Electronic Transactions on Numerical Analysis.
Volume 36, pp. 54-82, 2009-2010.
Copyright © 2009, Kent State University.
ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

ALTERNATING PROJECTED BARZILAI-BORWEIN METHODS FOR NONNEGATIVE MATRIX FACTORIZATION [*]

LIXING HAN [†], MICHAEL NEUMANN [‡], AND UPENDRA PRASAD [§]

Dedicated to Richard S. Varga on the occasion of his 80th birthday

Abstract. The Nonnegative Matrix Factorization (NMF) technique has been used in many areas of science, engineering, and technology. In this paper, we propose four algorithms for solving the nonsmooth nonnegative matrix factorization (nsNMF) problems. The nsNMF uses a smoothing parameter $\theta \in [0, 1]$ to control the sparseness in its matrix factors and it reduces to the original NMF if $\theta = 0$. Each of our algorithms alternately solves a nonnegative linear least squares subproblem in matrix form using a projected Barzilai–Borwein method with a nonmonotone line search or no line search. We have tested and compared our algorithms with the projected gradient method of Lin on a variety of randomly generated NMF problems. Our numerical results show that three of our algorithms, namely, APBB1, APBB2, and APBB3, are significantly faster than Lin's algorithm for large-scale, difficult, or exactly factorable NMF problems in terms of CPU time used. We have also tested and compared our APBB2 method with the multiplicative algorithm of Lee and Seung and Lin's algorithm for solving the nsNMF problem resulted from the ORL face database using both $\theta = 0$ and $\theta = 0.7$. The experiments show that when $\theta = 0.7$ is used, the APBB2 method can produce sparse basis images and reconstructed images which are comparable to the ones by the Lin and Lee–Seung methods in considerably less time. They also show that the APBB2 method can reconstruct better quality images and obtain sparser basis images than the methods of Lee–Seung and Lin when each method is allowed to run for a short period of time. Finally, we provide a numerical comparison between the APBB2 method and the Hierarchical Alternating Least Squares (HALS)/Rank-one Residue Iteration (RRI) method, which was recently proposed by Cichocki, Zdunek, and Amari and by Ho, Van Dooren, and Blondel independently.

Key words. nonnegative matrix factorization, smoothing matrix, nonnegative least squares problem, projected Barzilai–Borwein method, nonmonotone line search.

AMS subject classifications. 15A48, 15A23, 65F30, 90C30

1. Introduction.

Given a nonnegative matrix V, find nonnegative matrix factors W and H such that

$$V \approx WH. \tag{1.1}$$

This is the so–called *nonnegative matrix factorization* (NMF) which was first proposed by Paatero and Tapper [28, 29] and Lee and Seung [21]. The NMF has become an enormously useful tool in many areas of science, engineering, and technology since Lee and Seung published their seminal paper [22]. For recent surveys on NMF, see [2, 5, 14].

As the exact factorization $V = WH$ usually does not exist, it is typical to reformulate NMF as an optimization problem which minimizes some type of distance between V and WH. A natural choice of such a distance is the Euclidean distance, which results in the following optimization problem:

PROBLEM 1.1 (NMF). *Suppose that $V \in \mathbb{R}^{m,n}$ is nonnegative. Find W and H which*

[*] Received July 14, 2008. Revised version received January 20, 2009. Accepted for publication February 2, 2009. Published online January 18, 2010. Recommended by R. Plemmons.

[†] Department of Mathematics, University of Michigan–Flint, Flint, Michigan 48502, USA, (lxhan@umflint.edu). Work was supported by the National Natural Science Foundation of China (10771158) and a research grant from the Office of Research, University of Michigan–Flint.

[‡] Department of Mathematics, University of Connecticut, Storrs, Connecticut 06269–3009, USA, (neumann@math.uconn.edu). Research partially supported by NSA Grant No. 06G–232.

[§] Department of Mathematics, University of Connecticut, Storrs, Connecticut 06269–3009, USA, (upendra@math.uconn.edu).

solve

$$\min f(W,H) = \frac{1}{2}\|V - WH\|_F^2 \ \text{ s.t. } W \geq 0,\ H \geq 0, \tag{1.2}$$

where $W \in \mathbb{R}^{m,r}$, $H \in \mathbb{R}^{r,n}$, *and* $\|\cdot\|_F$ *is the Frobenius norm.*

One of the main features of NMF is its ability to learn objects by parts (see Lee and Seung [22]). Mathematically this means that NMF generates sparse factors W and H. It has been discovered in [24], however, that NMF does not always lead to sparse W and H. To remedy this situation, several approaches have been proposed. For instance, Li, Hou, and Zhang [24] and Hoyer [16, 17] introduce additional constraints in order to increase the sparseness of the factors W or H. In a more recent proposal by Pascual–Montano et al. [30], the *nonsmooth nonnegative matrix factorization* (nsNMF) introduces a natural way of controlling the sparseness. The nsNMF is defined as:

$$V \approx WSH, \tag{1.3}$$

where $S \in \mathbb{R}^{r,r}$ is a "smoothing" matrix of the form

$$S = (1-\theta)I + \frac{\theta}{r}J, \tag{1.4}$$

where I is the $r \times r$ identity matrix and $J \in \mathbb{R}^{r,r}$ is the matrix of all 1's. The variable θ is called the *smoothing parameter* and it satisfies $0 \leq \theta \leq 1$. If $\theta = 0$, then $S = I$ and the nsNMF reduces to the original NMF. The value of θ can control the sparseness of W and H. The larger θ is, the sparser W and H are.

In [30], Pascual–Montano et al. reformulate (1.3) as an optimization problem using the Kullback–Liebler divergence as the objective function. Here we reformulate (1.3) as the following optimization problem using the Euclidean distance:

PROBLEM 1.2 (nsNMF). *Suppose that* $V \in \mathbb{R}^{m,n}$ *is nonnegative. Choose the smoothing parameter* $\theta \in [0,1]$ *and define* $S = (1-\theta)I + \frac{\theta}{r}J$. *Find* W *and* H *which solve*

$$\min h_S(W,H) = \frac{1}{2}\|V - WSH\|_F^2, \ \text{ s.t. } W \geq 0,\ H \geq 0, \tag{1.5}$$

where $W \in \mathbb{R}^{m,r}$ *and* $H \in \mathbb{R}^{r,n}$.

REMARK 1.3. By the definitions of functions f and h_S, we have

$$h_S(W,H) = f(W,SH) = f(WS,H). \tag{1.6}$$

In [23], Lee and Seung proposed a multiplicative algorithm for solving the NMF problem 1.1. This algorithm starts with a positive pair (W^0, H^0) and iteratively generates a sequence of approximations (W^k, H^k) to a solution of Problem 1.1 using the following updates:

$$W^{k+1} = W^k .* (V(H^k)^T)./(W^k H^k (H^k)^T), \tag{1.7}$$

$$H^{k+1} = H^k .* ((W^{k+1})^T V)./((W^{k+1})^T W^{k+1} H^k). \tag{1.8}$$

The Lee–Seung algorithm is easy to implement and has been used in some applications. It has also been extended to nonnegative matrix factorization problems with additional constraints [17, 24]. Under certain conditions, Lin [26] proves that a modified Lee-Seung algorithm converges to a Karush-Kuhn-Tucker point of Problem 1.1. It has been found that the Lee–Seung algorithm can be slow [2, 25, 5]. A heuristic explanation why the Lee–Seung algorithm has a slow rate of convergence can be found in [13].

Several new algorithms aimed at achieving better performance have been proposed recently (see the survey papers [2, 5], the references therein, and [6, 20, 19, 14, 15]). In particular, the projected gradient method of Lin [25] is easy to implement and often significantly faster than the Lee–Seung algorithm. Under mild conditions, it converges to a Karush–Kuhn–Tucker point of Problem 1.1.

The Lin algorithm starts with a nonnegative pair (W^0, H^0) and generates a sequence of approximations (W^k, H^k) to a solution of Problem 1.1 using the following alternating nonnegative least squares (ANLS) approach:

$$W^{k+1} = \text{argmin}_{W \geq 0} f(W, H^k) = \text{argmin}_{W \geq 0} \frac{1}{2} \|V^T - (H^k)^T W^T\|_F^2 \tag{1.9}$$

and

$$H^{k+1} = \text{argmin}_{H \geq 0} f(W^{k+1}, H) = \text{argmin}_{H \geq 0} \frac{1}{2} \|V - W^{k+1} H\|_F^2. \tag{1.10}$$

The convergence of the ANLS approach to a Karush–Kuhn–Tucker point of the NMF problem 1.1 can be proved (see for example, [12, 25]).

We comment that there are several NMF algorithms which use the ANLS approach. The Lin algorithm differs from the other algorithms in its strategy of solving Subproblems (1.9) and (1.10). Lin uses a projected gradient method to solve each subproblem which is a generalization of the steepest descent method for unconstrained optimization. As is well known, the steepest descent method can be slow, even for quadratic objective functions. The projected gradient method for constrained optimization often inherits this behavior (see [18, 27]).

Using Newton or quasi–Newton methods to solve the subproblems can give a faster rate of convergence [20, 33]. However, these methods need to determine a suitable active set for the constraints at each iteration (see, for example, [4, 18]). The computational cost per iteration of these methods can be high for large-scale optimization problems and the subproblems (1.10) and (1.9) resulting from real life applications are often of large size. Another algorithm which also uses an active set strategy has been proposed in [19].

