

Article

A Two-Phase Algorithm for Robust Symmetric Non-Negative Matrix Factorization

Bingjie Li , Xi Shi and Zhenyue Zhang *

School of Mathematics Science, Yuquan Campus, Zhejiang University, Hangzhou 310038, China; 11535017@zju.edu.cn (B.L.); 22035035@zju.edu.cn (X.S.)

* Correspondence: zyzhang@zju.edu.cn

Abstract: As a special class of non-negative matrix factorization, symmetric non-negative matrix factorization (SymNMF) has been widely used in the machine learning field to mine the hidden non-linear structure of data. Due to the non-negative constraint and non-convexity of SymNMF, the efficiency of existing methods is generally unsatisfactory. To tackle this issue, we propose a two-phase algorithm to solve the SymNMF problem efficiently. In the first phase, we drop the non-negative constraint of SymNMF and propose a new model with penalty terms, in order to control the negative component of the factor. Unlike previous methods, the factor sequence in this phase is not required to be non-negative, allowing fast unconstrained optimization algorithms, such as the conjugate gradient method, to be used. In the second phase, we revisit the SymNMF problem, taking the non-negative part of the solution in the first phase as the initial point. To achieve faster convergence, we propose an interpolation projected gradient (IPG) method for SymNMF, which is much more efficient than the classical projected gradient method. Our two-phase algorithm is easy to implement, with convergence guaranteed for both phases. Numerical experiments show that our algorithm performs better than others on synthetic data and unsupervised clustering tasks.



Citation: Li, B.; Shi, X.; Zhang, Z. A Two-Phase Algorithm for Robust Symmetric Non-Negative Matrix Factorization. *Symmetry* **2021**, *13*, 1757. <https://doi.org/10.3390/sym13091757>

Academic Editor:
Juan Luis García Guirao

Received: 7 August 2021
Accepted: 17 September 2021
Published: 20 September 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: symmetric non-negative matrix factorization; low-rank approximation; non-linear conjugate gradient method; projected gradient method; data clustering

1. Introduction

A large amount of data in the real world, such as images and texts, are non-negative. Non-negative matrix factorization (NMF) [1] is a special low rank factorization approach for these non-negative data. NMF aims to decompose a non-negative matrix $X \in \mathbb{R}^{m \times n}$ into the product of two non-negative matrices $W \in \mathbb{R}^{m \times r}$ ($r \ll \min\{m, n\}$) and $H \in \mathbb{R}^{n \times r}$, that is,

$$\min_{W, H \geq 0} \|X - WH^T\|_F^2, \quad (1)$$

where $\|\cdot\|_F$ is the Frobenius norm of a matrix. Due to the non-negativity of W and H , NMF has better interpretability than unconstrained matrix factorization. For images, the columns of W can be interpreted as the basis images, and the columns of H can be seen as the linear combinations of these basis images. Since the number of basic images is much smaller than the dimensionality of the original images, H can be regarded as a low-dimensional representation of X . Generally, the clustering performance on H is better than that on the original data X . Due to the above advantages of NMF, it has been widely used in object recognition [2], face feature extraction [3], blind source separation [4], hyperspectral imaging processing [5], and many others. In recent years, in order to improve the representation ability of NMF, a variety of constrained NMF models have been proposed, including Sparse NMF [6], orthogonal NMF [7] and graph regularized NMF [8]. For a survey of NMF, we recommend [9,10].

In recent years, symmetric non-negative matrix factorization (SymNMF) has become a competitive low-rank approximation approach in pattern mining. Given a symmetric non-negative matrix $A \in \mathbb{R}^{n \times n}$, SymNMF aims to find a non-negative matrix $H \in \mathbb{R}^{n \times r}$, in order to solve

$$\min_{H \geq 0} \frac{1}{4} \|A - HH^T\|_F^2, \quad (2)$$

where $r \ll n$ is given in advance. SymNMF can be regarded as a special form of NMF in (1), requiring $W = H$. However, there are significant differences between the two approaches. In SymNMF, the matrix to be decomposed is the affinity matrix of the data, instead of the data matrix itself in NMF. SymNMF obtains a low-dimensional representation of the data, where the non-negative constraint on the representation makes the class information easy to distinguish. When different classes cannot be separated linearly, the effect of SymNMF is often better than that of the original NMF and its variants. In addition, SymNMF is closely related to fuzzy clustering [11]. Let x_1, \dots, x_n be n independent data points, and C_1, \dots, C_r are the r clusters to which the above data points belong. Assuming that the probability that the data point x_i belongs to the category C_j is p_{ij} , then we have $\sum_{j=1}^r p_{ij} = 1$. Given a similarity matrix $A = (a_{ij})$ such that a_{ij} equals the probability that x_i and x_j belongs to the same cluster, it has been proved in [12] that the SymNMF of A can recover $P = (p_{ij})$. Thus, the fuzzy clustering of x_1, \dots, x_n can be done.

SymNMF has been proven to be effective in cluster inference [13], image segmentation [14], genetic data analysis [15], and so on. It performs better than classical clustering approaches, such as K-means [16], PCA [17] and spectral clustering [18].

1.1. Existing Methods

Due to the powerful ability of SymNMF in pattern recognition, it has attracted widespread attention since its introduction. Most of the algorithms used for solving (2) are derived from variants of the NMF algorithms, mainly due to the correlation between SymNMF and NMF. Generally speaking, the mainstream algorithms used for SymNMF can be divided into three variants: multiplicative update, coordinate descent, and asymmetric relaxation. Next, we briefly introduce the ideas and progress of these three variants.

Multiplicative update methods. The earliest research on the use of SymNMF in data mining can be traced back to the work of Zass and Shashua [12]. They extended the multiplicative update (MU) algorithm from NMF to SymNMF. However, their algorithm for SymNMF performed poorly on real-world data sets, mainly because the MU algorithm has no convergence guarantee and the objective function declines very slowly. In [19], He et al. proposed two algorithms based on parallel MU; namely, α -SNMF and β -SNMF. Unlike the matrix–vector operations used by Zass and Shashua, α -SNMF and β -SNMF use matrix–matrix operations to update the non-negative factor, H . The experiments detailed in [19] demonstrated that the computational efficiency of α -SNMF and β -SNMF is higher than that of previous algorithms. In [20], based on the method of He et al., the authors proposed an accelerated MU (AMU) algorithm, which combines the Nesterov method and the restart strategy. Experiments have shown that the acceleration strategy proposed in [20] is more effective than α -SNMF and β -SNMF.

Coordinate descent methods. Inspired by the high efficiency of the coordinate descent method [21] in NMF, Vandaele et al. extended it to SymNMF [22]. The authors rewrote the symmetric non-negative matrix factorization as:

$$\min_{h_1, \dots, h_r \geq 0} \|A - \sum_{i=1}^r h_i h_i^T\|_F^2,$$

and changed one of $\{h_1, \dots, h_r\}$ in each update, where h_i ($1 \leq i \leq r$) is the i -th column vector of H . This approach can obtain a closed-form solution in each iteration. Their algorithm is named block coordinate descent (BCD) method. However, BCD is not guar-

anted to converge to the stationary point of SymNMF, mainly due to the possibility of multiple solutions in its update. To overcome this weakness of the BCD algorithm, Shi et al. proposed an inexact block coordinate descent (IBCD) algorithm [23], updating variables by successively minimizing a sequence of approximations of the objective function. The IBCD algorithm and its variants are guaranteed to achieve convergence to stationary solutions under very mild conditions.

