HAL
open science

Positive semidefinite matrix factorization: A link to phase retrieval and a block gradient algorithm

Dana Lahat, Cédric Févotte

To cite this version:

Dana Lahat, Cédric Févotte. Positive semidefinite matrix factorization: A link to phase retrieval and a block gradient algorithm. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2020, Barcelona (virtual), Spain. 10.1109/ICASSP40776.2020.9053938 . hal-02875241

HAL Id: hal-02875241

https://hal.science/hal-02875241v1

Submitted on 19 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

POSITIVE SEMIDEFINITE MATRIX FACTORIZATION: A LINK TO PHASE RETRIEVAL AND A BLOCK GRADIENT ALGORITHM

*Dana Lahat, Cédric Févotte**

IRIT, Université de Toulouse, CNRS, Toulouse, France

ABSTRACT

This paper deals with positive semidefinite matrix factorization (PSDMF). PSDMF writes each entry of a nonnegative matrix as the inner product of two symmetric positive semidefinite matrices. PSDMF generalizes nonnegative matrix factorization. Exact PSDMF has found applications in combinatorial optimization, quantum communication complexity, and quantum information theory, among others. In this paper, we show, for the first time, a link between PSDMF and the problem of matrix recovery from phaseless measurements, which includes phase retrieval. We demonstrate the usefulness of this observation by proposing a new type of local optimization scheme for PSDMF, which is based on a generalization of the Wirtinger flow method for phase retrieval. Numerical experiments show that our algorithm can performs as well as state-of-the-art algorithms, in certain setups. We suggest that this link between the two types of problems, which have until now been addressed separately, opens the door to new applications, algorithms, and insights.

***Index Terms*—** Positive semidefinite factorization, nonnegative factorizations, phase retrieval, rank minimization, semidefinite programming

1. INTRODUCTION

This paper deals with positive semidefinite matrix factorization (PSDMF) [1, 2]. PSDMF writes the (i, j)th entry x_{ij} of a nonnegative matrix $\mathbf{X} \in \mathbb{R}^{I \times J}$ as an inner product of two $K \times K$ symmetric positive semidefinite (psd) matrices $\mathbf{A}^{(i)}$ and $\mathbf{B}^{(j)}$, indexed by $i = [I]$, $j = [J]$:

$$x_{ij} \cong \langle \mathbf{A}^{(i)}, \mathbf{B}^{(j)} \rangle \triangleq \mathrm{tr}\{\mathbf{A}^{(i)}\mathbf{B}^{(j)}\} \tag{1}$$

where $[I] \triangleq 1, \ldots, I$, $\mathrm{tr}\{\cdot\}$ denotes the trace of a matrix, and $\cong$ means equal or approximately equal, depending on the context. The minimal number K such that a nonnegative matrix $\mathbf{X}$ admits an exact PSDMF is called the *psd rank* of $\mathbf{X}$ [1]. Each psd matrix $\mathbf{A}^{(i)}$ and $\mathbf{B}^{(j)}$ may have a different rank, denoted $R_A^{(i)}$ and $R_B^{(j)}$, respectively. We shall sometimes refer to the psd rank K as the "outer rank" and to $R_A^{(i)}$ and $R_B^{(j)}$ as "inner ranks" [3].

When $\mathbf{A}^{(i)}, \mathbf{B}^{(j)}$ are diagonal $\forall i, j$, PSDMF boils down to nonnegative matrix factorization (NMF) (e.g., [4]). Hence, NMF is a special case of PSDMF. Indeed, the original motivation for PSDMF [1] was generalizing a fundamental result [5] on the connection between nonnegative factorizations of the slack matrix of a polytope and of its polyhedral lifts. As demonstrated by [1], the existence of an exact PSDMF to a matrix associated with a convex set (i.e., its *slack matrix*) implies that this convex set can be represented as the image under a linear map of an affine slice of a given closed convex cone. When the psd rank K of this factorization is strictly smaller than the nonnegative rank of this slack matrix, this representation of the convex set is called a "*lift*". Due to the fundamental role of Yannakakis' result [5] in combinatorial optimization, its generalization attracts a significant amount of attention. Indeed, since it was first introduced, PSDMF has found applications in numerous fields, including semidefinite representations of polyhedra [1], semidefinite reformulations of linear programming [2], quantum communication complexity [2], combinatorial optimization [6, 2], quantum information theory [7], and more. A large amount of work has been dedicated to understanding the psd rank and its relation to other concepts of rank such as the ordinary matrix rank and the nonnegative rank (see, e.g., [8, 1], and follow-up work).

