Efficient and Non-Convex Coordinate Descent for Symmetric Nonnegative Matrix Factorization

Arnaud Vandaele, Nicolas Gillis, Qi Lei, Kai Zhong, and Inderjit Dhillon, *Fellow, IEEE*

***Abstract*—Given a symmetric nonnegative matrix A, symmetric nonnegative matrix factorization (symNMF) is the problem of finding a nonnegative matrix H, usually with much fewer columns than A, such that $A \approx HH^T$. SymNMF can be used for data analysis and in particular for various clustering tasks. Unlike standard NMF, which is traditionally solved by a series of quadratic (convex) subproblems, we propose to solve SymNMF by directly solving the nonconvex problem, namely, minimize $\|A - HH^T\|^2$, which is a fourth-order nonconvex problem. In this paper, we propose simple and very efficient coordinate descent schemes, which solve a series of fourth-order univariate subproblems exactly. We also derive convergence guarantees for our methods and show that they perform favorably compared to recent state-of-the-art methods on synthetic and real-world datasets, especially on large and sparse input matrices.**

***Index Terms*—Symmetric nonnegative matrix factorization, coordinate descent, completely positive matrices.**

I. Introduction

NONNEGATIVE matrix factorization (NMF) has become a standard technique in data mining by providing low-rank decompositions of nonnegative matrices: given a nonnegative matrix $X \in \mathbb{R}^{m \times n}_+$ and an integer $r < \min(m, n)$, the problem is to find $W \in \mathbb{R}^{m \times r}_+$ and $H \in \mathbb{R}^{n \times r}_+$ such that $X \approx WH^T$. In many applications, the nonnegativity constraints lead to a sparse and part-based representation, and a better interpretability of the factors, e.g., when analyzing images or documents [1].

In this paper, we work on a special case of NMF where the input matrix is a symmetric matrix A. Usually, the matrix A will be a *similarity matrix* where the (i, j)th entry is a measure of the similarity between the ith and the jth data points. This is a rather general framework, and the user can decide how to generate the matrix A from his data set by selecting an appropriate metric to compare two data points. As opposed to NMF, we are interested in a symmetric approximation HH^T with the factor H being nonnegative–hence symNMF is an NMF variant with $W = H$. If the data points are grouped into clusters, each rank-one factor $H(:, j)H(:, j)^T$ will ideally correspond to a cluster present in the data set. In fact, symNMF has been used successfully in many different settings and was proved to compete with standard clustering techniques such as normalized cut, spectral clustering, k-means and spherical k-means; see [2]–[8] and the references therein.

SymNMF also has tight connections with completely positive matrices [9], [10], that is, matrices of the form $A = HH^T, H \geq 0$, which play an important role in combinatorial optimization [11]. Note that the smallest r such that such a factorization exists is called the cp-rank of A. The focus of this paper is to provide efficient methods to compute good symmetric and nonnegative low-rank approximations HH^T with $H \geq 0$ of a given nonnegative symmetric matrix A.

Let us describe our problem more formally. Given a n-by-n symmetric nonnegative matrix A and a factorization rank r, symNMF looks for an n-by-r nonnegative matrix H such that $A \approx HH^T$. The error between A and its approximation HH^T can be measured in different ways but we focus in this paper on the Frobenius norm:

$$\min_{H \geq 0} \; F(H) \equiv \frac{1}{4} \left\| A - HH^T \right\|_F^2, \qquad (1)$$

which is arguably the most widely used in practice. Applying standard non-linear optimization schemes to (1), one can only hope to obtain stationary points, since the objective function of (1) is highly non-convex, and the problem is NP-hard [12]. For example, two such methods to find approximate solutions to (1) were proposed in [7]:

1) The first method is a Newton-like algorithm which exploits some second-order information without the prohibitive cost of the full Newton method. Each iteration of the algorithm has a computational complexity of $O(n^3 r)$ operations.
2) The second algorithm is an adaptation of the alternating nonnegative least squares (ANLS) method for NMF [13], [14] where the term $||W - H||_F^2$ penalizing the difference between the two factors in NMF is added to the objective function. That same idea was used in [15] where the author developed two methods to solve this penalized problem but without any available implementation or comparison.

In this paper, we analyze coordinate descent (CD) schemes for (1). Our motivation is that the most efficient methods for NMF are CD methods; see [16]–[19] and the references therein. The reason behind the success of CD methods for NMF is

Manuscript received October 26, 2016; revised March 08, 2016, May 04, 2016, and June 21, 2016; accepted June 22, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Cedric Fevotte. Nicolas Gillis acknowledges the support by the F.R.S.-FNRS, through the incentive grant for scientific research no F.4501.16. This research was supported by NSF grant CCF-1564000. *(Corresponding author: Arnaud Vandaele.)*

A. Vandaele and N. Gillis are with the Department of Mathematics and Operational Research, University of Mons, 7000 Mons, Belgium (e-mail: arnaud.vandaele@umons.ac.be; nicolas.gillis@umons.ac.be).

Q. Lei and K. Zhong are with the Institute for Computational Engineering and Science, University of Texas at Austin, Austin, TX 78712-1757 USA (e-mail: leiqi@ices.utexas.edu; zhongkai@ices.utexas.edu).

I. Dhillon is with the Institute for Computational Engineering and Science, University of Texas at Austin, Austin, TX 78712-1757 USA, and also with the Department of Computer Science, University of Texas at Austin, Austin, TX 78712-1757 USA (e-mail: inderjit@cs.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2016.2591510

twofold: (i) the updates can be written in closed-form and are very cheap to compute, and (ii) the interaction between the variables is low because many variables are expected to be equal to zero at a stationary point [20].

The paper is organized as follows. In Section II, we focus on the rank-one problem and present the general framework to implement an exact CD method for symNMF. The main proposed algorithm is described in Section III. Section IV discusses initialization and convergence issues. Section V presents extensive numerical experiments on synthetic and real data sets, which shows that our CD methods perform competitively with recent state-of-the-art techniques for symNMF.

II. Exact Coordinate Descent Methods for SymNMF

Exact coordinate descent (CD) techniques are among the most intuitive methods to solve optimization problems. At each iteration, all variables are fixed but one, and that variable is updated to its optimal value. The update of one variable at a time is often computationally cheap and easy to implement. However little interest was given to these methods until recently when CD approaches were shown competitive for certain classes of problems; see [21] for a recent survey. In fact, more and more applications are using CD approaches, especially in machine learning when dealing with large-scale problems.

Let us derive the exact cyclic CD method for symNMF. The approximation HH^T of the input matrix A can be written as the sum of r rank-one symmetric matrices:

$$A \approx \sum_{k=1}^{r} H_{:,k} H_{:,k}^T, \tag{2}$$

where $H_{:,k}$ is the kth column of H. If we assume that all columns of H are known except for the jth, the problem comes down to approximate a residual symmetric matrix $R^{(j)}$ with a rank-one nonnegative symmetric matrix $H_{:,j}H_{:,j}^T$:

$$\min_{H_{:,j} \geq 0} \left\| R^{(j)} - H_{:,j} H_{:,j}^T \right\|_F^2, \tag{3}$$

where

$$R^{(j)} = A - \sum_{k=1, k \neq j}^{r} H_{:,k} H_{:,k}^T. \tag{4}$$

For this reason and to simplify the presentation, we only consider the rank-one subproblem in the following Section II-A, before presenting on the overall procedure in Section II-B.

A. Rank-One Symmetric NMF

Given a n-by-n symmetric matrix $P \in \mathbb{R}^{n \times n}$, let us consider the rank-one symNMF problem

$$\min_{h \geq 0} \; f(h) \equiv \frac{1}{4} \left\| P - hh^T \right\|_F^2, \tag{5}$$

where $h \in \mathbb{R}^n_+$. If all entries of P are nonnegative, the problem can be solved for example with the truncated singular value decomposition; this follows from the Perron-Frobenius and Eckart-Young theorems. In our case, the residuals $R^{(j)}$ will in general have negative entries–see (4)–which makes the problem NP-hard in general [22]. The optimality conditions for (5) are given by

$$h \geq 0, \nabla f(h) \geq 0, \text{ and } h_i \nabla f(h)_i = 0 \text{ for all } i, \tag{6}$$

where $\nabla f(h)_i$ the ith component of the gradient $\nabla f(h)$. For any $1 \leq i \leq n$, the exact CD method consists in alternatively updating the variables in a cyclic way:

$$\text{for } i = 1, 2, \ldots, n: \quad h_i \leftarrow h_i^+,$$

where h_i^+ is the optimal value of h_i in (5) when all other variables are fixed. Let us show how to compute h_i^+. We have:

$$\nabla f(h)_i = h_i^3 + \underbrace{\left(\sum_{l=1, l \neq i}^{n} h_l^2 - P_{ii} \right)}_{a_i} h_i - \underbrace{\sum_{l=1, l \neq i} h_l P_{li}}_{b_i}, \tag{7}$$

where

$$a_i = \sum_{l=1, l \neq i}^{n} h_l^2 - P_{ii} = \|h\|^2 - h_i^2 - P_{ii}, \text{ and} \tag{8}$$

$$b_i = - \sum_{l=1, l \neq i} h_l P_{li} = h_i P_{ii} - h^T P_{:,i}. \tag{9}$$

If all the variables but h_i are fixed, by the complementary slackness condition (6), the optimal solution h_i^+ will be either 0 or a solution of the equation $\nabla f(h)_i = 0$, that is, a root of $x^3 + a_i x + b_i$. Since the roots of a third-degree polynomial can be computed in closed form, it suffices to first compute these roots and then evaluate $f(h)$ at these roots in order to identify the optimal solution h_i^+. The algorithm based on Cardano's method (see for example [23]) is described as Algorithm 1 and runs in $O(1)$ time. Therefore, given that a_i and b_i are known, h_i^+ can be computed in $O(1)$ operations.

The only inputs of Algorithm 1 are the quantities (8) and (9). However, the variables in (5) are not independent. When h_i is updated to h_i^+, the partial derivative of the other variables, that is, the entries of $\nabla f(h)$, must be updated. For $l \in \{i+1, \ldots, n\}$, we update:

$$a_l \leftarrow a_l + (h_i^+)^2 - h_i^2 \quad \text{and} \quad b_l \leftarrow b_l + P_{li}(h_i^+ - h_i). \tag{10}$$

This means that updating one variable will cost $O(n)$ operations due to the necessary run over the coordinates of h for updating the gradient. (Note that we could also simply evaluate the ith entry of the gradient when updating h_i, which also requires $O(n)$ operations; see Section III.) Algorithm 2 describes one iteration of CD applied on problem (5). In other words, if one wants to find a stationary point of problem (5), Algorithm 2 should be called until convergence, and this would correspond to applying a cyclic coordinate descent method to (5). In lines 2-2, the quantities a_i's and b_i's are precomputed. Because of the product $h^T P_{:,i}$ needed for every b_i, it takes $O(n^2)$ time. Then, from line 2 to line 2, Algorithm 1 is called for every variable and is followed by the updates described by (10). Finally, Algorithm 2 has a computational cost of $O(n^2)$ operations. Note that we cannot expect a lower computational cost since computing the gradient (and in particular the product Ph) requires $O(n^2)$ operations.

Algorithm 1: $x = BestPolynomialRoot(a, b)$.

INPUT: $a \in \mathbb{R}, b \in \mathbb{R}$
OUTPUT: $\arg\min_x \frac{x^4}{4} + \frac{ax^2}{2} + bx$ such that $x \geq 0$.
$\Delta = 4a^3 + 27b^2$
$d = \frac{1}{2}\left(-b + \sqrt{\frac{\Delta}{27}}\right)$
if $\Delta \leq 0$ **then**
  $r = 2\sqrt[3]{|d|}$
  $\theta = \frac{phaseangle(d)}{3}$
  $z^* = 0, y^* = 0$
  for $k = 0 : 2$ **do**
    $z = r\cos\left(\theta + \frac{2k\pi}{3}\right)$
    if $z \geq 0$ **and** $\frac{z^4}{4} + a\frac{z^2}{2} + bz < y^*$ **then**
      $z^* = z$
      $y^* = \frac{z^4}{4} + a\frac{z^2}{2} + bz$
    end if
  end for
  $x = z^*$
else
  $z = \sqrt[3]{d} + \sqrt[3]{\frac{1}{2}\left(-b - \sqrt{\frac{\Delta}{27}}\right)}$
  if $z \geq 0$ **and** $\frac{z^4}{4} + a\frac{z^2}{2} + bz < 0$ **then**
    $x = z$
  else
    $x = 0$
  end if
end if

Algorithm 2: $h = rankoneCDSymNMF(P, h_0)$.

INPUT: $P \in \mathbb{R}^{n\times n}, h_0 \in \mathbb{R}^n$
OUTPUT: $h \in \mathbb{R}^n_+$
$h = h_0$
for $i = 1 : n$ **do**
  $a_i = \|h\|_2^2 - h_i^2 - P_{ii}$
  $b_i = h_i P_{ii} - h^T P_{:,i}$
end for
for $i = 1 : n$ **do**
  $h_i^+ = BestPolynomialRoot(a_i, b_i)$
  for $l > i$ **do**
    $a_l \leftarrow a_l + (h_i^+)^2 - h_i^2$
    $b_l \leftarrow b_l + P_{li}(h_i^+ - h_i)$
  end for
  $h_i = h_i^+$
end for

Algorithm 3: $H = generalCDSymNMF(A, H_0)$.

INPUT: $A \in \mathbb{R}^{n\times n}, H_0 \in \mathbb{R}^{n\times r}$
OUTPUT: $H \in \mathbb{R}^{n\times r}_+$
$H = H_0$
$R = A - HH^T$
while stopping criterion not satisfied **do**
  for $j = 1 : r$ **do**
    $R^{(j)} \leftarrow R + H_{:,j}H_{:,j}^T$
    $H_{:,j} \leftarrow rankoneCDSymNMF(R^{(j)}, H_{:,j})$
    $R \leftarrow R^{(j)} - H_{:,j}H_{:,j}^T$
  end for
end while

B. First exact coordinate descent method for SymNMF

To tackle SymNMF (1), we apply Algorithm 2 on every column of H successively, that is, we apply Algorithm 2 with $h = H(:, j)$ and $P = R^{(j)}$ for $j = 1, \ldots, r$. The procedure is simple to describe, see Algorithm 3 which implements the exact cyclic CD method applied to SymNMF.

One can easily check that Algorithm 3 requires $O(n^2 r)$ operations to update the nr entries of H once:

- In step 4, the full residual matrix $R = A - HH^T$ is precomputed where the product HH^T requires $O(rn^2)$ operations.
- In step 7, the residual matrix $R^{(j)}$ can be computed using the fact that $R^{(j)} = R + H_{:,j}H_{:,j}^T$, which requires $O(n^2)$ operations.
- In step 8, Algorithm 2 is called, and requires $O(n^2)$ operations.
- In step 9, the full residual matrix $R = R^{(j)} - H_{:,j}H_{:,j}^T$ is updated, which requires $O(n^2)$ operations.

Algorithm 3 has some drawbacks. In particular, the heavy computation of the residual matrix R is unpractical for large sparse matrices (see below). In the next sections, we show how to tackle these issues and propose a more efficient CD method for symNMF, applicable to large sparse matrices.

III. Improved Implementation of Algorithm 3

The algorithm for symNMF developed in the previous section (Algorithm 3) is unpractical when the input matrix A is large and sparse; in the sense that although A can be stored in memory, Algorithm 3 will run out of memory for n large. In fact, the residual matrix R with n^2 entries computed in step 4 of Algorithm 3 is in general dense (for example if the entries of H are initialized to some positive entries–see Section IV), even if A is sparse. Sparse matrices usually have $O(n)$ non-zero entries and, when n is large, it is unpractical to store $O(n^2)$ entries (this is for example typical for document data sets where n is of the order of millions).

In this section we re-implement Algorithm 3 in order to avoid the explicit computation of the residual matrix R; see Algorithm 4. While Algorithm 3 runs in $O(rn^2)$ operations per iteration and requires $O(n^2)$ space in memory (whether or not A is sparse), Algorithm 4 runs in $O(r\max(K, nr))$ operations per iteration and requires $O(\max(K, nr))$ space in memory, where K is the number of non-zero entries of A. Hence,

- When A is dense, $K = O(n^2)$ and Algorithm 4 will have the same asymptotic computational cost of $O(rn^2)$ operations per iteration as Algorithm 3. However, it performs better in practice because the exact number of operations is smaller.

- When A is sparse, $K = O(n)$ and Algorithm 4 runs in $O(r^2 n)$ operations per iteration, which is significantly smaller than Algorithm 3 in $O(rn^2)$, so that it will be applicable to very large sparse matrices. In fact, in practice, n can be of the order of millions while r is usually smaller than a hundred. This will be illustrated in Section V for some numerical experiments on text data sets.

In the following, we first assume that A is dense when accounting for the computational cost of Algorithm 4. Then, we show that the computational cost is significantly reduced when A is sparse. Since we want to avoid the computation of the residual (4), reducing the problem into rank-one subproblems solved one after the other is not desirable. To evaluate the gradient of the objective function in (1) for the (i,j)th entry of H, we need to modify the expressions (8) and (9) by substituting $R^{(j)}$ with $A - \sum_{k=1,k\neq j}^{r} H_{:,k}H_{:,k}^T$. We have

$$\begin{aligned}\nabla_{H_{ij}} F(H) = &\ \nabla_{H_{ij}} \left(\frac{1}{4} ||A - HH^T||_F^2 \right) \\ = &\ H_{ij}^3 + a_{ij} H_{ij} + b_{ij},\end{aligned}$$

where

$$a_{ij} = \|H_{i,:}\|^2 + \|H_{:,j}\|^2 - 2H_{ij}^2 - A_{ii}, \text{and} \quad (11)$$

$$b_{ij} = H_{i,:}(H^T H)_{:,j} - H_{:,j}^T A_{:,i} - H_{ij}^3 - H_{ij} a_{ij}. \quad (12)$$

The quantities a_{ij} and b_{ij} will no longer be updated during the iterations as in Algorithm 3, but rather computed on the fly before each entry of H is updated. The reason is twofold:

- it avoids storing two n-by-r matrices, and
- the updates of the b_{ij}'s, as done in (10), cannot be performed in $O(n)$ operations without the matrix $R^{(j)}$.

However, in order to minimize the computational cost, the following quantities will be precomputed and updated during the course of the iterations:

- $\|H_{i,:}\|^2$ for all i and $\|H_{:,j}\|^2$ for all j: if the values of $\|H_{i,:}\|^2$ and $\|H_{:,j}\|^2$ are available, a_{ij} can be computed in $O(1)$; see (11). Moreover, when H_{ij} is updated to its optimal value H_{ij}^+, we only need to update $\|H_{i,:}\|^2$ and $\|H_{:,j}\|^2$ which can also be done in $O(1)$:

$$\|H_{i,:}\|^2 \leftarrow \|H_{i,:}\|^2 + (H_{ij}^+)^2 - H_{ij}^2, \quad (13)$$

$$\|H_{:,j}\|^2 \leftarrow \|H_{:,j}\|^2 + (H_{ij}^+)^2 - H_{ij}^2. \quad (14)$$

  Therefore, pre-computing the $\|H_{i,:}\|^2$'s and $\|H_{:,j}\|^2$'s, which require $O(rn)$ operations, allows us to compute the a_{ij}'s in $O(1)$.
- The r-by-r matrix $H^T H$: by maintaining $H^T H$, computing $H_{i,:}(H^T H)_{:,j}$ requires $O(r)$ operations. Moreover, when the (i,j)th entry of H is updated to H_{ij}^+, updating $H^T H$ requires $O(r)$ operations:

$$(H^T H)_{jk} \leftarrow (H^T H)_{jk} - H_{ik}(H_{ij}^+ - H_{ij}), \quad k = 1, \ldots, r. \quad (15)$$

To compute b_{ij}, we also need to perform the product $H_{:,j}^T A_{:,i}$; see (12). This requires $O(n)$ operations, which cannot be avoided and is the most expensive part of the algorithm.