Asymmetric relaxation methods. Unlike the above direct algorithms, Kuang et al. [14] relaxed (2) to

$$\min_{W, H \geq 0} \|A - WH^T\|_F^2 + \lambda \|W - H\|_F^2, \quad (3)$$

and solved it by updating W and H alternately, where $\lambda > 0$ is a pre-determined parameter. However, the stationary point (W^*, H^*) of the problem (3) cannot provide a SymNMF. In [14], the author enforced $W^* = H^*$ and adopted H^*H^{*T} as the approximation of A . However, H^* may not be a stationary point of the problem (2). The defect of the asymmetric relaxation was solved in [24], where Zhu et al. proved that, if the coefficient λ in the problem (3) is large enough, the (W^*, H^*) automatically satisfies $W^* = H^*$, and H^* is a stationary point of (2). In addition, Zhu et al. extended the hierarchical alternating least squares (HALS) method [24] from NMF to the symmetric case. The HALS algorithm is superior to the previous algorithm, in terms of speed and clustering accuracy, according to their report.

Most of the above algorithms guarantee the stationary point of the problem (2). However, they have two common flaws as follows:

- The existing SymNMF algorithm always restricts the iteration sequence to be non-negative. Since non-negative constraint problems are more difficult to solve than unconstrained problems, the efficiency of existing algorithms is still unsatisfactory, even though coordinate drops or asymmetric relaxation are used to speed up execution.
- The objective function of SymNMF is a quartic function with non-negative constraints, which is strongly non-convex and has a lot of local optimal solutions. Starting from different initial points, the existing algorithms will always find different local optima, leading to loss of robustness.

These two defects limit the application of SymNMF in practice. Hence, there is an urgent need for a new SymNMF algorithm with high efficiency and robustness.

1.2. Our Contributions

In this article, we aim to find a fast and robust algorithm for SymNMF. We achieve this goal through the use of a two-phase strategy. In the first phase, we drop the non-negative constraint and, instead, punish the negative part of H for making H as close to a non-negative matrix as possible. The unconstrained optimization problem we propose for use in this phase can be quickly implemented in the non-linear conjugate gradient method. In the second phase, we directly solve the problem (2), using the non-negative part of the factor obtained in the first phase as the initial point. Additionally, in the second phase, we propose an interpolation projected gradient (IPG) method for SymNMF. IPG uses the quadratic interpolation technique to determine the step length, which can efficiently obtain the stationary point of the problem (2). Compared with the existing algorithms, our two-phase method (TPM) has the following advantages:

- In both phases, TPM works quickly and guarantees convergence. In general, the total running time of TPM is much smaller than that of existing methods.
- Compared with existing algorithms, TPM can obtain SymNMF with a lower approximation error, and is more robust to the initial point.
- In real-world unsupervised clustering tasks, TPM performs significantly better than existing methods. TPM can not only obtain higher clustering accuracy in a shorter time, but also performs very robustly.

The rest of the paper is organized as follows: In the second section, we discuss the proposed two-phase method. The convergence analysis of the two phases is also given in this section. In the third section, we illustrate the superiority of our algorithm, through multiple simulations and real-world examples. Finally, we provide our conclusions in Section 4.

2. Two-Phase Method for SymNMF

Due to the non-convexity and non-negativity constraint of (2), existing SymNMF algorithms face two dilemmas: low efficiency and dependence on the initial point. The two-phase algorithm we propose in this section solves the above dilemma in a step-by-step manner. In the first phase, we propose an unconstrained model for SymNMF. It can be solved quickly, using the non-linear conjugate gradient method, and can provide a non-negative factor H of A with a small approximation error. In the second phase, we propose a first-order algorithm, named interpolation projected gradient (IPG) method, which is more efficient than the classical gradient projection method. Starting from the factor obtained in the first phase, IPG quickly converges to the stationary point of (2).

2.1. The Unconstrained Model for SymNMF

In the first phase of our algorithm, we consider dropping the non-negative constraint in (2) and add a penalty term, in order to ensure that the decomposition factor H of A contains as few negative entries as possible. Our proposed model is:

$$\min_{H \in \mathbb{R}^{n \times r}} \{f(H) := \frac{1}{4} \|A - HH^T\|_F^2 + \frac{\lambda}{2} \|[H]_-\|_F^2\}, \quad (4)$$

where $\lambda > 0$ is a pre-determined parameter and $[H]_-$ represents the negative part of H (i.e., $[H]_- = \min\{H, 0\}$). Obviously, $f(H)$ is an unconstrained differentiable function, and it is easy to verify the gradient of $f(H)$ as follows:

$$\nabla f(H) = (HH^T - A)H + \lambda[H]_-.$$

Hence, first-order optimization methods, such as non-linear conjugate gradient methods (NCG), can be used to solve (4).

Basically, NCG is of the form:

$$H_{k+1} = H_k + \alpha_k D_k, \quad (5)$$

where the step length $\alpha_k > 0$ is obtained by some line search methods, and the search direction D_k is determined by:

$$D_k = \begin{cases} -\nabla f(H_k), & k = 1, \\ -\nabla f(H_k) + \beta_k D_{k-1}, & k > 1, \end{cases} \quad (6)$$

where D_{k-1} is the search direction of the previous step. The step length α_k and combination coefficient β_k determine the convergence behavior of NCG. Among the various forms of β_k in the literature, the Polak–Ribiere–Polyak (PRP) formula [25]

$$\beta_k^{\text{PRP}} = \frac{\langle \nabla f(H_k), \nabla f(H_k) - \nabla f(H_{k-1}) \rangle}{\|\nabla f(H_{k-1})\|_F^2} \quad (7)$$

is popular, due to its high efficiency, where $\langle \cdot, \cdot \rangle$ in (7) is the matrix inner product, that is, for given matrix $A = (a_{ij})$ and $B = (b_{ij})$, $\langle A, B \rangle = \sum_i \sum_j a_{ij} b_{ij}$. However, for general continuous differentiable functions, the PRP formula does not guarantee convergence. Meanwhile, the direction D_k determined by (7) may not satisfy the descent property; that is, $\langle \nabla f(H_k), D_k \rangle \geq 0$ may appear for some k .

In order to improve the convergence properties of the PRP formula, we combine the PRP method with the steepest descent method, such that the search direction always satisfies the descent property. For the current iteration point, H_k , we consider the search direction

$$D_k(p) := -\nabla f(H_k) + \frac{1}{2^p} \beta_k^{\text{PRP}} D_{k-1},$$

where p is an integer. For $p = 0$, $D_k(p)$ is equivalent to the direction generated by the PRP formula. While $p \rightarrow +\infty$, $D_k(p)$ is the opposite direction of the gradient. Let us consider the function:

$$\phi_k(p) := \cos \langle D_k(p), -\nabla f(H_k) \rangle = \frac{\langle D_k(p), -\nabla f(H_k) \rangle}{\|D_k(p)\|_F \|\nabla f(H_k)\|_F}.$$

As $\lim_{p \rightarrow +\infty} \phi_k(p) = 1$, for a given $\mu \in (0, 1)$, there must be an integer p such that $\phi_k(p) > \mu$. For each k , we take:

$$D_k = D_k(p_k^*), \quad \text{s.t.} \quad p_k^* = \arg \min \{p \in \mathbb{N}_+ : \phi_k(p) > \mu\} \tag{8}$$

as the search direction in (5). For each k , the angle between D_k and $-\nabla f(H_k)$ is less than $\arccos(\mu)$.