Until now, the analysis and applications of PSDMF addressed only the case of exact decompositions. Due to the wide range of applications of the nonnegative rank and NMF, several authors (e.g., [3]) have raised the question if PSDMF can be useful for data analysis, or any other kinds of applications, beyond those that have been suggested so far. In this paper, we show, for the first time (to the best of our knowledge), a link between PSDMF and the problem of matrix recovery from phaseless measurements (e.g., [9, 10, 11, 12, 13]), which includes phase retrieval (e.g., [14]). We explain this link in Section 2. We point out that quantum-related problems such as quantum tomography, and the combinatorial nature of phase retrieval, have already been discussed in the literature (e.g., [15]). However, to the best of our knowledge, they have not been associated with PSDMF.

In order to show the usefulness of this link, we develop, in Section 3, a new type of local optimization scheme for PSDMF, which is based on a generalization [12, 13] of Wirtinger flow [14]. The algorithm that we propose is based on an alternating block gradient descent. Our proposed algorithm is different from existing algorithms in the literature [3], which consist of the fast projected gradient method (FPGM) and coordinate descent (CD) schemes. Preliminary numerical experiments in Section 4 indicate that the performance of our algorithm is comparable to the state-of-the-art [3]. Although phase retrieval is associated with complex-valued data, we restrict our algorithm to the real-valued case, as in [3]. Nevertheless, based on the link that we find with phase retrieval, generalizing our method to the complex case is straightforward. While our algorithm can certainly be improved, our message is that PSDMF — and the applications in which it is currently being used— has strong links with phase retrieval (and its numerous generalizations), and that these links should be further explored.

*This work has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No. 681839 (project FACTORY).

2. ALGORITHMIC STATE-OF-THE-ART AND A LINK TO PHASE RETRIEVAL

In order to find an (approximate) PSDMF of a given nonnegative matrix, [3] suggested to minimize the cost function

$$f = \sum_{i=1}^{I}\sum_{j=1}^{J}(x_{ij} - \langle \mathbf{A}^{(i)}, \mathbf{B}^{(j)}\rangle)^2 \tag{2}$$

with respect to (w.r.t.) the variables $\mathbf{A}^{(i)}, \mathbf{B}^{(j)}$ $\forall i, j$. Vandaele et al. [3] address the optimization of (2) in an alternating scheme, in which one set of parameters, without loss of generality (w.l.o.g.) those indexed by $[I]$, is fixed, while the other set is optimized.

The FPGM algorithm proposed by [3] is a gradient-based approach in which the gradient is taken, w.l.o.g., w.r.t. the K^2J-dimensional variable consisting of all the entries of $\{\mathbf{B}^{(j)}\}_{j=1}^{J}$, given $\{\mathbf{A}^{(i)}\}_{i=1}^{K}$. The gradient step is followed by projecting the outcome, for each j, on the cone of $K \times K$ symmetric psd matrices, $\mathcal{S}_+^K$. Each alternation is a convex problem, consisting of a series of successive refinements via a gradient descent scheme. The number of such refinement steps is set in [3] to $\max(1, K\Delta)$, where $\Delta > 0$ is a parameter that can be adjusted by the user. We refer the reader to [3] for further details about FPGM. We now make the new observation that FPGM optimizes simultaneously, in each alternation, the convex problem

$$\min_{\mathbf{B}^{(j)} \in \mathcal{S}_+^K} \|\mathbf{x}_j - \mathcal{A}(\mathbf{B}^{(j)})\|_2^2 \quad \text{for all } j\,, \tag{3}$$

where $\|\cdot\|_2$ is the Euclidean norm, $\mathbf{x}_j \in \mathbb{R}^{I\times 1}$ is the jth column vector of $\mathbf{X}$, and where $\mathcal{A} : \mathbb{R}^{K\times K} \longrightarrow \mathbb{R}^{I\times 1}$ is a given affine transformation that maps matrices to vectors (e.g., [13, Sec. 1]). This notation implies that the jth column of $\mathbf{X}$ in (1) can be written as $\mathcal{A}(\mathbf{B}^{(j)}) \cong \mathbf{x}_j$. We point out that (3) is reminiscent of the matrix recovery problem

$$\min_{\mathbf{M}\in\mathbb{R}^{K\times K}} \text{rank}(\mathbf{M}) \quad \text{s.t.} \quad \mathcal{A}(\mathbf{M}) = \mathbf{x}_j\,. \tag{4}$$