Algorithm 4: $H = cyclicCDSymNMF(A, H_0)$.

1: INPUT: $A \in \mathbb{R}^{n\times n}, H_0 \in \mathbb{R}^{n\times r}$
2: OUTPUT: $H \in \mathbb{R}^{n\times r}$
3: $H = H_0$
4: **for** $j = 1 : r$ **do**
5: $\quad C_j = \|H_{:,j}\|^2$
6: **end for**
7: **for** $i = 1 : n$ **do**
8: $\quad L_i = \|H_{i,:}\|^2$
9: **end for**
10: $D = H^T H$
11: **while** stopping criterion not satisfied **do**
12: $\quad$ **for** $j = 1 : r$ **do**
13: $\quad\quad$ **for** $i = 1 : n$ **do**
14: $\quad\quad\quad a_{ij} \leftarrow C_j + L_i - 2H_{ij}^2 - A_{ii}$
15: $\quad\quad\quad b_{ij} \leftarrow H_{i,:}^T (D)_{j,:} - H_{:,j}^T A_{:,i} - H_{ij}^3 - H_{ij} a_{ij}$
16: $\quad\quad\quad H_{ij}^+ \leftarrow BestPolynomialRoot(a_{ij}, b_{ij})$
17: $\quad\quad\quad C_j \leftarrow C_j + (H_{ij}^+)^2 - H_{ij}^2$
18: $\quad\quad\quad L_i \leftarrow L_i + (H_{ij}^+)^2 - H_{ij}^2$
19: $\quad\quad\quad D_{j,:} \leftarrow D_{j,:} - H_{i,:}(H_{ij}^+ - H_{ij})$
20: $\quad\quad\quad D_{:,j} \leftarrow D_{j,:}$
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: **end while**

In summary, by precomputing the quantities $\|H_{i,:}\|^2$, $\|H_{:,j}\|^2$ and $H^T H$, it is possible to apply one iteration of CD over the nr variables in $O(n^2 r)$ operations. The computational cost is the same as in Algorithm 3, in the dense case, but no residual matrix is computed; see Algorithm 4.

From line 4 to line 10, the precomputations are performed in $O(nr^2)$ time where computing $H^T H$ is the most expensive part. Then the two loops iterate over all the entries to update each variable once. Computing b_{ij} (in line 4) is the bottleneck of the CD scheme as it is the only part in the two loops which requires $O(n)$ time. However, when the matrix A is sparse, the cost of computing $H_{:,j}^T A_{:,i}$ for all i, that is computing $H_{:,j}^T A$, drops to $O(K)$ where K is the number of nonzero entries in A. Taking into account the term $H_{i,:}(H^T H)_{j,:}$ to compute b_{ij} that requires $O(r)$ operations, we have that Algorithm 4 requires $O(r \max(K, nr))$ operations per iteration.

IV. Initialization and Convergence

In this section, we discuss initialization and convergence of Algorithm 4. We also provide a small modification for Algorithm 4 to perform better (especially when random initialization is used).

a) Initialization: In most previous works, the matrix H is initialized randomly, using the uniform distribution in the interval [0, 1] for each entry of H [7]. Note that, in practice, to obtain an unbiased initial point, the matrix H should be multiplied by

a constant β^* such that

$$\beta^* = \arg\min_{\beta \geq 0} ||A - (\beta H_0)(\beta H_0)^T||_F$$
$$= \sqrt{\frac{\langle A, H_0 H_0^T \rangle}{\langle H_0 H_0^T, H_0 H_0^T \rangle}} = \sqrt{\frac{\langle AH_0, H_0 \rangle}{||H_0^T H_0||_F^2}}. \quad (16)$$

This allows the initial approximation $H_0 H_0^T$ to be well scaled compared to A. When using such an initialization, we observed that using random shuffling of the columns of H before each iteration (that is, optimizing the columns of H in a different order each time we run Algorithm 4) performs in general much better; see Section V.

Remark 1 (Other Heuristics to Accelerate Coordinate Descent Methods): sDuring the course of our research, we have tried several heuristics to accelerate Algorithm 4, including three of the most popular strategies:

- *Gauss-Southwell strategies*. We have updated the variables by ordering them according to some criterion (namely, the decrease of the objective function, and the magnitude of the corresponding entry of the gradient).
- *Variable selection*. Instead of optimizing all variables at each step, we carefully selected a subset of the variables to optimize at each iteration (again using a criterion based on the decrease of the objective function or the magnitude of the corresponding entry of the gradient).
- *Random shuffling*. We have shuffled randomly the order in which the variables are updated in each column. This strategy was shown to be superior in several context, although a theoretical understanding of this phenomenon remains elusive [21].

However, these heuristics (and combinations of them) would not improve significantly the effectiveness of Algorithm 4 hence we do not present them here.

Random initialization might not seem very reasonable, especially for our CD scheme. In fact, at the first step of our CD method, the optimal values of the entries of the first column $H_{:,1}$ of H are computed sequentially, trying to solve

$$\min_{H_{:,1} \geq 0} ||R^{(1)} - H_{:,1} H_{:,1}^T||_F^2 \text{ with } R^{(1)} = A - \sum_{k=2}^{r} H_{:,k} H_{:,k}^T.$$

Hence we are trying to approximate a matrix $R^{(1)}$ which is the difference between A and a randomly generated matrix $\sum_{k=2}^{r} H_{:,k} H_{:,k}^T$: this does not really make sense. In fact, we are trying to approximate a matrix which is highly perturbed with a randomly generated matrix.

It would arguably make more sense to initialize H at zero, so that, when optimizing over the entries of $H_{:,1}$ at the first step, we only try to approximate the matrix A itself. It turns out that this simple strategy allows to obtain a faster initial convergence than the random initialization strategy. However, we observe the following: this solution tends to have a very particular structure where the first factor is dense and the next ones are sparser. The explanation is that the first factor is given more importance since it is optimized first hence it will be close to the best rank-one approximation of A, which is in general positive (if A is irreducible, by Perron-Frobenius and Eckart-Young theorems). Hence initializing H at zero tends to produce unbalanced factors. However, this might be desirable in some cases as the next factors are in general significantly sparser than with random initialization. To illustrate this, let us perform the following numerical experiment: we use the CBCL face data set (see Section V) that contains 2429 facial images, 19 by 19 pixels each. Let us construct the nonnegative matrix $X \in \mathbb{R}^{361 \times 2429}$ where each column is a vectorized image. Then, we construct the matrix $A = XX^T \in \mathbb{R}^{361 \times 361}$ that contains the similarities between the pixel intensities among the facial images. Hence symNMF of A will provide us with a matrix H where each column of H corresponds to a 'cluster' of pixels sharing some similarities. Fig. 1 shows the columns of H obtained (after reshaping them as images) with zero initialization (left) and random initialization (right) with $r = 49$ as in [1]. We observe that the solutions are very different, although the relative approximation error $||A - HH^T||_F / ||A||_F$ are similar (6.2% for zero initialization vs. 7.5% for random initialization, after 2000 iterations). Depending on the application at hand, one of the two solutions might be more desirable: for example, for the CBCL data set, it seems that the solution obtained with zero initialization is more easily interpretable as facial features, while with the random initialization it can be interpreted as average/mean faces.

This example also illustrates the sensitivity of Algorithm 4 to initialization: different initializations can lead to very different solutions. This is an unavoidable feature for any algorithm trying to find a good solution to an NP-hard problem at a relatively low computational cost.

Finally, we would like to point out that the ability to initialize our algorithm at zero is a very nice feature. In fact, since $H = 0$ is a (first-order) stationary point of (1), *this shows that our coordinate descent method can escape some first-order stationary points, because it uses higher-order information*. For example, any gradient-based method cannot be initialized at zero (the gradient is 0), also the ANLS-based algorithm from [7] cannot escape from zero.

b) Convergence: By construction, the objective function is nonincreasing under the updates of Algorithm 4 while it is bounded from below. Moreover, since our initial estimate H_0 is initially scaled (16), we have $||A - H_0 H_0^T||_F \leq ||A||_F$ and therefore any iterate H of Algorithm 4 satisfies

$$||HH^T||_F - ||A||_F \leq ||A - HH^T||_F \leq ||A - H_0 H_0^T||_F \leq ||A||_F.$$

Since $H \geq 0$, we have for all k

$$||H_{:k} H_{:k}^T||_F \leq ||\sum_{k=1}^{r} H_{:k} H_{:k}^T||_F = ||HH^T||_F,$$

which implies that $||H_{:k}||_2 \leq \sqrt{2||A||_F}$ for all k hence all iterates of Algorithm 4 belong in a compact set. Therefore, Algorithm 4 generates a converging subsequence (Bolzano-Weierstrass theorem). (Note that, even if the initial iterate is not scaled, all iterates belong to a compact set, replacing $2||A||_F$ by $||A||_F + ||A - H_0 H_0^T||_F$.)

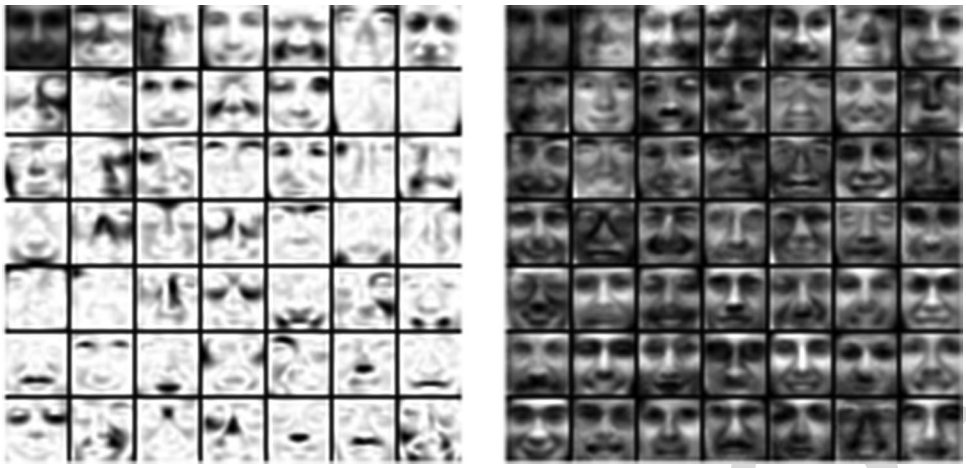


Fig. 1. Comparison of the basis elements obtained with symNMF on the CBCL data set ($r = 49$) with (left) zero initialization and (right) random initialization.

Unfortunately, in its current form, it is difficult to prove convergence of our algorithm to a stationary point. In fact, to guarantee the convergence of an exact cyclic coordinate method to a stationary point, three sufficient conditions are (i) the objective function is continuously differentiable over the feasible set, (ii) the sets over which the blocks of variables are updated are compact as well as convex,[1] and (iii) the minimum computed at each iteration for a given block of variables is uniquely attained; see Prop. 2.7.1 in [24], [25]. Conditions (i-ii) are met for Algorithm 4. Unfortunately, it is not necessarily the case that the minimizer of a fourth order polynomial is unique. (Note however that for a randomly generated polynomial, this happens with probability 0. We have observed numerically that this in fact never happens in our numerical experiments, although there are counter examples.)

A possible way to obtain convergence is to apply the maximum block improvement (MBI) method, that is, at each iteration, only update the variable that leads to the largest decrease of the objective function [26]. Although this is theoretically appealing, this makes the algorithm computationally much more expensive hence much slower in practice. (A possible fix is to use MBI not for every iteration, but every Tth iteration for some fixed T.)

Although the solution of symNMF might not be unique and stationary points might not be isolated, we have always observed in our numerical experiments that the sequence of iterates generated by Algorithm 4 converged to a unique limit point. In that case, we can prove that this limit point is a stationary point.

Proposition 1: Let $(H_{(0)}, H_{(1)}, \ldots)$ be a sequence of iterates generated by Algorithm 4. If that sequence converges to a unique accumulation point, it is a stationary point of symNMF (1).

Proof: This proof follows similar arguments as the proof of convergence of exact cyclic CD for NMF [19]. Let $\bar{H}$ be the accumulation point of the sequence $(H_{(0)}, H_{(1)}, \ldots)$, that is,

$$\lim_{k\to\infty} H_{(k)} \quad = \quad \bar{H}.$$

[1] An alternative assumption to the condition (ii) under which the same result holds is when the function is monotonically nonincreasing in the interval from one iterate to the next [24].

Note that, by construction,

$$F(H_{(1)}) \geq F(H_{(2)}) \geq \cdots \geq F(\bar{H}).$$

Note also that we consider that only one variable has been updated between $H_{(k+1)}$ and $H_{(k)}$.

Assume $\bar{H}$ is not a stationary point of (1): therefore, there exists (i, j) such that

- $\bar{H}_{i,j} = 0$ and $\nabla F(\bar{H})_{i,j} < 0$, or
- $\bar{H}_{i,j} > 0$ and $\nabla F(\bar{H})_{i,j} \neq 0$.

In both cases, since F is smooth, there exists $p \neq 0$ such that

$$F(\bar{H} + pE^{ij}) = F(\bar{H}) - \epsilon < F(\bar{H}),$$

for some $\epsilon > 0$, where E^{ij} is the matrix of all zeros except at the (i, j)th entry where it is equal to one and $\bar{H} + pE^{ij} \geq 0$.

Let us define $(H_{(n_0)}, H_{(n_1)}, \ldots)$ a subsequence of $(H_{(0)}, H_{(1)}, \ldots)$ as follows: $H_{(n_k)}$ is the iterate for which the (i, j)th entry is updated to obtain $H_{(n_k+1)}$. Since Algorithm 4 updates the entries of H column by column, we have $n_k = (j-1)n + i - 1 + nrk$ for $k = 0, 1, \ldots$.

By continuity of F and the convergence of the sequence $H_{(n_k)}$, there exists K sufficiently large so that for all $k > K$:

$$F(H_{(n_k)} + pE^{ij}) \;\leq\; F(\bar{H}) - \frac{\epsilon}{2}. \tag{17}$$

In fact, the continuity of F implies that for all $\xi > 0$, there exists $\delta > 0$ sufficiently small such that $||\bar{H} - H_{(n_k)}||_2 < \delta \Rightarrow |F(\bar{H}) - F(H_{(n_k)})| < \xi$. It suffices to choose n_k sufficiently large so that δ is sufficiently small (since $H_{(n_k)}$ converges to $\bar{H}$) for the value $\xi = \epsilon/2$.

Let us flip the sign of (17) and add $F(H_{(n_k)})$ on both sides to obtain

$$F(H_{(n_k)}) - F(H_{(n_k)} + pE^{ij}) \;\geq\; F(H_{(n_k)}) - F(\bar{H}) + \frac{\epsilon}{2}.$$

By construction of the subsequence, the (i, j)th entry of $H_{(n_k)}$ is updated first (the other entries are updated afterward) to obtain $H_{(n_{k+1})}$ which implies that

$$F(H_{(n_{k+1})}) \leq F(H_{(n_k+1)}) \leq F(H_{(n_k)} + pE^{ij})$$

TABLE I
IMAGE DATASETS

Data	# pixels	m	n
ORL[1]	112×92	10304	400
Umist[2]	112×92	10304	575
CBCL[3]	19×19	361	2429
Frey[2]	28×20	560	1965

[1]http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[2]http://www.cs.toronto.edu/roweis/data.html
[3]http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html

hence

$$\begin{aligned} F(H_{(n_k)}) - F(H_{(n_{k+1})}) &\geq F(H_{(n_k)}) - F(H_{(n_k)} + pE^{ij}) \\ &\geq F(H_{(n_k)}) - F(\bar{H}) + \frac{\epsilon}{2} \\ &\geq \frac{\epsilon}{2}, \end{aligned}$$

since $F(\bar{H}) \leq F(H_{(n_k)})$. We therefore have that for all $k > K$,

$$F(H_{(n_{k+1})}) \leq F(H_{(n_k)}) - \frac{\epsilon}{2},$$

a contradiction since F is bounded below. ■

Note that Proposition 1 is useful in practice since it can easily be checked whether Algorithm 4 converges to a unique accumulation point, plotting for example the norm between the different iterates.

V. NUMERICAL RESULTS

This section shows the effectiveness of Algorithm 4 on several data sets compared to the state-of-the-art techniques. It is organized as follows. In Section V-A, we describe the real data sets and, in Section V-B, the tested symNMF algorithms. In Section V-C, we describe the settings we use to compare the symNMF algorithms. In Section V-D, we provide and discuss the experimental results.

A. Data Sets

We will use exactly the same data sets as in [18]. Because of space limitation, we only give the results for one value of the factorization rank r, more numerical experiments are available on the arXiv version of this paper [27]. In [18], authors use four dense data sets and six sparse data sets to compare several NMF algorithms. In this section, we use these data sets to generate similarity matrices A on which we compare the different symNMF algorithms. Given a nonnegative data set $X \in \mathbb{R}_+^{m \times n}$, we construct the symmetric similarity matrix $A = X^T X \in \mathbb{R}_+^{n \times n}$, so that the entries of A are equal to the inner products between data points. Table I summarizes the dense data sets, corresponding to widely used facial images in the data mining community. Table II summarizes the characteristics of the different sparse data sets, corresponding to document datasets and described in details in [28].

TABLE II
TEXT MINING DATA SETS (SPARSITY IS GIVEN AS THE PERCENTAGE OF ZEROS)

Data	m	n	#nonzero	sparsity X	sparsity $X^T X$
classic	7094	41681	223839	99.92	99.50
sports	8580	14870	1091723	99.14	84.51
reviews	4069	18483	758635	98.99	84.24
hitech	2301	10080	331373	98.57	80.32
ohscal	11162	11465	674365	99.47	91.58
la1	3204	31472	484024	99.52	95.72

B. Tested symNMF Algorithms

We compare the following algorithms

1) (Newton) This is the Newton-like method from [7].
2) (ANLS) This is the method based on the ANLS method for NMF adding the penalty $||W - H||_F^2$ in the objective function (see Introduction) from [7]. Note that ANLS has the drawback to depend on a parameter that is nontrivial to tune, namely, the penalty parameter for the term $||W - H||_F^2$ in the objective function (we used the default tuning strategy recommended by the authors).
3) (tSVD) This method, recently introduced in [29], first computes the rank-r truncated SVD of $A \approx A_r = U_r \Sigma_r U_r^T$ where U_r contains the first r singular vectors of A and Σ_r is the r-by-r diagonal matrix containing the first r singular values of A on its diagonal. Then, instead of solving (1), the authors solve a 'closeby' optimization problem replacing A with A_r

$$\min_{H \geq 0} ||A_r - HH^T||_F.$$

Once the truncated SVD is computed, each iteration of this method is extremely cheap as the main computational cost is in a matrix-matrix product $B_r Q$, where $B_r = U_r \Sigma_r^{1/2}$ and Q is an r-by-r rotation matrix, which can be computed in $O(nr^2)$ operations. Note also that they use the initialization $H_0 = \max(0, B_r)$—we flipped the signs of the columns of U_r to maximize the ℓ_2 norm of the nonnegative part [30].
4) (BetaSNMF) This algorithm is presented in ([31, Algorithm 4], and is based on multiplicative updates (similarly as for the original NMF algorithm proposed by Lee and Seung [32]). Note that we have also implemented the multiplicative update rules from [33] (and already derived in [3]). However, we do not report the numerical results here because it was outperformed by BetaSNMF in all our numerical experiments, an observation already made in [31].
5) (CD-X-Y) This is Algorithm 4. X is either 'Cyclic' or 'Shuffle' and indicates whether the columns of H are optimized in a cyclic way or if they are shuffled randomly before each iteration. Y is for the initialization: Y is 'rand' for random initialization and is '0' for zero initialization; see Section IV for more details. Hence, we will compare four variants of Algorithm 4: CD-Cyclic-0, CD-Shuffle-0, CD-Cyclic-Rand and CD-Shuffle-Rand.

Because Algorithm 4 requires to perform many loops (nr at each step), Matlab is not a well-suited language. Therefore, we have developed a C implementation, that can be called from Matlab (using Mex files). Note that the algorithms above are better suited for Matlab since the main computational cost

resides in matrix-matrix products, and in solving linear systems of equations (for ANLS and Newton).