After D_k is determined, we need to find a suitable step length α_k . Generally, α_k should at least follow an inexact line search criteria. The weak Wolfe condition [26],

$$f(H_k + \alpha D_k) \leq f(H_k) + \rho \alpha \langle \nabla f(H_k), D_k \rangle \tag{9}$$

$$\langle \nabla f(H_k + \alpha D_k), D_k \rangle \geq \sigma \langle \nabla f(H_k), D_k \rangle, \tag{10}$$

has been commonly used, where ρ and σ satisfy $0 < \rho < \sigma < 1$. In our previous work [27], we proposed a simple strategy combining bisection with interpolation, which can be used to obtain α_k satisfying (9) and (10). Initially, we set $\alpha'_{k,0} = 0$, which satisfies (9) but not (10), and selects a relatively large $\alpha''_{k,0} > 0$, making (9) invalid. Starting from the interval $[\alpha'_{k,0}, \alpha''_{k,0}]$, we iteratively generate a sequence of intervals $\{[\alpha'_{k,\ell}, \alpha''_{k,\ell}]\}$, such that each $\alpha'_{k,\ell}$ satisfies (9) but does not satisfy (10). At the same time, $\alpha''_{k,\ell}$ does not satisfy (9). For the current interval $[\alpha'_{k,\ell}, \alpha''_{k,\ell}]$, we consider the quadratic function $q(\alpha)$ that satisfies the interpolation conditions:

$$q(\alpha'_{k,\ell}) = f(H_k + \alpha'_{k,\ell} D_k),$$

$$q'(\alpha'_{k,\ell}) = \langle \nabla f(H_k + \alpha'_{k,\ell} D_k), D_k \rangle,$$

$$q(\alpha''_{k,\ell}) = f(H_k + \alpha''_{k,\ell} D_k).$$

The minimizer $c_{k,\ell} = \arg \min_{\alpha} q(\alpha)$ can be directly obtained by:

$$c_{k,\ell} = \alpha'_{k,\ell} + \frac{\alpha''_{k,\ell} - \alpha'_{k,\ell}}{2} \frac{-(\alpha''_{k,\ell} - \alpha'_{k,\ell})q'(\alpha'_{k,\ell})}{q(\alpha''_{k,\ell}) - q(\alpha'_{k,\ell}) - (\alpha''_{k,\ell} - \alpha'_{k,\ell})q'(\alpha'_{k,\ell})}. \tag{11}$$

We slightly modified $c_{k,\ell}$ to:

$$\tilde{c}_{k,\ell} = \max \{c_{k,\ell}, \eta \alpha'_{k,\ell} + (1 - \eta) \alpha''_{k,\ell}\} \in (\alpha'_{k,\ell}, \alpha''_{k,\ell}), \tag{12}$$

where $\eta = \frac{\sigma}{2(\sigma - \rho)}$. If the Wolfe–Powell conditions (9) and (10) hold for $\alpha = \tilde{c}_{k,\ell}$, we can obtain $\alpha_k = \tilde{c}_{k,\ell}$. Otherwise, we continue to shrink $[\alpha'_{k,\ell}, \alpha''_{k,\ell}]$ as follows:

$$[\alpha'_{k,\ell+1}, \alpha''_{k,\ell+1}] = \begin{cases} [\alpha'_{k,\ell}, \tilde{c}_{k,\ell}], & \text{if (9) does not hold for } \alpha = \tilde{c}_{k,\ell}; \\ [\tilde{c}_{k,\ell}, \alpha''_{k,\ell}], & \text{otherwise.} \end{cases} \tag{13}$$

Using the Lemma 3 in [27], for a decreasing direction D_k , we can always obtain α_k satisfying (9) and (10) in finite steps.

Summarizing the strategy for determining D_k and α_k given above, we propose a modified non-linear conjugate gradient method for solving (4). The detail of the algorithm can be seen in Algorithm 1, and the parameters of the algorithm are set as $\rho = 0.1$, $\sigma = 0.4$, $\mu = 10^{-3}$, $\epsilon_1 = 10^{-4}$, and $k_{\max}^{(1)} = 500$. The sequence generated by Algorithm 1 converges to the stationary point of $f(H)$, according to the following lemma [28].

Algorithm 1 Modified non-linear conjugate gradient method for solving (4).

Require: initial point H_0 ; parameters λ , $0 < \rho < \sigma < 1$, $\mu \in (0, 1)$, $\epsilon_1 > 0$, $k = 0$; and maximum number of iterations $k_{\max}^{(1)}$.

Ensure: an approximate solution H^* .

- 1: While $k \leq k_{\max}^{(1)}$,
 - 2: Compute $f(H_k)$, $\nabla f(H_k)$, and set D_k by (8).
 - 3: If $\|\nabla f(H_k)\|_\infty < \epsilon_1$ or $k = k_{\max}^{(1)}$, set $H^* = H$ and terminate the iteration.
 - 4: Otherwise, compute α_k by alternatively repeating (12) and (13), until we obtain α , satisfying (9) and (10), then update $H_{k+1} = H_k + \alpha_k D_k$.
 - 5: Set $k = k + 1$.
 - 6: End for
-

Lemma 1. Let each α_k satisfy the Wolfe–Powell conditions (9) and (10) and each D_k satisfy

$$\cos\langle D_k, -\nabla f(H_k) \rangle \leq \mu$$

for a fixed $\mu \in (0, 1)$. If ∇f exists and is uniformly continuous on the level set $\{H : f(H) \leq f(H_0)\}$, where H_0 is the initial point, then the sequence $\{H_k\}$ generated by (5) is convergent and $\lim_{k \rightarrow \infty} \|\nabla f(H_k)\|_F = 0$.

By the definition of D_k in (8), it is obvious that $\cos\langle D_k, -\nabla f(H_k) \rangle \leq \mu$ holds for each k . Moreover, the Wolfe–Powell conditions (9) and (10) hold for each α_k , from the result in [27]. Therefore, we know that $f(H_k)$ generated by Algorithm 1 is monotone decreasing and $\lim_{k \rightarrow \infty} \|\nabla f(H_k)\|_F = 0$.

Generally, Algorithm 1 converges very fast. However, the accumulation point H^* generated by Algorithm 1 might not be non-negative. However, H^* is approximately non-negative, due to the penalty term in (4). Hence, we directly truncate H^* as $[H^*]_+ = \max\{H^*, 0\}$ and regard $[H^*]_+[H^*]_+^T$ as an approximation of A .

2.2. The Interpolation Projected Gradient Method for SymNMF

The symmetric non-negative matrix $[H^*]_+[H^*]_+^T$ obtained by Algorithm 1 approximates A well. However, we observe that $[H^*]_+$ is generally not a stationary point of the original SymNMF problem (2). This implies that we can find $H \geq 0$ such that $\|A - HH^T\|_F < \|A - [H^*]_+[H^*]_+^T\|_F$. This phenomenon inspires us to solve (2) directly, starting from $[H^*]_+$, until we find the stationary point of (2).

Among all SymNMF algorithms, the projected gradient (PG) method [29] is simple to implement and has a low computational cost. However, PG suffers from slow convergence, mainly due to the inefficiency of the method it uses to determine the step length α_k . In this subsection, we revisit PG and modify it to a significantly faster interpolation projected gradient (IPG) method for SymNMF. Let

$$g(H) = \frac{1}{4} \|A - HH^T\|_F^2. \quad (14)$$

It is easy to verify that its gradient has the form:

$$\nabla g(H) = (HH^T - A)H.$$

Given that H_k and $0 < \nu < 1$, the PG searches for $\alpha_k = \alpha$ such that

$$H_k(\alpha) := [H_k - \alpha \nabla g(H_k)]_+ \quad (15)$$

satisfies

$$g(H_k(\alpha)) \leq g(H_k) + \nu \langle H_k(\alpha) - H_k, \nabla g(H_k) \rangle, \quad (16)$$

and update $H_{k+1} = H_k(\alpha_k)$. Given an initial step length $\hat{\alpha} > 0$, PG seeks α_k by backtracking. That is, if $\hat{\alpha}$ does not satisfy (16), we update the step length using $\hat{\alpha} \leftarrow \tau \hat{\alpha}$, where $\tau \in (0, 1)$ is a parameter. It has been proven, in [30], that the required α_k satisfying (16) can be found by backtracking in finite steps. However, PG is generally inefficient, as the derivative information of $g(H)$ is not used by the backtracking strategy.

