}$, a slack matrix of the regular 5-gon is given by:

$$S_5 = \begin{pmatrix} 0 & 1 & \phi & 1 & 0 \\ 0 & 0 & 1 & \phi & 1 \\ 1 & 0 & 0 & 1 & \phi \\ \phi & 1 & 0 & 0 & 1 \\ 1 & \phi & 1 & 0 & 0 \end{pmatrix}.$$

A $\mathcal{S}^4_+$-factorization of S_5 is given by the following factors:

$$a^i = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \frac{1}{1+\sqrt{\phi}} \\ \frac{1}{1+\sqrt{\phi}} \\ 1 \\ \phi - \frac{1}{\sqrt{\phi}} \end{pmatrix} \right\},$$

$$b^j = \left\{ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1-(\sqrt{\phi})^3}{2} & \sqrt{1 - \left(\frac{1-(\sqrt{\phi})^3}{2}\right)^2} \\ \sqrt{\phi} & 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} \sqrt{\phi} \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ a \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ \sqrt{\phi} \\ -1 \end{pmatrix} \right\}.$$

[1] Example 5.2 provides an explicit PSD factorization of size 4 for S_8. For S_7, we were not able to obtain such an exact factorization of size 4, although we have tried many different initializations. It is possible that $\mathrm{rank}_{\mathrm{psd}}(S_7) = 5$ since there is no result about the monotonicity of the PSD rank of regular n-gons (this is, as far as we know, an open question). In fact, [12] showed that monotonicity does not hold for the PSD rank over the complex numbers with $\mathrm{rank}^{\mathbb{C}}_{\mathrm{psd}}(S_6) = 3 < 4 \leq \mathrm{rank}^{\mathbb{C}}_{\mathrm{psd}}(S_5)$.

Example 5.2 A slack matrix of the 8-gon is given by

$$S_8 = \begin{pmatrix} 0 & 1 & 1+\sqrt{2} & 2+\sqrt{2} & 2+\sqrt{2} & 1+\sqrt{2} & 1 & 0 \\ 0 & 0 & 1 & 1+\sqrt{2} & 2+\sqrt{2} & 2+\sqrt{2} & 1+\sqrt{2} & 1 \\ 1 & 0 & 0 & 1 & 1+\sqrt{2} & 2+\sqrt{2} & 2+\sqrt{2} & 1+\sqrt{2} \\ 1+\sqrt{2} & 1 & 0 & 0 & 1 & 1+\sqrt{2} & 2+\sqrt{2} & 2+\sqrt{2} \\ 2+\sqrt{2} & 1+\sqrt{2} & 1 & 0 & 0 & 1 & 1+\sqrt{2} & 2+\sqrt{2} \\ 2+\sqrt{2} & 2+\sqrt{2} & 1+\sqrt{2} & 1 & 0 & 0 & 1 & 1+\sqrt{2} \\ 1+\sqrt{2} & 2+\sqrt{2} & 2+\sqrt{2} & 1+\sqrt{2} & 1 & 0 & 0 & 1 \\ 1 & 1+\sqrt{2} & 2+\sqrt{2} & 2+\sqrt{2} & 1+\sqrt{2} & 1 & 0 & 0 \end{pmatrix}.$$

Let $\alpha_1 = \sqrt{1+\sqrt{2}}, \alpha_2 = \sqrt{2+\sqrt{2}}, \alpha_3 = \frac{1}{\alpha_1} - \alpha_1, \alpha_4 = \sqrt{\sqrt{2}}$ and $\alpha_5 = \sqrt{1 - \frac{1}{\alpha_1^2}}$. A $\mathcal{S}_+^4$-factorization of S_8 is given by the following factors:

$$a^i = \left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -\alpha_1 \\ -\alpha_1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ \alpha_3 \\ \alpha_3 \\ \frac{-1}{\alpha_1} \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \\ -2 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} -1 \\ \frac{-1}{\alpha_1} \\ \frac{-1}{\alpha_1} \\ \frac{1}{\alpha_1} \end{pmatrix} \right\},$$