Note that the NMF problem 1.1 can be decoupled into

$$\min_{w_i \geq 0, h_i \geq 0} \frac{1}{2} \|V - \sum_{i=1}^{r} w_i h_i\|_F^2, \tag{1.11}$$

where the w_i's are the columns of W and h_i's the rows of H. Utilizing this special structure, Ho, Van Doreen, and Blondel [14, 15] recently proposed the so-called Rank-one Residue Iteration (RRI) algorithm for solving the NMF problem. This algorithm was independently proposed by Cichocki, Zdunek, and Amari in [6], where it is called the Hierarchical Alternating Least Squares (HALS) algorithm. In one loop, the HALS/RRI algorithm solves

$$\min_{h \geq 0} \frac{1}{2} \|R_t - w_t h\|_F^2 \tag{1.12}$$

and

$$\min_{w\geq 0} \frac{1}{2}\|R_t - wh_t\|_F^2 \tag{1.13}$$

for $t = 1, 2, \ldots, r$, where $R_t = V - \sum_{i\neq t} w_i h_i$. A distinguished feature of this algorithm is that the solution to (1.12) or (1.13) has an explicit formula which is easy to implement. Numerical experiments in [6, 14, 15] show that the HALS/RRI algorithm is significantly faster than the Lee–Seung method and some other methods based on ANLS approach. Under mild conditions, it has been proved that this algorithm converges to a Karush-Kuhn-Tucker point of Problem 1.1 (see [14, 15]). For detailed description and analysis of the RRI algorithm, we refer to the thesis of Ho [14].

It has been reported by many researchers in the optimization community that the Barzilai–Borwein gradient method with a non–monotone line search is a reliable and efficient algorithm for large-scale optimization problems (see for instance [3, 10]).

In this paper, motivated by the success of Lin's projected gradient method and good performance of the Barzilai–Borwein method for optimization, we propose four projected Barzilai–Borwein algorithms within the ANLS framework for the nsNMF problem 1.2. We shall call these algorithms *Alternating Projected Barzilai–Borwein* (APBB) algorithms for nsNMF. They alternately solve the nonnegative least squares subproblems:

$$W^{k+1} = \text{argmin}_{W\geq 0} f(W, SH^k) = \text{argmin}_{W\geq 0}\frac{1}{2}\|V^T - (SH^k)^T W^T\|_F^2 \tag{1.14}$$

and

$$H^{k+1} = \text{argmin}_{H\geq 0} f(W^{k+1}S, H) = \text{argmin}_{H\geq 0}\frac{1}{2}\|V - (W^{k+1}S)H\|_F^2. \tag{1.15}$$

Each of these two subproblems is solved using a projected Barzilai–Borwein method.

Note that if the smoothing parameter $\theta = 0$, then our algorithms solve the original NMF problem 1.1. We will test the APBB algorithms on various examples using randomly generated data and the real life ORL database [36] and CBCL database [35] and compare them with the Lee–Seung, Lin, and HALS/RRI algorithms. Our numerical results show that three of the APBB algorithms, namely, APBB1, APBB2 and APBB3 are faster than the Lin algorithm, in particular, for large-scale, difficult, and exactly factorable problems in terms of CPU time used. They also show that the APBB2 method can outperform the HALS/RRI method on large-scale problems.

It is straightforward to extend the original Lee–Seung algorithm for NMF to a multiplicative algorithm for nsNMF by replacing H^k by SH^k in (1.7) and W^{k+1} by $W^{k+1}S$ in (1.8). The resulting Lee–Seung type multiplicative updates for nsNMF are:

$$W^{k+1} = W^k .* (V(SH^k)^T)./(W^kSH^k(SH^k)^T) \tag{1.16}$$

and

$$H^{k+1} = H^k .* ((W^{k+1}S)^T V)./((W^{k+1}S)^T W^{k+1}SH^k). \tag{1.17}$$

The Lin algorithm for NMF can also be extended to nsNMF by solving Subproblems (1.14) and (1.15) using his projected gradient approach. We will also compare the APBB2, Lee–Seung, and Lin methods on their abilities to generate sparse W and H factors when they are applied to the nsNMF problem resulting from the ORL database with $\theta > 0$.

This paper is organized as follows. In Section 2 we give a short introduction of the Barzilai–Borwein gradient method for optimization. We also present two sets of first order optimality conditions for the nsNMF problem which do not need the Lagrange multipliers and can be used in the termination criteria of an algorithm for solving nsNMF. In Section 3 we first propose four projected Barzilai–Borwein methods for solving the nonnegative least squares problems. We then incorporate them into an ANLS framework and obtain four APBB algorithms for nsNMF. In Section 4 we present some numerical results to illustrate the performance of these algorithms and compare them with the Lin and Lee–Seung methods. We also provide numerical results comparing the APBB2 method with the HALS/RRI method. In Section 5 we give some final remarks.

This project was started in late 2005, after Lin published his manuscript [25] and corresponding MATLAB code online. It has come to our attention recently that Zdunek and Cichocki [34] have independently considered using a projected Barzilai–Borwein approach to the NMF problem. There are differences between their approach and ours which we will discuss at the end of Subsection 3.2. The performance of these two approaches will be discussed in Section 5.

2. Preliminaries .

As mentioned in the introduction, the APBB algorithms alternately solve the subproblems for nsNMF using a projected Barzilai–Borwein method. In this section we first give a short introduction to the Barzilai–Borwein gradient method for optimization. We then present two sets of optimality conditions for the nsNMF problem (1.5) which do not use the Lagrange multipliers. They can be used in the termination criteria for an nsNMF algorithm.

2.1. The Barzilai-Borwein algorithm for optimization.

The gradient methods for the unconstrained optimization problem

$$\min_{x\in\mathbb{R}^n} f(x), \tag{2.1}$$

are iterative methods of the form

$$x^{k+1} = x^k + \lambda^k d^k, \tag{2.2}$$

where $d^k = -\nabla f(x^k)$ is the search direction and λ^k is the stepsize. The classic steepest descent method computes the stepsize using the exact line search

$$\lambda^k_{SD} = \mathrm{argmin}_{\lambda\in\mathbb{R}} f(x^k + \lambda d^k). \tag{2.3}$$

This method is usually very slow and not recommended for solving (2.1) (see for example, [18, 27]).

In 1988, Barzilai and Borwein [1] proposed a strategy of deriving the stepsize λ^k from a two–point approximation to the secant equation underlying quasi–Newton methods, which leads to the following choice of λ^k:

$$\lambda^k_{BB} = \frac{s^T s}{y^T s}, \tag{2.4}$$

with $s = x^k - x^{k-1}$ and $y = \nabla f(x^k) - \nabla f(x^{k-1})$. We shall call the resulting gradient method the Barzilai–Borwein (BB) method. In [1], the BB method is shown to converge

ETNA
Kent State University
http://etna.math.kent.edu

R–superlinearly for 2-dimensional convex quadratic objective functions. For general strictly convex quadratic objective functions, it has been shown that the BB method is globally convergent in [31] and its convergence rate is R–linear in [7].

To ensure the global convergence of the BB method when the objective function f is not quadratic, Raydan [32] proposes a globalization strategy that uses a nonmonotone line search technique of [11]. This strategy is based on an Amijo-type nonmonotone line search: Find $\alpha \in (0, 1]$ such that

$$f(x^k + \alpha\lambda_{BB}^k d^k) \leq \max_{0\leq j\leq \min(k,M)} f(x^{k-j}) + \gamma\alpha\lambda_{BB}^k \nabla f(x^k)^T d^k, \tag{2.5}$$

where $\gamma \in (0, 1/2)$, and define

$$x^{k+1} = x^k + \alpha\lambda_{BB}^k d^k. \tag{2.6}$$

In this line search $\alpha = 1$ is always tried first and used if it satisfies (2.5).

Under mild conditions, Raydan [32] shows that this globalized BB method is convergent. He also provides extensive numerical results showing that it substantially outperforms the steepest descent method. It is also comparable and sometimes preferable to a modified Polak–Ribiere (PR+) conjugate gradient method and the well known CONMIN routine for large-scale unconstrained optimization problems.

There have been several approaches to extend the BB method to optimization problems with simple constraints. In particular, Birgin, Martínez, and Raydan [3] propose two projected BB (PBB) methods for the following optimization problem on a convex set:

$$\min_{x\in\Omega} f(x), \tag{2.7}$$

where Ω be a convex subset of $\mathbb{R}^n$. They call their methods SPG1 and SPG2. The kth iteration of the SPG2 method is of the form

$$x^{k+1} = x^k + \alpha d^k, \tag{2.8}$$

where d^k is the search direction of the form

$$d^k = P_\Omega[x^k - \lambda_{BB}^k \nabla f(x^k)] - x^k, \tag{2.9}$$

where λ_{BB}^k is the BB stepsize which is calculated using (2.4) and P_Ω is the projection operator that projects a vector $x \in \mathbb{R}^n$ onto the convex region Ω. The stepsize $\alpha = 1$ is always tried first and used if it satisfies (2.5). Otherwise $\alpha \in (0, 1)$ is computed using a backtracking strategy until it satisfies the nonmonotone line search condition (2.5).

The kth iteration of the SPG1 method is of the form

$$x^{k+1} = P_\Omega[x^k - \alpha\lambda_{BB}^k \nabla f(x^k)] \tag{2.10}$$

and it uses a backtracking nonmonotone line search similar to the SPG2 method.

2.2. Optimality conditions for nsNMF. For a general smooth constrained optimization problem, its first order optimality conditions are expressed with the Lagrange multipliers (see, for example, [27]). However, the nsNMF problem 1.2 has only nonnegativity constraints. We can express its optimality conditions without using Lagrange multipliers.

By the definition of h_S and remark 1.3, the gradient of h_S in Problem 1.2 with respect to W is given by

$$\nabla_W h_S(W,H) = \nabla_W f(W,SH) = WSHH^TS^T - VH^TS^T \tag{2.11}$$

and the gradient of h_S with respect to H is given by

$$\nabla_H h_S(W,H) = \nabla_H f(WS,H) = S^TW^TWSH - S^TW^TV. \tag{2.12}$$

If (W,H) is a local minimizer of nsNMF 1.2, then according to the Karush–Kuhn–Tucker theorem for constrained optimization (see for example, [27]), there exist Lagrange multipliers matrices μ and ν such that

$$WSHH^TS^T - VH^TS^T = \mu, \qquad W.*\mu = 0, \qquad W \geq 0, \qquad \mu \geq 0 \tag{2.13}$$

and

$$S^TW^TWSH - S^TW^TV = \nu, \qquad H.*\nu = 0, \qquad H \geq 0, \qquad \nu \geq 0. \tag{2.14}$$

We can rewrite these conditions without using the Lagrange multipliers. This results in the following lemma.