Given that $\hat{\alpha}$ does not satisfy (16), let us consider the minimizer c of $g(H(\alpha))$ on $[0, \hat{\alpha}]$; that is,

$$c = \arg \min_{\alpha \in [0, \hat{\alpha}]} g(H_k(\alpha)). \quad (17)$$

It is not easy to find c , as $g(H)$ is a quartic function and $H_k(\alpha)$ is a polyline. To tackle this issue, we approximate $g(H_k(\alpha))$ using the following two steps. First, we approximate $H_k(\alpha)$ with a straight line:

$$\hat{H}_k(\alpha) = H_k + \frac{\alpha}{\hat{\alpha}} (H_k(\hat{\alpha}) - H_k)$$

on $[0, \hat{\alpha}]$, noting that $\hat{H}_k(0) = H_k(0) = H_k$ and $\hat{H}_k(\hat{\alpha}) = H_k(\hat{\alpha})$. Then, we approximate $g(\hat{H}_k(\alpha))$ using a quadratic function $q(\alpha)$ that satisfies the interpolation conditions:

$$\begin{aligned} q(0) &= g(H_k), & q(\hat{\alpha}) &= g(H_k(\hat{\alpha})) \\ q'(0) &= \langle \nabla g(H_k), \frac{1}{\hat{\alpha}} (H_k(\hat{\alpha}) - H_k) \rangle. \end{aligned}$$

$q(\alpha)$ can be seen as a rough approximation of $g(H_k(\alpha))$ and can be minimized by a closed-form solution. We have:

Lemma 2. *The quadratic function $q(\alpha)$ opens upward and has a minimizer, as follows:*

$$c = -\frac{\hat{\alpha}}{2} \frac{\langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle}{g(H_k(\hat{\alpha})) - g(H_k) - \langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle}. \quad (18)$$

Proof. By the definition of $q(\alpha)$, this can be represented as:

$$\begin{aligned} q(\alpha) &= g(H_k) + \langle \nabla g(H_k), \frac{1}{\hat{\alpha}} (H_k(\hat{\alpha}) - H_k) \rangle \alpha \\ &\quad + (g(H_k(\hat{\alpha})) - g(H_k) - \langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle) \frac{\alpha^2}{\hat{\alpha}^2}. \end{aligned}$$

To prove Lemma 2, we first prove:

$$\langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle \leq 0. \quad (19)$$

To show (19), we denote $U = H_k(\hat{\alpha}) - H_k$, $V = H_k - \hat{\alpha} \nabla g(H_k)$, $W = H_k$, and $Y = \nabla g(H_k)$. Then, we have:

$$\langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle = \sum_s \sum_t u_{st} y_{st},$$

and

$$u_{st}y_{st} = \begin{cases} -\hat{\alpha}y_{st}^2, & v_{st} \geq 0; \\ -w_{st}y_{st}, & v_{st} < 0. \end{cases}$$

For negative $v_{st} = w_{st} - \hat{\alpha}y_{st}$, we have $0 \leq w_{st} < \hat{\alpha}y_{st}$; thus, $-w_{st}y_{st} \leq 0$ always holds. Therefore, each $u_{st}y_{st}$ is non-positive and (19) is true. As $\hat{\alpha}$ does not satisfy (16), we further obtain:

$$g(H_k(\hat{\alpha})) - g(H_k) > \nu \langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle \geq \langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle,$$

which implies that

$$g(H_k(\hat{\alpha})) - g(H_k) - \langle \nabla g(H_k), (H_k(\hat{\alpha}) - H_k) \rangle > 0.$$

Hence, $q(\alpha)$ opens upward, and its minimizer can be directly obtained as (18). \square

To keep c from being too large or too small, we modify c to:

$$\hat{c} = \min\{\max\{c, \hat{\alpha}\tau_1\}, \hat{\alpha}\tau_2\}, \quad (20)$$

where τ_1 and τ_2 satisfying $0 < \tau_1 < \tau_2 < 1$, are two parameters. If \hat{c} satisfies (16), we set $\alpha_k = \hat{c}$. Otherwise, we update:

$$\hat{\alpha} = \hat{c}, \quad (21)$$

and repeat (18), (20), and (21), until we find that α_k satisfies (16).

We summarize the whole procedure of IPG method in Algorithm 2, with the parameters set as $\nu = 0.1$, $\tau_1 = 0.01$, $\tau_2 = 0.1$, and $k_{\max}^{(2)} = 5000$. We also terminate the iteration while the optimal gap [23] is sufficiently small; that is:

$$\|H_k - [H_k - (H_k H_k^T - A)H_k]_+\|_{\infty} < \epsilon_2,$$

where $[\cdot]_+$ represents the non-negative part of a matrix, $\|\cdot\|_{\infty}$ represents the largest absolute value of all entries in a matrix, and $\epsilon_2 = 10^{-8}$. It has been shown in [31] that this gap is equal to zero if and only if a stationary point of (2) is achieved. IPG guarantees convergence to the stationary point of (2). We have the following theorem:

Theorem 1. Any accumulation point H^{**} generated by Algorithm 2 is a stationary point of problem (2).

Proof. For the current H_k , we claim that the step length α_k satisfying (16) can be found within a finite number of iterations. If the initial $\hat{\alpha} = \max\{2\alpha_{k-1}, \gamma\}$ satisfies (16), α_k is directly obtained. Otherwise, by (20) and (21), it is easy to see that $\hat{\alpha}$ approaches 0 with updates. Meanwhile, there exists an interval $(0, \epsilon)$, such that (16) holds for any α in it, where ϵ is a sufficiently small positive number. Hence, one can always find α_k satisfying (16) within finite iterations. In addition, there exists $\alpha' \in (\alpha_k, \frac{\alpha_k}{\tau_1}]$, such that α' does not satisfy (16). The above characteristics of α_k satisfy the convergence condition of Theorem 11.5.5 in [28], and the correctness of Theorem 1 can be directly obtained. \square

Algorithm 2 Interpolation projected gradient (IPG) method for SymNMF.

Require: initial point H , parameters $\nu \in (0, 1)$, $0 < \tau_1 < \tau_2 < 1$, $\epsilon_2 > 0$, $k = 0$, and the maximum number of iterations $k_{\max}^{(2)}$.

Ensure: an approximate solution H^* .

- 1: While $k \leq k_{\max}^{(2)}$,
- 2: Compute $g(H_k)$ and $\nabla g(H_k)$. If $\|H_k - [H_k - (H_k H_k^T - A)H_k]_+\|_\infty < \epsilon_2$, terminate the iteration.
- 3: Set $\hat{\alpha} = \max\{2\alpha_{k-1}, 10^{-3}\}$.
- 4: Alternately repeat (18), (20), and (21), until $\hat{\alpha}$ satisfies (16). Then, set $\alpha_k = \hat{\alpha}$ and update $H_{k+1} = [H_k - \alpha_k \nabla g(H_k)]_+$.
- 5: Set $k = k + 1$.
- 6: End while
- 7: $H^* = H_k$.

Compared with the classical PG method, IPG performs better. To see this, we generated 20 synthetic symmetric non-negative matrices and tested IPG and PG on them. Each of the generated matrices $A = HH^T$ was constructed using a random non-negative factor $H \in \mathbb{R}^{200 \times 50}$, where the entries of H were uniformly distributed in $[0, 1]$. We set $\tau_1 = 0.01$ and $\tau_2 = 0.1$ for IPG, and selected $\tau = 0.01, 0.02, 0.05, 0.1$ for PG. The left subgraph of Figure 1 shows the average SymNMF error

$$E(H) = \frac{\|A - HH^T\|_F}{\|A\|_F}$$

of each algorithm over the CPU time. The efficiency of IPG was always higher than that of PG, regardless of the parameters of PG.