$$b^j = \left\{ \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ -1 & 0 \\ -1 & \alpha_1 \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -\alpha_2 \end{pmatrix}, \begin{pmatrix} \alpha_1 & 0 \\ 1 & 0 \\ 0 & 0 \\ 1 & \alpha_1 \end{pmatrix}, \begin{pmatrix} -\alpha_2 & 0 \\ -\alpha_4 & 1 \\ 0 & -1 \\ -\alpha_4 & 0 \end{pmatrix}, \begin{pmatrix} \alpha_2 & 0 \\ 0 & \alpha_2 \\ \frac{1}{\alpha_4} & \frac{-\alpha_1^2}{\alpha_2} \\ \frac{2}{\alpha_4} & \frac{-\sqrt{2}}{2} \end{pmatrix}, \right.$$

$$\left. \begin{pmatrix} \alpha_1 & 0 \\ -\sqrt{2} & \alpha_4 \\ \sqrt{2} & -\alpha_4 \\ \sqrt{2} & -\alpha_4 \end{pmatrix}, \begin{pmatrix} -1 & 0 \\ \alpha_1 & 0 \\ -\alpha_1 & 1 \\ -\alpha_1 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ \frac{-1}{\alpha_1} & \alpha_5 \\ \alpha_1 & 0 \\ -\alpha_3 & \alpha_5 \end{pmatrix} \right\},$$

Example 5.3 A slack matrix of the 10-gon with $\phi = \frac{1+\sqrt{5}}{2}$ is given by

$$S_{10} = \begin{pmatrix} 0 & 0 & \phi^{-2} & 1 & \phi & 2 & 2 & \phi & 1 & \phi^{-2} \\ \phi^{-2} & 0 & 0 & \phi^{-2} & 1 & \phi & 2 & 2 & \phi & 1 \\ 1 & \phi^{-2} & 0 & 0 & \phi^{-2} & 1 & \phi & 2 & 2 & \phi \\ \phi & 1 & \phi^{-2} & 0 & 0 & \phi^{-2} & 1 & \phi & 2 & 2 \\ 2 & \phi & 1 & \phi^{-2} & 0 & 0 & \phi^{-2} & 1 & \phi & 2 \\ 2 & 2 & \phi & 1 & \phi^{-2} & 0 & 0 & \phi^{-2} & 1 & \phi \\ \phi & 2 & 2 & \phi & 1 & \phi^{-2} & 0 & 0 & \phi^{-2} & 1 \\ 1 & \phi & 2 & 2 & \phi & 1 & \phi^{-2} & 0 & 0 & \phi^{-2} \\ \phi^{-2} & 1 & \phi & 2 & 2 & \phi & 1 & \phi^{-2} & 0 & 0 \\ 0 & \phi^{-2} & 1 & \phi & 2 & 2 & \phi & 1 & \phi^{-2} & 0 \end{pmatrix}.$$

Let $\alpha_1 = (\sqrt{2}\phi)^{-1/2}$, $\alpha_2 = (\sqrt{2}/\phi)^{1/2}$, $\alpha_3 = \sqrt{2/\phi}$, $\alpha_4 = \sqrt{\sqrt{2}\phi}$, $\alpha_5 = -\phi^{3/2}$ and $\alpha_6 = \sqrt{\sqrt{5}-1}$. A $\mathcal{S}_+^5$-factorization of S_{10} is given by the following factors:

$$a^i = \left\{ \begin{pmatrix} 0 & \alpha_1^{-1} & 0 \\ 0 & \alpha_4 & 0 \\ 1 & \alpha_5 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & (\alpha_1\phi)^{-1} & 0 \\ \alpha_2 & \alpha_4\phi^{-1} & 0 \\ -1 & \alpha_5\phi^{-1} & 0 \\ 0 & -\phi^{-1} & 0 \\ 0 & 0 & \sqrt{\phi} \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ \alpha_2 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & \phi^{-1} & 0 \\ 0 & 0 & \sqrt{\phi} \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \right.$$