LEMMA 2.1. *If (W,H) is a local minimizer for Problem nsNMF 1.2, then we have*

$$\min\{WSHH^TS^T - VH^TS^T, W\} = 0 \tag{2.15}$$

and

$$\min\{S^TW^TWSH - S^TW^TV, H\} = 0, \tag{2.16}$$

where $\min$ *is the entry-wise operation on matrices.*

In order to obtain the second set of optimality conditions, we need the concept of projected gradient. Let the feasible region of Problem 1.2 be $C = \{(W,H) : W \geq 0,\ H \geq 0\}$. Then for any $(W,H) \in C$, the projected gradient of h_S is defined as

$$[\nabla^P_W h_S(W,H)]_{ij} = \begin{cases} [\nabla_W h_S(W,H)]_{ij} & \text{if } W_{ij} > 0, \\ \min\{0, [\nabla_W h_S(W,H)]_{ij}\} & \text{if } W_{ij} = 0, \end{cases} \tag{2.17}$$

$$[\nabla^P_H h_S(W,H)]_{ij} = \begin{cases} [\nabla_H h_S(W,H)]_{ij} & \text{if } H_{ij} > 0, \\ \min\{0, [\nabla_H h_S(W,H)]_{ij}\} & \text{if } H_{ij} = 0. \end{cases} \tag{2.18}$$

From this definition, if W is a positive matrix, then $\nabla^P_W h_S(W,H) = \nabla_W h_S(W,H)$. Similarly, $\nabla^P_H h_S(W,H) = \nabla_H h_S(W,H)$ if H is positive.

We are now ready to state the second set of optimality conditions.

LEMMA 2.2. *If $(W,H) \in C$ is a local minimizer for Problem nsNMF 1.2, then we have that*

$$\nabla^P_W h_S(W,H) = 0 \tag{2.19}$$

and

$$\nabla_H^P h_S(W,H) = 0. \tag{2.20}$$

Each of the two sets of optimality conditions can be used in the termination conditions for an algorithm solving the nsNMF problem. In this paper, we will use the second set, that is, conditions (2.19) and (2.20) to terminate our APBB algorithms. We shall call a pair (W,H) satisfying (2.19) and (2.20) a stationary point of the nsNMF problem 1.2.

3. Alternating Projected Barzilai-Borwein algorithms for nsNMF .

3.1. Nonnegative least squares problem. As mentioned earlier, each of our APBB algorithms for the nsNMF problem 1.2 is based on a PBB method for solving Subproblems (1.14) and (1.15), alternately. Both subproblems are a nonnegative least squares (NLS) problem in matrix form which can be uniformly defined as in the following:

PROBLEM 3.1 (NLS). *Suppose that $B \in \mathbb{R}^{m,n}$ and $A \in \mathbb{R}^{m,r}$ are nonnegative. Find a matrix $X \in \mathbb{R}^{r,n}$ which solves*

$$\min_{X\geq 0} Q(X) = \frac{1}{2}\|B - AX\|_F^2. \tag{3.1}$$

The quadratic objective function Q can be rewritten as

$$Q(X) = \frac{1}{2}\langle B - AX, B - AX\rangle = \frac{1}{2}\langle B, B\rangle - \langle A^T B, X\rangle + \frac{1}{2}\langle X, (A^T A)X\rangle,$$

where $\langle \cdot,\cdot\rangle$ is the matrix inner product which is defined by

$$\langle S, T\rangle = \sum_{i,j} S_{ij}T_{ij} \tag{3.2}$$

for two matrices $S, T \in \mathbb{R}^{m,n}$.

The gradient of Q is of the form:

$$\nabla Q(X) = A^T AX - A^T B \tag{3.3}$$

and the projected gradient of Q for X in the feasible region $\mathbb{R}_+^{r,n} = \{X \in \mathbb{R}^{r,n} : X \geq 0\}$ is of the form

$$[\nabla^P Q(X)]_{ij} = \begin{cases} [\nabla Q(X)]_{ij} & \text{if } X_{ij} > 0, \\ \min\{0, [\nabla Q(X)]_{ij}\} & \text{if } X_{ij} = 0. \end{cases} \tag{3.4}$$

The NLS problem is a convex optimization problem. Therefore, each of its local minimizers is also a global minimizer. Moreover, the Karush–Kuhn–Tucker conditions are both necessary and sufficient for X to be a minimizer of (3.1). We give a necessary and sufficient condition for a minimizer of NLS Problem 3.1 using the projected gradient $\nabla^P Q(X)$ here, which can be used in the termination criterion for an algorithm for the NLS problem.

LEMMA 3.2. *The matrix $X \in \mathbb{R}_+^{r,n}$ is a minimizer of the NLS problem 3.1 if and only if*

$$\nabla^P Q(X) = 0. \tag{3.5}$$

The solution of NLS Problem 3.1 can be categorized into three approaches. The first approach is to reformulate it in vector form using vectorization:

$$\min_{X\geq 0}\frac{1}{2}\|B-AX\|_F^2=\min_{\mathrm{vec}(X)\geq 0}\frac{1}{2}\|\mathrm{vec}(B)-(I\otimes A)\mathrm{vec}(X)\|_2^2, \tag{3.6}$$

where $\otimes$ is the Kronecker product and $\mathrm{vec}(\cdot)$ is the vectorization operation. However, this approach is not very practical in the context of NMF due to the large size of the matrix $I\otimes A$. In addition, this approach does not use a nice property of the NMF problem: It can be decoupled as in (1.11).

The second approach is to decouple the problem into the sum of NLS problems in vector form as

$$\min_{X\geq 0}\frac{1}{2}\|B-AX\|_F^2=\sum_{j=1}^{n}\min_{X_j\geq 0}\frac{1}{2}\|B_j-AX_j\|_2^2 \tag{3.7}$$

and then solve for each individual column X_j.

The third approach is to solve Problem 3.1 in matrix form directly. For instance, Lin uses this approach to solve (3.1) by a projected gradient method. He calls his method *nlssubprob* in [25]. Our solution of (3.1) also uses this approach.

We comment that the second and third approaches have a similar theoretical complexity. In implementation, however, the third approach is preferred. This is particularly true if MATLAB is used since using loops in MATLAB is time consuming.

3.2. Projected Barzilai-Borwein algorithms for NLS . We present four projected Barzilai-Borwein algorithms for the NLS problem which will be called PBBNLS algorithms. We first notice that solving the optimization problem 3.1 is equivalent to solving the following convex NLS problem:

$$\min_{X\geq 0}\tilde{Q}(X)=-\langle A^TB,X\rangle+\frac{1}{2}\langle X,A^TAX\rangle. \tag{3.8}$$

The PBBNLS algorithms solve Problem (3.8). This strategy can avoid the computation of $\langle B,B\rangle$ and save time if B is of large size. The gradient of $\tilde{Q}$ and projected gradient of $\tilde{Q}$ in $\mathbb{R}_+^{r,n}$ are the same as those of Q, defined in (3.3) and (3.4), respectively.

Our first algorithm PBBNLS1 uses a backtracking strategy in the nonmonotone line search which is similar to the one suggested by Birgin, Martinez, and Raydan [3] in their algorithms SPG1. In PBBNLS1, if the next iteration generated by the projected BB method $P[X^k-\lambda^k\nabla\tilde{Q}(X^k)]$ does not satisfy the nonmontone line search condition (3.10), then a stepsize $\alpha\in(0,1)$ is computed using a backtracking approach with $P[X^k-\alpha\lambda^k\nabla\tilde{Q}(X^k)]$ satisfying (3.10). This algorithm is given below:

ALGORITHM 3.3 (PBBNLS1).
Set the parameters:

$$\gamma = 10^{-4}, M = 10, \lambda_{max} = 10^{20}, \lambda_{min} = 10^{-20}.$$

Step 1. **Initialization:** Choose the initial approximation $X^0 \in \mathbb{R}_+^{r,n}$ and the tolerance $\rho > 0$. Compute

$$\lambda^0 = 1/(\max_{(i,j)} \nabla \tilde{Q}(X^0)).$$

Set $k = 0$.
Step 2. **Check Termination:** If the projected gradient norm satisfies

$$\|\nabla^P \tilde{Q}(X^k)\|_F < \rho, \tag{3.9}$$

stop. Otherwise,
Step 3. **Line Search:**
3.1. Set $D^k = P[X^k - \lambda^k \nabla \tilde{Q}(X^k)] - X^k$ and $\alpha = 1$.
3.2. If

$$\tilde{Q}(P[X^k - \alpha\lambda^k \nabla \tilde{Q}(X^k)]) \le \max_{0 \le j \le \min(k,M)} \tilde{Q}(X^{k-j}) + \gamma\alpha\langle \nabla \tilde{Q}(X^k), D^k\rangle, \tag{3.10}$$

set

$$X^{k+1} = P[X^k - \alpha\lambda^k \nabla \tilde{Q}(X^k)] \tag{3.11}$$

and go to Step 4. Otherwise set

$$\alpha = \alpha/4 \tag{3.12}$$

and repeat Step 3.2.
Step 4. **Computing the Barzilai–Borwein Stepsize:**
Compute

$$s^k = X^{k+1} - X^k \text{ and } y^k = \nabla \tilde{Q}(X^{k+1}) - \nabla \tilde{Q}(X^k).$$

If $\langle s^k, y^k\rangle \le 0$, set $\lambda^{k+1} = \lambda_{max}$, else set

$$\lambda^{k+1} = \min(\lambda_{max}, \max(\lambda_{min}, \langle s^k, s^k\rangle / \langle s^k, y^k\rangle)). \tag{3.13}$$

Step 5. Set k = k+1 and go to Step 2.