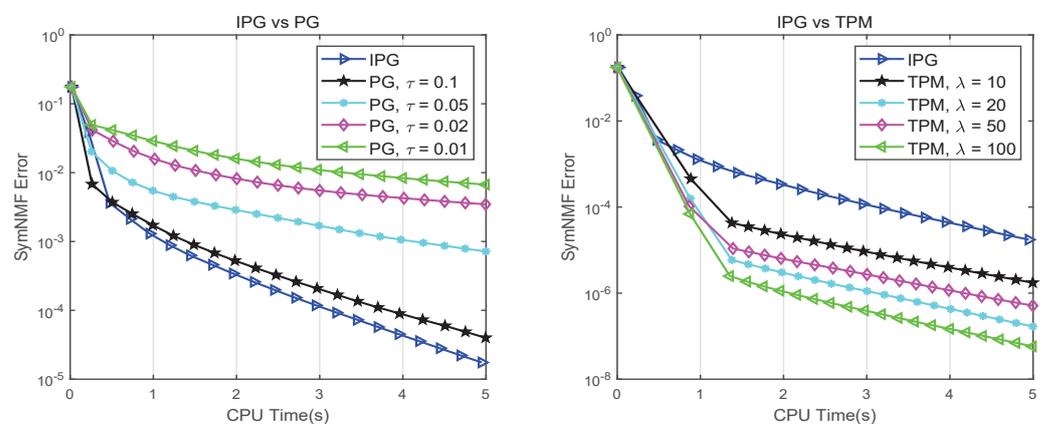


Figure 1. The average SymNMF error versus run-time on the synthetic data.

2.3. The Two-Phase Method for Robust SymNMF

Combining Algorithms 1 and 2, we propose our two-phase method (TPM) for SymNMF. Each of these two phases has a unique effect on obtaining a stable SymNMF. The first phase alleviates the non-convexity of the problem by discarding the non-negativity constraints. Meanwhile, in this phase, we designed an NCG algorithm to quickly obtain a symmetric non-negative approximation of A . Although the convergence point of the first phase is not a stationary point of (2), it can be used as a good initial point for the second phase. In the second phase, we proposed the IPG algorithm, which continues to reduce the SymNMF error and converge to the stationary point of (2) very fast. The combination of these two phases enables us to efficiently solve the SymNMF problem. We summarize the whole procedure of TPM in Algorithm 3.

We also compared TPM with IPG on the synthetic symmetric non-negative matrices generated above. We selected $\lambda = 10, 20, 50, 100$ for TPM. The right subgraph of Figure 1

shows the SymNMF error of each algorithm over the CPU time. The efficiency of TPM was always higher than that of IPG, regardless of the parameters of TPM.

Algorithm 3 Two-Phase Method (TPM) for SymNMF.

Require: initial point H ; parameters $\lambda, 0 < \rho < \sigma < 1, \mu \in (0, 1), \nu = 0.1, 0 < \tau_1 < \tau_2 < 1, \epsilon_1 > 0, \epsilon_2 > 0$; and maximum number of iterations for each phase $k_{\max}^{(1)}, k_{\max}^{(2)}$.

Ensure: a SymNMF factor of A : H^{**} .

- 1: Starting with H , run Algorithm 1 until $\|\nabla f(H_k)\|_\infty < \epsilon_1$ or the number of iterations exceeds $k_{\max}^{(1)}$. Denote the last H_k obtained by Algorithm 1 as H^* .
 - 2: Set $H_+^* = \max\{H^*, 0\}$.
 - 3: Starting with H_+^* , run Algorithm 2 until $\|H_k - [H_k - (H_k H_k^T - A)H_k]_+\|_\infty < \epsilon_2$ or the number of iterations exceeds $k_{\max}^{(2)}$. Denote the last H_k obtained by Algorithm 2 as H^{**} .
-

3. Numerical Experiments and Comparisons

In this section, we report the numerical performance of our method on both synthetic data and real-world data, compared to five state-of-the-art SymNMF approaches. The algorithms used for comparison included alternating non-negative least squares (ANLS) [14], block coordinate descent (BCD) [22], inexact block coordinate descent (IBCD) [23], hierarchical alternating non-negative least squares (HALS) [24], and accelerated multiplicative update (AMU) algorithms [20]. For synthetic data sets, we were concerned about the decline of the objective function and the convergence behavior near the stationary point. For real-world data sets, besides the above measurements, we focused on the clustering performance, in terms of accuracy and stability.

For ANLS, BCD, IBCD, and HALS, we used the respective MATLAB code provided by the authors. For AMU, we wrote the code ourselves and adopted the default setting for the parameters recommended by the authors. The parameter λ of TPM could be automatically set based on the sparsity of the matrix to be decomposed. Generally, one can set λ as

$$\lambda \approx \frac{10nnz}{n^2}, \quad (22)$$

where nnz is the number of non-zeros entries in the matrix. In our experiments, according to (22), we simply set $\lambda = 10$ for synthetic examples (dense matrices) and $\lambda = 0.01$ for real-world data sets (sparse matrices). We implemented all the experiments in MATLAB 2020b on a Windows system using a PC with a 1.60 GHz Intel Core i5-8250U CPU and 8 GB of RAM.

3.1. Synthetic Data

For the synthetic data sets, as in Section 2.2, we generated 20 symmetric non-negative matrices $A = HH^T$ by randomly constructing a non-negative factor $H \in \mathbb{R}^{n \times r}$, where $n = 200, r = 50$, and the entries of H are uniformly distributed in $[0, 1]$. Note that A is a completely positive matrix [32]; that is, A has an exact SymNMF.

To comprehensively evaluate the various algorithms, we adopted two criteria for the performance of SymNMF. The first one was the SymNMF error:

$$E(t) = \frac{\|A - H(t)H(t)^T\|_F}{\|A\|_F}, \quad (23)$$

where $H(t)$ represents the factor H achieved by an algorithm for a given initialization within t seconds. The other criterion was the optimal gap:

$$G(t) = \|H(t) - [H(t) - (H(t)H(t)^T - A)H(t)]_+\|_\infty \quad (24)$$

mentioned in Section 2.2. Recall that $G(t)$ is equal to zero if and only if $H(t)$ is a stationary point of (2). For each matrix, we executed all algorithms 10 times with different initial points, and stopped them when their elapsed time exceeded 5 s. The initial point was randomly generated in the form of $\kappa^* H_0$ [22], where H_0 is uniformly distributed in $[0, 1]$ and κ^* satisfies

$$\kappa^* = \arg \min_{\kappa > 0} \|A - (\kappa H_0)(\kappa H_0)^T\|_F = \sqrt{\frac{\langle AH_0, H_0 \rangle}{\|H_0^T H_0\|_F^2}}. \quad (25)$$

Figure 2 shows the average $E(t)$ and $G(t)$ over 200 trials for each algorithm. We observed the following:

- The efficiency of TPM was much better than that of all other algorithms. This advantage was not only reflected in the decrease in the SymNMF error, but also in the speed at which the iteration approached a stationary point.
- TPM was capable to obtain SymNMF with better quality, compared to other algorithms. The average SymNMF error of TPM was less than 10^{-5} , while the best result of the other SymNMF algorithms could only achieve an average SymNMF error of about 10^{-3} .

The above observations demonstrate that TPM not only converged faster than other approaches but also could obtain SymNMF with lower error.