$$\begin{pmatrix} \alpha_2 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & \phi & 0 \\ 0 & 0 & \alpha_3 \\ 0 & 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 & \alpha_1^{-1} \\ 0 & 0 & \alpha_2 \\ 1 & 0 & \sqrt{2\phi}-1 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{pmatrix},$$

$$\begin{pmatrix} 0 & 0 & \alpha_4 \\ \alpha_2 & 0 & 2^{1/4} \\ -1 & 0 & \sqrt{2\phi}-\sqrt{\phi} \\ 0 & \phi^{-1} & 0 \\ 0 & 0 & -\sqrt{\phi} \end{pmatrix}, \begin{pmatrix} 0 & (\alpha_1\phi)^{-1} & \alpha_4 \\ \alpha_2 & \alpha_4\phi^{-1} & 2^{1/4} \\ -1 & \alpha_5\phi^{-1} & \sqrt{2\phi}-\sqrt{\phi} \\ 0 & -\phi^{-1} & 0 \\ 0 & 0 & -\sqrt{\phi} \end{pmatrix}, \begin{pmatrix} 0 & \alpha_1^{-1} & \alpha_1^{-1} \\ 0 & \alpha_4 & \alpha_2 \\ 1 & \alpha_5 & \sqrt{2\phi}-1 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix},$$

$$\left. \begin{pmatrix} \alpha_2 & 0 & \alpha_3\phi^{-1} \\ 0 & 0 & \alpha_4\alpha_3 \\ 0 & \phi & \alpha_5\alpha_3 \\ 0 & 0 & -\alpha_3 \\ 0 & 0 & 0 \end{pmatrix} \right\},$$

$$b^j = \left\{ \begin{pmatrix} 0 \\ \alpha_1 \\ 0 \\ \sqrt{\phi} \\ 0 \end{pmatrix}, \begin{pmatrix} \alpha_1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \alpha_1 \\ \phi^{-1} \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \alpha_1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \alpha_1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ \alpha_1 \\ 0 \\ 0 \\ \phi^{-1} \end{pmatrix}, \begin{pmatrix} \alpha_1 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ \alpha_1 \\ \phi^{-1} \\ 0 \\ \alpha_6 \end{pmatrix}, \begin{pmatrix} \alpha_1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}, \begin{pmatrix} 0 \\ \alpha_1 \\ 0 \\ \sqrt{\phi} \\ \phi^{-1} \end{pmatrix} \right\}.$$

5.2 Adaptation for completely PSD matrices

For NMF involving a symmetric n-by-n matrix X, the additional constraint requiring W and H to be equal to each other leads to an optimization problem known as symmetric NMF (SymNMF). Specific numerical algorithms have been designed for this problem having applications in data mining [17,20,30]. When an exact factorization is possible, that is, $X = HH^T$ for a nonnegative n-by-k matrix H, the matrix X is said to be completely positive. The smallest integer k for which such an exact factorization exists is referred as the cp-rank of X [1].

By analogy with completely positive matrices, a completely positive semidefinite matrix X is defined as a n-by-n symmetric matrix for which there exists a set $A^1, \ldots, A^n \in \mathcal{S}_+^k$ such that $X_{i,j} = \langle A^i, A^j \rangle$. The smallest integer k for which it is possible to write such a factorization is called the cpsd-rank of X; see [15,25] for a recent survey. As opposed to problem (4), the symmetric version

$$\min_{\substack{A^i \in \mathcal{S}^k_+ \\ i=1,\ldots,n}} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(X_{i,j} - \left\langle A^i, A^j \right\rangle \right)^2, \tag{13}$$