Our second and third NLS solvers PBBNLS2[1] and PBBNLS3 differ from PBBNLS1 in their use of line search. The steps of PBBNLS2 and PBBNLS3 are the same as the ones in PBBNLS1, except Step 3. Therefore we will only state Step 3 in PBBNLS2 and PBBNLS3.

[1]The code for PBBNLS2 is available at: `http://www.math.uconn.edu/~neumann/nmf/PBBNLS2.m`

In PBBNLS2, the step size α by the line search is computed along the projected BB gradient direction $D^k = P[X^k - \lambda^k \nabla \tilde{Q}(X^k)] - X^k$ using the backtracking strategy.

ALGORITHM 3.4 (PBBNLS2).
Step 3. **Line Search:**
3.1. Set $D^k = P[X^k - \lambda^k \nabla \tilde{Q}(X^k)] - X^k$ and $\alpha = 1$.
3.2. If

$$\tilde{Q}(X^k + \alpha D^k) \leq \max_{0 \leq j \leq \min(k,M)} \tilde{Q}(X^{k-j}) + \gamma \alpha \langle \nabla \tilde{Q}(X^k), D^k \rangle, \tag{3.14}$$

set

$$X^{k+1} = X^k + \alpha D^k \tag{3.15}$$

and go to Step 4.
Otherwise set $\alpha = \alpha/4$ and repeat Step 3.2.

In PBBNLS3, if $\alpha = 1$ does not satisfy condition (3.10), then an exact type line search along the projected BB gradient direction D^k is used. Since the objective function $\tilde{Q}$ is quadratic, the stepsize can be obtained by

$$\alpha = -\frac{\langle \nabla \tilde{Q}(X^k), D^k \rangle}{\langle D^k, A^T A D^k \rangle}. \tag{3.16}$$

ALGORITHM 3.5 (PBBNLS3).
Step 3. **Line Search:**
3.1. Set $D^k = P[X^k - \lambda^k \nabla \tilde{Q}(X^k)] - X^k$.
3.2. If

$$\tilde{Q}(X^k + D^k) \leq \max_{0 \leq j \leq \min(k,M)} \tilde{Q}(X^{k-j}) + \gamma \langle \nabla \tilde{Q}(X^k), D^k \rangle, \tag{3.17}$$

set $X^{k+1} = X^k + D^k$ and go to Step 4.
Otherwise compute α using (3.16). Set $X^{k+1} = P[X^k + \alpha D^k]$ and go to Step 4.

Our fourth NLS solver PBBNLS4 is a projected BB method without using a line search. This method is closest to the original BB method. The reason we include this method here is that Raydan [31] has proved that the original BB method is convergent for unconstrained optimization problems with strictly quadratic objective functions and it often performs well for quadratic unconstrained optimization problems (see [10] and the references therein). Note that the NLS problem (3.1) has a convex quadratic objective function.

ALGORITHM 3.6 (PBBNLS4).
No Step 3: line search.

In the four PBBNLS methods we employ the following strategies to save computational time:

- To avoid repetition, the constant matrices A^TA and A^TB are calculated once and used throughout.
- The function value of $\tilde{Q}(X^k+\alpha D^k)$ in (3.10) may need to be evaluated several times in each iteration. We rewrite this function in the following form:

$$\begin{aligned}\tilde{Q}(X^k+\alpha D_k) &= -\langle A^TB, X^k+\alpha D_k\rangle + \frac{1}{2}\langle X^k+\alpha D_k, A^TA(X^k+\alpha D_k)\rangle \\ &= \tilde{Q}(X^k) + \alpha\langle \nabla\tilde{Q}(X^k), D^k\rangle + \frac{1}{2}\alpha^2\langle D^k, A^TAD_k\rangle. \end{aligned} \tag{3.18}$$

  In this form, only one calculation of $\langle\nabla\tilde{Q}(X^k), D^k\rangle$ and $\langle D^k, A^TAD_k\rangle$ is needed in the backtracking line search, which brings down the cost of the computation.

REMARK 3.7. In [34], Zdunek and Cichocki have also considered to use a projected Barzilai-Borwein method to solve the NLS problem (3.1). Here are differences between their approach and ours:

1). In the use of BB stepsizes: our approach considers X as one long vector, i.e., the columns of X are stitched into a long vector, although in real computation, this is not implemented explicitly. As a result, we have a scalar BB stepsize.

On the other hand, their approach uses the decoupling strategy as illustrated in (3.7). For each such problem

$$\min \frac{1}{2}\|B_j - AX_j\|_2^2,$$

they use a BB stepsize. Therefore, they use a vector of n BB stepsizes.

2). In the use of line search: our method uses a nonmonotone line search and we have a scalar steplength α due to the use of line search. Our strategy always uses the BB stepsize if it satisfies the nonmonotone line search condition, that is, using $\alpha = 1$.

The preference of using the BB stepsize whenever possible has been discussed in the literature in optimization, see, for example, [10].

The line search of Zdunek and Cichocki [34] is not a nonmonotone line search and it does not always prefer a BB steplength. Again, their method generates a vector of line search steplengths.

3.3. Alternating Projected Barzilai-Borwein algorithms for nsNMF. We are now ready to present the APBB algorithms for nsNMF which are based on the four PBBNLS methods respectively. We denote these methods by APBBi, for $i = 1, 2, 3,$ or 4. In APBBi, the solver PBBNLSi is used to solve the NLS Subproblems (3.20) and (3.21) alternately[2].

[2]The code for APBB2 is available at: `http://www.math.uconn.edu/~neumann/nmf/APBB2.m`

ALGORITHM 3.8 (APBBi, $i = 1, 2, 3,$ or 4).

```
Step 1. Initialization. Choose the smoothing parameter $\theta \in [0,1]$, the tolerance $\epsilon > 0$, and
the initial nonnegative matrices $W^0$ and $H^0$. Set $k = 0$.
Step 2. Check Termination: If projected gradient norm satisfies
    (3.19)    $PGN^k < \epsilon * PGN^0,$
where $PGN^k = \|[\nabla^C_W h_S(W^k, H^k), (\nabla^C_H h_S(W^k, H^k))^T]\|_F$, stop. Otherwise
Step 3. Main Iteration:
  3.1. Update $W$: Use PBBNLSi to solve
    (3.20)    $\min_{W \geq 0} f(W, SH^k),$
    for $W^{k+1}$.
  3.2. Update $H$: Use PBBNLSi to solve
    (3.21)    $\min_{H \geq 0} f(W^{k+1}S, H),$
    for $H^{k+1}$.
Step 4. Set $k = k + 1$ and go to Step 2.
```

REMARK 3.9. At early stages of the APBB methods, it is not cost effective to solve Subproblems (3.20) and (3.21) very accurately. In the implementation of APBBi we use an efficient strategy which was first introduced by Lin [25]. Let ρ_W and ρ_H be the tolerances for (3.20) and (3.21) by PBBNLSi, respectively. This strategy initially sets

$$\rho_W = \rho_H = \max(0.001, \epsilon)\|[\nabla_W h_S(W^0, H^0), (\nabla_H h_S(W^0, H^0))^T]\|_F.$$

If PBBNLSi solves (3.20) in one step, then we reduce ρ_W by setting $\rho_W = 0.1\rho_W$. A similar method is used for ρ_H.

4. Numerical results .

We have tested the APBB methods and compared them with the Lee–Seung method and the Lin method on both randomly generated problems and a real life problem using the ORL database [36]. We have also compared the APBB2 method with the HALS/RRI method. The numerical tests were done on a Dell XPS 420 desktop with 3 GB of RAM and a 2.4 GHz Core Quad CPU running Windows Vista. All the algorithms were implemented in MATLAB, where the code of Lin's method can be found in [25]. In this section, we report the numerical results.

4.1. Comparison of PBBNLSi methods and Lin's nlssubprob method on NLS problems.

The overall performance of our APBBi methods ($i = 1, 2, 3, 4$) and Lin's method depends on the efficiency of the NLS solvers PBBNLSi and *nlssubprob* respectively.

We tested the PBBNLSi methods and Lin's *nlssubprob* method on a variety of randomly generated NLS problems. For each problem, we tested each algorithm using same randomly generated starting point X^0, with tolerance $\rho = 10^{-4}$. We also used 1000 as the maximal number of iterations allowed for each algorithm.

ETNA
Kent State University
http://etna.math.kent.edu

The numerical results are given in Table 4.1. In this table, the first column lists how the test problems and the initial points X^0 were generated. The integer nit denotes the number of iterations needed when the termination criterion (3.9) was met. The notation ≥ 1000 is used to indicate that the algorithm was terminated because the number of iterations reached 1000 but (3.9) had not been met. The numbers, PGN, RN, and CPUTIME denote the final projected gradient norm $\|\nabla^P Q(X^k)\|_F$, the final value of $\|B - AX^k\|_F$, and the CPU time used at the termination of each algorithm, respectively.

TABLE 4.1
Comparison of PBBNLSi and nlssubprob on randomly generated NLS problems.