Newton and ANLS are both available from http://math.ucla.edu/dakuang/, while we have implemented tSVD and BetaSNMF ourselves.

For all algorithms using random initializations for the matrix H, we used the same initial matrices. Note however that, in all the figures presented in this section, we will display the error after the first iteration, which is the reason why the curves do not start at the same value.

C. Experimental Setup

In order to compare for the average performance of the different algorithms, we denote $e_{\min}$ the smallest error obtained by all algorithms over all initializations, and define

$$E(t) = \frac{e(t) - e_{\min}}{||A||_F - e_{\min}}, \tag{18}$$

where $e(t)$ is the error $||A - HH^T||_F$ achieved by an algorithm for a given initialization within t seconds (and hence $e(0) = ||A - H_0 H_0^T||_F$ where H_0 is the initialization). The quantity $E(t)$ is therefore a normalized measure of the evolution of the objective function of a given algorithm on a given data set.

The advantage of this measure is that it separates better the different algorithms, when using a log scale, since it goes to zero for the best algorithm (except for algorithms that are initialized randomly as we will report the average value of $E(t)$ over several random initializations; see below). We would like to stress out that the measure $E(t)$ from (18) has to be interpreted with care. In fact, an algorithm for which $E(t)$ converges to zero simply means that it is the algorithm able to find the best solution among all algorithms (in other words, to identify a region with a better local minima). In fact, the different algorithms are initialized with different initial points: in particular, tSVD uses an SVD-based initialization. It does not necessarily mean that it converges the fastest: to compare (initial) convergence, one has to look at the values $E(t)$ for t small. However, the measure $E(t)$ allows to better visualize the different algorithms. For example, displaying the relative error $||A - HH^T||/||A||_F$ allows to compare the initial convergence, but then the errors for all algorithms tend to converge at similar values and it is difficult to identify visually which one converges to the best solution.

For the algorithms using random initialization (namely, Newton, ANLS, CD-Cyclic-Rand and CD-Shuffle-Rand), we will run the algorithms 10 times and report the average value of $E(t)$. For all data sets, we will run each algorithm for 100 seconds, or for longer to allow the CD-based approaches to perform at least 100 iterations.

All tests are performed using Matlab on a PC Intel CORE i5-4570 CPU @3.2GHz $\times$ 4, with 7.7G RAM. The codes are available online from https://sites.google.com/site/nicolasgillis/.

Remark 2 (Computation of the Error): Note that to compute $||A - HH^T||_F$, one should not compute HH^T explicitly (especially if A is sparse) and use instead

$$\begin{aligned} ||A - HH^T||_F^2 &= ||A||_F^2 - 2\langle A, HH^T\rangle + ||HH^T||_F^2 \\ &= ||A||_F^2 - 2\langle AH, H\rangle + ||H^T H||_F^2. \end{aligned}$$

D. Comparison

We now compare the different symNMF algorithms listed in Section V-B according to the measure given in (18) on the data sets described in Section V-B, and on synthetic data sets.

1) Real Data Sets: We start with the real data sets.

a) Dense Image Data Sets: Fig. 2 displays the results for the dense real data sets. Table III gives the number of iterations performed by each algorithm within the 500 seconds, and Table IV the final relative error $||A - HH^T||/||A||_F$ in percent.

We observe the following:

- In all cases, tSVD performs best and is able to generate the solution with the smallest objective function value among all algorithms. This might be a bit surprising since it works only with an approximation of the original data: it appears that for these real dense data sets, this approximation can be computed efficiently and allows tSVD to converge extremely fast to a very good solution.
  One of the reasons tSVD is so effective is because each iteration is n times cheaper (once the truncated SVD is computed) hence it can perform many more iterations; see Table III. Another crucial reason is that image data sets can be very well approximated by low-rank matrices (see Section V-D2 for a confirmation of this behavior). Therefore, for images, tSVD is the best method to use as it provides a very good solution extremely fast.
- When it comes to initial convergence, CD-Cyclic-0 and CD-Shuffle-0 perform best: they are able to generate very fast a good solution. In all cases, they are the fastest to generate a solution at a relative error of 1% of the final solution of tSVD. Moreover, the fact that tSVD does not generate any solution as long as the truncated SVD is not computed could be critical for larger data sets. For example, for CBCL with $n = 2429$ and $r = 60$, the truncated SVD takes about 6 seconds to compute while, in the mean time, CD-Cyclic-0 and CD-Shuffle-0 generate a solution with relative error of 0.3% from the final solution obtained by tSVD after 500 seconds.
- For these data sets, CD-Cyclic-0 and CD-Shuffle-0 perform exactly the same: for the zero initialization, it seems that shuffling the columns of H does not play a crucial role.
- When initialized randomly, we observe that the CD method performs significantly better with random shuffling. Moreover, CD-Shuffle-Rand converges initially slower than CD-Shuffle-0 but is often able to converge to a better solution; in particular for the ORL and Umistim data sets.
- Newton converges slowly, the main reason being that each iteration is very costly, namely $O(n^3 r)$ operations.

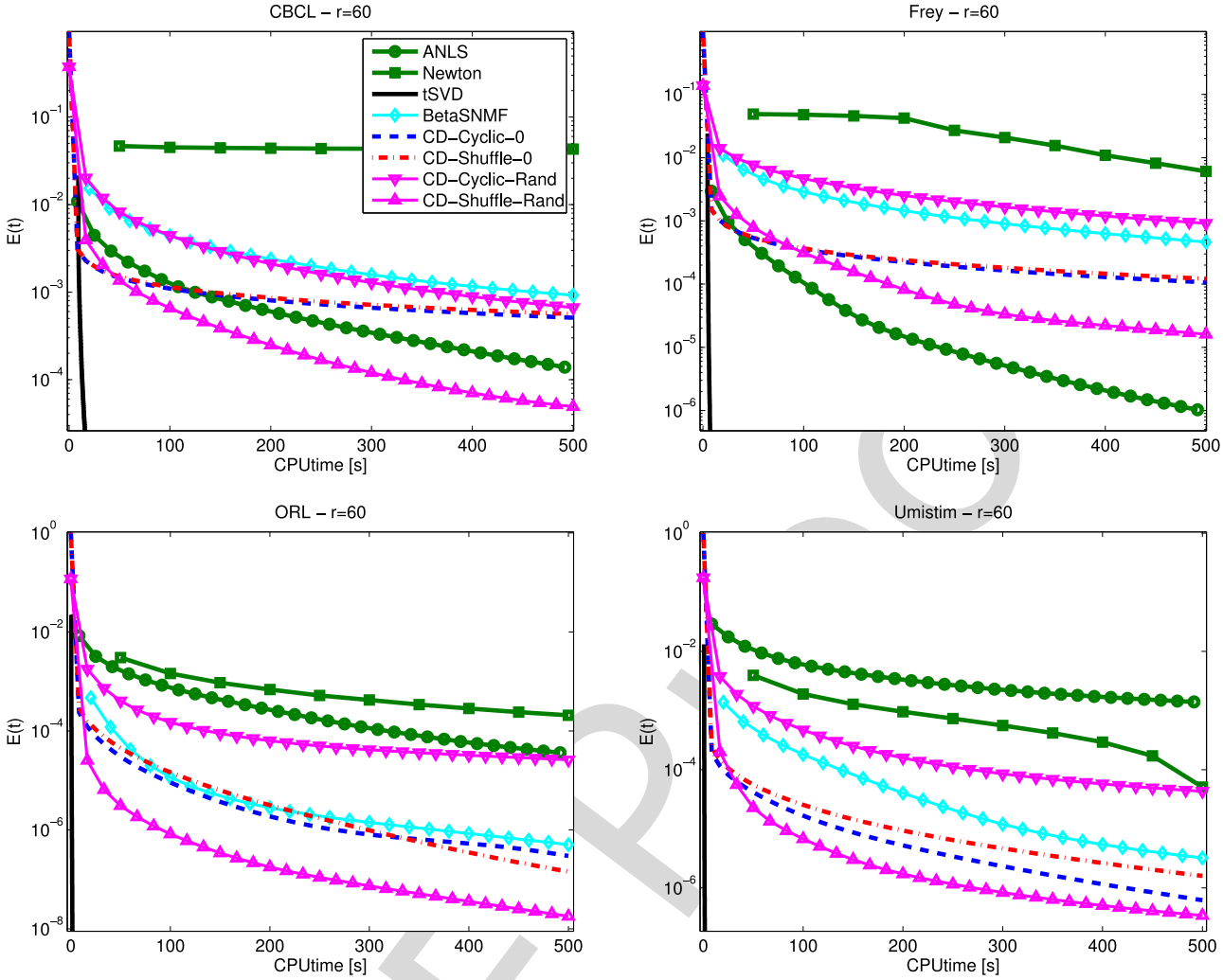


Fig. 2. Evolution of the measure (18) of the different symNMF algorithms on the dense real data sets for $r = 60$.

TABLE III
AVERAGE NUMBER OF ITERATIONS PERFORMED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE DENSE REAL DATA SETS

$r = 60$	ANLS	Newton	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
ORL	56995	14377	204960	234400	17738	15741	16235	16062
Umist	33555	8993	225968	132830	9193	8756	8951	8955
CBCL	3965	21	93252	10254	726	722	775	784
Frey	5692	456	173030	15465	1303	1290	1213	1216

TABLE IV
AVERAGE RELATIVE ERROR IN PERCENT ($100 * ||A - HH^T||_F / ||A||_F$) OF THE FINAL SOLUTION OBTAINED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE DENSE REAL DATA SETS. FOR ALGORITHMS BASED ON RANDOM INITIALIZATIONS, THE STANDARD DEVIATION IS GIVEN

$r = 60$	ANLS	Newton	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
ORL	0.144 ± 4e-4	0.168	**0.14**	0.141 ± 4e-5	0.141	0.141	0.143 ± 4e-4	**0.14** ± 4e-6
Umist	0.165 ± 6e-3	0.098	**0.04**	0.041 ± 8e-5	0.041	0.041	0.045 ± 3e-4	0.041 ± 3e-5
CBCL	0.059 ± 4e-4	4.34	**0.046**	0.138 ± 1e-3	0.097	0.102	0.112 ± 7e-3	0.051 ± 6e-4
Frey	0.057 ± 6e-5	0.66	**0.056**	0.103 ± 5e-4	0.067	0.069	0.148 ± 2e-3	0.058 ± 2e-4

- ANLS performs relatively well: it never converges initially faster than CD-based approaches but is able to generate a better final solution for the Frey data set.
- BetaSNMF does not perform well on these data sets compared to tSVD and CD methods, although performing better than Newton and 2 out of 4 times better than ANLS.
- For algorithms based on random initializations, the standard deviation between several runs is rather small, illustrating the fact that these algorithms converge to solutions with similar final errors.

Conclusion: for image data sets, tSVD performs the best. However, CD-Cyclic-0 allows a very fast initial

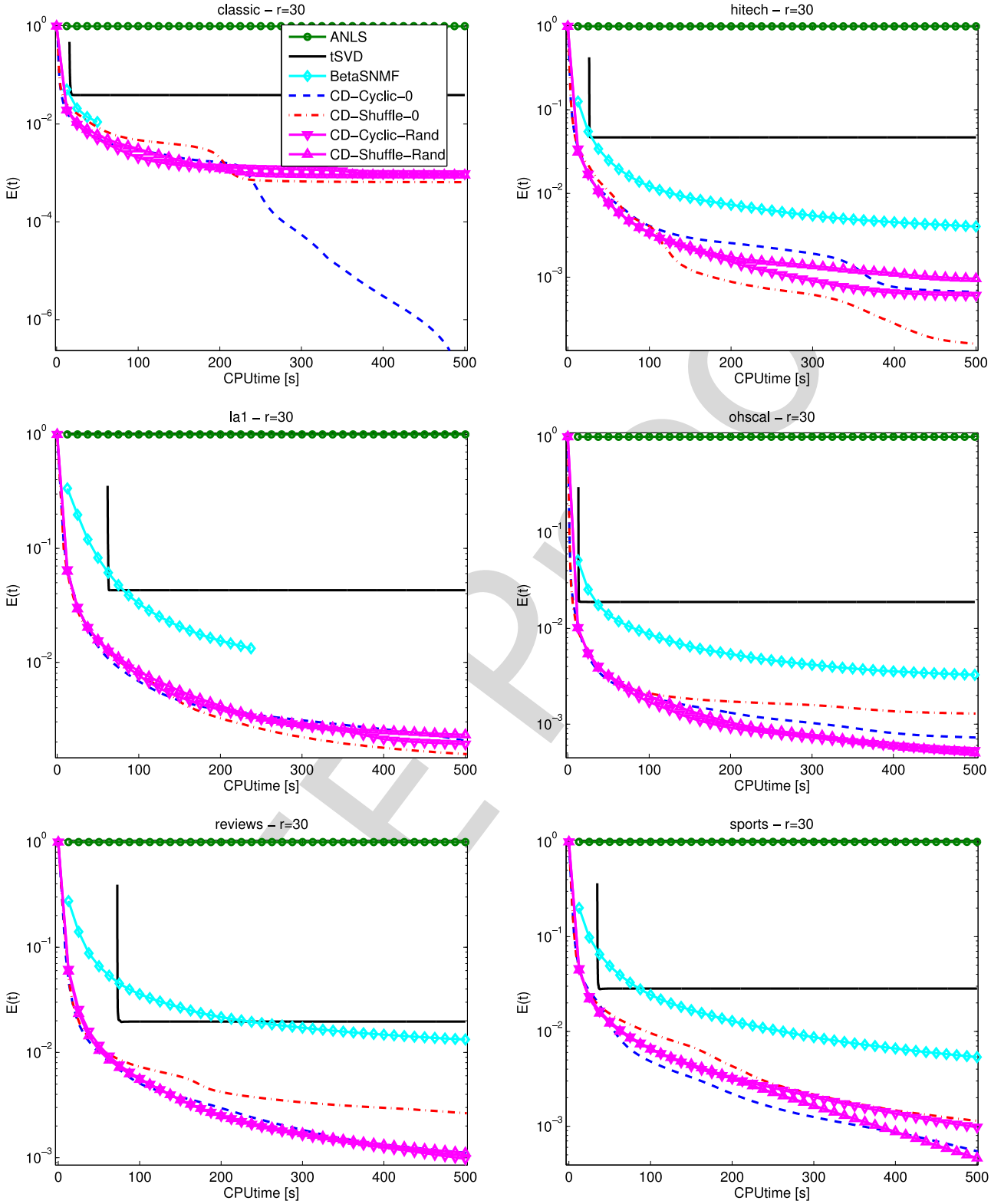


Fig. 3. Evolution of the measure (18) of the different symNMF algorithms on real sparse data sets for $r = 30$.

convergence and can be used to obtain very quickly a good solution.

b) Sparse document data sets: Fig. 3 displays the results for the real sparse data sets. Table V gives the number of iterations performed by each algorithm within the 500 seconds, and Table VI the final relative error $||A - HH^T||/||A||_F$ in percent.

It is interesting to note that, for some data sets (namely, la1 and reviews), computing the truncated SVD of A is not possible with Matlab within 60 seconds hence tSVD is not able to return

TABLE V
AVERAGE NUMBER OF ITERATIONS PERFORMED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE SPARSE REAL DATA SETS

$r = 30$	ANLS	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
classic	550	21345	163	389	390	386	386
sports	254	57358	540	170	171	163	163
reviews	162	41519	353	114	114	114	114
hitech	458	81975	898	281	282	284	284
ohscal	680	75935	1462	495	494	494	494
la1	154	24667	163	126	126	127	127

TABLE VI
AVERAGE RELATIVE ERROR IN PERCENT ($100 * ||A - HH^T||_F / ||A||_F$) OF THE FINAL SOLUTION OBTAINED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE SPARSE REAL DATA SETS. FOR ALGORITHMS BASED ON RANDOM INITIALIZATIONS, THE STANDARD DEVIATION IS GIVEN

$r = 30$	ANLS	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
classic	99.9 ± 6e-4	39.8	38 ± 0.14	**37.3**	37.4	37.4 ± 0.03	37.4 ± 0.04
sports	99.9 ± 1e-3	19.2	17.3 ± 0.11	**16.9**	**16.9**	**16.9** ± 0.04	**16.9** ± 0.04
reviews	99.9 ± 1e-3	17.1	16.5 ± 0.16	**15.5**	15.7	**15.5** ± 0.05	**15.5** ± 0.03
hitech	99.5 ± 3e-3	33.3	30.3 ± 0.1	**30**	**30**	30.1 ± 0.03	30.1 ± 0.03
ohscal	99.95 ± 6e-4	22.2	20.9 ± 0.06	**20.7**	20.8	**20.7** ± 0.04	**20.7** ± 0.03
la1	99.9 ± 8e-4	34	31.9 ± 0.2	31.2	**31.1**	31.2 ± 0.07	31.2 ± 0.05

any solution before that time; see Remark 3 for a discussion. Moreover, Newton is not displayed because it is not designed for sparse matrices and runs out of memory [7].

We observe the following:

- tSVD performs very poorly. The reason is twofold: (1) the truncated SVD is very expensive to compute and (2) sparse matrices are usually not close to being low-rank hence tSVD converges to a very poor solution (see Section V-D2 for a confirmation of this behavior).
- ANLS performs very poorly and is not able to generate a good solution. In fact, it has difficulties to decrease the objective function (on the figures, it seems it does not decrease, but it actually decreases very slowly).
- BetaSNMF performs better than ANLS but does not compete with CD methods. (Note that, for the classic and la1 data sets, BetaSNMF was stopped prematurely because there was a division by zero which could have been avoided but we have strictly used the description of Algorithm 4 in [31]).
- All CD-based approaches are very effective and perform similarly. It seems that, in these cases, nor the initialization nor the order in which the columns of H are updated plays a significant role.

  In fact, for algorithms initialized randomly, Fig. 3 reports the average over 10 runs but, on average, random initialization performs similarly as the initialization with zero.

  In one case (classic data set), CD-Cyclic-0 is able to generate a better final solution.

Conclusion: for sparse document data sets, CD-based approaches outperform significantly the other tested methods.

Remark 3 (SVD Computation in tSVD): It has to be noted that, in our numerical experiments, the matrix A is constructed using the formula $A = X^T X$, where the columns of the matrix X are the data points. In other words, we use the simple similarity measure $y^T z$ between two data points y and z. In that case, the SVD of A can be obtained from the SVD of X, hence can be made (i) more efficient (when X has more columns than rows, that is, $m \ll n$), and (ii) numerically more accurate (because the condition number of $X^T X$ is equal to the square of that of X); see, e.g., [34, Lecture 31]. Moreover, in case of sparse data, this avoids the fill-in, as observed in Table II where $X^T X$ is denser than X. Therefore, in this particular situation when $A = X^T X$ and X is sparse and/or $m \ll n$, it is much better to compute the SVD of A based on the SVD of X. Table VII gives the computational time in both cases. In this particular scenario, it would make sense to use tSVD as an initialization procedure for CD methods to obtain rapidly a good initial iterate. However, looking at Fig. 3 and Table VI indicates that this would not necessarily be advantageous for the CD-based methods in all cases. For example, for the classic data set, tSVD would achieve a relative error of 39.8% within about 6 seconds while CD methods obtain a similar relative error within that computing time. For the hitech data set however, this would be rather helpful since tSVD would only take about 1 second to obtain a relative error of 33.3% while CD methods require about 9 seconds to do so.

TABLE VII
COMPUTATIONAL TIME REQUIRED TO COMPUTE THE RANK-30 TRUNCATED SVD OF X AND $X^T X$ USING MATLAB

`svds(.,30)`	classic	hitech	la1	ohscal	reviews	sports
`X'*X`	17.14	18.54	63.33	15	67.32	31.77
`X`	5.55	0.82	3.08	2.87	1.39	2.98

However, the goal of this paper is to provide an efficient algorithm for the general symNMF problem, without assuming any particular structure on the matrix A (in practice the similarity measure between data points is usually not simply their inner product). Therefore, we have not assumed that the matrix A had this particular structure and only provide numerical comparison in that case.