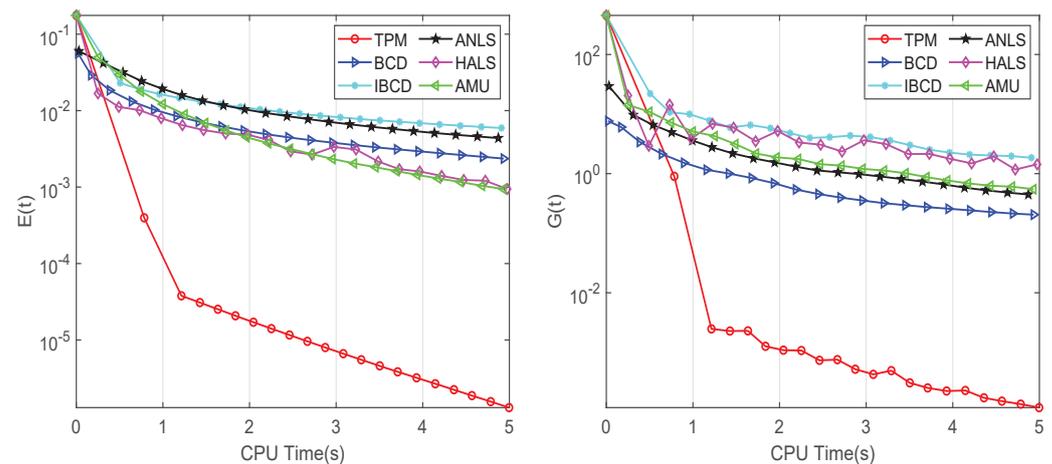


Figure 2. The average $E(t)$ and $G(t)$ (from left to right) of six SymNMF algorithms versus run-time on the synthetic data.

3.2. Real-World Data

In this subsection, we evaluated the performance of TPM and other SymNMF algorithms in real-world data sets from three perspectives: the SymNMF error, the clustering accuracy, and the stability. Given the affinity matrix $A \in \mathbb{R}^{n \times n}$ of a data set $\{x_1, \dots, x_n\}$, SymNMF searched for a non-negative factor $H \in \mathbb{R}^{n \times r}$ satisfying $A \approx HH^T$, where r is the number of classes in the data set. The label of H was determined by the position of the largest component of each row of H ; that is,

$$\ell(x_i) = \arg \max_j \{h_{i1}, \dots, h_{ij}, \dots, h_{ir}\}.$$

where h_{ij} ($1 \leq i \leq n, 1 \leq j \leq r$) is the entry of the i -th row and j -th column of H .

We conducted experiments on four data sets: COIL-20 [33] (<https://www.cs.columbia.edu/CAVE/software/softlib/coil-20.php>, accessed on 29 March 2021), ORL [34] (<http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>, accessed on 19 April 2021), PIE [35] (http://www.ri.cmu.edu/research_project_detail.html?project_id=418&menu_id=261, accessed on 27 April 2021), and TDT2 [36] (<https://www ldc.upenn.edu/>

[collaborations/past-projects](#), accessed on 7 May 2021). Detailed information of these data sets is given in Table 1.

Table 1. Data sets used in our experiments.

Data Name	Type	Dimension	# Data Points	# Clusters
COIL-20	object	32×32	1440	20
ORL	face	92×112	400	40
PIE	face	32×32	2856	68
TDT2	document	36,771	8741	20

The affinity matrix for each data set was constructed following the procedures given in [14]; that is,

- For the image data sets COIL-20, ORL, and PIE, we defined:

$$e_{ij} = \begin{cases} \exp\left(-\frac{\|x_i - x_j\|_2^2}{\sigma_i \sigma_j}\right), & i \in \mathcal{N}_k(j) \text{ or } j \in \mathcal{N}_k(i); \\ 0, & \text{otherwise,} \end{cases}$$

where $\mathcal{N}_k(i)$ is the set of k -nearest neighbors of x_i , and σ_i is the Euclidean distance between x_i and its \hat{k} -th neighbor. We adopted $\hat{k} = 7$, as suggested in [37]. The affinity between x_i and x_j was defined by

$$a_{ij} = d_i^{1/2} e_{ij} d_j^{1/2}, \quad (26)$$

where $d_i = \sum_{s=1}^n e_{is}$.

- For the document data set TDT2, all the document vectors x_i were normalized to have unit 2-norm, and we defined:

$$e_{ij} = \begin{cases} x_i^T x_j, & i \in \mathcal{N}_k(j) \text{ or } j \in \mathcal{N}_k(i); \\ 0, & \text{otherwise.} \end{cases}$$

The affinity between x_i and x_j was also defined as in (26).

The parameter k in $\mathcal{N}_k(i)$ determined the sparseness of the affinity matrix. Unlike the recommended $k = \lceil \log_2(n) \rceil + 1$ in [38], we set k as $\lceil \log_2(\frac{n}{n_c}) \rceil + 1$, where n_c is the class number of the data set. Compared with the original setting, our k was smaller, and the connected data points were more likely to be class consistent.

We executed all algorithms 20 times with different randomly constructed initial points, and stopped them when their elapsed time exceeds the time limit. For COIL-20, ORL, PIE, and TDT2, according to their sizes, the time limits were set as 20, 10, 50, and 120 s, respectively. To compare the average performance of the different algorithms, we denoted

$$\hat{E}(t) = E(t) - E_{\min},$$

where $E(t)$ is defined as in (23) and E_{\min} is the smallest error obtained by all algorithms over all initializations. Figure 3 displays the average $\hat{E}(t)$ for each algorithm. In all cases, TPM performed best and can generate the solution with the lowest SymNMF error among all of the algorithms.

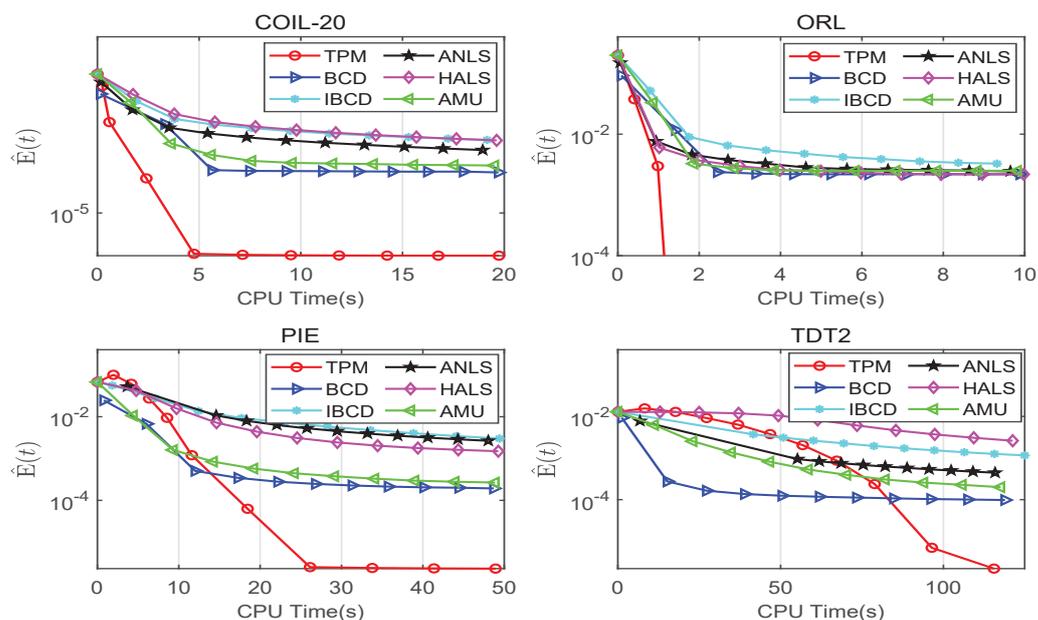


Figure 3. The average $\hat{E}(t)$ of six SymNMF algorithms versus run-time on real-world data sets.