Problem	Algorithm	nit	CPU Time	PGN	RN
	nlssubprob	631	0.390	0.000089	39.341092
A = rand(100,15)	PBBNLS1	103	0.062	0.000066	39.341092
B = rand(100,200)	PBBNLS2	93	0.047	0.000098	39.341092
X^0=rand(15,200)	PBBNLS3	150	0.078	0.000097	39.341092
	PBBNLS4	129	0.047	0.000070	39.341092
	nlssubprob	$\geq$ 1000	7.535	0.023547	107.023210
A = rand(300,50)	PBBNLS1	196	1.108	0.000080	107.023210
B = rand(300,500)	PBBNLS2	177	0.952	0.000098	107.023210
X^0=rand(50,500)	PBBNLS3	210	1.154	0.000099	107.023210
	PBBNLS4	195	0.733	0.000033	107.023210
	nlssubprob	$\geq$ 1000	7.472	0.321130	288.299174
A = rand(2000,50)	PBBNLS1	138	0.780	0.000012	288.299174
B = rand(2000,500)	PBBNLS2	162	0.905	0.000080	288.299174
X^0=rand(50,500)	PBBNLS3	143	0.827	0.000032	288.299174
	PBBNLS4	$\geq$ 1000	3.526	565.334391	293.426207
	nlssubprob	$\geq$ 1000	168.044	2.666440	487.457979
A = rand(3000,100)	PBBNLS1	278	37.440	0.000097	487.457908
B = rand(1000,3000)	PBBNLS2	308	37.674	0.000083	487.457908
X^0=rand(100,3000)	PBBNLS3	373	49.234	0.000095	487.457908
	PBBNLS4	388	30.483	0.000052	487.457908
	nlssubprob	$\geq$ 1000	169.479	5.569313	698.384960
A = rand(2000,100)	PBBNLS1	368	49.983	0.000028	698.384923
B = rand(2000,3000)	PBBNLS2	298	37.035	0.000096	698.384923
X^0=rand(100,3000)	PBBNLS3	311	41.044	0.000036	698.384923
	PBBNLS4	275	22.215	0.000028	698.384923
Exactly Solvable	nlssubprob	817	0.468	0.000089	0.000037
A = rand(100,15)	PBBNLS1	37	0.016	0.000008	0.000003
B = A * rand(15,200)	PBBNLS2	44	0.016	0.000006	0.000003
X^0=rand(15,200)	PBBNLS3	44	0.016	0.000041	0.000012
	PBBNLS4	40	0.016	0.000090	0.000016
Exactly Solvable	nlssubprob	$\geq$ 1000	6.942	6.906346	1.303933
A = rand(300,50)	PBBNLS1	57	0.312	0.000022	0.000005
B = A * rand(50,500)	PBBNLS2	64	0.328	0.000070	0.000017
X^0=rand(50,500)	PBBNLS3	63	0.328	0.000084	0.000024
	PBBNLS4	61	0.234	0.000018	0.000006
Exactly Solvable	nlssubprob	$\geq$ 1000	7.129	29.389267	0.997834
A = rand(2000,50)	PBBNLS1	34	0.218	0.000095	0.000007
B = A * rand(50,500)	PBBNLS2	33	0.218	0.000031	0.000003
X^0=rand(50,500)	PBBNLS3	38	0.250	0.000051	0.000004
	PBBNLS4	33	0.172	0.000002	0.000000

We observe from Table 4.1 that the PBBNLS1, PBBNLS2, and PBBNLS3 methods perform similarly. Each of them is significantly faster than the *nlssubprob* method. The performance of PBBNLS4 is somewhat inconsistent: If it works, it is the fastest method. However, we notice that it does not always perform well which can be seen from the results of problem # 3 of the table.

We also tested the same set of problems using other initial points X^0 and other tolerance levels ranging from $\rho = 10^{-1}$ to $\rho = 10^{-8}$. The relative performance of the PBBNLSi methods and the *nlssubprob* method is similar to the one illustrated in Table 4.1.

As the APBBi methods and Lin's method use PBBNLSi or *nlssubprob* to solve the Subproblems (3.20) and (3.21) alternately, we expect the APBBi methods to be fatser than Lin's method. We will illustrate this in the remaining subsections.

4.2. Comparison of the APBBi methods and Lin's method on randomly generated NMF problems. In order to assess the performance of the APBBi ($i = 1, 2, 3, 4$) methods for solving the NMF problem we tested them on four types of randomly generated NMF problems: small and medium size, large size, exactly factorable, and difficult. We also tested the multiplicative method of Lee–Seung and Lin's method and discovered that the Lee–Seung method is outperformed by Lin's method substantially. Therefore, we only report the comparison of the APBBi methods with Lin's method in this subsection.

In the implementation of APBBi methods, we allowed PBBNLSi to iterate at most 1000 iterations for solving each Subproblem (3.20) or (3.21). The same maximum number of iterations was used in Lin's subproblem solver *nlssubprob*.

The stopping criteria we used for all algorithms were the same. The algorithm was terminated when one of the following two conditions was met:

(i). The approximate projected gradient norm satisfies:

$$\|[\nabla_W^C h_S(W^k, H^{k-1}), (\nabla_H^C h_S(W^k, H^k))^T]\|_F < \epsilon \cdot PGN^0, \tag{4.1}$$

where ϵ is the tolerance and PGN^0 is the initial projected gradient norm. The recommended tolerance value is $\epsilon = 10^{-7}$ or $\epsilon = 10^{-8}$ for general use of the APBBi methods. If high precision in the NMF factorization is needed, we suggest to use even smaller tolerance such as $\epsilon = 10^{-10}$.

(ii). The maximum allowed CPU time is reached.

The numerical results with tolerance $\epsilon = 10^{-7}$ and maximum allowed CPU time of 600 seconds are reported in Tables 4.2–4.5. In these tables, the first column lists how the the test problems and the initial pair (W^0, H^0) were generated. The value CPUTIME denotes the CPU time used when the termination criterion (4.1) was met. The notation ≥ 600 is used to indicate that the algorithm was terminated because the CPU time reached 600 but (4.1) had not been met. The numbers PGN, RN, ITER, and NITER denote the final approximate projected gradient norm $\|[\nabla_W^C h_S(W^k, H^{k-1}), (\nabla_H^C h_S(W^k, H^k))^T]\|_F$, the final value of $\|V - W^k H^k\|_F$, the number of iterations, and the total number of sub–iterations for solving (3.20) and (3.21) at the termination of each algorithm, respectively.

Table 4.2 illustrates the performance of APBBi methods and Lin's method for small and medium size NMF problems. From this table we observe that for small size NMF problems, the APBBi methods are competitive to Lin's method. However, as the problem size grows, the APBBi methods converge significantly faster than Lin's method in terms of CPUTIME.

Table 4.3 demonstrates the behavior of the five algorithms on large size NMF problems. We observe from this table that clearly APBB4 is rather slow comparing to the APBBi ($i = 1, 2, 3$) and Lin methods for large problems. This can be explained by the inconsistent performance of PBBNLS4 for solving NLS problems. We also observe that APBB2 is

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.2
Comparison of APBBi and Lin on small and medium size NMF problems.

Problem	Algorithm	iter	niter	CPUTime	PGN	RN
	Lin	359	6117	1.248	0.00028	15.925579
V = rand(100,50)	APBB1	646	9818	1.716	0.000108	15.927577
W^0 = rand(100,10)	APBB2	439	5961	1.014	0.000207	15.934012
H^0 =rand(10,50)	APBB3	475	5660	1.076	0.000278	15.934005
	APBB4	562	12256	1.591	0.000181	15.910972
	Lin	576	15996	6.708	0.001299	33.885342
V = rand(100,200)	APBB1	375	6455	2.683	0.000905	33.901354
W^0 = rand(100,15)	APBB2	631	10561	4.274	0.001	33.890555
H^0 =rand(15,200)	APBB3	499	8784	3.588	0.00157	33.902192
	APBB4	430	13751	3.775	0.001386	33.883501
	Lin	1150	85874	302.736	0.03136	94.856419
V = rand(300,500)	APBB1	101	2156	7.878	0.020744	95.092793
W^0 = rand(300,40)	APBB2	110	2429	8.362	0.028711	95.070843
H^0 =rand(40,500)	APBB3	103	2133	7.722	0.028945	95.102193
	APBB4	118	7770	16.084	0.031105	95.011186
	Lin	2651	81560	169.292	0.010645	77.82193
V = rand(1000,100)	APBB1	543	7613	17.94	0.009247	77.815999
W^0 = rand(1000,20)	APBB2	488	6953	15.444	0.010538	77.82096
H^0 =rand(20,100)	APBB3	425	5252	13.088	0.005955	77.822524
	APBB4	351	10229	16.224	0.005741	77.830637
	Lin	177	7729	10.624	0.032641	161.578261
V = abs(randn(300,300))	APBB1	183	3350	4.508	0.016747	161.647472
W^0 = abs(randn(300,20))	APBB2	176	3215	4.15	0.017717	161.690957
H^0 =abs(randn(20,300))	APBB3	194	2996	4.118	0.024201	161.686909
	APBB4	209	6351	6.068	0.032873	161.621572
	Lin	934	13155	62.182	0.127067	199.116691
V = abs(randn(300,500))	APBB1	646	7874	36.379	0.138395	199.168636
W^0 = abs(randn(300,40))	APBB2	577	8068	33.353	0.08876	199.069136
H^0 =abs(randn(40,500))	APBB3	792	9768	43.805	0.100362	199.123688
	APBB4	438	14787	35.256	0.075076	199.163277
	Lin	1438	24112	64.99	0.04327	162.820691
V = abs(randn(1000,100))	APBB1	235	2822	7.254	0.044061	162.845427
W^0 = abs(randn(1000,20))	APBB2	247	2929	7.176	0.036261	162.845179
H^0 =abs(randn(20,100))	APBB3	258	2809	7.472	0.046555	162.841468
	APBB4	247	5193	9.516	0.043423	162.901371

slightly faster than APBB1 and APBB3 for this set of problems and all these three methods are significantly faster than Lin's method.

Table 4.4 compares the APBBi methods and Lin's method on some difficult NMF problems. We observe that the APBBi methods substantially outperform Lin's method on this set of problems.

In Table 4.5 we compare the five algorithms on some exactly factorable NMF problems. We observe that in this case, APBB1, APBB2, and APBB3 methods are substantially faster than Lin's method and APBB4.