This problem arises in a broad range of applications, and forms a challenge in optimization due to its nonconvexity. In particular, the phase retrieval problem can be recast as (4) if we add the constraint that $\mathbf{M}$ be psd (and if $\mathbf{M}$ is complex-valued and has rank-1) [16]. This formulation of phase retrieval *lifts* up the task of recovering a vector (or a matrix) from quadratic constraints into low-rank matrix recovery from affine constraints via semidefinite programming; hence, the phase retrieval framework based on this principle was termed PhaseLift [16]. We thus make the observation that each alternation in FPGM can be regarded as a lifted approach, but without the rank minimization constraint. Vandaele et al. [3] mention that a drawback of FPGM is that it does not allow easily to adjust the rank of the psd matrices, while there is interest in obtaining low-rank terms, which usually arise in applications. Therefore, they suggest a factor-based framework, that we shall describe later on in this section. However, we point out that it is likely that this goal can be achieved also using algorithms already existing in the literature (e.g., [16], or their generalizations and variants) by sequentially optimizing a problem of the form (4) for each psd matrix.

In PSDMF applications, the psd matrices are often of low rank, i.e., $R_A^{(i)}, R_B^{(j)} < K$ for some i and/or j (e.g., [8]). Hence, another approach suggested by [3] is to write the psd matrices as

$$\mathbf{A}^{(i)} \triangleq \mathbf{U}^{(i)}\mathbf{U}^{(i)\top} \quad \text{and} \quad \mathbf{B}^{(j)} \triangleq \mathbf{V}^{(j)}\mathbf{V}^{(j)\top}\,, \tag{5}$$

where $\mathbf{U}^{(i)} \in \mathbb{R}^{K\times R_A^{(i)}}$ and $\mathbf{V}^{(i)} \in \mathbb{R}^{K\times R_B^{(i)}}$ are factor matrices (factors, for short), and to optimize the cost function w.r.t. these factors. Note that this approach requires the inner ranks to be known, whereas this is not the case in FPGM. With this notation, the cost function can be written as

$$f = \sum_{i=1}^{I}\sum_{j=1}^{J}\left(x_{ij} - \text{tr}\{\mathbf{V}^{(j)\top}\mathbf{A}^{(i)}\mathbf{V}^{(j)}\}\right)^2 \tag{6a}$$

$$= \sum_{j=1}^{J}\|\mathbf{x}_j - \mathcal{A}(\mathbf{V}^{(j)}\mathbf{V}^{(j)\top})\|_2^2\,. \tag{6b}$$

This change of variables is common is semidefinite programming (e.g., [17]). The second algorithm proposed by [3] is an alternating scheme based on CD. It optimizes (6) w.r.t. the entries of $\{\mathbf{V}^{(j)}\}_{j=1}^{J}$ given $\{\mathbf{U}^{(i)}\}_{i=1}^{I}$, and vice versa in the other alternation. We now make the following new observation. Consider the optimization of (6b) for a specific j (or when $J = 1$), when the factors associated with i are fixed. When all factors have inner rank equal to one, the complex-valued counterpart of this optimization problem, which is a special case of (6b), is nothing but the (generalized) phase retrieval problem [14], which consists in solving quadratic equations of the form $x_{ij} \cong |\langle \mathbf{u}^{(i)}, \mathbf{v}^{(j)}\rangle|^2$, $i = [I]$, where $\mathbf{u}^{(i)}, \mathbf{v}^{(j)}$ are complex valued vectors of length K, and $\mathbf{u}^{(i)}$ are known. It is thus not surprising that one can find a CD algorithm [18] for phase retrieval that is very close to that of [3], with differences arising from the particularities of phase retrieval.

3. BLOCK GRADIENT DESCENT ALGORITHM

Based on our observations in Section 2, we develop a new real-valued algorithm for PSDMF, which is based on the principles set by [14] for phase retrieval, and that were later extended to the general low-rank case by [13, 12]. Without loss of generality (W.l.o.g.), optimizing (6) w.r.t. $\mathbf{V}^{(j)}$ can be written as:

$$\min_{\mathbf{V}^{(j)}\in\mathbb{R}^{K\times R_B^{(j)}}} f(\mathbf{V}^{(j)}) \triangleq \|\mathbf{x}_j - \mathcal{A}(\mathbf{V}^{(j)}\mathbf{V}^{(j)\top})\|_2^2\,. \tag{7}$$