Besides the SymNMF error, we also compared the clustering performance for each algorithm. We adopted two criteria—average correction (AC) and normalized mutual information (NMI) [39]—to measure the clustering performance. The average values over all the 20 trials are listed in Table 2. We observed that TPM performed better than the other algorithms. On COIL-20, ORL, PIE, and TDT2, we increased the highest clustering accuracy of existing methods by 8.46%, 3.09%, 6.87%, and 6.15% respectively. In order to further illustrate the clustering stability of all algorithms, we drew the results of 20 runs from different starting points in a box plot, as shown in Figure 4. Unlike the larger variance of other algorithms, the clustering results of TPM were very concentrated. This phenomenon demonstrated that TPM was very robust to the initial point.

Table 2. Clustering performance measured by AC (%) and NMI (%) on real-world databases.

Metric	Data Set	TPM	ANLS	BCD	IBCD	HALS	AMU
AC	COIL-20	82.98	74.52	60.25	65.77	56.16	71.04
	ORL	78.00	74.57	72.55	74.91	74.30	74.29
	PIE	86.91	80.04	48.32	50.27	56.61	78.80
	TDT2	93.30	87.15	56.31	69.57	33.59	79.93
NMI	COIL-20	91.63	88.13	78.75	81.00	73.22	86.61
	ORL	90.20	89.01	87.82	88.80	88.67	88.77
	PIE	94.96	93.11	76.56	77.53	81.26	92.55
	TDT2	88.33	85.81	62.18	70.53	34.69	80.92

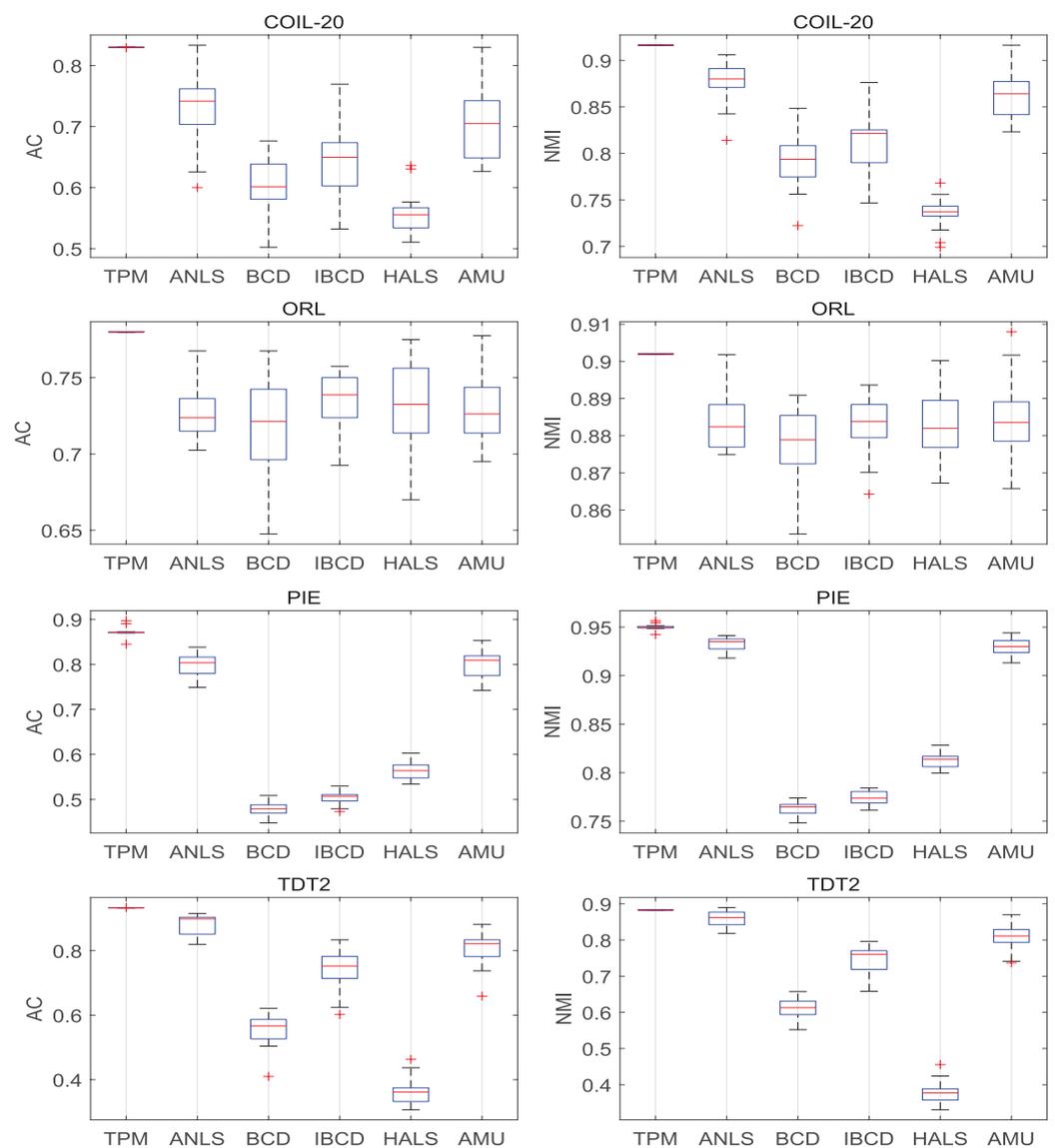


Figure 4. Box-plot of the clustering results of six algorithms on four real-world data sets.

4. Conclusions

In this paper, we proposed a two-phase algorithm for SymNMF. Our method is more efficient and robust than state-of-the-art algorithms, and performed well in real-world clustering tasks. Moreover, our algorithm is guaranteed to converge to the stationary solutions of the SymNMF problem. However, there are still some issues that need to be explored. First, although our algorithm has high efficiency, a theoretical analysis of the convergence rate should be conducted. Second, in real-world clustering tasks, the structure of the graph can significantly affect the clustering results obtained by the SymNMF algorithm. Thirdly, we need to consider how to reduce the time complexity and the storage requirements of the algorithm when processing large-scale data. The distributed-memory parallel strategy proposed in [40] provides a good inspiration. In the future, we intend to carry out further research on the above-mentioned issues.

Author Contributions: Funding acquisition, Z.Z.; methodology B.L.; supervision, Z.Z.; validation, B.L. and X.S.; writing—original draft, B.L.; writing—review and editing, B.L., X.S. and Z.Z. All authors have read and agreed to the published version of the manuscript.