In summary, based on the performance of the APBBi methods and Lin's method on the four types of randomly generated NMF problems using the aforementioned stopping criteria, we conclude that APBB1, APBB2, and APBB3 methods converge significantly faster than Lin's method for large size, difficult, and exactly factorable problems in terms of CPU time used. They are also competitive with Lin's method on small size problems and faster than Lin's method on medium size problems. Based on the overall performance, it seems that APBB1 and APBB2 perform similarly and both of them are a little better than APBB3. The

TABLE 4.3
Comparison of APBBi and Lin on large size NMF problems.

Problem	Algorithm	iter	niter	CPUTime	PGN	RN
	Lin	542	25420	≥ 600	0.293784	388.404591
V = rand(2000,1000)	APBB1	63	1789	48.36	0.169287	388.753789
W^0 = rand(2000,40)	APBB2	60	1732	43.93	0.176915	388.788783
H^0 =rand(40,1000)	APBB3	70	1708	45.989	0.215956	388.755657
	APBB4	337	46797	≥ 600	1.625634	388.389849
	Lin	40	3960	≥ 600	21.122063	665.880519
V = rand(3000,2000)	APBB1	95	2049	367.148	1.648503	665.212812
W^0 = rand(3000,100)	APBB2	80	1936	308.664	1.67074	665.233771
H^0 =rand(100,2000)	APBB3	94	1993	343.53	1.623925	665.20267
	APBB4	10	10020	≥ 600	32322.8139	711.14501
	Lin	11	1659	≥ 600	84.633034	848.130974
V = rand(2000,5000)	APBB1	29	1194	469.36	7.132444	839.962863
W^0 = rand(2000,200)	APBB2	30	1267	457.395	9.070003	839.917857
H^0 =rand(200,5000)	APBB3	32	1470	535.473	6.558566	839.754882
	APBB4	11	3458	≥ 600	631245.173	898.640696
	Lin	45	1925	292.892	5.742485	1394.828534
V = abs(randn(2000,3000))	APBB1	52	1243	185.704	7.227765	1394.594862
W^0 = abs(randn(2000,100))	APBB2	51	1418	187.513	8.661069	1394.68873
H^0 =abs(randn(100,3000))	APBB3	55	1267	186.156	8.624533	1394.694934
	APBB4	22	11117	≥ 600	17712.80413	1421.28342

TABLE 4.4
Comparison of APBBi and Lin on difficult NMF problems.

Problem	Algorithm	iter	niter	CPUTime	PGN	RN
	Lin	6195	198110	26.255	0.00181	718.456033
V = csc(rand(50,50))	APBB1	2669	32219	4.103	0.001332	718.456035
W^0 = rand(50,10)	APBB2	2749	32679	3.994	0.001183	718.456035
H^0 =rand(10,50)	APBB3	2644	31259	3.853	0.001061	718.456035
	APBB4	3048	30851	2.98	0.001445	718.456034
	Lin	22093	1717514	243.861	0.003822	1165.453591
V = csc(rand(100,50))	APBB1	21991	135041	26.115	0.002869	1165.453605
W^0 = rand(100,10)	APBB2	22410	136280	24.898	0.002869	1165.453605
H^0 =rand(10,50)	APBB3	22401	118545	23.338	0.001773	1165.453605
	APBB4	21483	172171	22.729	0.002435	1165.453604
	Lin	26370	645478	291.067	0.004951	3525.44525
V = csc(rand(100,200))	APBB1	6905	105236	47.471	0.004517	3525.445263
W^0 = rand(100,15)	APBB2	7541	117095	51.309	0.004901	3525.445263
H^0 =rand(15,200)	APBB3	9636	156118	72.587	0.004927	3525.445262
	APBB4	9761	100586	38.704	0.004854	3525.445261
	Lin	25868	509206	≥ 600	0.032144	9990.273219
V = csc(rand(200,300))	APBB1	6226	97030	99.185	0.024480	9990.273226
W^0 = rand(200,20)	APBB2	6178	97649	97.298	0.021796	9990.273226
H^0 =rand(20,300)	APBB3	6185	122773	121.447	0.024276	9990.273225
	APBB4	7683	102010	85.645	0.023077	9990.273222

performance of APBB4 is inconsistent: Sometimes it works well and sometimes it does not perform well. Therefore APBB4 is not recommended for solving NMF problems.

As a final remark in this subsection, it can be seen that Lin's method can sometimes give a slightly smaller final value of $\|V - W^k H^k\|_F$ than the APBBi methods when each algorithm terminates due to condition (4.1) being met. Several factors can attribute to this phenomenon. First, these algorithms can converge to different stationary points (W^*, H^*) of the NMF problem. The values of $\|V - W^* H^*\|_F$ at these points can be slightly different. Second, even if each of them converges to a stationary point (W^*, H^*) with the same $\|V - W^* H^*\|_F$

TABLE 4.5
Comparison of APBBi and Lin on exactly factorable NMF problems.

Problem	Algorithm	iter	niter	CPUTime	PGN	RN
Exactly Factorable	Lin	761	143759	9.781	0.00002	0.000146
V = rand(20,5)*rand(5,50)	APBB1	837	17861	1.232	0.000014	0.001601
W^0 = rand(20,5)	APBB2	869	18722	1.17	0.000014	0.001395
H^0 =rand(5,50)	APBB3	1045	20461	1.295	0.000011	0.001514
	APBB4	215	55065	2.387	0.000013	0.000385
Exactly Factorable	Lin	837	115740	15.709	0.000078	0.000573
V = rand(100,10)*rand(10,50)	APBB1	524	13180	2.122	0.000076	0.005784
W^0 = rand(100,10)	APBB2	560	13633	1.981	0.000075	0.005586
H^0 =rand(10,50)	APBB3	597	14917	2.246	0.000075	0.00545
	APBB4	304	76860	6.958	0.000082	0.000877
Exactly Factorable	Lin	2735	243572	122.586	0.000366	0.003565
V = rand(100,15)*rand(15,200)	APBB1	1926	54640	23.837	0.000371	0.04548
W^0 = rand(100,15)	APBB2	2217	65976	25.381	0.000357	0.045799
H^0 =rand(15,200)	APBB3	1879	49998	19.906	0.000333	0.045776
	APBB4	2065	279023	75.972	0.000368	0.016646
Exactly Factorable	Lin	1790	320943	$\geq$ 600	0.032549	0.34627
V = rand(200,30) *rand(30,500)	APBB1	1687	67167	148.763	0.00209	0.24903
W^0 = rand(200,30)	APBB2	1640	64987	125.971	0.002076	0.250934
H^0 =rand(30,500)	APBB3	1985	108521	208.87	0.002295	0.257472
	APBB4	682	632573	$\geq$ 600	1.283378	0.409974

value (for example, when each converges to a global minimizer of the NMF problem), they can still have different final $\|V - W^kH^k\|_F$ values because the termination condition (4.1) measures how close (W^k, H^k) gets to a stationary point. It is possible that a pair of (W^k, H^k) is closer to a stationary point than another pair but has a slightly larger $\|V - W^kH^k\|_F$ value as the function $\|V - WH\|_F$ is nonlinear. Third, Lin's method is a monotonically decreasing method and the APBBi methods are nonmonotone methods. As Lin's method often takes significantly longer time to satisfy termination condition (4.1) than APBBi methods do, Lin's method may result in slightly smaller objective function value than the other methods at the given tolerance level.

4.3. Comparison of the APBB2, Lee-Seung, and Lin methods on the ORL database.

As APBB1, APBB2, and APBB3 have similar performance based on the results in Subsection 4.2, we chose APBB2 to test and compare it with the Lee–Seung and Lin methods on a real life problem: the reduced size ORL face database (see [36, 17]). In this subsection, we report these experiments.

The reduced size ORL database consists of 400 facial images of 40 people, each person with 10 different orientation, expressions and brightness/contrast levels. The image dimensions are 56×46. The matrix V whose columns represent these images has $56 \times 46 = 2576$ rows and 400 columns. The original NMF or the nsNMF is applied to reconstruct images from this database by using a small rank r. In our experiments, we used $r = 49$. When NMF or nsNMF is used, $W \in \mathbb{R}^{2576,49}$ gives the basis images and $H \in \mathbb{R}^{49,400}$ is the encoding matrix.

In our first experiment, we used the APBB2, Lin, and Lee–Seung methods to solve the NMF problem resulting from the ORL database. We set the tolerance $\epsilon = 10^{-8}$ and let each algorithm run for 8 seconds. The facial images were then reconstructed using the final W^k and H^k obtained by each algorithm at its termination.