Remark 4 (Low-Rank Models for Full-Rank Matrices): Although sparse data sets are usually not low rank, it still makes sense to try to find a low-rank structure that is close to a given data set, as this often allows to extract some pertinent information. In particular, in document classification and clustering, low-rank models have proven to be extremely useful; see the discussion in the Introduction and the references

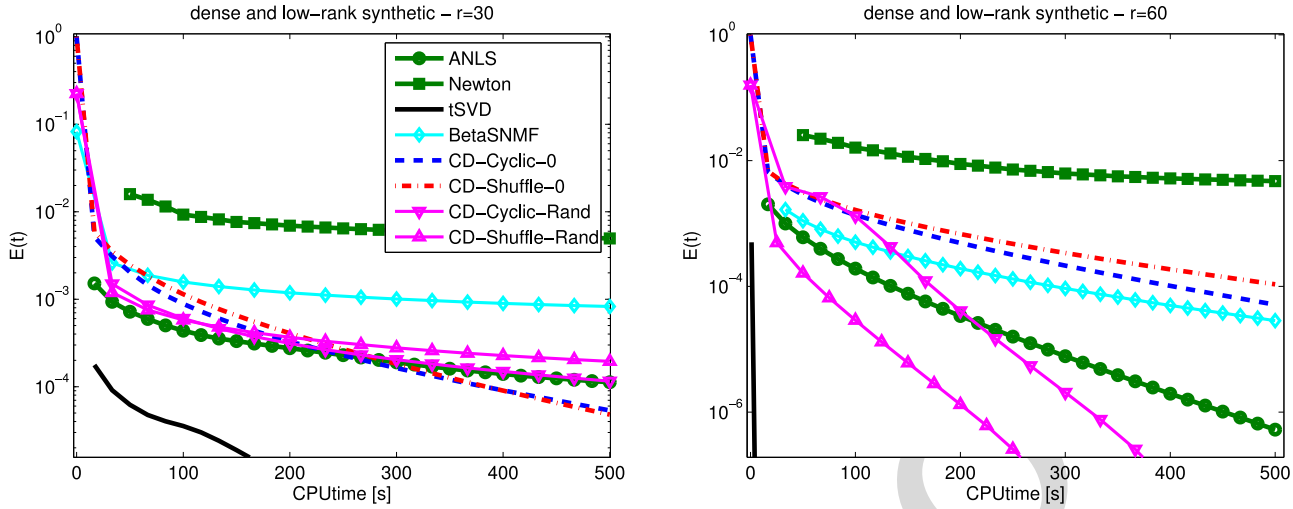


Fig. 4. Evolution of the measure (18) of the different symNMF algorithms on dense and low-rank synthetic data sets for $r = 30$ (left) and $r = 60$ (right).

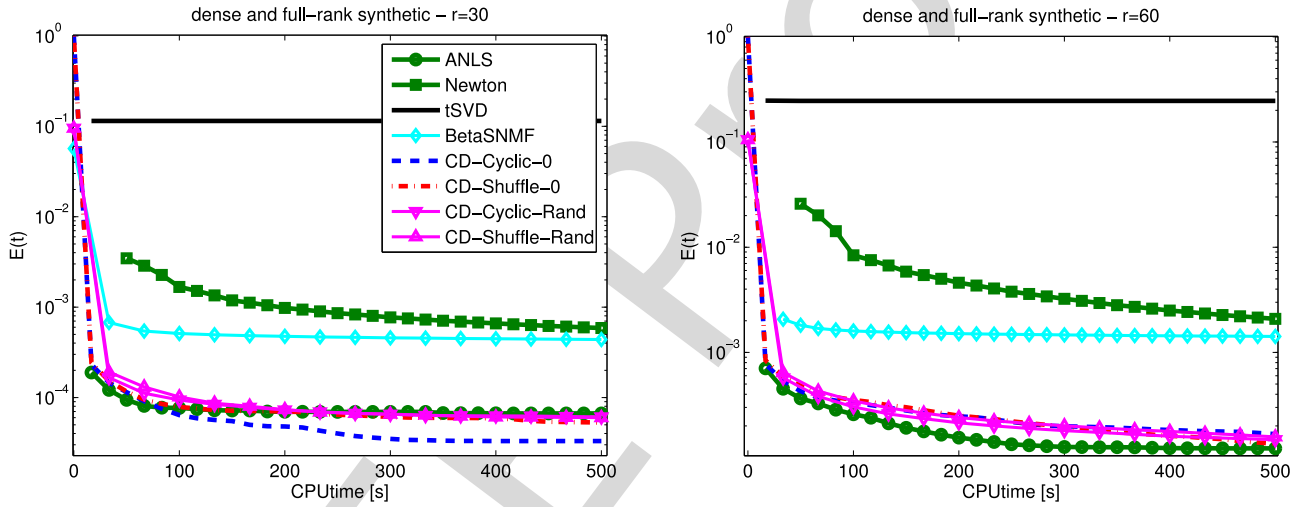


Fig. 5. Evolution of the measure (18) of the different symNMF algorithms on dense full-rank synthetic data sets for $r = 30$ (left) and $r = 60$ (right).

therein. Another important application where low-rank models have proven extremely useful although the data sets are usually not low-rank is *recommender systems* [35] and community detection (see, e.g., [36]). We also refer the reader to the recent survey on low-rank models [37].

2) Synthetic data sets: low-rank vs. full rank matrices: In this section, we perform some numerical experiments on synthetic data sets. Our main motivation is to confirm the (expected) behavior observed on real data: tSVD performs extremely well for low-rank matrices and poorly on full-rank matrices.

a) Low-rank input matrices: The most natural way to generate nonnegative symmetric matrices of given cp-rank is to generate H_* randomly and then compute $A = H_* H_*^T$. In this section, we use the Matlab function $H_* = \texttt{rand}(n, r)$ with $n = 500$ and $r = 30, 60$, that is, each entry of H_* is generated uniformly at random in the interval $[0, 1]$. We have generated 10 such matrices for each rank, and Fig. 4 displays the average value for the measure (18) but we use here $e_{\min} = 0$ since it is the known optimal value.

We observe that, in all cases, tSVD outperforms all methods. Moreover, it seems that the SVD-based initialization is very effective. The reason is that A has exactly rank r and hence its best rank-r approximation is exact. Moreover, tSVD only works in the correct subspace in which H_* belongs hence converges much faster than the other methods.

Except for Newton, the other algorithms perform similarly. It is worth noting that the same behavior we observed for real dense data sets is present here: CD-Shuffle-Rand performs better than CD-Cyclic-Rand, while shuffling the columns of H before each iteration does not play a crucial role with the zero initialization.

b) Full-Rank Input Matrices: A simple way to generate nonnegative symmetric matrices of full rank is to generate a matrix B randomly and then compute $A = B + B^T$. In this section, we use the Matlab function $B = \texttt{rand}(n)$ with $n = 500$. We have generated 10 such matrices for each rank, and Fig. 5 displays the average value for the measure $E(t)$ from (18). Fig. 5 displays the results.

We observe that, in all cases, tSVD performs extremely poorly while all other methods (except for Newton and BetaSNMF) perform similarly. The reason is that tSVD works only with the best rank-r approximation of A, which is poor when A has full rank.

3) Summary of results: Clearly, tSVD and CD-based approaches are the most effective, although ANLS sometimes performs competitively for the dense data sets. However, tSVD performs extremely well only when the input matrix is low rank (cf. low-rank synthetic data sets) or close to being low rank (cf. image data sets). There are three cases when it performs very poorly:

- It cannot perform a symNMF when the factorization rank r is larger than the rank of A, that is, when $r > \text{rank}(A)$, which may be necessary for matrices with high cp-rank (in fact, the cp-rank can be much higher than the rank [9]).
- If the truncated SVD is a poor approximation of A, the algorithm will perform poorly since it does not use any other information; see the results for the full rank synthetic data sets and the sparse real data sets.
- The algorithm returns no solution as long as the SVD is not computed. In some cases, the cost of computing the truncated SVD is high and tSVD could terminate before any solution to symNMF is produced; see the sparse real data sets.

To conclude, CD-based approaches are overall the most reliable and most effective methods to solve symNMF (1). For dense data sets, initialization at zero allows a faster initial convergence, while CD-Shuffle-Rand generates in average the best solution and CD-Cyclic-Rand does not perform well and is not recommended. For sparse data sets, all CD variants perform similarly and outperform the other tested algorithms.

VI. Conclusion and Further Research

In this paper, we have proposed very efficient exact coordinate descent methods for symNMF (1) that performs competitively with state-of-the-art methods.

Some interesting directions for further research are the following:

- The study of sparse symNMF, where one is looking for a sparser matrix H. A natural model would for example use the sparsity-inducing ℓ_1 norm and try to solve

$$\min_{H \geq 0} \ \frac{1}{4}||A - HH^T||_F^2 + \sum_{j=1}^{r} \Lambda_j ||H_{:,j}||_1 \ , \qquad (19)$$

  for some penalty parameter $\Lambda \in \mathbb{R}^r_+$. Algorithm 4 can be easily adapted to handle (19), by replacing the b_{ij}'s with $b_{ij} + \Lambda_j$. In fact, the derivative of the penalty term only influences the constant part in the gradient; see (12). However, it seems the solutions of (19) are very sensitive to the parameter Λ which are therefore difficult to tune. Note that another way to identify sparser factors is simply to increase the factorization rank r, or to sparsify the input matrix A (only keeping the important edges in the graph induced by A; see [38] and the references therein)—in fact, a sparser matrix A induces sparser factors since

$$A_{ij} = 0 \Rightarrow H_{i,:}H_{j,:}^T \approx 0 \Rightarrow H_{ik} \approx 0 \text{ or } H_{jk} \approx 0 \, \forall k.$$

  This is an interesting observation: $A_{ij} = 0$ implies a (soft) orthogonality constraints on the rows of H. This is rather natural: if item i does not share any similarity with item j ($A_{ij} = 0$), then they should be assigned to different clusters ($H_{ik} \approx 0$ or $H_{jk} \approx 0$ for all k).
- The design of more efficient algorithms for symNMF. For example, a promising direction would be to combine the idea from [29] that use a compressed version of A with very cheap per-iteration cost with our more reliable CD method, to combine the best of both worlds.

Acknowledgment

The authors would like to thank the reviewers for their insightful feedback that helped them improve the paper significantly.

References

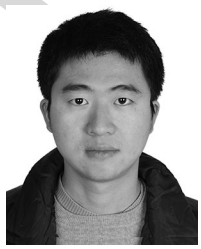
[1] D. Lee and H. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.
[2] R. Zass and A. Shashua, "A unifying approach to hard and probabilistic clustering," in *Proc. 10th IEEE Int. Conf. Comput. Vis. (ICCV)*, vol. 1, 2005, pp. 294–301.
[3] B. Long, Z. M. Zhang, X. Wu, and P. S. Yu, "Relational clustering by symmetric convex coding," in *Proc. 24th ACM Int. Conf. Mach. Learn.*, 2007, pp. 569–576.
[4] Y. Chen, M. Rege, M. Dong, and J. Hua, "Non-negative matrix factorization for semi-supervised data clustering," *Knowl. Inf. Syst.*, vol. 17, no. 3, pp. 355–379, 2008.
[5] Z. Yang, T. Hao, O. Dikmen, X. Chen, and E. Oja, "Clustering by nonnegative matrix factorization using graph random walk," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1079–1087.
[6] D. Kuang, H. Park, and C. Ding, "Symmetric nonnegative matrix factorization for graph clustering." in *Proc. SIAM Conf. Data Min. (SDM)*, vol. 12, 2012, pp. 106–117.
[7] D. Kuang, S. Yun, and H. Park, "SymNMF: nonnegative low-rank approximation of a similarity matrix for graph clustering," *J. Global Optim.*, vol. 62, no. 3, pp. 545–574, 2014.
[8] X. Yan, J. Guo, S. Liu, X. Cheng, and Y. Wang, "Learning topics in short texts by non-negative matrix factorization on term correlation matrix," in *Proc. SIAM Int. Conf. Data Min.*, 2013. Q1
[9] A. Berman and N. Shaked-Monderer, *Completely Positive Matrices*. Singapore: World Scientific, 2003.
[10] V. Kalofolias and E. Gallopoulos, "Computing symmetric nonnegative rank factorizations," *Linear Algebra Appl.*, vol. 436, no. 2, pp. 421–435, 2012.
[11] S. Burer, "On the copositive representation of binary and continuous nonconvex quadratic programs," *Math. Programm.*, vol. 120, no. 2, pp. 479–495, 2009.
[12] P. Dickinson and L. Gijben, "On the computational complexity of membership problems for the completely positive cone and its dual," *Comput. Optim. Appl.*, vol. 57, no. 2, pp. 403–415, 2014.
[13] J. Kim and H. Park, "Toward faster nonnegative matrix factorization: A new algorithm and comparisons," in *Proc. 8th IEEE Int. Conf. Data Min. (ICDM)*, 2008, pp. 353–362.
[14] J. Kim and H. Park, "Fast nonnegative matrix factorization: An active-set-like method and comparisons," *SIAM J. Scientif. Comput.*, vol. 33, no. 6, pp. 3261–3281, 2011.
[15] N.-D. Ho, "Nonnegative matrix factorization: Algorithms and applications," Ph.D. dissertation, Université Catholique de Louvain, Louvain, France, 2008.
[16] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale Nonnegative Matrix and Tensor Factorizations," *IEICE Trans. Fundam. Electron.*, vol. E92-A, no. 3, pp. 708–721, 2009.

[17] L. Li and Y.-J. Zhang, "FastNMF: Highly efficient monotonic fixed-point nonnegative matrix factorization algorithm with good applicability," *J. Electron. Imag.*, vol. 18, no. 3, pp. 033 004–033 004, 2009.
[18] N. Gillis and F. Glineur, "Accelerated multiplicative updates and hierarchical ALS algorithms for nonnegative matrix factorization," *Neural Comput.*, vol. 24, no. 4, pp. 1085–1105, 2012.
[19] C.-J. Hsieh and I. Dhillon, "Fast coordinate descent methods with variable selection for non-negative matrix factorization," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2011, pp. 1064–1072.
[20] N. Gillis, "Nonnegative matrix factorization: Complexity, algorithms and applications," Ph.D. dissertation, Université Catholique de Louvain, Louvain, France, 2011. [Online]. Available: https://sites.google.com/site/nicolasgillis/
[21] S. Wright, "Coordinate descent algorithms," *Math. Programm.*, vol. 151, no. 1, pp. 3–34, 2015.
[22] M. Belachew and N. Gillis, "Solving the maximum clique problem with symmetric rank-one nonnegative matrix approximation," arXiv:1505.07077, 2015.
[23] G. Cardano, *Ars Magna or the Rules of Algebra*. New York, NY, USA: Dover, 1968.
[24] D. Bertsekas, *Corrections for the Book Nonlinear Programming*, 2nd ed., 1999. [Online]. Available: http://www.athenasc.com/nlperrata.pdf
[25] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.
[26] B. Chen, S. He, Z. Li, and S. Zhang, "Maximum block improvement and polynomial optimization," *SIAM J. Optim.*, vol. 22, no. 1, pp. 87–107, 2012.
[27] A. Vandaele, N. Gillis, Q. Lei, K. Zhong, and I. Dhillon, "Coordinate descent methods for symmetric nonnegative matrix factorization," 2015, arXiv:1509.01404.
[28] S. Zhong and J. Ghosh, "Generative model-based document clustering: a comparative study," *Knowl. Inf. Syst.*, vol. 8, no. 3, pp. 374–384, 2005.
[29] K. Huang, N. Sidiropoulos, and A. Swami, "Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition," *IEEE Trans. Signal Process.*, vol. 62, no. 1, pp. 211–224, 2014.
[30] R. Bro, E. Acar, and T. Kolda, "Resolving the sign ambiguity in the singular value decomposition," *J. Chemometr.*, vol. 22, no. 2, pp. 135–140, 2008.
[31] Z. He, S. Xie, R. Zdunek, G. Zhou, and A. Cichocki, "Symmetric nonnegative matrix factorization: Algorithms and applications to probabilistic clustering," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 2117–2131, 2011.
[32] D. Lee and H. Seung, "Algorithms for Non-negative Matrix Factorization," in *Proc. Adv. Neural Inf. Process.*, vol. 13, 2001.
[33] Z. Yang and E. Oja, "Quadratic nonnegative matrix factorization," *Pattern Recognit.*, vol. 45, no. 4, pp. 1500–1510, 2012.
[34] L. Trefethen and D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, vol. 50, 1997.
[35] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Comput.*, no. 8, pp. 30–37, 2009.
[36] I. Psorakis, S. Roberts, M. Ebden, and B. Sheldon, "Overlapping community detection using Bayesian non-negative matrix factorization," *Phys. Rev. E*, vol. 83, no. 6, p. 066114, 2011.
[37] M. Udell, C. Horn, R. Zadeh, and S. Boyd, "Generalized low rank models," *Found. Trends Mach. Learn.*, 2015, to appear, arXiv:1410.0342.
[38] J. Batson, D. Spielman, N. Srivastava, and S. Teng, "Spectral sparsification of graphs: Theory and algorithms," *Commun. ACM*, vol. 56, no. 8, pp. 87–94, 2013.

Arnaud Vandaele received the M.Sc. degree in computer science engineering from Université de Mons, Belgium, in 2008 and the M.Sc. degree in applied mathematics engineering from Université catholique de Louvain, Belgium, in 2011. He is currently a Ph.D. student at the Université de Mons, Belgium and his research include numerical optimization and linear algebra.

Nicolas Gillis received his Ph.D. from Université catholique de Louvain (Belgium) in 2011. He is currently an associate professor at the Department of Mathematics and Operational Research, Faculté polytechnique, Université de Mons, Belgium. His research interests lie in optimization, numerical linear algebra, machine learning and data mining.

Qi Lei received the B.S degree of Mathematics from Zhejiang University, Zhejiang, China in 2014. She is currently a Ph.D. student at the University of Texas at Austin. Since 2014, she joined the Center for Big Data Analytics under the supervision of Professor Inderjit Dhillon and her research interests lie in large-scale optimization and linear algebra.



Kai Zhong received the B.S. degree in physics from Peking University in China in 2012. He is currently working towards the doctoral degree at the Institute for Computational Engineering and Science in the University of Texas at Austin. His research interests include machine learning, data mining and numerical optimization.

Inderjit Dhillon (F'XX) is the Gottesman Family Centennial Professor of Computer Science and Mathematics at UT Austin, where he is also the Director of the ICES Center for Big Data Analytics. His main research interests are in big data, machine learning, network analysis, linear algebra and optimization. He received his B.Tech. degree from IIT Bombay, and Ph.D. from UC Berkeley. He has received several prestigious awards, including the ICES Distinguished Research Award, the SIAM Outstanding Paper Prize, the Moncrief Grand Challenge Award, the SIAM Linear Algebra Prize, the University Research Excellence Award, and the NSF Career Award. He has published over 140 journal and conference papers, and has served on the Editorial Board of the *Journal of Machine Learning Research*, the IEEE TRANSACTIONS OF PATTERN ANALYSIS AND MACHINE INTELLIGENCE, *Foundations and Trends in Machine Learning* and the *SIAM Journal for Matrix Analysis and Applications*. He is an SIAM Fellow and an ACM Fellow.

QUERIES

Q1. Author: For [8], and all conference paper references, provide page numbers if printed in proceeding or location of conference where presented if not printed.

Q2. Author: For [24], provide publisher name and location.

Q3. Author: Only one source allowed per reference; please either delete the arXiv paper or split into two.

Q4. Author: Please provide initial year of IEEE membership grade.