is no longer convex even when all factors A^i's are fixed but one. However, it is possible to adapt the methods developed in Sect. 3.2 in order to handle (13). We propose to keep the problem with two sets of variables but we add a penalty term to (7) with a scalar $\gamma > 0$ in order to enforce the similarity between a^i and b^i for $i = 1, \ldots, n$, similarly as done for Symmetric NMF in [17,20]:

$$\min_{\substack{a^i \in \mathbb{R}^{k \times r_i} \\ i=1,\ldots,n}} f = \sum_{i=1}^{n} \sum_{j=1}^{n} \left(X_{i,j} - \sum_{h=1}^{r_i} \sum_{l=1}^{r_j} \left({a^i_{:,h}}^T b^j_{:,l} \right)^2 \right)^2 + \gamma \sum_{i=1}^{n} \|a^i - b^i\|_2^2.$$

This modification of the objective function has limited consequences on Algorithms 5 and 6 since the additional terms are quadratic. The entry of the gradient corresponding to the variable $a^i_{p,q}$ is given by

$$\nabla_{a^i_{p,q}} f = c_3 {a^i_{p,q}}^3 + c_2 {a^i_{p,q}}^2 + (c_1 + 2\gamma) a^i_{p,q} + (c_0 - 2\gamma b^i_{p,q}).$$

With this change, we are now able to compute symmetric factorizations.

Example 5.4 The symmetric 6-by-6 matrix

$$P_4 = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & 1 & 0 & 1 \\ 1 & 1 & 2 & 0 & 1 & 1 \\ 1 & 1 & 0 & 2 & 1 & 1 \\ 1 & 0 & 1 & 1 & 2 & 1 \\ 0 & 1 & 1 & 1 & 1 & 2 \end{pmatrix},$$

as defined in Sect. 4.2 has a symmetric factorization with $k = 4$ with the factors

$$a^i = \left\{ \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} \right\}.$$

The choice of the parameter γ can be made in different ways and should be increased in the course of the optimization process in order to ensure that a^i converges to b^i for all i. For this particular example, we simply used $\gamma = 1$ which gave us the desired result.

5.3 Adaptation for the square root rank

Given a nonnegative matrix X, a *Hadamard square root* of X is defined as a matrix obtained by replacing the (i, j)th entry of X by either $\sqrt{X_{i,j}}$, or $-\sqrt{X_{i,j}}$. Hence there are 2^N possible Hadamard square root matrices for a matrix X with N non-zero entries. The *square root rank* of a nonnegative matrix X is defined as the minimum rank among the ranks of all the Hadamard square root of X. If a nonnegative matrix X has square root rank k, then there is an exact PSD factorization of X with rank-1 factors of size k; see Proposition 6.2. in [6] (hence the square root rank of X is an upper bound on the psd-rank of X). Therefore, we can use Algorithms 5 and 6 with $r_i = 1$ for all i to try to compute the square root rank of X. Note that computing this quantity is NP-hard as well [6].

Example 5.5 For the 8-gon (and its slack matrix S_8; see Example 5.2), we have computed such a rank-one decomposition with $k = 6$ and $r_i = 1$ for all i. Note that to compute this decomposition, we had to use many different starting points (around a thousand) and manually fix some entries of the a^i's and b^j's to zero. In order to present this decomposition, let us define

$$S = \begin{pmatrix} 0 & -1 & -1 & 1 & -1 & -1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & -1 & 1 & -1 \\ 1 & 0 & 0 & -1 & -1 & 1 & -1 & 1 \\ 1 & -1 & 0 & 0 & -1 & -1 & 1 & -1 \\ 1 & -1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & -1 & 0 & 0 & -1 & -1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 \\ -1 & 1 & -1 & 1 & -1 & -1 & 0 & 0 \end{pmatrix},$$