Equation (7) is the low-rank, factor-based, counterpart of (3). Tu et al. [13] and Zheng and Lafferty [12] proposed to optimize the nonconvex objective in (7) via gradient descent, where the gradient of f w.r.t. $\mathbf{V}^{(j)}$, denoted $\nabla f(\mathbf{V}^{(j)}) = \frac{\partial f}{\partial \mathbf{V}^{(j)}}$, is given by (e.g., [12, 13])

$$\nabla f(\mathbf{V}^{(j)}) = \sum_{i=1}^{I} 4(-x_{ij} + \text{tr}\{\mathbf{A}^{(i)}\mathbf{V}^{(j)}\mathbf{V}^{(j)\top}\})\mathbf{A}^{(i)}\mathbf{V}^{(j)}\,. \tag{8}$$

We thus propose a PSDMF scheme based on successive application of the gradient descent method in [13, 12] for optimizing (7) for each j using the gradient in (8), and then alternating to $\mathbf{U}^{(i)}$ for all i. We refer to it as alternating block gradient (ABG). Algorithm 1 describes one alternation in our block gradient descent scheme for updating the factors indexed by $j = [J]$ given initial values $\mathbf{V}_0^{(1)}, \ldots, \mathbf{V}_0^{(J)}$ and for fixed $\{\mathbf{U}^{(i)}\}_{i=1}^{I}$. The number of gradient refinements (or "passes") over each factor $\mathbf{V}^{(j)}$ is denoted as D. We consider two options: the first, denoted `d->j`, corresponds to Lines 5 and 6 in Algorithm 1; it iterates on $d = [D]$ as the outer loop and on $j = [J]$ in the inner one. The other option is denoted `j->d`.

Algorithm 1 Gradient descent scheme for optimizing one set of factors

Input: $\mathbf{X} \in \mathbb{R}_+^{I \times J}, \mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(I)}, \mathbf{V}_0^{(1)}, \ldots, \mathbf{V}_0^{(J)}$.
Output: $\mathbf{V}^{(1)}, \ldots, \mathbf{V}^{(J)}$.
1: **for** $i = 1 : I$ **do**
2: $\quad \mathbf{A}^{(i)} \leftarrow \mathbf{U}^{(i)}\mathbf{U}^{(i)\top}$
3: **end for**
4: $\tau \leftarrow$ Choose initial step size (e.g., Algorithm 2)
5: **for** $d = 1 : D$ **do** ▷ the `d->j` option
6: $\quad$ **for** $j = 1 : J$ **do**
7: $\quad \nabla f(\mathbf{V}_d^{(j)}) \leftarrow \sum_{i=1}^{I} 4(-x_{ij} + \mathrm{tr}\{\mathbf{A}^{(i)}\mathbf{V}_d^{(j)}\mathbf{V}_d^{(j)\top}\})\mathbf{A}^{(i)}\mathbf{V}_d^{(j)}$
8: $\quad\quad \mu_{d+1}^{(j)} \leftarrow$ Choose step size (e.g., τ, or Algorithm 3)
9: $\quad\quad \mathbf{V}_{d+1}^{(j)} \leftarrow \mathbf{V}_d^{(j)} - \mu_{d+1}^{(j)} \nabla f(\mathbf{V}_d^{(j)})$
10: $\quad$ **end for**
11: **end for**

Algorithm 2 Choosing initial step size τ

1: $j' \leftarrow \mathcal{U}\{1, J\}$
2: $\mathbf{x}_1 \leftarrow \mathbf{V}_d^{(j')}$
3: $\mathbf{x}_2 \leftarrow \mathbf{V}_d^{(j')} + \mathbf{E}$ $\quad$ where $e_{kr} \sim \mathcal{N}(0, \sigma^2)$
4: $\widehat{L} \leftarrow \|\nabla f(\mathbf{x}_2) - \nabla f(\mathbf{x}_1)\| / \|\mathbf{x}_2 - \mathbf{x}_1\|$
5: $\widehat{L} = \max(\widehat{L}, 10^{-30})$
6: $\tau \leftarrow C_L / \widehat{L}$

Algorithm 3 Backtracking line search

1: $t \leftarrow \tau$, $\quad$ choose α, β
2: **while** $f(\mathbf{V}_d^{(j)} - t\nabla f(\mathbf{V}_d^{(j)})) > f(\mathbf{V}_d^{(j)}) - \alpha t \|\nabla f(\mathbf{V}_d^{(j)})\|_F^2$ **do**
3: $\quad t = \beta t$
4: **end while**