Efficient and Non-Convex Coordinate Descent for Symmetric Nonnegative Matrix Factorization

Arnaud Vandaele, Nicolas Gillis, Qi Lei, Kai Zhong, and Inderjit Dhillon, *Fellow, IEEE*

***Abstract*—Given a symmetric nonnegative matrix A, symmetric nonnegative matrix factorization (symNMF) is the problem of finding a nonnegative matrix H, usually with much fewer columns than A, such that $A \approx HH^T$. SymNMF can be used for data analysis and in particular for various clustering tasks. Unlike standard NMF, which is traditionally solved by a series of quadratic (convex) subproblems, we propose to solve SymNMF by directly solving the nonconvex problem, namely, minimize $\|A - HH^T\|^2$, which is a fourth-order nonconvex problem. In this paper, we propose simple and very efficient coordinate descent schemes, which solve a series of fourth-order univariate subproblems exactly. We also derive convergence guarantees for our methods and show that they perform favorably compared to recent state-of-the-art methods on synthetic and real-world datasets, especially on large and sparse input matrices.**

***Index Terms*—Symmetric nonnegative matrix factorization, coordinate descent, completely positive matrices.**

I. Introduction

NONNEGATIVE matrix factorization (NMF) has become a standard technique in data mining by providing low-rank decompositions of nonnegative matrices: given a nonnegative matrix $X \in \mathbb{R}^{m \times n}_+$ and an integer $r < \min(m, n)$, the problem is to find $W \in \mathbb{R}^{m \times r}_+$ and $H \in \mathbb{R}^{n \times r}_+$ such that $X \approx WH^T$. In many applications, the nonnegativity constraints lead to a sparse and part-based representation, and a better interpretability of the factors, e.g., when analyzing images or documents [1].

In this paper, we work on a special case of NMF where the input matrix is a symmetric matrix A. Usually, the matrix A will be a *similarity matrix* where the (i, j)th entry is a measure of the similarity between the ith and the jth data points. This is a rather general framework, and the user can decide how to generate the matrix A from his data set by selecting an appropriate metric to compare two data points. As opposed to NMF, we are interested in a symmetric approximation HH^T with the factor H being nonnegative–hence symNMF is an NMF variant with $W = H$. If the data points are grouped into clusters, each rank-one factor $H(:, j)H(:, j)^T$ will ideally correspond to a cluster present in the data set. In fact, symNMF has been used successfully in many different settings and was proved to compete with standard clustering techniques such as normalized cut, spectral clustering, k-means and spherical k-means; see [2]–[8] and the references therein.

SymNMF also has tight connections with completely positive matrices [9], [10], that is, matrices of the form $A = HH^T, H \geq 0$, which play an important role in combinatorial optimization [11]. Note that the smallest r such that such a factorization exists is called the cp-rank of A. The focus of this paper is to provide efficient methods to compute good symmetric and nonnegative low-rank approximations HH^T with $H \geq 0$ of a given nonnegative symmetric matrix A.

Let us describe our problem more formally. Given a n-by-n symmetric nonnegative matrix A and a factorization rank r, symNMF looks for an n-by-r nonnegative matrix H such that $A \approx HH^T$. The error between A and its approximation HH^T can be measured in different ways but we focus in this paper on the Frobenius norm:

$$\min_{H \geq 0} \; F(H) \equiv \frac{1}{4} \left\| A - HH^T \right\|_F^2, \qquad (1)$$

which is arguably the most widely used in practice. Applying standard non-linear optimization schemes to (1), one can only hope to obtain stationary points, since the objective function of (1) is highly non-convex, and the problem is NP-hard [12]. For example, two such methods to find approximate solutions to (1) were proposed in [7]:

1) The first method is a Newton-like algorithm which exploits some second-order information without the prohibitive cost of the full Newton method. Each iteration of the algorithm has a computational complexity of $O(n^3 r)$ operations.
2) The second algorithm is an adaptation of the alternating nonnegative least squares (ANLS) method for NMF [13], [14] where the term $||W - H||_F^2$ penalizing the difference between the two factors in NMF is added to the objective function. That same idea was used in [15] where the author developed two methods to solve this penalized problem but without any available implementation or comparison.

In this paper, we analyze coordinate descent (CD) schemes for (1). Our motivation is that the most efficient methods for NMF are CD methods; see [16]–[19] and the references therein. The reason behind the success of CD methods for NMF is

Manuscript received October 26, 2016; revised March 08, 2016, May 04, 2016, and June 21, 2016; accepted June 22, 2016. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Cedric Fevotte. Nicolas Gillis acknowledges the support by the F.R.S.-FNRS, through the incentive grant for scientific research no F.4501.16. This research was supported by NSF grant CCF-1564000. *(Corresponding author: Arnaud Vandaele.)*

A. Vandaele and N. Gillis are with the Department of Mathematics and Operational Research, University of Mons, 7000 Mons, Belgium (e-mail: arnaud.vandaele@umons.ac.be; nicolas.gillis@umons.ac.be).

Q. Lei and K. Zhong are with the Institute for Computational Engineering and Science, University of Texas at Austin, Austin, TX 78712-1757 USA (e-mail: leiqi@ices.utexas.edu; zhongkai@ices.utexas.edu).

I. Dhillon is with the Institute for Computational Engineering and Science, University of Texas at Austin, Austin, TX 78712-1757 USA, and also with the Department of Computer Science, University of Texas at Austin, Austin, TX 78712-1757 USA (e-mail: inderjit@cs.utexas.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TSP.2016.2591510

twofold: (i) the updates can be written in closed-form and are very cheap to compute, and (ii) the interaction between the variables is low because many variables are expected to be equal to zero at a stationary point [20].

The paper is organized as follows. In Section II, we focus on the rank-one problem and present the general framework to implement an exact CD method for symNMF. The main proposed algorithm is described in Section III. Section IV discusses initialization and convergence issues. Section V presents extensive numerical experiments on synthetic and real data sets, which shows that our CD methods perform competitively with recent state-of-the-art techniques for symNMF.

II. Exact Coordinate Descent Methods for SymNMF

Exact coordinate descent (CD) techniques are among the most intuitive methods to solve optimization problems. At each iteration, all variables are fixed but one, and that variable is updated to its optimal value. The update of one variable at a time is often computationally cheap and easy to implement. However little interest was given to these methods until recently when CD approaches were shown competitive for certain classes of problems; see [21] for a recent survey. In fact, more and more applications are using CD approaches, especially in machine learning when dealing with large-scale problems.

Let us derive the exact cyclic CD method for symNMF. The approximation HH^T of the input matrix A can be written as the sum of r rank-one symmetric matrices:

$$A \approx \sum_{k=1}^{r} H_{:,k} H_{:,k}^T, \tag{2}$$

where $H_{:,k}$ is the kth column of H. If we assume that all columns of H are known except for the jth, the problem comes down to approximate a residual symmetric matrix $R^{(j)}$ with a rank-one nonnegative symmetric matrix $H_{:,j}H_{:,j}^T$:

$$\min_{H_{:,j} \geq 0} \left\| R^{(j)} - H_{:,j} H_{:,j}^T \right\|_F^2, \tag{3}$$

where

$$R^{(j)} = A - \sum_{k=1, k \neq j}^{r} H_{:,k} H_{:,k}^T. \tag{4}$$

For this reason and to simplify the presentation, we only consider the rank-one subproblem in the following Section II-A, before presenting on the overall procedure in Section II-B.

A. Rank-One Symmetric NMF

Given a n-by-n symmetric matrix $P \in \mathbb{R}^{n \times n}$, let us consider the rank-one symNMF problem

$$\min_{h \geq 0} \ f(h) \equiv \frac{1}{4} \left\| P - hh^T \right\|_F^2, \tag{5}$$

where $h \in \mathbb{R}^n_+$. If all entries of P are nonnegative, the problem can be solved for example with the truncated singular value decomposition; this follows from the Perron-Frobenius and Eckart-Young theorems. In our case, the residuals $R^{(j)}$ will in general have negative entries–see (4)–which makes the problem NP-hard in general [22]. The optimality conditions for (5) are given by

$$h \geq 0, \nabla f(h) \geq 0, \text{ and } h_i \nabla f(h)_i = 0 \text{ for all } i, \tag{6}$$

where $\nabla f(h)_i$ the ith component of the gradient $\nabla f(h)$. For any $1 \leq i \leq n$, the exact CD method consists in alternatively updating the variables in a cyclic way:

$$\text{for } i = 1, 2, \ldots, n: \quad h_i \leftarrow h_i^+,$$

where h_i^+ is the optimal value of h_i in (5) when all other variables are fixed. Let us show how to compute h_i^+. We have:

$$\nabla f(h)_i = h_i^3 + \underbrace{\left(\sum_{l=1, l \neq i}^{n} h_l^2 - P_{ii} \right)}_{a_i} h_i - \underbrace{\sum_{l=1, l \neq i} h_l P_{li}}_{b_i}, \tag{7}$$

where

$$a_i = \sum_{l=1, l \neq i}^{n} h_l^2 - P_{ii} = \|h\|^2 - h_i^2 - P_{ii}, \text{ and} \tag{8}$$

$$b_i = - \sum_{l=1, l \neq i} h_l P_{li} = h_i P_{ii} - h^T P_{:,i}. \tag{9}$$

If all the variables but h_i are fixed, by the complementary slackness condition (6), the optimal solution h_i^+ will be either 0 or a solution of the equation $\nabla f(h)_i = 0$, that is, a root of $x^3 + a_i x + b_i$. Since the roots of a third-degree polynomial can be computed in closed form, it suffices to first compute these roots and then evaluate $f(h)$ at these roots in order to identify the optimal solution h_i^+. The algorithm based on Cardano's method (see for example [23]) is described as Algorithm 1 and runs in $O(1)$ time. Therefore, given that a_i and b_i are known, h_i^+ can be computed in $O(1)$ operations.

The only inputs of Algorithm 1 are the quantities (8) and (9). However, the variables in (5) are not independent. When h_i is updated to h_i^+, the partial derivative of the other variables, that is, the entries of $\nabla f(h)$, must be updated. For $l \in \{i+1, \ldots, n\}$, we update:

$$a_l \leftarrow a_l + (h_i^+)^2 - h_i^2 \quad \text{and} \quad b_l \leftarrow b_l + P_{li}(h_i^+ - h_i). \tag{10}$$

This means that updating one variable will cost $O(n)$ operations due to the necessary run over the coordinates of h for updating the gradient. (Note that we could also simply evaluate the ith entry of the gradient when updating h_i, which also requires $O(n)$ operations; see Section III.) Algorithm 2 describes one iteration of CD applied on problem (5). In other words, if one wants to find a stationary point of problem (5), Algorithm 2 should be called until convergence, and this would correspond to applying a cyclic coordinate descent method to (5). In lines 2-2, the quantities a_i's and b_i's are precomputed. Because of the product $h^T P_{:,i}$ needed for every b_i, it takes $O(n^2)$ time. Then, from line 2 to line 2, Algorithm 1 is called for every variable and is followed by the updates described by (10). Finally, Algorithm 2 has a computational cost of $O(n^2)$ operations. Note that we cannot expect a lower computational cost since computing the gradient (and in particular the product Ph) requires $O(n^2)$ operations.

Algorithm 1: $x = BestPolynomialRoot(a, b)$.

INPUT: $a \in \mathbb{R}, b \in \mathbb{R}$
OUTPUT: $\arg\min_x \frac{x^4}{4} + \frac{ax^2}{2} + bx$ such that $x \geq 0$.
$\Delta = 4a^3 + 27b^2$
$d = \frac{1}{2}\left(-b + \sqrt{\frac{\Delta}{27}}\right)$
if $\Delta \leq 0$ **then**
  $r = 2\sqrt[3]{|d|}$
  $\theta = \frac{phaseangle(d)}{3}$
  $z^* = 0, y^* = 0$
  for $k = 0 : 2$ **do**
    $z = r\cos\left(\theta + \frac{2k\pi}{3}\right)$
    if $z \geq 0$ **and** $\frac{z^4}{4} + a\frac{z^2}{2} + bz < y^*$ **then**
      $z^* = z$
      $y^* = \frac{z^4}{4} + a\frac{z^2}{2} + bz$
    end if
  end for
  $x = z^*$
else
  $z = \sqrt[3]{d} + \sqrt[3]{\frac{1}{2}\left(-b - \sqrt{\frac{\Delta}{27}}\right)}$
  if $z \geq 0$ **and** $\frac{z^4}{4} + a\frac{z^2}{2} + bz < 0$ **then**
    $x = z$
  else
    $x = 0$
  end if
end if

Algorithm 2: $h = rankoneCDSymNMF(P, h_0)$.

INPUT: $P \in \mathbb{R}^{n\times n}, h_0 \in \mathbb{R}^n$
OUTPUT: $h \in \mathbb{R}^n_+$
$h = h_0$
for $i = 1 : n$ **do**
  $a_i = \|h\|_2^2 - h_i^2 - P_{ii}$
  $b_i = h_i P_{ii} - h^T P_{:,i}$
end for
for $i = 1 : n$ **do**
  $h_i^+ = BestPolynomialRoot(a_i, b_i)$
  for $l > i$ **do**
    $a_l \leftarrow a_l + (h_i^+)^2 - h_i^2$
    $b_l \leftarrow b_l + P_{li}(h_i^+ - h_i)$
  end for
  $h_i = h_i^+$
end for

Algorithm 3: $H = generalCDSymNMF(A, H_0)$.

INPUT: $A \in \mathbb{R}^{n\times n}, H_0 \in \mathbb{R}^{n\times r}$
OUTPUT: $H \in \mathbb{R}^{n\times r}_+$
$H = H_0$
$R = A - HH^T$
while stopping criterion not satisfied **do**
  for $j = 1 : r$ **do**
    $R^{(j)} \leftarrow R + H_{:,j}H_{:,j}^T$
    $H_{:,j} \leftarrow rankoneCDSymNMF(R^{(j)}, H_{:,j})$
    $R \leftarrow R^{(j)} - H_{:,j}H_{:,j}^T$
  end for
end while

B. First exact coordinate descent method for SymNMF

To tackle SymNMF (1), we apply Algorithm 2 on every column of H successively, that is, we apply Algorithm 2 with $h = H(:, j)$ and $P = R^{(j)}$ for $j = 1, \ldots, r$. The procedure is simple to describe, see Algorithm 3 which implements the exact cyclic CD method applied to SymNMF.

One can easily check that Algorithm 3 requires $O(n^2 r)$ operations to update the nr entries of H once:

- In step 4, the full residual matrix $R = A - HH^T$ is precomputed where the product HH^T requires $O(rn^2)$ operations.
- In step 7, the residual matrix $R^{(j)}$ can be computed using the fact that $R^{(j)} = R + H_{:,j}H_{:,j}^T$, which requires $O(n^2)$ operations.
- In step 8, Algorithm 2 is called, and requires $O(n^2)$ operations.
- In step 9, the full residual matrix $R = R^{(j)} - H_{:,j}H_{:,j}^T$ is updated, which requires $O(n^2)$ operations.

Algorithm 3 has some drawbacks. In particular, the heavy computation of the residual matrix R is unpractical for large sparse matrices (see below). In the next sections, we show how to tackle these issues and propose a more efficient CD method for symNMF, applicable to large sparse matrices.

III. Improved Implementation of Algorithm 3

The algorithm for symNMF developed in the previous section (Algorithm 3) is unpractical when the input matrix A is large and sparse; in the sense that although A can be stored in memory, Algorithm 3 will run out of memory for n large. In fact, the residual matrix R with n^2 entries computed in step 4 of Algorithm 3 is in general dense (for example if the entries of H are initialized to some positive entries–see Section IV), even if A is sparse. Sparse matrices usually have $O(n)$ non-zero entries and, when n is large, it is unpractical to store $O(n^2)$ entries (this is for example typical for document data sets where n is of the order of millions).

In this section we re-implement Algorithm 3 in order to avoid the explicit computation of the residual matrix R; see Algorithm 4. While Algorithm 3 runs in $O(rn^2)$ operations per iteration and requires $O(n^2)$ space in memory (whether or not A is sparse), Algorithm 4 runs in $O(r\max(K, nr))$ operations per iteration and requires $O(\max(K, nr))$ space in memory, where K is the number of non-zero entries of A. Hence,

- When A is dense, $K = O(n^2)$ and Algorithm 4 will have the same asymptotic computational cost of $O(rn^2)$ operations per iteration as Algorithm 3. However, it performs better in practice because the exact number of operations is smaller.

- When A is sparse, $K = O(n)$ and Algorithm 4 runs in $O(r^2 n)$ operations per iteration, which is significantly smaller than Algorithm 3 in $O(rn^2)$, so that it will be applicable to very large sparse matrices. In fact, in practice, n can be of the order of millions while r is usually smaller than a hundred. This will be illustrated in Section V for some numerical experiments on text data sets.

In the following, we first assume that A is dense when accounting for the computational cost of Algorithm 4. Then, we show that the computational cost is significantly reduced when A is sparse. Since we want to avoid the computation of the residual (4), reducing the problem into rank-one subproblems solved one after the other is not desirable. To evaluate the gradient of the objective function in (1) for the (i,j)th entry of H, we need to modify the expressions (8) and (9) by substituting $R^{(j)}$ with $A - \sum_{k=1,k\neq j}^{r} H_{:,k}H_{:,k}^T$. We have

$$\begin{aligned}\nabla_{H_{ij}} F(H) = & \nabla_{H_{ij}} \left(\frac{1}{4} ||A - HH^T||_F^2 \right) \\ = & H_{ij}^3 + a_{ij} H_{ij} + b_{ij},\end{aligned}$$

where

$$a_{ij} = \|H_{i,:}\|^2 + \|H_{:,j}\|^2 - 2H_{ij}^2 - A_{ii}, \text{and} \quad (11)$$

$$b_{ij} = H_{i,:}(H^T H)_{:,j} - H_{:,j}^T A_{:,i} - H_{ij}^3 - H_{ij} a_{ij}. \quad (12)$$

The quantities a_{ij} and b_{ij} will no longer be updated during the iterations as in Algorithm 3, but rather computed on the fly before each entry of H is updated. The reason is twofold:

- it avoids storing two n-by-r matrices, and
- the updates of the b_{ij}'s, as done in (10), cannot be performed in $O(n)$ operations without the matrix $R^{(j)}$.

However, in order to minimize the computational cost, the following quantities will be precomputed and updated during the course of the iterations:

- $\|H_{i,:}\|^2$ for all i and $\|H_{:,j}\|^2$ for all j: if the values of $\|H_{i,:}\|^2$ and $\|H_{:,j}\|^2$ are available, a_{ij} can be computed in $O(1)$; see (11). Moreover, when H_{ij} is updated to its optimal value H_{ij}^+, we only need to update $\|H_{i,:}\|^2$ and $\|H_{:,j}\|^2$ which can also be done in $O(1)$:

$$\|H_{i,:}\|^2 \leftarrow \|H_{i,:}\|^2 + (H_{ij}^+)^2 - H_{ij}^2, \quad (13)$$

$$\|H_{:,j}\|^2 \leftarrow \|H_{:,j}\|^2 + (H_{ij}^+)^2 - H_{ij}^2. \quad (14)$$

  Therefore, pre-computing the $\|H_{i,:}\|^2$'s and $\|H_{:,j}\|^2$'s, which require $O(rn)$ operations, allows us to compute the a_{ij}'s in $O(1)$.
- The r-by-r matrix $H^T H$: by maintaining $H^T H$, computing $H_{i,:}(H^T H)_{:,j}$ requires $O(r)$ operations. Moreover, when the (i,j)th entry of H is updated to H_{ij}^+, updating $H^T H$ requires $O(r)$ operations:

$$(H^T H)_{jk} \leftarrow (H^T H)_{jk} - H_{ik}(H_{ij}^+ - H_{ij}), \quad k = 1, \ldots, r. \quad (15)$$

To compute b_{ij}, we also need to perform the product $H_{:,j}^T A_{:,i}$; see (12). This requires $O(n)$ operations, which cannot be avoided and is the most expensive part of the algorithm.

Algorithm 4: $H = cyclicCDSymNMF(A, H_0)$.