$$W = \begin{pmatrix} 0 & -1 & -\alpha_1 & \alpha_2 & -\alpha_2 & -\alpha_1 \\ 0 & 0 & 1 & \alpha_1 & \alpha_2 & -\alpha_2 \\ 1 & 0 & 0 & -1 & -\alpha_1 & \alpha_2 \\ \alpha_1 & -1 & 0 & 0 & -1 & -\alpha_1 \\ \alpha_2 & -\alpha_1 & 1 & 0 & 0 & 1 \\ \alpha_2 & \alpha_2 & \alpha_1 & -1 & 0 & 0 \\ \alpha_1 & \alpha_2 & \alpha_2 & \alpha_1 & 1 & 0 \\ -1 & \alpha_1 & -\alpha_2 & \alpha_2 & -\alpha_1 & -1 \end{pmatrix}, \text{ and } H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \frac{\alpha_2}{\alpha_1 - 1} & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & \frac{1-\alpha_1}{\alpha_2} \\ 0 & 0 & 1 & 0 & 0 & 0 & \frac{1+\alpha_1}{1-\alpha_1} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & \frac{\alpha_2}{\alpha_1 - 1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \frac{1+\alpha_1}{\alpha_2} \end{pmatrix},$$

with $\alpha_1 = \sqrt{1 + \sqrt{2}}$ and $\alpha_2 = \sqrt{2 + \sqrt{2}}$. Denoting $\sqrt[+]{S_8}$ the nonnegative Hadamard square root of S_8, one can check that $S \circ \sqrt[+]{S_8} = WH$ implying that the square root rank of S_8 is at most 6. Our algorithms were not able to compute such a decomposition for $k = 5$ (relative error always at least 0.6%).

6 Conclusion

In this work, we introduced different algorithms for solving numerically the PSD factorization problem (2). These algorithms are based on an alternating strategy in order to solve convex subproblems. The first method proposed uses PSD matrices as variables and implements a fast projected gradient method. The second idea is to apply the coordinate descent (CD) framework after having expressed the problem as an unconstrained optimization problem. Numerical experiments have been conducted to assess the performances of the different methods, and we observed that CD with the Gauss-Southwell rule performs consistently the best. Finally, we have illustrated the ability of our algorithms to help in the computation of non-trivial factorizations for regular n-gons, for symmetric PSD factorizations and for the square root rank. Note that an earlier version of our code was also used successfully in [21].

An important direction for future research is the development of a globalization framework, such as in [29] for NMF, in order to escape local minima and generate, in average, better solutions than with a simple multi-start strategy as used in this paper.

References

1. Berman, A., Shaked-Monderer, N.: Completely Positive Matrices. World Scientific, Singapore (2003)
2. Burer, S., Monteiro, R.: A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. Math. Program. **95**(2), 329–357 (2003)
3. Cardano, G.: Ars Magna or the Rules of Algebra. Dover Publications, Mineola (1968)
4. Cichocki, A., Phan, A.H.: Fast local algorithms for large scale nonnegative matrix and tensor factorizations. IEICE Trans. Fundam. Electron. **E92–A**(3), 708–721 (2009)
5. Cichocki, A., Zdunek, R., Amari, S.i.: Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization. In: International Conference on Independent Component Analysis and Signal Separation, pp. 169–176. Springer (2007)
6. Fawzi, H., Gouveia, J., Parrilo, P., Robinson, R., Thomas, R.: Positive semidefinite rank. Math. Program. **153**(1), 133–177 (2015)
7. Fawzi, H., Gouveia, J., Robinson, R.: Rational and real positive semidefinite rank can be different. Oper. Res. Lett. **44**(1), 59–60 (2016)
8. Fiorini, S., Massar, S., Pokutta, S., Tiwary, H., de Wolf, R.: Linear vs. semidefinite extended formulations: exponential separation and strong lower bounds. In: Proceedings of the 44th Annual ACM Symposium on Theory of Computing, pp. 95–106. ACM (2012)
9. Fiorini, S., Rothvoss, T., Tiwary, H.: Extended formulations for polygons. Discrete Comput. Geom. **48**(3), 658–668 (2012)
10. Gillis, N.: The why and how of nonnegative matrix factorization. In: Suykens, J., Signoretto, M., Argyriou, A. (eds.) Regularization, Optimization, Kernels, and Support Vector Machines‘. Chapman & Hall/CRC, Boca Raton (2014). Machine Learning and Pattern Recognition Series
11. Gillis, N., Glineur, F.: Accelerated multiplicative updates and hierarchical ALS algorithms for nonnegative matrix factorization. Neural Comput. **24**(4), 1085–1105 (2012)
12. Goucha, A., Gouveia, J., Silva, P.: On ranks of regular polygons. SIAM J. Discrete Math. **31**(4), 2612–2625 (2017)
13. Gouveia, J., Parrilo, P., Thomas, R.: Lifts of convex sets and cone factorizations. Math. Oper. Res. **38**(2), 248–264 (2013)
14. Gouveia, J., Robinson, R., Thomas, R.: Worst-case results for positive semidefinite rank. Math. Program. **153**(1), 201–212 (2015)
15. Gribling, S., de Laat, D., Laurent, M.: Matrices with high completely positive semidefinite rank. Linear Algebra Appl. **513**, 122–148 (2017)
16. Grippo, L., Sciandrone, M.: On the convergence of the block nonlinear gauss-seidel method under convex constraints. Oper. Res. Lett. **26**(3), 127–136 (2000)
17. Ho, N.D.: Nonnegative matrix factorization algorithms and applications. Ph.D. thesis, Univertsité catholique de Louvain (2008)
18. Hsieh, C.J., Dhillon, I.: Fast coordinate descent methods with variable selection for non-negative matrix factorization. In: Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1064–1072. ACM (2011)