Choosing the optimal step size in the gradient scheme to optimize (7) is critical to the good convergence of the algorithm. Due to the non-convexity of (7), the Lipschitz constant of the gradient, denoted $L(\nabla f)$, is not easy to compute (e.g., [14, 13, 12, 18]). We thus adopt the approach in PhasePack [19] for estimating the initial step size numerically, using a local estimation of $L(\nabla f)$, and adjust it to our PSDMF framework. Using the property

$$\|\nabla f(\mathbf{x}_2) - \nabla f(\mathbf{x}_1)\|_2 / \|\mathbf{x}_2 - \mathbf{x}_1\|_2 \leq L(\nabla f)\,, \tag{9}$$

$L(\nabla f)$ can be estimated numerically by generating two random vectors $\mathbf{x}_1, \mathbf{x}_2$. We adjusted this approach to PSDMF, as described in Algorithm 2: (i) in each alternation, we estimate a single value of $L(\nabla f)$. This is based on an assumption that all factors optimized in the same alternation satisfy the same model. Of course, if this assumption is not valid, $L(\nabla f)$ may have to be computed for each factor individually; (ii) instead of choosing $\mathbf{x}_1, \mathbf{x}_2$ randomly, as in [19], we set $\mathbf{x}_1$ to be one of the factors that we actually optimize. We choose the index of this factor from the discrete uniform distribution on the interval $[1, J]$, $\mathcal{U}\{1, J\}$, at each alternation (Line 1 in Algorithm 2). In our numerical experiments, this scheme turned out to yield a balanced average number of backtracking steps per alternation, as opposed to what we observed when fixing the index j' to a constant number; (iii) in order to increase the chance of having a local estimation of $L(\nabla f)$, $\mathbf{x}_2$ is chosen as a random point that is "not too far" from $\mathbf{x}_1$, by adding a small perturbation to $\mathbf{x}_1$. This perturbation is a random matrix $\mathbf{E}$, of the same size of the factor chosen to represent $\mathbf{x}_1$, and whose entries are independent and identically distributed (i.i.d.) Gaussian with a small variance σ^2 (Line 3 in Algorithm 2), and (iv) we set the initial step size to $C_L/\widehat{L}$, where $C_L > 0$ is a constant chosen by the user. We adopt the approach of [19] and use backtracking line search to guarantee stability. Our backtracking line search scheme is described in Algorithm 3. This scheme turned out to yield satisfying results in our preliminary numerical experiments, which we present in Section 4.

Convergence analysis for our proposed ABG framework is beyond the scope of this paper. Vandaele et al. [3] do not provide theoretical convergence results for their algorithms, either. However, based on our observation that all the algorithms proposed so far for PSDMF are related to methods that have already been studied in the context of phase retrieval and its generalizations (e.g., [13, 12, 18, 14]), it is likely that at least partial understanding of the convergence of these PSDMF algorithms can be based on existing results in the literature.

The computational complexity of one gradient iteration on one factor j in Algorithm 1 is essentially the same as one block iteration of the gradient descent algorithm in [12], $\mathcal{O}(K^2R) + \mathcal{O}(IK^2)$ (for simplicity, we take $R_A^{(i)} = R = R_B^{(j)}\ \forall i, j$). Taking into account J factors and D refinement steps, we obtain $\mathcal{O}(DJK^2R) + \mathcal{O}(DJIK^2)$. Our step size estimation requires another computation of the gradient, but only once for all the DJ "for" iterations. We found, numerically, that this estimation of the step size reduces the number of backtracking steps to an average of one: this means that we only have to do about twice the number of operations, on average, due to backtracking. Hence, the backtracking line search procedure increase the computational load only by factor of two. Taking into account two alternations per overall iteration, and assuming that $I = J$, the overall computational complexity per iteration (which consists of two alternations of the form Algorithm 1) remains at the order of $\mathcal{O}(DIK^2R) + \mathcal{O}(DI^2K^2)$. For comparison, the computational complexity of FPGM is $\mathcal{O}(IJK^2) + \mathcal{O}(IK^4) + \mathcal{O}(\min(K^4, I^2)) + \mathcal{O}(\Delta JK^5)$ where $\Delta = 5$ was chosen as the default value in the numerical experiments in [3]. The dominant terms in the computational complexity of CD are $\mathcal{O}(IK^4R) + \mathcal{O}(IK^2 \max(J, K^2))$. We omit the details of the calculations for lack of space. Note that the complexity evaluations in [3] assume $R = K$, whereas here we do not make this assumption.