In Figure 4.1, some sample original images are given. In Figure 4.2, the reconstructed

FIGURE 4.1. *Sample original images from ORL database.*

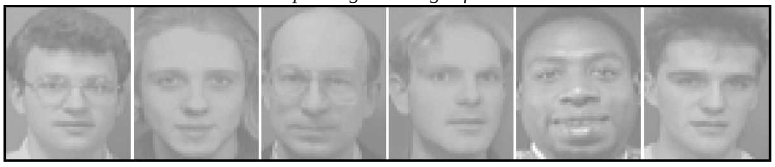
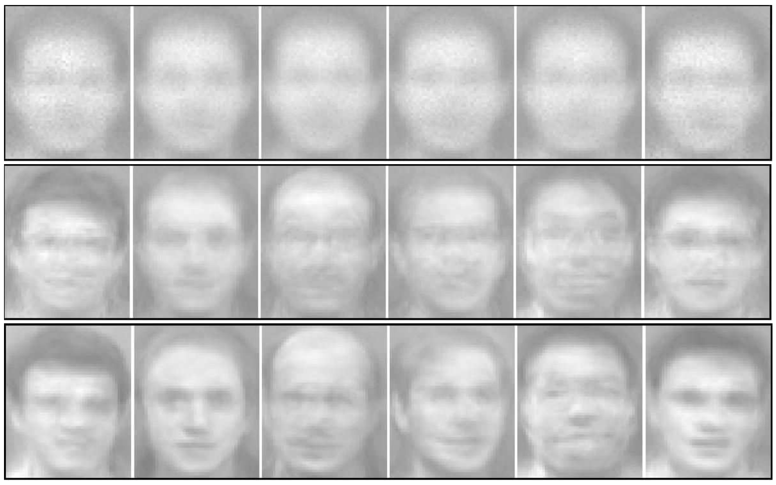


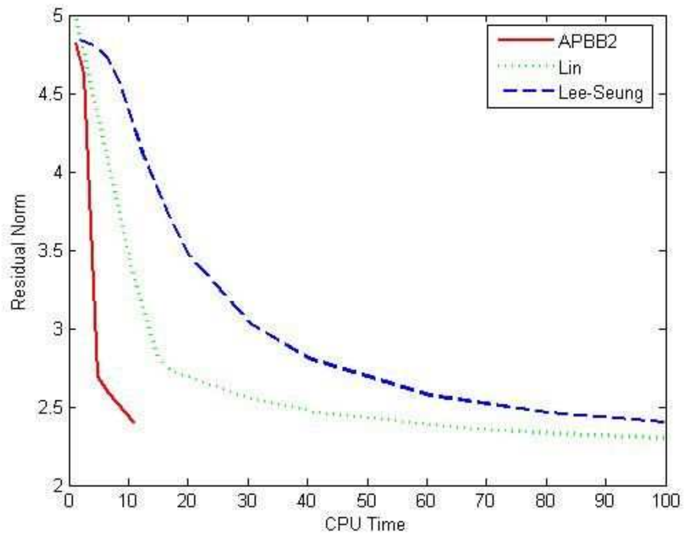
FIGURE 4.2. *Reconstruction of images with $r = 49$ and time = 8 seconds: From top to bottom by Lee-Seung, Lin and APBB2.*



images by running each of Lee–Seung, Lin, and APBB2 for 8 seconds are given. From Figures 4.1 and 4.2, it is clear that APBB2 gives better quality reconstructed images than Lin and Lee–Seung do. We can also observe that Lin obtains better reconstruction than Lee–Seung does. These observations can be explained by the fact that APBB2 converges faster than Lin and Lin faster than Lee–Seung. In Figure 4.3, we plot the residual norm $RN = \|V - WH\|_F$ versus CPU time for each algorithm for this experiment using a range of CPU time between 0 and 100 seconds. Note that APBB2 terminated after about 11 seconds for the tolerance $\epsilon = 10^{-8}$. From Figure 4.3, we can see that for this database, to attain a similar level of reconstruction quality to APBB2 using 8 seconds, Lin needs about 40 seconds and Lee–Seung needs much longer time.

We tried to answer four questions when we carried out the second experiment on the ORL database. The first two questions are: Can the APBB2, Lin, and Lee–Seung methods produce sparser basis images when they are applied to the nsNMF problem with a positive smoothing parameter θ if they are allowed to run sufficient amount of computational time? Can the APBB2 method generate sparse basis images and reconstruct images which are comparable to the ones by the Lin and Lee–Seung but use significantly less time? We will use Figures 4.4, 4.5, 4.6, and Table 4.6 to answer these questions. The next two questions are: How good and sparse are the basis images generated by each of the three methods when they are allowed to run a relatively short period of time? How good are the images reconstructed by each of the three methods when they are allowed to run a relatively short period of time? Figures 4.7,

FIGURE 4.3. *Residual Norm versus CPU Time for Lee-Seung, Lin and APBB2 using* $\epsilon = 10^{-8}$.



4.8, and Table 4.7 answer these questions.

In Figures 4.4, 4.5, 4.6, and Table 4.6, we report the results of applying the Lee–Seung, Lin, and APBB2 methods to solve the nsNMF problem resulted from the ORL database. These figures and the table were constructed by setting the tolerance $\epsilon = 10^{-10}$, running the Lee-Seung and the Lin algorithms for 600 seconds and the APBB2 algorithm for 150 seconds, and using the smoothing parameter $\theta = 0$ and $\theta = 0.7$ respectively. Note that when $\theta = 0$, the nsNMF reduces to the original NMF.

Figure 4.4 gives the basis images generated by Lee–Seung, Lin, and APBB2 for solving the nsNMF problem with $\theta = 0$ (i.e., the original NMF). Figure 4.5 gives the basis images generated by Lee–Seung, Lin, and APBB2 for solving the nsNMF problem with $\theta = 0.7$. Figure 4.6 gives the reconstructed sample images by Lee–Seung, Lin, and APBB2 for solving the nsNMF problem with $\theta = 0.7$.

We observe from Figures 4.4 and 4.5 that when $\theta = 0.7$ is used, all three algorithms can produce much sparser basis images than they do with $\theta = 0$. This confirms that the nsNMF can increase the ability of learning by parts of the original NMF (see [30]). If we examine this example more carefully, however, we can see that both APBB2 and Lin methods give sparser basis images than the Lee–Seung method. We also observe from Figure 4.6 that the APBB2 method obtains reconstructed images which are comparable to the ones by the Lin method and by the Lee-Seung method. In summary, these figures show that in this example, the APBB2 method can reconstruct images and generate sparse basis images which are comparable to the Lin and Lee–Seung methods but uses considerably less time.

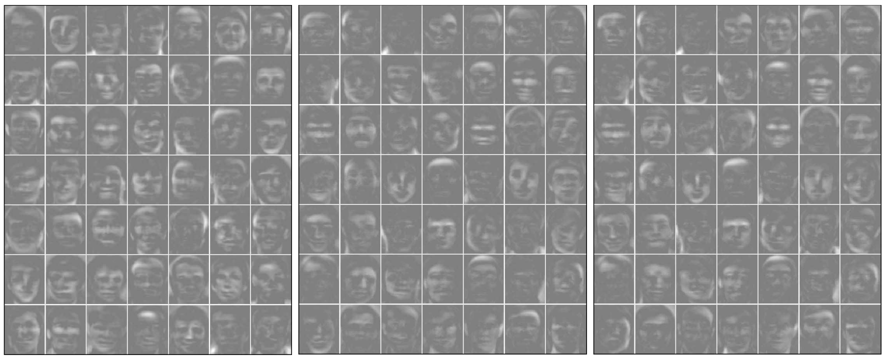
These observations can be explained by the numerical results in Table 4.6. In this table, the sparseness of a matrix A is measured by

$$spA = \frac{\text{Number of Zero Entries in A}}{\text{Total Number of Entries in A}}, \tag{4.2}$$

where A_{ij} is considered a zero entry if $|A_{ij}| < 10^{-6}$ in our experiment. From this table, we

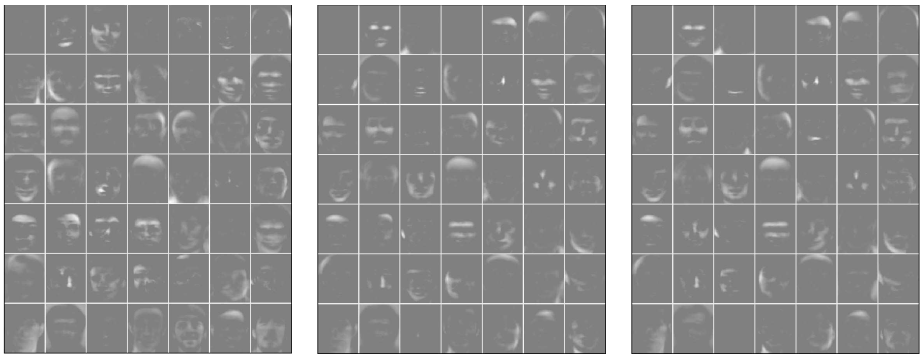
ETNA
Kent State University
http://etna.math.kent.edu

FIGURE 4.4. *Basis images generated by three algorithms for solving original NMF (i.e. $\theta = 0$ in nsNMF) from ORL database using $\epsilon = 10^{-10}$.*



(a) Lee-Seung (600 seconds) (b) Lin (600 seconds) (c) APBB2 (150 seconds)

FIGURE 4.5. *Basis images generated by three algorithms for solving nsNMF from ORL database with $\theta = 0.7$ and using $\epsilon = 10^{-10}$.*



(a) Lee-Seung (600 seconds) (b) Lin (600 seconds) (c) APBB2 (150 seconds)

can see that when $\theta = 0.7$ is used, each of the three algorithms can increase the sparseness of W (and H) substantially comparing to $\theta = 0$. We also observe that both the Lin and the APBB2 methods give sparser W and sparser H than the Lee-Seung method does. In addition, the APBB2 method obtains slightly smaller residual norm $\|V - WSH\|_F$ than the Lin and Lee–Seung methods. Here we would like to emphasize that these numerical results were obtained by running APBB2 method for 150 seconds and the Lin and Lee–Seung methods for 600 seconds.

In Figure 4.7, we give the basis images generated by the Lee–Seung, Lin, and APBB2 methods for solving the nsNMF with $\theta = 0.7$ and using 30 seconds of maximum allowed CPU time. We can see that the APBB2 method can produce sparser basis images than Lin's method in this case. Moreover, the Lee–Seung method can not produce any sparseness in the basis matrix W within 30 seconds of CPU time, as illustrated in Table 4.7.