1: INPUT: $A \in \mathbb{R}^{n\times n}, H_0 \in \mathbb{R}^{n\times r}$
2: OUTPUT: $H \in \mathbb{R}^{n\times r}$
3: $H = H_0$
4: **for** $j = 1 : r$ **do**
5: $\quad C_j = \|H_{:,j}\|^2$
6: **end for**
7: **for** $i = 1 : n$ **do**
8: $\quad L_i = \|H_{i,:}\|^2$
9: **end for**
10: $D = H^T H$
11: **while** stopping criterion not satisfied **do**
12: $\quad$ **for** $j = 1 : r$ **do**
13: $\quad\quad$ **for** $i = 1 : n$ **do**
14: $\quad\quad\quad a_{ij} \leftarrow C_j + L_i - 2H_{ij}^2 - A_{ii}$
15: $\quad\quad\quad b_{ij} \leftarrow H_{i,:}^T (D)_{j,:} - H_{:,j}^T A_{:,i} - H_{ij}^3 - H_{ij} a_{ij}$
16: $\quad\quad\quad H_{ij}^+ \leftarrow BestPolynomialRoot(a_{ij}, b_{ij})$
17: $\quad\quad\quad C_j \leftarrow C_j + (H_{ij}^+)^2 - H_{ij}^2$
18: $\quad\quad\quad L_i \leftarrow L_i + (H_{ij}^+)^2 - H_{ij}^2$
19: $\quad\quad\quad D_{j,:} \leftarrow D_{j,:} - H_{i,:}(H_{ij}^+ - H_{ij})$
20: $\quad\quad\quad D_{:,j} \leftarrow D_{j,:}$
21: $\quad\quad$ **end for**
22: $\quad$ **end for**
23: **end while**

In summary, by precomputing the quantities $\|H_{i,:}\|^2$, $\|H_{:,j}\|^2$ and $H^T H$, it is possible to apply one iteration of CD over the nr variables in $O(n^2 r)$ operations. The computational cost is the same as in Algorithm 3, in the dense case, but no residual matrix is computed; see Algorithm 4.

From line 4 to line 10, the precomputations are performed in $O(nr^2)$ time where computing $H^T H$ is the most expensive part. Then the two loops iterate over all the entries to update each variable once. Computing b_{ij} (in line 4) is the bottleneck of the CD scheme as it is the only part in the two loops which requires $O(n)$ time. However, when the matrix A is sparse, the cost of computing $H_{:,j}^T A_{:,i}$ for all i, that is computing $H_{:,j}^T A$, drops to $O(K)$ where K is the number of nonzero entries in A. Taking into account the term $H_{i,:}(H^T H)_{j,:}$ to compute b_{ij} that requires $O(r)$ operations, we have that Algorithm 4 requires $O(r \max(K, nr))$ operations per iteration.

IV. Initialization and Convergence

In this section, we discuss initialization and convergence of Algorithm 4. We also provide a small modification for Algorithm 4 to perform better (especially when random initialization is used).

a) Initialization: In most previous works, the matrix H is initialized randomly, using the uniform distribution in the interval [0, 1] for each entry of H [7]. Note that, in practice, to obtain an unbiased initial point, the matrix H should be multiplied by

a constant β^* such that

$$\beta^* = \arg\min_{\beta \geq 0} ||A - (\beta H_0)(\beta H_0)^T||_F$$
$$= \sqrt{\frac{\langle A, H_0 H_0^T \rangle}{\langle H_0 H_0^T, H_0 H_0^T \rangle}} = \sqrt{\frac{\langle AH_0, H_0 \rangle}{||H_0^T H_0||_F^2}}. \quad (16)$$

This allows the initial approximation $H_0 H_0^T$ to be well scaled compared to A. When using such an initialization, we observed that using random shuffling of the columns of H before each iteration (that is, optimizing the columns of H in a different order each time we run Algorithm 4) performs in general much better; see Section V.

Remark 1 (Other Heuristics to Accelerate Coordinate Descent Methods): sDuring the course of our research, we have tried several heuristics to accelerate Algorithm 4, including three of the most popular strategies:

- *Gauss-Southwell strategies*. We have updated the variables by ordering them according to some criterion (namely, the decrease of the objective function, and the magnitude of the corresponding entry of the gradient).
- *Variable selection*. Instead of optimizing all variables at each step, we carefully selected a subset of the variables to optimize at each iteration (again using a criterion based on the decrease of the objective function or the magnitude of the corresponding entry of the gradient).
- *Random shuffling*. We have shuffled randomly the order in which the variables are updated in each column. This strategy was shown to be superior in several context, although a theoretical understanding of this phenomenon remains elusive [21].

However, these heuristics (and combinations of them) would not improve significantly the effectiveness of Algorithm 4 hence we do not present them here.

Random initialization might not seem very reasonable, especially for our CD scheme. In fact, at the first step of our CD method, the optimal values of the entries of the first column $H_{:,1}$ of H are computed sequentially, trying to solve

$$\min_{H_{:,1} \geq 0} ||R^{(1)} - H_{:,1} H_{:,1}^T||_F^2 \text{ with } R^{(1)} = A - \sum_{k=2}^{r} H_{:,k} H_{:,k}^T.$$

Hence we are trying to approximate a matrix $R^{(1)}$ which is the difference between A and a randomly generated matrix $\sum_{k=2}^{r} H_{:,k} H_{:,k}^T$: this does not really make sense. In fact, we are trying to approximate a matrix which is highly perturbed with a randomly generated matrix.

It would arguably make more sense to initialize H at zero, so that, when optimizing over the entries of $H_{:,1}$ at the first step, we only try to approximate the matrix A itself. It turns out that this simple strategy allows to obtain a faster initial convergence than the random initialization strategy. However, we observe the following: this solution tends to have a very particular structure where the first factor is dense and the next ones are sparser. The explanation is that the first factor is given more importance since it is optimized first hence it will be close to the best rank-one approximation of A, which is in general positive (if A is irreducible, by Perron-Frobenius and Eckart-Young theorems). Hence initializing H at zero tends to produce unbalanced factors. However, this might be desirable in some cases as the next factors are in general significantly sparser than with random initialization. To illustrate this, let us perform the following numerical experiment: we use the CBCL face data set (see Section V) that contains 2429 facial images, 19 by 19 pixels each. Let us construct the nonnegative matrix $X \in \mathbb{R}^{361 \times 2429}$ where each column is a vectorized image. Then, we construct the matrix $A = XX^T \in \mathbb{R}^{361 \times 361}$ that contains the similarities between the pixel intensities among the facial images. Hence symNMF of A will provide us with a matrix H where each column of H corresponds to a 'cluster' of pixels sharing some similarities. Fig. 1 shows the columns of H obtained (after reshaping them as images) with zero initialization (left) and random initialization (right) with $r = 49$ as in [1]. We observe that the solutions are very different, although the relative approximation error $||A - HH^T||_F / ||A||_F$ are similar (6.2% for zero initialization vs. 7.5% for random initialization, after 2000 iterations). Depending on the application at hand, one of the two solutions might be more desirable: for example, for the CBCL data set, it seems that the solution obtained with zero initialization is more easily interpretable as facial features, while with the random initialization it can be interpreted as average/mean faces.

This example also illustrates the sensitivity of Algorithm 4 to initialization: different initializations can lead to very different solutions. This is an unavoidable feature for any algorithm trying to find a good solution to an NP-hard problem at a relatively low computational cost.

Finally, we would like to point out that the ability to initialize our algorithm at zero is a very nice feature. In fact, since $H = 0$ is a (first-order) stationary point of (1), *this shows that our coordinate descent method can escape some first-order stationary points, because it uses higher-order information*. For example, any gradient-based method cannot be initialized at zero (the gradient is 0), also the ANLS-based algorithm from [7] cannot escape from zero.

b) Convergence: By construction, the objective function is nonincreasing under the updates of Algorithm 4 while it is bounded from below. Moreover, since our initial estimate H_0 is initially scaled (16), we have $||A - H_0 H_0^T||_F \leq ||A||_F$ and therefore any iterate H of Algorithm 4 satisfies

$$||HH^T||_F - ||A||_F \leq ||A - HH^T||_F \leq ||A - H_0 H_0^T||_F \leq ||A||_F.$$

Since $H \geq 0$, we have for all k

$$||H_{:k} H_{:k}^T||_F \leq ||\sum_{k=1}^{r} H_{:k} H_{:k}^T||_F = ||HH^T||_F,$$

which implies that $||H_{:k}||_2 \leq \sqrt{2||A||_F}$ for all k hence all iterates of Algorithm 4 belong in a compact set. Therefore, Algorithm 4 generates a converging subsequence (Bolzano-Weierstrass theorem). (Note that, even if the initial iterate is not scaled, all iterates belong to a compact set, replacing $2||A||_F$ by $||A||_F + ||A - H_0 H_0^T||_F$.)

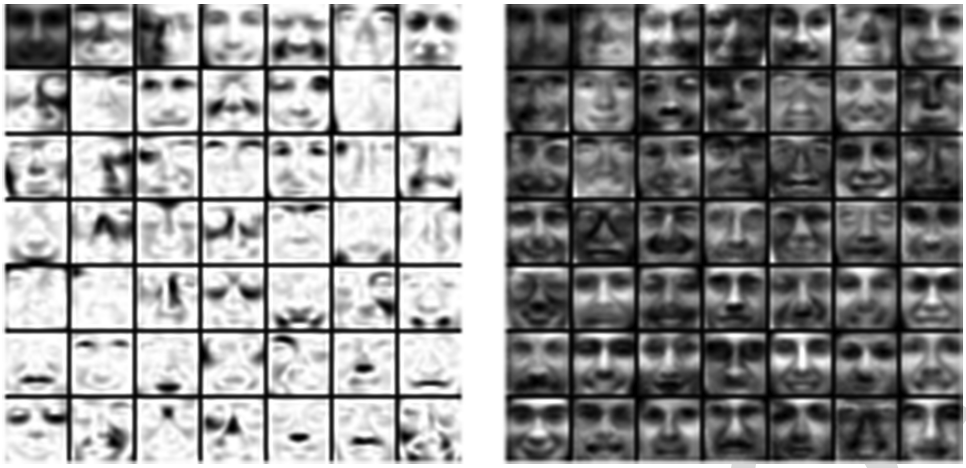


Fig. 1. Comparison of the basis elements obtained with symNMF on the CBCL data set ($r = 49$) with (left) zero initialization and (right) random initialization.

Unfortunately, in its current form, it is difficult to prove convergence of our algorithm to a stationary point. In fact, to guarantee the convergence of an exact cyclic coordinate method to a stationary point, three sufficient conditions are (i) the objective function is continuously differentiable over the feasible set, (ii) the sets over which the blocks of variables are updated are compact as well as convex,[1] and (iii) the minimum computed at each iteration for a given block of variables is uniquely attained; see Prop. 2.7.1 in [24], [25]. Conditions (i-ii) are met for Algorithm 4. Unfortunately, it is not necessarily the case that the minimizer of a fourth order polynomial is unique. (Note however that for a randomly generated polynomial, this happens with probability 0. We have observed numerically that this in fact never happens in our numerical experiments, although there are counter examples.)

A possible way to obtain convergence is to apply the maximum block improvement (MBI) method, that is, at each iteration, only update the variable that leads to the largest decrease of the objective function [26]. Although this is theoretically appealing, this makes the algorithm computationally much more expensive hence much slower in practice. (A possible fix is to use MBI not for every iteration, but every Tth iteration for some fixed T.)

Although the solution of symNMF might not be unique and stationary points might not be isolated, we have always observed in our numerical experiments that the sequence of iterates generated by Algorithm 4 converged to a unique limit point. In that case, we can prove that this limit point is a stationary point.

Proposition 1: Let $(H_{(0)}, H_{(1)}, \ldots)$ be a sequence of iterates generated by Algorithm 4. If that sequence converges to a unique accumulation point, it is a stationary point of symNMF (1).

Proof: This proof follows similar arguments as the proof of convergence of exact cyclic CD for NMF [19]. Let $\bar{H}$ be the accumulation point of the sequence $(H_{(0)}, H_{(1)}, \ldots)$, that is,

$$\lim_{k\to\infty} H_{(k)} \quad = \quad \bar{H}.$$

[1] An alternative assumption to the condition (ii) under which the same result holds is when the function is monotonically nonincreasing in the interval from one iterate to the next [24].

Note that, by construction,

$$F(H_{(1)}) \geq F(H_{(2)}) \geq \cdots \geq F(\bar{H}).$$

Note also that we consider that only one variable has been updated between $H_{(k+1)}$ and $H_{(k)}$.

Assume $\bar{H}$ is not a stationary point of (1): therefore, there exists (i, j) such that

- $\bar{H}_{i,j} = 0$ and $\nabla F(\bar{H})_{i,j} < 0$, or
- $\bar{H}_{i,j} > 0$ and $\nabla F(\bar{H})_{i,j} \neq 0$.

In both cases, since F is smooth, there exists $p \neq 0$ such that

$$F(\bar{H} + pE^{ij}) = F(\bar{H}) - \epsilon < F(\bar{H}),$$

for some $\epsilon > 0$, where E^{ij} is the matrix of all zeros except at the (i, j)th entry where it is equal to one and $\bar{H} + pE^{ij} \geq 0$.

Let us define $(H_{(n_0)}, H_{(n_1)}, \ldots)$ a subsequence of $(H_{(0)}, H_{(1)}, \ldots)$ as follows: $H_{(n_k)}$ is the iterate for which the (i, j)th entry is updated to obtain $H_{(n_k+1)}$. Since Algorithm 4 updates the entries of H column by column, we have $n_k = (j-1)n + i - 1 + nrk$ for $k = 0, 1, \ldots$.

By continuity of F and the convergence of the sequence $H_{(n_k)}$, there exists K sufficiently large so that for all $k > K$:

$$F(H_{(n_k)} + pE^{ij}) \;\leq\; F(\bar{H}) - \frac{\epsilon}{2}. \tag{17}$$

In fact, the continuity of F implies that for all $\xi > 0$, there exists $\delta > 0$ sufficiently small such that $||\bar{H} - H_{(n_k)}||_2 < \delta \Rightarrow |F(\bar{H}) - F(H_{(n_k)})| < \xi$. It suffices to choose n_k sufficiently large so that δ is sufficiently small (since $H_{(n_k)}$ converges to $\bar{H}$) for the value $\xi = \epsilon/2$.

Let us flip the sign of (17) and add $F(H_{(n_k)})$ on both sides to obtain

$$F(H_{(n_k)}) - F(H_{(n_k)} + pE^{ij}) \;\geq\; F(H_{(n_k)}) - F(\bar{H}) + \frac{\epsilon}{2}.$$

By construction of the subsequence, the (i, j)th entry of $H_{(n_k)}$ is updated first (the other entries are updated afterward) to obtain $H_{(n_{k+1})}$ which implies that

$$F(H_{(n_{k+1})}) \leq F(H_{(n_k+1)}) \leq F(H_{(n_k)} + pE^{ij})$$

TABLE I
IMAGE DATASETS

Data	# pixels	m	n
ORL[1]	112×92	10304	400
Umist[2]	112×92	10304	575
CBCL[3]	19×19	361	2429
Frey[2]	28×20	560	1965

[1]http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html
[2]http://www.cs.toronto.edu/roweis/data.html
[3]http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html

hence

$$\begin{aligned} F(H_{(n_k)}) - F(H_{(n_{k+1})}) &\geq F(H_{(n_k)}) - F(H_{(n_k)} + pE^{ij}) \\ &\geq F(H_{(n_k)}) - F(\bar{H}) + \frac{\epsilon}{2} \\ &\geq \frac{\epsilon}{2}, \end{aligned}$$

since $F(\bar{H}) \leq F(H_{(n_k)})$. We therefore have that for all $k > K$,

$$F(H_{(n_{k+1})}) \leq F(H_{(n_k)}) - \frac{\epsilon}{2},$$

a contradiction since F is bounded below. ■

Note that Proposition 1 is useful in practice since it can easily be checked whether Algorithm 4 converges to a unique accumulation point, plotting for example the norm between the different iterates.

V. NUMERICAL RESULTS

This section shows the effectiveness of Algorithm 4 on several data sets compared to the state-of-the-art techniques. It is organized as follows. In Section V-A, we describe the real data sets and, in Section V-B, the tested symNMF algorithms. In Section V-C, we describe the settings we use to compare the symNMF algorithms. In Section V-D, we provide and discuss the experimental results.

A. Data Sets

We will use exactly the same data sets as in [18]. Because of space limitation, we only give the results for one value of the factorization rank r, more numerical experiments are available on the arXiv version of this paper [27]. In [18], authors use four dense data sets and six sparse data sets to compare several NMF algorithms. In this section, we use these data sets to generate similarity matrices A on which we compare the different symNMF algorithms. Given a nonnegative data set $X \in \mathbb{R}_+^{m \times n}$, we construct the symmetric similarity matrix $A = X^T X \in \mathbb{R}_+^{n \times n}$, so that the entries of A are equal to the inner products between data points. Table I summarizes the dense data sets, corresponding to widely used facial images in the data mining community. Table II summarizes the characteristics of the different sparse data sets, corresponding to document datasets and described in details in [28].

TABLE II
TEXT MINING DATA SETS (SPARSITY IS GIVEN AS THE PERCENTAGE OF ZEROS)

Data	m	n	#nonzero	sparsity X	sparsity $X^T X$
classic	7094	41681	223839	99.92	99.50
sports	8580	14870	1091723	99.14	84.51
reviews	4069	18483	758635	98.99	84.24
hitech	2301	10080	331373	98.57	80.32
ohscal	11162	11465	674365	99.47	91.58
la1	3204	31472	484024	99.52	95.72

B. Tested symNMF Algorithms

We compare the following algorithms

1) (Newton) This is the Newton-like method from [7].
2) (ANLS) This is the method based on the ANLS method for NMF adding the penalty $||W - H||_F^2$ in the objective function (see Introduction) from [7]. Note that ANLS has the drawback to depend on a parameter that is nontrivial to tune, namely, the penalty parameter for the term $||W - H||_F^2$ in the objective function (we used the default tuning strategy recommended by the authors).
3) (tSVD) This method, recently introduced in [29], first computes the rank-r truncated SVD of $A \approx A_r = U_r \Sigma_r U_r^T$ where U_r contains the first r singular vectors of A and Σ_r is the r-by-r diagonal matrix containing the first r singular values of A on its diagonal. Then, instead of solving (1), the authors solve a 'closeby' optimization problem replacing A with A_r

$$\min_{H \geq 0} ||A_r - HH^T||_F.$$

Once the truncated SVD is computed, each iteration of this method is extremely cheap as the main computational cost is in a matrix-matrix product $B_r Q$, where $B_r = U_r \Sigma_r^{1/2}$ and Q is an r-by-r rotation matrix, which can be computed in $O(nr^2)$ operations. Note also that they use the initialization $H_0 = \max(0, B_r)$—we flipped the signs of the columns of U_r to maximize the ℓ_2 norm of the nonnegative part [30].
4) (BetaSNMF) This algorithm is presented in ([31, Algorithm 4], and is based on multiplicative updates (similarly as for the original NMF algorithm proposed by Lee and Seung [32]). Note that we have also implemented the multiplicative update rules from [33] (and already derived in [3]). However, we do not report the numerical results here because it was outperformed by BetaSNMF in all our numerical experiments, an observation already made in [31].
5) (CD-X-Y) This is Algorithm 4. X is either 'Cyclic' or 'Shuffle' and indicates whether the columns of H are optimized in a cyclic way or if they are shuffled randomly before each iteration. Y is for the initialization: Y is 'rand' for random initialization and is '0' for zero initialization; see Section IV for more details. Hence, we will compare four variants of Algorithm 4: CD-Cyclic-0, CD-Shuffle-0, CD-Cyclic-Rand and CD-Shuffle-Rand.

Because Algorithm 4 requires to perform many loops (nr at each step), Matlab is not a well-suited language. Therefore, we have developed a C implementation, that can be called from Matlab (using Mex files). Note that the algorithms above are better suited for Matlab since the main computational cost

resides in matrix-matrix products, and in solving linear systems of equations (for ANLS and Newton).

Newton and ANLS are both available from http://math.ucla.edu/dakuang/, while we have implemented tSVD and BetaSNMF ourselves.

For all algorithms using random initializations for the matrix H, we used the same initial matrices. Note however that, in all the figures presented in this section, we will display the error after the first iteration, which is the reason why the curves do not start at the same value.