4. NUMERICAL EXPERIMENTS

We implemented our ABG algorithm in Matlab. We test six variants of our ABG method: for each value of $D = 1, 5, 25$, we test with outer loop on j and inner loop on d, and vice versa (Lines 5 and 6 in Algorithm 1). We compare ABG with (i) FPGM with $\Delta = 5$ (i.e., $K\Delta$ refinements per alternation [3]), and (ii) two variants of CD, cyclic and Gauss-Southwell (GS) ("greedy" CD). The parameter controlling the number of "greedy" iterations is set to 0.5. The same values of parameters were used by [3]. Vandaele et al. [3] implemented their CD algorithm in c. They explain that the large number of "for" loops in CD causes it to run very slowly on Matlab. Vandaele et al. [3] thus compare their CD algorithm coded in c with FPGM coded in Matlab. Here, in order to compare algorithms on the same platform, we implemented CD in Matlab. We use the c version of CD and the Matlab code for FPGM from *https://sites.google.com/site/exactnmf/psd-factorization*. The only difference between the c and Matlab implementations is the CPU run time. They are identical in terms of error evolution and number of iterations. Hence, only comparisons w.r.t. CPU run time in Figs. 1 and 2 distinguish between c and Matlab versions of CD.

We set $\alpha = 0.1$, $\beta = 0.2$ for backtracking line search (Algorithm 3) and $C_L = 1$, $\sigma^2 = 0.05$, in the estimation of the initial step size (Algorithm 2). Each numerical experiment consists of 30

Monte Carlo (MC) trials: the input matrix does not change, and the variability is only due to the different random initialization of the input factors. We initialize the factors with numbers drawn from the standard normal distribution $\mathcal{N}(0,1)$, as in [3]. The stopping criterion is $\frac{f^{t+1}-f^t}{f^1} < \texttt{TolFun}$, where t is the iteration index.

Our first experiment is low-rank approximation of a 13×13 matrix ($I = 13 = J$), generated from factors with ranks $K = 3$ and $R_A^{(i)} = 2 = R_B^{(j)}\ \forall i, j$, whose entries are drawn independently from the standard normal distribution. The factorization uses the true K and $R_B^{(j)}\ \forall j$ but a smaller value of $R_A^{(i)} = 1\ \forall i$. In this experiment, we do not compare with FPGM because our factors have inner rank less than K; as explained in Section 2, FPGM can only represent factors of size $K \times K$. Note that the input matrix provides 169 observations (or measurements) and consists of 121 degrees of freedom (d.o.f.), whereas the estimating model has only 95 d.o.f.. Here, $\texttt{TolFun}= 10^{-10}$. Figure 1 illustrates our results.

In the second experiment, our input is S12, the slack matrix of the regular 12-gon (twelve-sided polygon). This is a nonnegative 12×12 matrix with zero diagonal and subdiagonal. Slack matrices [5] represent geometric properties of polyhedra. Their psd rank has numerous applications, such as those mentioned briefly in Section 1. Vandaele et al. [3] conjecture that the smallest K for which S12 admits an exact PSDMF, i.e., its psd rank, is $K = 5$. For the sake of algorithmic comparison, we approximate S12 with factors of ranks $K = 4 = R_A^{(i)} = R_B^{(j)}\ \forall i, j$. Hence, as in the previous example, we do not expect to obtain exact factorization. Here, the number of observations is 144, which is much less than the number of d.o.f. in the factorization model, 224. The reason we cannot obtain an exact factorization is not only the special structure of PSDMF, but also due to the fact that S12 has zero diagonal and subdiagonal: these zeros impose further constraints on the factors. Here, $\texttt{TolFun}= 10^{-9}$. Figure 2 illustrates our results.

The first subfigure (clockwise from top left) in Figs. 1 and 2 demonstrates the evolution of the error $\frac{\|\mathbf{X}-\mathbf{X}_t\|_F}{\|\mathbf{X}_t\|_F}$, where $\mathbf{X}_t$ is the estimate of $\mathbf{X}$ at the tth iteration, in one MC trial, versus CPU time. The second subfigure illustrates, in box plots, the CPU time needed to reach the stopping criterion for all MC trials. The 3rd and 4th subfigures describe, in box plots, the total number of iterations and the final error achieved, for all MC trials. We use log or log-log scale in all figures. These subfigures illustrate a trend that we observed in all our numerical experiments: the c versions of CD are always the fastest; however, they are not always the ones to achieve the smallest error. On the other hand, our Matlab implementation of CD is the slowest to reach the stopping criterion (it is probably possible to encode it a bit faster in Matlab, but we doubt that the improvement will change the overall trend). The convergence speed of ABG is somewhere in between, with differences arising from the specific setup. Figure 2 indicates that FPGM seems to be comparable to the more efficient variants of ABG. The fact that ABG works on matrices ("blocks") probably explains why it outperforms the Matlab versions of CD.