19. Kaibel, V.: Extended formulations in combinatorial optimization. Optima **85**, 2–7 (2011)
20. Kuang, D., Yun, S., Park, H.: SymNMF: nonnegative low-rank approximation of a similarity matrix for graph clustering. J. Glob. Optim. **62**(3), 545–574 (2015)
21. Kubjas, K., Robeva, E., Robinson, R.: Positive semidefinite rank and nested spectrahedra. Linear Multilinear Algebra. 1–23 (2017)
22. Lee, T., Theis, D.O.: Support-based lower bounds for the positive semidefinite rank of a nonnegative matrix. (2012). arXiv preprint arXiv:1203.3961
23. Löfberg, J.: Yalmip: A toolbox for modeling and optimization in matlab. In: IEEE International Symposium on Computer Aided Control Systems Design, 2004, pp. 284–289. IEEE (2004)
24. Nesterov, Y.: A method of solving a convex programming problem with convergence rate 0(1/k2). Sov. Math. Dokl. **27**, 372–376 (1983)
25. Prakash, A., Sikora, J., Varvitsiotis, A., Wei, Z.: Completely positive semidefinite rank. Math. Program. 1–35 (2016)
26. Shitov, Y.: The complexity of positive semidefinite matrix factorization. SIAM J. Optim. **27**(3), 1898–1909 (2017)
27. Sturm, J.F.: Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones. Optim. Methods Softw. **11**(1–4), 625–653 (1999)
28. Vandaele, A., Gillis, N., Glineur, F.: On the linear extension complexity of regular n-gons. Linear Algebra Appl. **521**, 217–239 (2017)
29. Vandaele, A., Gillis, N., Glineur, F., Tuyttens, D.: Heuristics for exact nonnegative matrix factorization. J. Glob. Optim. **65**(2), 369–400 (2016)
30. Vandaele, A., Gillis, N., Lei, Q., Zhong, K., Dhillon, I.: Efficient and non-convex coordinate descent for symmetric nonnegative matrix factorization. IEEE Trans. Signal Process. **64**(21), 5571–5584 (2016)
31. Wright, S.: Coordinate descent algorithms. Math. Program. **151**(1), 3–34 (2015)
32. Yannakakis, M.: Expressing combinatorial optimization problems by linear programs. J. Comput. Syst. Sci. **43**(3), 441–466 (1991)