C. Experimental Setup

In order to compare for the average performance of the different algorithms, we denote $e_{\min}$ the smallest error obtained by all algorithms over all initializations, and define

$$E(t) = \frac{e(t) - e_{\min}}{||A||_F - e_{\min}}, \tag{18}$$

where $e(t)$ is the error $||A - HH^T||_F$ achieved by an algorithm for a given initialization within t seconds (and hence $e(0) = ||A - H_0 H_0^T||_F$ where H_0 is the initialization). The quantity $E(t)$ is therefore a normalized measure of the evolution of the objective function of a given algorithm on a given data set.

The advantage of this measure is that it separates better the different algorithms, when using a log scale, since it goes to zero for the best algorithm (except for algorithms that are initialized randomly as we will report the average value of $E(t)$ over several random initializations; see below). We would like to stress out that the measure $E(t)$ from (18) has to be interpreted with care. In fact, an algorithm for which $E(t)$ converges to zero simply means that it is the algorithm able to find the best solution among all algorithms (in other words, to identify a region with a better local minima). In fact, the different algorithms are initialized with different initial points: in particular, tSVD uses an SVD-based initialization. It does not necessarily mean that it converges the fastest: to compare (initial) convergence, one has to look at the values $E(t)$ for t small. However, the measure $E(t)$ allows to better visualize the different algorithms. For example, displaying the relative error $||A - HH^T||/||A||_F$ allows to compare the initial convergence, but then the errors for all algorithms tend to converge at similar values and it is difficult to identify visually which one converges to the best solution.

For the algorithms using random initialization (namely, Newton, ANLS, CD-Cyclic-Rand and CD-Shuffle-Rand), we will run the algorithms 10 times and report the average value of $E(t)$. For all data sets, we will run each algorithm for 100 seconds, or for longer to allow the CD-based approaches to perform at least 100 iterations.

All tests are performed using Matlab on a PC Intel CORE i5-4570 CPU @3.2GHz $\times$ 4, with 7.7G RAM. The codes are available online from https://sites.google.com/site/nicolasgillis/.

Remark 2 (Computation of the Error): Note that to compute $||A - HH^T||_F$, one should not compute HH^T explicitly (especially if A is sparse) and use instead

$$\begin{aligned} ||A - HH^T||_F^2 &= ||A||_F^2 - 2\langle A, HH^T\rangle + ||HH^T||_F^2 \\ &= ||A||_F^2 - 2\langle AH, H\rangle + ||H^T H||_F^2. \end{aligned}$$

D. Comparison

We now compare the different symNMF algorithms listed in Section V-B according to the measure given in (18) on the data sets described in Section V-B, and on synthetic data sets.

1) Real Data Sets: We start with the real data sets.

a) Dense Image Data Sets: Fig. 2 displays the results for the dense real data sets. Table III gives the number of iterations performed by each algorithm within the 500 seconds, and Table IV the final relative error $||A - HH^T||/||A||_F$ in percent.

We observe the following:

- In all cases, tSVD performs best and is able to generate the solution with the smallest objective function value among all algorithms. This might be a bit surprising since it works only with an approximation of the original data: it appears that for these real dense data sets, this approximation can be computed efficiently and allows tSVD to converge extremely fast to a very good solution.
  One of the reasons tSVD is so effective is because each iteration is n times cheaper (once the truncated SVD is computed) hence it can perform many more iterations; see Table III. Another crucial reason is that image data sets can be very well approximated by low-rank matrices (see Section V-D2 for a confirmation of this behavior). Therefore, for images, tSVD is the best method to use as it provides a very good solution extremely fast.
- When it comes to initial convergence, CD-Cyclic-0 and CD-Shuffle-0 perform best: they are able to generate very fast a good solution. In all cases, they are the fastest to generate a solution at a relative error of 1% of the final solution of tSVD. Moreover, the fact that tSVD does not generate any solution as long as the truncated SVD is not computed could be critical for larger data sets. For example, for CBCL with $n = 2429$ and $r = 60$, the truncated SVD takes about 6 seconds to compute while, in the mean time, CD-Cyclic-0 and CD-Shuffle-0 generate a solution with relative error of 0.3% from the final solution obtained by tSVD after 500 seconds.
- For these data sets, CD-Cyclic-0 and CD-Shuffle-0 perform exactly the same: for the zero initialization, it seems that shuffling the columns of H does not play a crucial role.
- When initialized randomly, we observe that the CD method performs significantly better with random shuffling. Moreover, CD-Shuffle-Rand converges initially slower than CD-Shuffle-0 but is often able to converge to a better solution; in particular for the ORL and Umistim data sets.
- Newton converges slowly, the main reason being that each iteration is very costly, namely $O(n^3 r)$ operations.

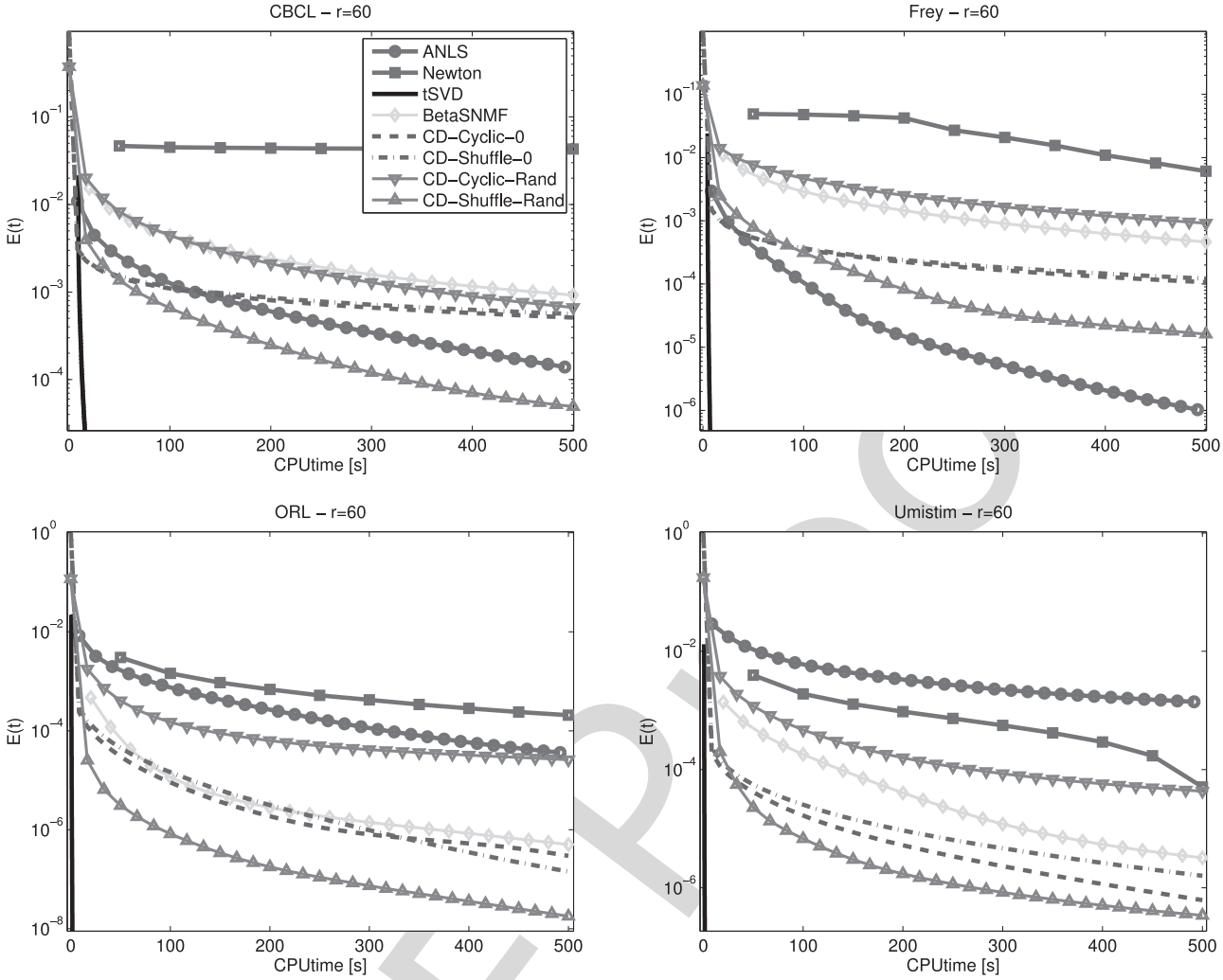


Fig. 2. Evolution of the measure (18) of the different symNMF algorithms on the dense real data sets for $r = 60$.

TABLE III
AVERAGE NUMBER OF ITERATIONS PERFORMED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE DENSE REAL DATA SETS

$r = 60$	ANLS	Newton	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
ORL	56995	14377	204960	234400	17738	15741	16235	16062
Umist	33555	8993	225968	132830	9193	8756	8951	8955
CBCL	3965	21	93252	10254	726	722	775	784
Frey	5692	456	173030	15465	1303	1290	1213	1216

TABLE IV
AVERAGE RELATIVE ERROR IN PERCENT ($100 * ||A - HH^T||_F / ||A||_F$) OF THE FINAL SOLUTION OBTAINED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE DENSE REAL DATA SETS. FOR ALGORITHMS BASED ON RANDOM INITIALIZATIONS, THE STANDARD DEVIATION IS GIVEN

$r = 60$	ANLS	Newton	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
ORL	0.144 ± 4e-4	0.168	**0.14**	0.141 ± 4e-5	0.141	0.141	0.143 ± 4e-4	**0.14** ± 4e-6
Umist	0.165 ± 6e-3	0.098	**0.04**	0.041 ± 8e-5	0.041	0.041	0.045 ± 3e-4	0.041 ± 3e-5
CBCL	0.059 ± 4e-4	4.34	**0.046**	0.138 ± 1e-3	0.097	0.102	0.112 ± 7e-3	0.051 ± 6e-4
Frey	0.057 ± 6e-5	0.66	**0.056**	0.103 ± 5e-4	0.067	0.069	0.148 ± 2e-3	0.058 ± 2e-4

- ANLS performs relatively well: it never converges initially faster than CD-based approaches but is able to generate a better final solution for the Frey data set.
- BetaSNMF does not perform well on these data sets compared to tSVD and CD methods, although performing better than Newton and 2 out of 4 times better than ANLS.
- For algorithms based on random initializations, the standard deviation between several runs is rather small, illustrating the fact that these algorithms converge to solutions with similar final errors.

Conclusion: for image data sets, tSVD performs the best. However, CD-Cyclic-0 allows a very fast initial

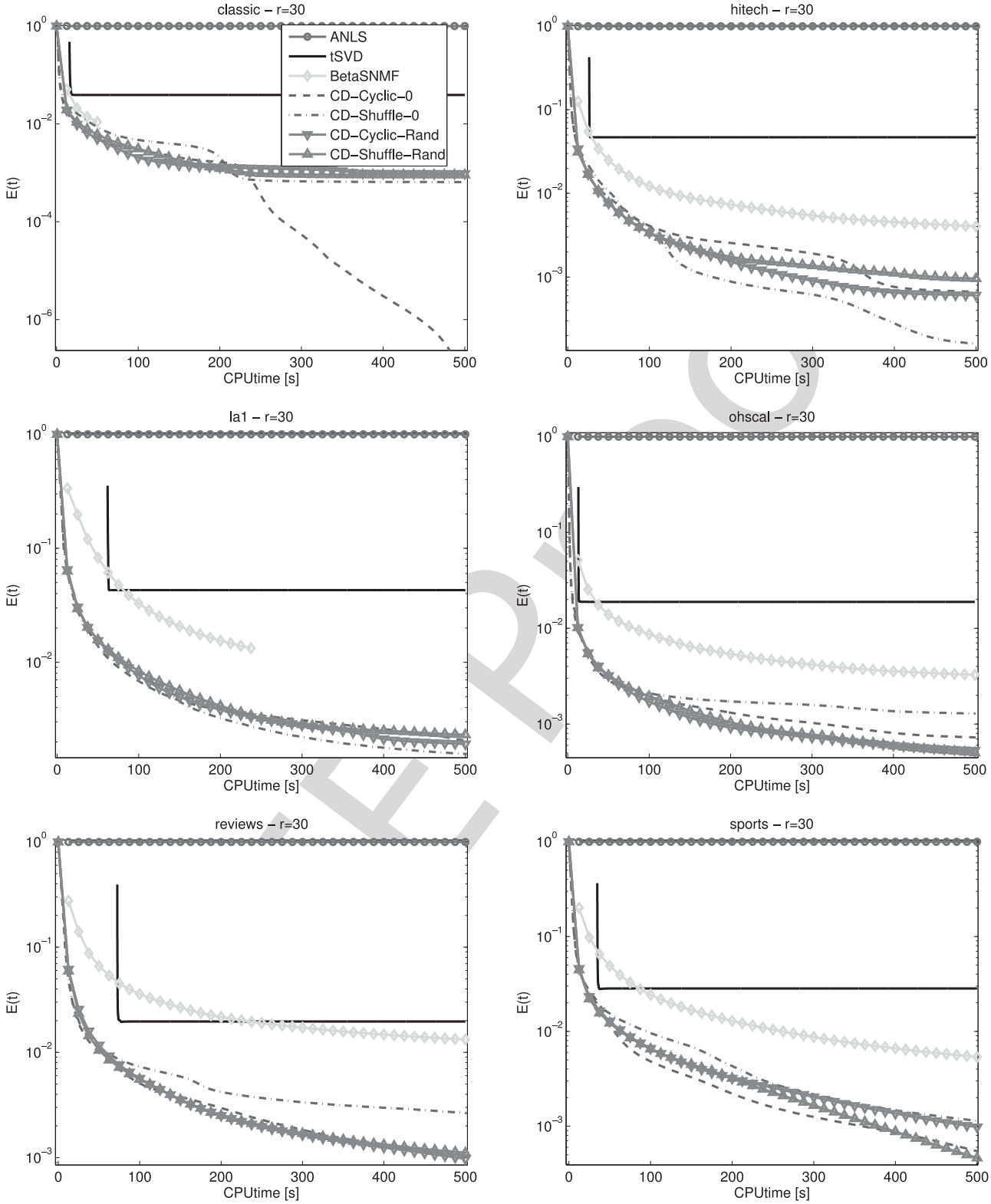


Fig. 3. Evolution of the measure (18) of the different symNMF algorithms on real sparse data sets for $r = 30$.

convergence and can be used to obtain very quickly a good solution.

b) Sparse document data sets: Fig. 3 displays the results for the real sparse data sets. Table V gives the number of iterations performed by each algorithm within the 500 seconds, and Table VI the final relative error $||A - HH^T||/||A||_F$ in percent.

It is interesting to note that, for some data sets (namely, la1 and reviews), computing the truncated SVD of A is not possible with Matlab within 60 seconds hence tSVD is not able to return

TABLE V
AVERAGE NUMBER OF ITERATIONS PERFORMED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE SPARSE REAL DATA SETS

$r = 30$	ANLS	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
classic	550	21345	163	389	390	386	386
sports	254	57358	540	170	171	163	163
reviews	162	41519	353	114	114	114	114
hitech	458	81975	898	281	282	284	284
ohscal	680	75935	1462	495	494	494	494
la1	154	24667	163	126	126	127	127

TABLE VI
AVERAGE RELATIVE ERROR IN PERCENT ($100 * ||A - HH^T||_F / ||A||_F$) OF THE FINAL SOLUTION OBTAINED BY EACH ALGORITHM WITHIN 500 SECONDS FOR THE SPARSE REAL DATA SETS. FOR ALGORITHMS BASED ON RANDOM INITIALIZATIONS, THE STANDARD DEVIATION IS GIVEN

$r = 30$	ANLS	tSVD	BetaSNMF	CD-Cyc.-0	CD-Shuf.-0	CD-Cyc.-Rand	CD-Shuf.-Rand
classic	99.9 ± 6e-4	39.8	38 ± 0.14	**37.3**	37.4	37.4 ± 0.03	37.4 ± 0.04
sports	99.9 ± 1e-3	19.2	17.3 ± 0.11	**16.9**	**16.9**	**16.9** ± 0.04	**16.9** ± 0.04
reviews	99.9 ± 1e-3	17.1	16.5 ± 0.16	**15.5**	15.7	**15.5** ± 0.05	**15.5** ± 0.03
hitech	99.5 ± 3e-3	33.3	30.3 ± 0.1	**30**	**30**	30.1 ± 0.03	30.1 ± 0.03
ohscal	99.95 ± 6e-4	22.2	20.9 ± 0.06	**20.7**	20.8	**20.7** ± 0.04	**20.7** ± 0.03
la1	99.9 ± 8e-4	34	31.9 ± 0.2	31.2	**31.1**	31.2 ± 0.07	31.2 ± 0.05

any solution before that time; see Remark 3 for a discussion. Moreover, Newton is not displayed because it is not designed for sparse matrices and runs out of memory [7].

We observe the following:

- tSVD performs very poorly. The reason is twofold: (1) the truncated SVD is very expensive to compute and (2) sparse matrices are usually not close to being low-rank hence tSVD converges to a very poor solution (see Section V-D2 for a confirmation of this behavior).
- ANLS performs very poorly and is not able to generate a good solution. In fact, it has difficulties to decrease the objective function (on the figures, it seems it does not decrease, but it actually decreases very slowly).
- BetaSNMF performs better than ANLS but does not compete with CD methods. (Note that, for the classic and la1 data sets, BetaSNMF was stopped prematurely because there was a division by zero which could have been avoided but we have strictly used the description of Algorithm 4 in [31]).
- All CD-based approaches are very effective and perform similarly. It seems that, in these cases, nor the initialization nor the order in which the columns of H are updated plays a significant role.

  In fact, for algorithms initialized randomly, Fig. 3 reports the average over 10 runs but, on average, random initialization performs similarly as the initialization with zero.

  In one case (classic data set), CD-Cyclic-0 is able to generate a better final solution.

Conclusion: for sparse document data sets, CD-based approaches outperform significantly the other tested methods.

Remark 3 (SVD Computation in tSVD): It has to be noted that, in our numerical experiments, the matrix A is constructed using the formula $A = X^T X$, where the columns of the matrix X are the data points. In other words, we use the simple similarity measure $y^T z$ between two data points y and z. In that case, the SVD of A can be obtained from the SVD of X, hence can be made (i) more efficient (when X has more columns than rows, that is, $m \ll n$), and (ii) numerically more accurate (because the condition number of $X^T X$ is equal to the square of that of X); see, e.g., [34, Lecture 31]. Moreover, in case of sparse data, this avoids the fill-in, as observed in Table II where $X^T X$ is denser than X. Therefore, in this particular situation when $A = X^T X$ and X is sparse and/or $m \ll n$, it is much better to compute the SVD of A based on the SVD of X. Table VII gives the computational time in both cases. In this particular scenario, it would make sense to use tSVD as an initialization procedure for CD methods to obtain rapidly a good initial iterate. However, looking at Fig. 3 and Table VI indicates that this would not necessarily be advantageous for the CD-based methods in all cases. For example, for the classic data set, tSVD would achieve a relative error of 39.8% within about 6 seconds while CD methods obtain a similar relative error within that computing time. For the hitech data set however, this would be rather helpful since tSVD would only take about 1 second to obtain a relative error of 33.3% while CD methods require about 9 seconds to do so.

TABLE VII
COMPUTATIONAL TIME REQUIRED TO COMPUTE THE RANK-30 TRUNCATED SVD OF X AND $X^T X$ USING MATLAB

`svds(.,30)`	classic	hitech	la1	ohscal	reviews	sports
`X'*X`	17.14	18.54	63.33	15	67.32	31.77
`X`	5.55	0.82	3.08	2.87	1.39	2.98

However, the goal of this paper is to provide an efficient algorithm for the general symNMF problem, without assuming any particular structure on the matrix A (in practice the similarity measure between data points is usually not simply their inner product). Therefore, we have not assumed that the matrix A had this particular structure and only provide numerical comparison in that case.