We observe that ABG tends to converge faster with a smaller number of refinements D, despite the increase in the number of iterations. In the low-rank scenario of Fig. 1, the final error achieved by all algorithms is distributed more or less the same, whereas in Fig. 2 we see more distinct differences in final error among algorithms and variants of ABG, in favor of larger D. Perhaps surprisingly, we do not observe any clear trend that can tell us which of the variants, `j->d` and `d->j`, is better.

Although these preliminary numerical experiments are limited in scope, they indicate that the proposed ABG framework does not fall, in terms of computational complexity and ability to converge, at least to a local optimum, behind the state-of-the-art [3]. Follow-up work will deal with convergence guarantees and understanding of the various variants of ABG.

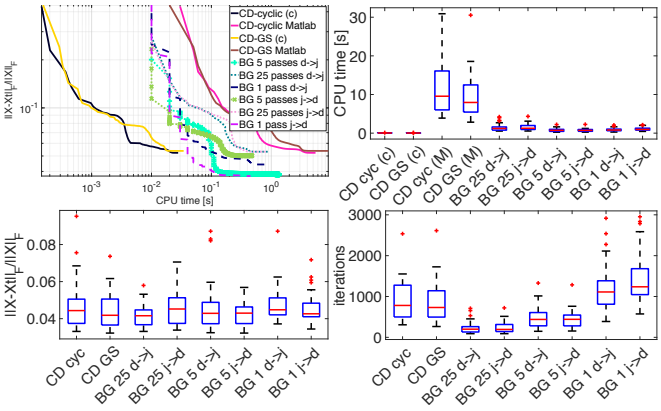

Fig. 1: PSDMF of a 13×13 matrix generated from random factors of ranks $K = 3$, $R_A^{(i)} = 2 = R_B^{(j)}\ \forall i, j$, factorized with $R_A^{(i)} = 1\ \forall i$. Clockwise from top left: (i) evolution of the error versus CPU time, for one MC trial; (ii)–(iv) CPU time, error, and number of iterations, when stopping criterion is achieved, for all MC trials.

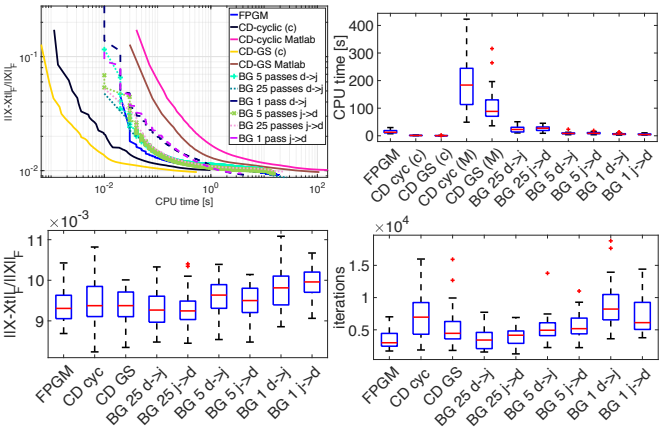

Fig. 2: PSDMF of the slack matrix of the 12-gon, S12, factorized with $K = 4 = R_A^{(i)} = R_B^{(j)}\ \forall i, j$. Clockwise from top left: (i) evolution of the error versus CPU time, for one MC trial; (ii)–(iv) CPU time, error, and number of iterations, when stopping criterion is achieved, for all MC trials.

In a broader perspective, our observations and numerical results clarify that results from phase retrieval and related problems can be used to develop new algorithms for PSDMF. Conversely, results already developed for PSDMF may be applied to phase retrieval, low-rank matrix recovery from magnitude-only (phaseless) measurements, and related problems. Having said that, we do not say that PSDMF solves phase retrieval (or any of its low-rank generalizations): unless sufficient constraints are imposed, PSDMF is a highly non-unique problem. A more substantial link between PSDMF and phase retrieval may be established by considering recent elaborate models, as in [10, 20]. We leave further discussion of this matter for future work.

Acknowledgment The authors would like to thank Vincent Y. F. Tan for bringing the paper of Vandaele et al. to our attention and for his valuable feedback on our manuscript.

5. REFERENCES

[1] J. Gouveia, P. A. Parrilo, and R. R. Thomas, "Lifts of convex sets and cone factorizations," *Mathematics of Operations Research*, vol. 38, no. 2, pp. 248–264, May 2013.