Funding: The work was supported in part by NSFC project 11971430 and Major Scientific Research Project of Zhejiang Lab (No. 2019KB0AB01).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Lee, D.D.; Seung, H.S. Learning the parts of objects by non-negative matrix factorization. *Nature* **1999**, *401*, 788–791. [[CrossRef](#)] [[PubMed](#)]
2. Liu, W.; Zheng, N. Non-negative matrix factorization based methods for object recognition. *Pattern Recognit. Lett.* **2004**, *25*, 893–897. [[CrossRef](#)]
3. David, G.; Jordi, V. Non-negative Matrix Factorization for Face Recognition. In Proceedings of the 5th Catalanian Conference on AI: Topics in Artificial Intelligence, Castellón, Spain, 24–25 October 2002; pp. 336–344.
4. Zdunek, R.; Cichocki, A. Non-negative Matrix Factorization with Quasi-Newton Optimization. In Proceedings of the Artificial Intelligence and Soft Computing, Zakopane, Poland, 25–29 June 2006; pp. 870–879.
5. Jose, M.B.D.; Plaza, A.; Dobigeon, N.; Parente, M.; Du, Q.; Gader, P.; Chanussot, J. Hyperspectral Unmixing Overview: Geometrical, Statistical, and Sparse Regression-Based Approaches. *IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens.* **2012**, *5*, 354–379.
6. Hoyer, P.O. Non-Negative Matrix Factorization with Sparseness Constraints. *J. Mach. Learn. Res.* **2004**, *5*, 1457–1469.
7. Ding, C.; Li, T.; Peng, W.; Park, H. Orthogonal nonnegative matrix t-factorizations for clustering. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, 20–23 August 2006; pp. 126–135.
8. Cai, D.; He, X.; Han, J.; Huang, T.S. Graph Regularized Nonnegative Matrix Factorization for Data Representation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2011**, *33*, 1548–1560.
9. Wang, Y.; Zhang, Y. Nonnegative Matrix Factorization: A Comprehensive Review. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 1336–1353. [[CrossRef](#)]
10. Esposito, F. A Review on Initialization Methods for Nonnegative Matrix Factorization: Towards Omics Data Experiments. *Mathematics* **2021**, *9*, 1006. [[CrossRef](#)]
11. Yang, M.S. A survey of fuzzy clustering. *Math. Comput. Model.* **1993**, *18*, 1–16. [[CrossRef](#)]
12. Zass, R.; Shashua, A. A unifying approach to hard and probabilistic clustering. In Proceedings of the IEEE International Conference on Computer Vision, Beijing, China, 17–21 October 2005; pp. 294–301.
13. Raviteja, V.; Kim, Ø.R.; Gopinath, C.; Boian, A. Determination of the number of clusters by symmetric non-negative matrix factorization. In Proceedings of the Big Data III: Learning, Analytics, and Applications, Online, 12 April 2021; Volume 11730, p. 15.
14. Kuang, D.; Yun, S.; Park, H. SymNMF: Nonnegative low-rank approximation of a similarity matrix for graph clustering. *J. Glob. Optim.* **2015**, *62*, 545–574. [[CrossRef](#)]
15. Ma, Y.; Hu, X.; He, T.; Jiang, X. Hessian Regularization Based Symmetric Nonnegative Matrix Factorization for Clustering Gene Expression and Microbiome Data. *Methods* **2016**, *111*, 80–84. [[CrossRef](#)]
16. Lloyd, S.P. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [[CrossRef](#)]
17. Jolliffe, I.T. Principal Component Analysis. *J. Mark. Res.* **2002**, *87*, 513.
18. Ng, A.Y.; Jordan, M.I.; Weiss, Y. On Spectral Clustering: Analysis and an Algorithm. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 14 April 2001; pp. 849–856.
19. He, Z.; Xie, S.; Zdunek, R.; Zhou, G.; Cichocki, A. Symmetric Nonnegative Matrix Factorization: Algorithms and Applications to Probabilistic Clustering. *IEEE Trans. Neural Netw.* **2011**, *22*, 2117–2131.
20. Wang, P.; He, Z.; Lu, J.; Tan, B.; Bai, Y.; Tan, J.; Liu, T.; Lin, Z. An Accelerated Symmetric Nonnegative Matrix Factorization Algorithm Using Extrapolation. *Symmetry* **2020**, *12*, 1187. [[CrossRef](#)]
21. Wright, S.J. Coordinate Descent Algorithms. *Math. Program.* **2015**, *151*, 3–34. [[CrossRef](#)]
22. Vandaele, A.; Gillis, N.; Lei, Q.; Zhong, K.; Dhillon, I. Efficient and Non-Convex Coordinate Descent for Symmetric Nonnegative Matrix Factorization. *IEEE Trans. Signal Process.* **2016**, *64*, 5571–5584. [[CrossRef](#)]
23. Shi, Q.; Sun, H.; Lu, S.; Hong, M.; Razaviyayn, M. Inexact Block Coordinate Descent Methods for Symmetric Nonnegative Matrix Factorization. *IEEE Trans. Signal Process.* **2017**, *65*, 5995–6008. [[CrossRef](#)]
24. Zhu, Z.; Li, X.; Liu, K.; Li, Q. Dropping Symmetry for Fast Symmetric Nonnegative Matrix Factorization. *arXiv* **2018**, arXiv:1811.05642.
25. Polyak, B.T. The conjugate gradient method in extremal problems. *USSR Comput. Math. Math. Phys.* **1969**, *9*, 94–112. [[CrossRef](#)]
26. Nocedal, J.; Wright, S. *Numerical Optimization*; Springer Science & Business Media: New York, NY, USA, 2006.
27. Zhang, Z.; Li, B. Exterior Point Method for Completely Positive Factorization. *arXiv* **2021**, arXiv:2102.08048.
28. Sun, W.; Yuan, Y. *Optimization Theory and Methods: Nonlinear Programming*; Springer Science & Business Media: New York, NY, USA, 2006.
29. Kuang, D.; Ding, C.; Park, H. Symmetric Nonnegative Matrix Factorization for Graph Clustering. In Proceedings of the 2012 SIAM International Conference on Data Mining (SDM), Anaheim, CA, USA, 26–28 April 2012; pp. 106–117.
30. Calamai, P.; More, J. Projected gradient methods for linearly constrained problems. *Math. Program.* **1987**, *39*, 93–116. [[CrossRef](#)]
31. Razaviyayn, M.; Hong, M.; Luo, Z.Q.; Pang, J.S. Parallel Successive Convex Approximation for Nonsmooth Nonconvex Optimization. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 8–13 December 2014; pp. 1440–1448.

32. Shaked-Monderer, N.; Berman, A. *Copositive and Completely Positive Matrices*; World Scientific: Hackensack, NJ, USA, 2021.
33. Nene, S.; Nayar, S.; Murase, H. *Columbia Object Image Library (COIL-20)*; Tech. Rep. CUCS-005-96; Columbia University: New York, NY, USA, 1996.
34. Samaria, F.; Harter, A. Parameterisation of a stochastic model for human face identification. In Proceedings of the Second IEEE Workshop on Applications of Computer Vision, Sarasota, FL, USA, 5–7 December 1994; pp. 138–142.
35. Sim, T.; Baker, S.; Bsat, M. The CMU Pose, Illumination, and Expression (PIE) database. In Proceedings of the Fifth IEEE International Conference on Automatic Face Gesture Recognition, Washinton, DC, USA, 20–21 May 2002; pp. 46–51.
36. Fiscus, J.; Doddington, G.; Garofolo, J.; Alvin, M. NIST's 1998 Topic Detection and Tracking evaluation (TDT2). In Proceedings of the 1999 DARPA Broadcast News Workshop, Herndon, VA, USA, 28 February–3 March 1999; pp. 19–24.
37. Zelnik-Manor, L.; Perona, P. Self-Tuning Spectral Clustering. In Proceedings of the Advances in Neural Information Processing Systems, Vancouver, BC, Canada, 13–18 December 2004; pp. 1601–1608.
38. Luxburg, U.V. A Tutorial on Spectral Clustering. *Stat. Comput.* **2004**, *17*, 395–416. [[CrossRef](#)]
39. Wu, W.; Jia, Y.; Kwong, S.; Hou, J. Pairwise Constraint Propagation-Induced Symmetric Nonnegative Matrix Factorization. *IEEE Trans. Neural Netw. Learn. Syst.* **2018**, *29*, 6348–6361. [[CrossRef](#)]
40. Eswar, S.; Hayashi, K.; Ballard, G.; Kannan, R.; Vuduc, R.; Park, H. Distributed-Memory Parallel Symmetric Nonnegative Matrix Factorization. In Proceedings of the SC20: The International Conference for High Performance Computing, Networking, Storage and Analysis, Atlanta, GA, USA, 9–19 November 2020; pp. 1–14.