Remark 4 (Low-Rank Models for Full-Rank Matrices): Although sparse data sets are usually not low rank, it still makes sense to try to find a low-rank structure that is close to a given data set, as this often allows to extract some pertinent information. In particular, in document classification and clustering, low-rank models have proven to be extremely useful; see the discussion in the Introduction and the references

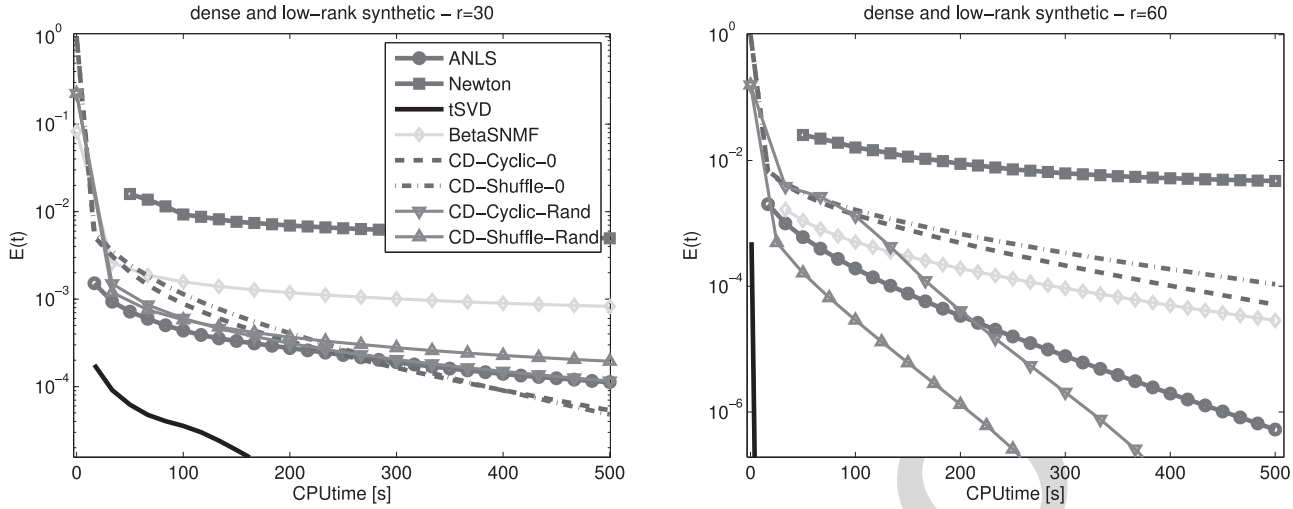


Fig. 4. Evolution of the measure (18) of the different symNMF algorithms on dense and low-rank synthetic data sets for $r = 30$ (left) and $r = 60$ (right).

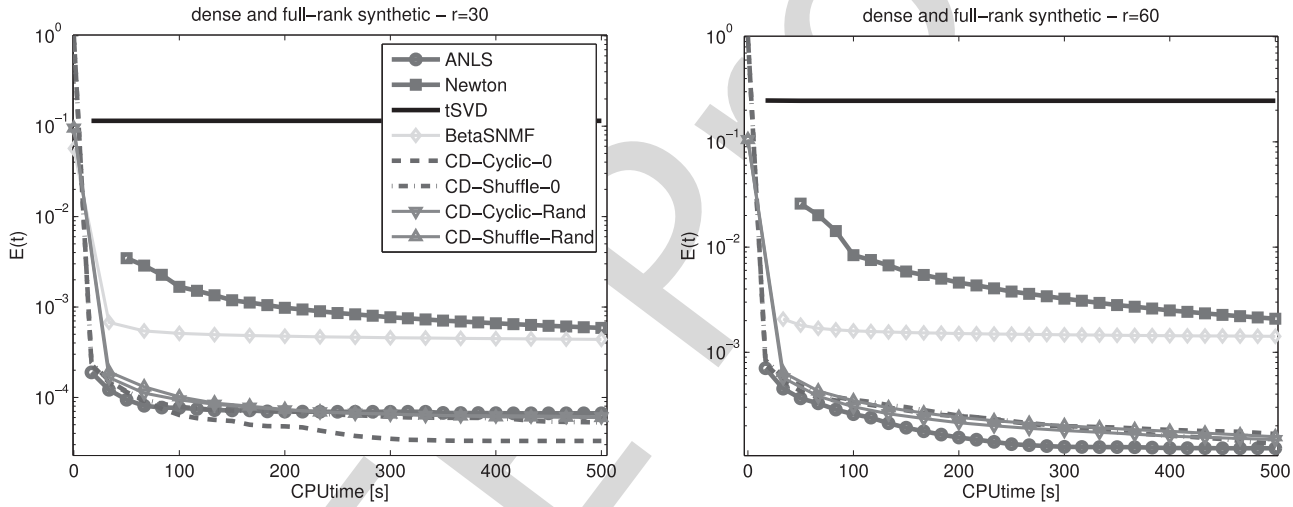


Fig. 5. Evolution of the measure (18) of the different symNMF algorithms on dense full-rank synthetic data sets for $r = 30$ (left) and $r = 60$ (right).

therein. Another important application where low-rank models have proven extremely useful although the data sets are usually not low-rank is *recommender systems* [35] and community detection (see, e.g., [36]). We also refer the reader to the recent survey on low-rank models [37].

2) Synthetic data sets: low-rank vs. full rank matrices: In this section, we perform some numerical experiments on synthetic data sets. Our main motivation is to confirm the (expected) behavior observed on real data: tSVD performs extremely well for low-rank matrices and poorly on full-rank matrices.

a) Low-rank input matrices: The most natural way to generate nonnegative symmetric matrices of given cp-rank is to generate H_* randomly and then compute $A = H_* H_*^T$. In this section, we use the Matlab function $H_* = \texttt{rand}(n, r)$ with $n = 500$ and $r = 30, 60$, that is, each entry of H_* is generated uniformly at random in the interval [0, 1]. We have generated 10 such matrices for each rank, and Fig. 4 displays the average value for the measure (18) but we use here $e_{\min} = 0$ since it is the known optimal value.

We observe that, in all cases, tSVD outperforms all methods. Moreover, it seems that the SVD-based initialization is very effective. The reason is that A has exactly rank r and hence its best rank-r approximation is exact. Moreover, tSVD only works in the correct subspace in which H_* belongs hence converges much faster than the other methods.

Except for Newton, the other algorithms perform similarly. It is worth noting that the same behavior we observed for real dense data sets is present here: CD-Shuffle-Rand performs better than CD-Cyclic-Rand, while shuffling the columns of H before each iteration does not play a crucial role with the zero initialization.

b) Full-Rank Input Matrices: A simple way to generate nonnegative symmetric matrices of full rank is to generate a matrix B randomly and then compute $A = B + B^T$. In this section, we use the Matlab function $B = \texttt{rand}(n)$ with $n = 500$. We have generated 10 such matrices for each rank, and Fig. 5 displays the average value for the measure $E(t)$ from (18). Fig. 5 displays the results.

We observe that, in all cases, tSVD performs extremely poorly while all other methods (except for Newton and BetaSNMF) perform similarly. The reason is that tSVD works only with the best rank-r approximation of A, which is poor when A has full rank.

3) Summary of results: Clearly, tSVD and CD-based approaches are the most effective, although ANLS sometimes performs competitively for the dense data sets. However, tSVD performs extremely well only when the input matrix is low rank (cf. low-rank synthetic data sets) or close to being low rank (cf. image data sets). There are three cases when it performs very poorly:

- It cannot perform a symNMF when the factorization rank r is larger than the rank of A, that is, when $r > \text{rank}(A)$, which may be necessary for matrices with high cp-rank (in fact, the cp-rank can be much higher than the rank [9]).
- If the truncated SVD is a poor approximation of A, the algorithm will perform poorly since it does not use any other information; see the results for the full rank synthetic data sets and the sparse real data sets.
- The algorithm returns no solution as long as the SVD is not computed. In some cases, the cost of computing the truncated SVD is high and tSVD could terminate before any solution to symNMF is produced; see the sparse real data sets.

To conclude, CD-based approaches are overall the most reliable and most effective methods to solve symNMF (1). For dense data sets, initialization at zero allows a faster initial convergence, while CD-Shuffle-Rand generates in average the best solution and CD-Cyclic-Rand does not perform well and is not recommended. For sparse data sets, all CD variants perform similarly and outperform the other tested algorithms.

VI. Conclusion and Further Research

In this paper, we have proposed very efficient exact coordinate descent methods for symNMF (1) that performs competitively with state-of-the-art methods.

Some interesting directions for further research are the following:

- The study of sparse symNMF, where one is looking for a sparser matrix H. A natural model would for example use the sparsity-inducing ℓ_1 norm and try to solve

$$\min_{H \geq 0} \ \frac{1}{4}||A - HH^T||_F^2 + \sum_{j=1}^{r} \Lambda_j ||H_{:,j}||_1 \ , \qquad (19)$$

  for some penalty parameter $\Lambda \in \mathbb{R}^r_+$. Algorithm 4 can be easily adapted to handle (19), by replacing the b_{ij}'s with $b_{ij} + \Lambda_j$. In fact, the derivative of the penalty term only influences the constant part in the gradient; see (12). However, it seems the solutions of (19) are very sensitive to the parameter Λ which are therefore difficult to tune. Note that another way to identify sparser factors is simply to increase the factorization rank r, or to sparsify the input matrix A (only keeping the important edges in the graph induced by A; see [38] and the references therein)—in fact, a sparser matrix A induces sparser factors since

$$A_{ij} = 0 \Rightarrow H_{i,:}H_{j,:}^T \approx 0 \Rightarrow H_{ik} \approx 0 \text{ or } H_{jk} \approx 0 \, \forall k.$$

  This is an interesting observation: $A_{ij} = 0$ implies a (soft) orthogonality constraints on the rows of H. This is rather natural: if item i does not share any similarity with item j ($A_{ij} = 0$), then they should be assigned to different clusters ($H_{ik} \approx 0$ or $H_{jk} \approx 0$ for all k).
- The design of more efficient algorithms for symNMF. For example, a promising direction would be to combine the idea from [29] that use a compressed version of A with very cheap per-iteration cost with our more reliable CD method, to combine the best of both worlds.

Acknowledgment

The authors would like to thank the reviewers for their insightful feedback that helped them improve the paper significantly.

References

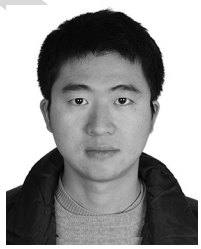
[1] D. Lee and H. Seung, "Learning the parts of objects by nonnegative matrix factorization," *Nature*, vol. 401, pp. 788–791, 1999.
[2] R. Zass and A. Shashua, "A unifying approach to hard and probabilistic clustering," in *Proc. 10th IEEE Int. Conf. Comput. Vis. (ICCV)*, vol. 1, 2005, pp. 294–301.
[3] B. Long, Z. M. Zhang, X. Wu, and P. S. Yu, "Relational clustering by symmetric convex coding," in *Proc. 24th ACM Int. Conf. Mach. Learn.*, 2007, pp. 569–576.
[4] Y. Chen, M. Rege, M. Dong, and J. Hua, "Non-negative matrix factorization for semi-supervised data clustering," *Knowl. Inf. Syst.*, vol. 17, no. 3, pp. 355–379, 2008.
[5] Z. Yang, T. Hao, O. Dikmen, X. Chen, and E. Oja, "Clustering by nonnegative matrix factorization using graph random walk," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1079–1087.
[6] D. Kuang, H. Park, and C. Ding, "Symmetric nonnegative matrix factorization for graph clustering." in *Proc. SIAM Conf. Data Min. (SDM)*, vol. 12, 2012, pp. 106–117.
[7] D. Kuang, S. Yun, and H. Park, "SymNMF: nonnegative low-rank approximation of a similarity matrix for graph clustering," *J. Global Optim.*, vol. 62, no. 3, pp. 545–574, 2014.
[8] X. Yan, J. Guo, S. Liu, X. Cheng, and Y. Wang, "Learning topics in short texts by non-negative matrix factorization on term correlation matrix," in *Proc. SIAM Int. Conf. Data Min.*, 2013.
[9] A. Berman and N. Shaked-Monderer, *Completely Positive Matrices*. Singapore: World Scientific, 2003.
[10] V. Kalofolias and E. Gallopoulos, "Computing symmetric nonnegative rank factorizations," *Linear Algebra Appl.*, vol. 436, no. 2, pp. 421–435, 2012.
[11] S. Burer, "On the copositive representation of binary and continuous nonconvex quadratic programs," *Math. Programm.*, vol. 120, no. 2, pp. 479–495, 2009.
[12] P. Dickinson and L. Gijben, "On the computational complexity of membership problems for the completely positive cone and its dual," *Comput. Optim. Appl.*, vol. 57, no. 2, pp. 403–415, 2014.
[13] J. Kim and H. Park, "Toward faster nonnegative matrix factorization: A new algorithm and comparisons," in *Proc. 8th IEEE Int. Conf. Data Min. (ICDM)*, 2008, pp. 353–362.
[14] J. Kim and H. Park, "Fast nonnegative matrix factorization: An active-set-like method and comparisons," *SIAM J. Scientif. Comput.*, vol. 33, no. 6, pp. 3261–3281, 2011.
[15] N.-D. Ho, "Nonnegative matrix factorization: Algorithms and applications," Ph.D. dissertation, Université Catholique de Louvain, Louvain, France, 2008.
[16] A. Cichocki and A.-H. Phan, "Fast local algorithms for large scale Nonnegative Matrix and Tensor Factorizations," *IEICE Trans. Fundam. Electron.*, vol. E92-A, no. 3, pp. 708–721, 2009.

[17] L. Li and Y.-J. Zhang, "FastNMF: Highly efficient monotonic fixed-point nonnegative matrix factorization algorithm with good applicability," *J. Electron. Imag.*, vol. 18, no. 3, pp. 033 004–033 004, 2009.
[18] N. Gillis and F. Glineur, "Accelerated multiplicative updates and hierarchical ALS algorithms for nonnegative matrix factorization," *Neural Comput.*, vol. 24, no. 4, pp. 1085–1105, 2012.
[19] C.-J. Hsieh and I. Dhillon, "Fast coordinate descent methods with variable selection for non-negative matrix factorization," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discov. Data Min.*, 2011, pp. 1064–1072.
[20] N. Gillis, "Nonnegative matrix factorization: Complexity, algorithms and applications," Ph.D. dissertation, Université Catholique de Louvain, Louvain, France, 2011. [Online]. Available: https://sites.google.com/site/nicolasgillis/
[21] S. Wright, "Coordinate descent algorithms," *Math. Programm.*, vol. 151, no. 1, pp. 3–34, 2015.
[22] M. Belachew and N. Gillis, "Solving the maximum clique problem with symmetric rank-one nonnegative matrix approximation," arXiv:1505.07077, 2015.
[23] G. Cardano, *Ars Magna or the Rules of Algebra*. New York, NY, USA: Dover, 1968.
[24] D. Bertsekas, *Corrections for the Book Nonlinear Programming*, 2nd ed., 1999. [Online]. Available: http://www.athenasc.com/nlperrata.pdf
[25] D. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA, USA: Athena Scientific, 1999.
[26] B. Chen, S. He, Z. Li, and S. Zhang, "Maximum block improvement and polynomial optimization," *SIAM J. Optim.*, vol. 22, no. 1, pp. 87–107, 2012.
[27] A. Vandaele, N. Gillis, Q. Lei, K. Zhong, and I. Dhillon, "Coordinate descent methods for symmetric nonnegative matrix factorization," 2015, arXiv:1509.01404.
[28] S. Zhong and J. Ghosh, "Generative model-based document clustering: a comparative study," *Knowl. Inf. Syst.*, vol. 8, no. 3, pp. 374–384, 2005.
[29] K. Huang, N. Sidiropoulos, and A. Swami, "Non-negative matrix factorization revisited: Uniqueness and algorithm for symmetric decomposition," *IEEE Trans. Signal Process.*, vol. 62, no. 1, pp. 211–224, 2014.
[30] R. Bro, E. Acar, and T. Kolda, "Resolving the sign ambiguity in the singular value decomposition," *J. Chemometr.*, vol. 22, no. 2, pp. 135–140, 2008.
[31] Z. He, S. Xie, R. Zdunek, G. Zhou, and A. Cichocki, "Symmetric nonnegative matrix factorization: Algorithms and applications to probabilistic clustering," *IEEE Trans. Neural Netw.*, vol. 22, no. 12, pp. 2117–2131, 2011.
[32] D. Lee and H. Seung, "Algorithms for Non-negative Matrix Factorization," in *Proc. Adv. Neural Inf. Process.*, vol. 13, 2001.
[33] Z. Yang and E. Oja, "Quadratic nonnegative matrix factorization," *Pattern Recognit.*, vol. 45, no. 4, pp. 1500–1510, 2012.
[34] L. Trefethen and D. Bau III, *Numerical Linear Algebra*. Philadelphia, PA, USA: SIAM, vol. 50, 1997.
[35] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Comput.*, no. 8, pp. 30–37, 2009.
[36] I. Psorakis, S. Roberts, M. Ebden, and B. Sheldon, "Overlapping community detection using Bayesian non-negative matrix factorization," *Phys. Rev. E*, vol. 83, no. 6, p. 066114, 2011.
[37] M. Udell, C. Horn, R. Zadeh, and S. Boyd, "Generalized low rank models," *Found. Trends Mach. Learn.*, 2015, to appear, arXiv:1410.0342.
[38] J. Batson, D. Spielman, N. Srivastava, and S. Teng, "Spectral sparsification of graphs: Theory and algorithms," *Commun. ACM*, vol. 56, no. 8, pp. 87–94, 2013.

Arnaud Vandaele received the M.Sc. degree in computer science engineering from Université de Mons, Belgium, in 2008 and the M.Sc. degree in applied mathematics engineering from Université catholique de Louvain, Belgium, in 2011. He is currently a Ph.D. student at the Université de Mons, Belgium and his research include numerical optimization and linear algebra.

Nicolas Gillis received his Ph.D. from Université catholique de Louvain (Belgium) in 2011. He is currently an associate professor at the Department of Mathematics and Operational Research, Faculté polytechnique, Université de Mons, Belgium. His research interests lie in optimization, numerical linear algebra, machine learning and data mining.

Qi Lei received the B.S degree of Mathematics from Zhejiang University, Zhejiang, China in 2014. She is currently a Ph.D. student at the University of Texas at Austin. Since 2014, she joined the Center for Big Data Analytics under the supervision of Professor Inderjit Dhillon and her research interests lie in large-scale optimization and linear algebra.



Kai Zhong received the B.S. degree in physics from Peking University in China in 2012. He is currently working towards the doctoral degree at the Institute for Computational Engineering and Science in the University of Texas at Austin. His research interests include machine learning, data mining and numerical optimization.

Inderjit Dhillon (F'XX) is the Gottesman Family Centennial Professor of Computer Science and Mathematics at UT Austin, where he is also the Director of the ICES Center for Big Data Analytics. His main research interests are in big data, machine learning, network analysis, linear algebra and optimization. He received his B.Tech. degree from IIT Bombay, and Ph.D. from UC Berkeley. He has received several prestigious awards, including the ICES Distinguished Research Award, the SIAM Outstanding Paper Prize, the Moncrief Grand Challenge Award, the SIAM Linear Algebra Prize, the University Research Excellence Award, and the NSF Career Award. He has published over 140 journal and conference papers, and has served on the Editorial Board of the *Journal of Machine Learning Research*, the IEEE TRANSACTIONS OF PATTERN ANALYSIS AND MACHINE INTELLIGENCE, *Foundations and Trends in Machine Learning* and the *SIAM Journal for Matrix Analysis and Applications*. He is an SIAM Fellow and an ACM Fellow.

QUERIES

Q1. Author: For [8], and all conference paper references, provide page numbers if printed in proceeding or location of conference where presented if not printed.

Q2. Author: For [24], provide publisher name and location.

Q3. Author: Only one source allowed per reference; please either delete the arXiv paper or split into two.

Q4. Author: Please provide initial year of IEEE membership grade.