[2] S. Fiorini, S. Massar, S. Pokutta, H. R. Tiwary, and R. De Wolf, "Linear vs. semidefinite extended formulations: exponential separation and strong lower bounds," in *Proc. 44th Annu. ACM Symp. Theory of Computing*. ACM, May 2012, pp. 95–106.

[3] A. Vandaele, F. Glineur, and N. Gillis, "Algorithms for positive semidefinite factorization," *Computational Optimization and Applications*, vol. 71, no. 1, pp. 193–219, Sep 2018.

[4] P. Paatero and U. Tapper, "Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values," *Environmetrics*, vol. 5, no. 2, pp. 111–126, Jun. 1994.

[5] M. Yannakakis, "Expressing combinatorial optimization problems by linear programs," *Journal of Computer and System Sciences*, vol. 43, no. 3, pp. 441–466, Dec. 1991.

[6] V. Kaibel, "Extended formulations in combinatorial optimization," *arXiv:1104.1023 [math.CO]*, Apr. 2011.

[7] R. Jain, Y. Shi, Z. Wei, and S. Zhang, "Efficient protocols for generating bipartite classical distributions and quantum states," *IEEE Trans. Inf. Theory*, vol. 59, no. 8, pp. 5171–5178, Aug 2013.

[8] H. Fawzi, J. Gouveia, P. A. Parrilo, R. Z. Robinson, and R. R. Thomas, "Positive semidefinite rank," *Mathematical Programming*, vol. 153, no. 1, pp. 133–177, Oct 2015.

[9] D. Yang, *Structured Low-Rank Matrix Recovery via Optimization Methods*, Ph.D. thesis, Colorado School of Mines, 2018.

[10] S. Nayer, P. Narayanamurthy, and N. Vaswani, "Phaseless PCA: Low-rank matrix recovery from column-wise phaseless measurements," in *Proc. ICML*, K. Chaudhuri and R. Salakhutdinov, Eds., Long Beach, California, USA, 09–15 Jun 2019, vol. 97 of *Proceedings of Machine Learning Research*, pp. 4762–4770, PMLR.

[11] G. Wang, G. B. Giannakis, and Y. C. Eldar, "Solving systems of random quadratic equations via truncated amplitude flow," *IEEE Trans. Inf. Theory*, vol. 64, no. 2, pp. 773–794, Feb 2018.

[12] Q. Zheng and J. Lafferty, "A convergent gradient descent algorithm for rank minimization and semidefinite programming from random linear measurements," in *Proc. NeurIPS*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., pp. 109–117. Curran Associates, Inc., 2015.

[13] S. Tu, R. Boczar, M. Simchowitz, M. Soltanolkotabi, and B. Recht, "Low-rank solutions of linear matrix equations via Procrustes flow," in *Proc. ICML*, M. F. Balcan and K. Q. Weinberger, Eds., New York, New York, USA, 20–22 Jun 2016, vol. 48 of *Proceedings of Machine Learning Research*, pp. 964–973, PMLR.

[14] E. J. Candès, X. Li, and M. Soltanolkotabi, "Phase retrieval via Wirtinger flow: Theory and algorithms," *IEEE Trans. Inf. Theory*, vol. 61, no. 4, pp. 1985–2007, April 2015.

[15] E. Candès, Y. Eldar, T. Strohmer, and V. Voroninski, "Phase retrieval via matrix completion," *SIAM Rev.*, vol. 57, no. 2, pp. 225–251, 2015.

[16] E. J. Candès, T. Strohmer, and V. Voroninski, "Phaselift: Exact and stable signal recovery from magnitude measurements via convex programming," *Communications on Pure and Applied Mathematics*, vol. 66, no. 8, pp. 1241–1274, 2013.

[17] S. Burer and R. D. C. Monteiro, "A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization," *Mathematical Programming*, vol. 95, no. 2, pp. 329–357, Feb 2003.

[18] W.-J. Zeng and H. C. So, "Coordinate descent algorithms for phase retrieval," *Signal Processing*, vol. 169, pp. 107418, 4 2020.

[19] R. Chandra, Z. Zhong, J. Hontz, V. McCulloch, C. Studer, and T. Goldstein, "PhasePack: A phase retrieval library," in *Proc. ACSSC*, Pacific Grove, CA, USA, Oct 2017, pp. 1617–1621.

[20] T. Bendory, D. Edidin, and Y. C. Eldar, "Blind phaseless short-time Fourier transform recovery," *IEEE Trans. Inf. Theory*, 2019, DOI: 10.1109/TIT.2019.2947056.