Finally, Figure 4.8 gives the reconstructed sample images by Lee–Seung, Lin, and APBB2 using the nsNMF with $\theta = 0.7$ and 30 seconds of maximum allowed CPU time. We observe

FIGURE 4.6. *Reconstructed images by three algorithms for solving nsNMF from ORL database with $\theta = 0.7$ and using $\epsilon = 10^{-10}$: From top to bottom by Lee-Seung (600 seconds), Lin (600 Seconds), and APBB2 (150 seconds).*

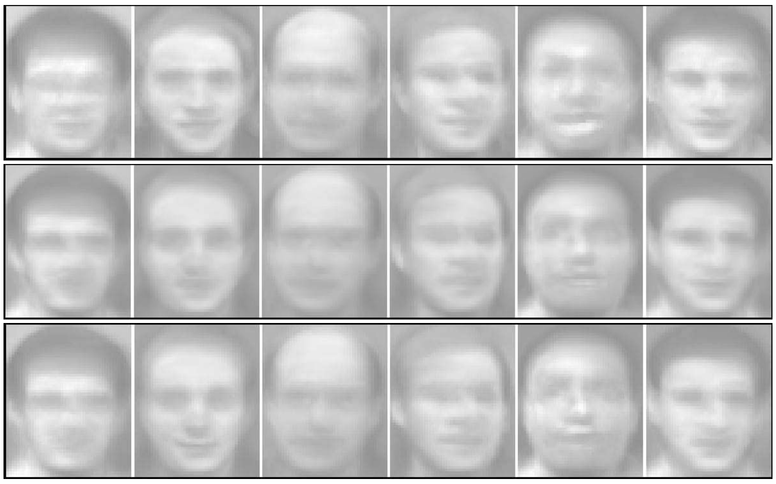
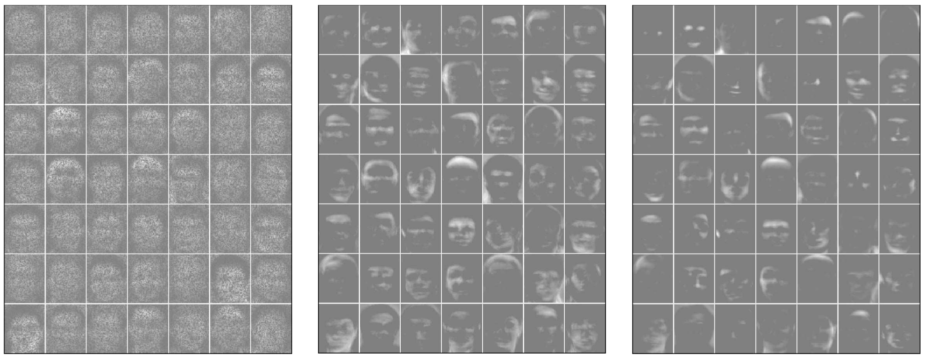


TABLE 4.6
Comparison of APBB2 (150 seconds), Lin (600 seconds), and Lee-Seung (600 seconds) for solving nsNMF from ORL database using tolerance $\epsilon = 10^{-10}$.

θ	Algorithm	PGN	RN	Sparseness in W	Spareness in H
0	APBB2	0.021793	2.189993	39	35
	Lin	0.021746	2.211365	40	35
	Lee-Seung	10.838066	2.247042	34	20
0.7	APBB2	0.002024	2.859032	81	56
	Lin	0.022953	2.877591	81	55
	Lee-Seung	16.703386	2.928300	60	32

FIGURE 4.7. *Basis images generated by three algorithms for solving nsNMF from ORL database with $\theta = 0.70$, using $\epsilon = 10^{-10}$ and Maximum Allowed CPU Time = 30 seconds.*



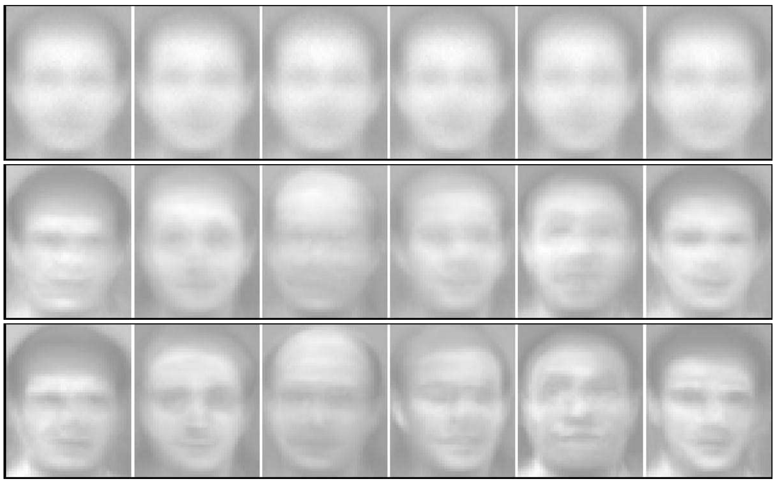
(a) Lee-Seung (b) Lin (c) APBB2

that the APBB2 method produces better reconstruction than Lin's method. It seems that when this relatively short period of time is used, the Lee–Seung method has not been able to generate identifiable reconstructed images.

TABLE 4.7
Comparison of three algorithms for solving nsNMF from ORL database with $\theta = 0.70$, using $\epsilon = 10^{-10}$ and Maximum Allowed CPU Time = 30 seconds.

Algorithm	PGN	RN	Sparseness in W	Sparseness in H
APBB2	0.023983	2.979195	82	50
Lin	2.296475	3.497314	71	32
Lee–Seung	10.894495	4.698601	0	0

FIGURE 4.8. *Reconstructed images by three algorithms for solving nsNMF from ORL database with $\theta = 0.70$, using $\epsilon = 10^{-10}$ and Maximum Allowed CPU Time = 30 seconds: From top to bottom by Lee–Seung, Lin, and APBB2.*



4.4. Comparison of the APBB2 and HALS/RRI methods. We implemented the HALS/RRI method in MATLAB. In our implementation, we followed Algorithm 7 of Ho [14, page 69]. The original Algorithm 7 of Ho may result in zero vectors h_t or w_t. To avoid this type of rank-deficient approximation, we used a strategy introduced in Ho's thesis [14, expression (4.5) on page 72].

For both APBB2 and HALS/RRI, we used similar stopping criteria described in Subsection 4.2, except for that the approximate gradient in condition (4.1) was changed to the gradient:

$$\|[\nabla^C_W h_S(W^k, H^k), (\nabla^C_H h_S(W^k, H^k))^T]\|_F < \epsilon \cdot PGN^0. \tag{4.3}$$

Since both the APBB2 and HALS/RRI methods are reasonably fast, we set the maximum allowed CPU time to be 100 seconds.

We first tested the APBB2 and HALS/RRI methods on three groups of randomly generated NMF problems:

- **Group 1** (Problems P_1–P_{11}):
  $V = \text{rand}(m, n)$, $W_0 = \text{rand}(m, r)$, $H_0 = \text{rand}(r, n)$.
- **Group 2** (Problems P_{12}–P_{16}):
  $V = \text{csc}(\text{rand}(m, n))$, $W_0 = \text{rand}(m, r)$, $H_0 = \text{rand}(r, n)$.
- **Group 3** (Problems P_{17}–P_{23}):
  $V = \text{rand}(m, r) * \text{rand}(r, n)$, $W_0 = \text{rand}(m, r)$, $H_0 = \text{rand}(r, n)$.

The numerical results of these experiments are reported in Table 4.8. From this table

TABLE 4.8
Comparison of APBB2 and HALS/RRI on Randomly Generated NMF Problems using $\epsilon = 10^{-6}$.

Problem	Algorithm	iter	niter	CPUTime	PGN	RN
$P_1, m = 20$	HALS/RRI	1157		0.655	0.000282	6.704077
$n = 50, r = 5$	APBB2	186	2323	0.156	0.000285	6.704077
$P_2, m = 100$	HALS/RRI	1925		4.165	0.003081	15.965216
$n = 50, r = 10$	APBB2	450	6664	1.201	0.003079	15.964402
$P_3, m = 50$	HALS/RRI	652		1.373	0.003241	15.907085
$n = 100, r = 10$	APBB2	689	11946	1.872	0.003263	15.911952
$P_4, m = 100$	HALS/RRI	3888		36.301	0.025348	31.949834
$n = 200, r = 20$	APBB2	616	12077	6.583	0.020062	31.945314
$P_5, m = 300$	HALS/RRI	2322		100.028	0.793530	98.254061
$n = 500, r = 30$	APBB2	1186	23309	65.442	0.185916	98.218000
$P_6, m = 300$	HALS/RRI	1019		100.059	0.627775	91.692878
$n = 500, r = 50$	APBB2	945	19856	100.059	1.305468	91.719337
$P_7, m = 100$	HALS/RRI	4493		100.012	0.212449	77.953204
$n = 1000, r = 20$	APBB2	942	19239	25.491	0.063113	77.958453
$P_8, m = 2000$	HALS/RRI	141		100.199	230.102735	388.743544
$n = 1000, r = 40$	APBB2	136	3510	100.667	8.724113	388.426680
$P_9, m = 3000$	HALS/RRI	3		180.462	2444.874883	700.382102
$n = 2000, r = 100$	APBB2	22	588	104.318	347.053280	667.647866
$P_{10}, m = 2000$	HALS/RRI	3		181.273	2608.736609	696.220345
$n = 3000, r = 100$	APBB2	21	734	103.585	272.621912	666.717715
$P_{11}, m = 2000$	HALS/RRI	2		497.253	32270.733529	968.339290
$n = 5000, r = 200$	APBB2	9	270	125.690	2110.725410	855.042449
$P_{12}, m = 20$	HALS/RRI	33167		18.377	0.005618	275.408370
$n = 50, r = 5$	APBB2	17906	234816	16.536	0.004837	275.408370
$P_{13}, m = 100$	HALS/RRI	30164		64.475	0.031324	1672.570769
$n = 50, r = 10$	APBB2	844	15720	2.356	0.033597	1672.570786
$P_{14}, m = 50$	HALS/RRI	21451		45.568	0.019479	1575.266873
$n = 100, r = 10$	APBB2	1656	22326	4.009	0.012779	1575.266875
$P_{15}, m = 100$	HALS/RRI	4841		27.675	0.052223	4351.003826
$n = 200, r = 15$	APBB2	4978	67865	34.039	0.046538	4351.003826
$P_{16}, m = 200$	HALS/RRI	7323		100.012	0.625107	10736.711898
$n = 300, r = 20$	APBB2	5210	98518	100.012	0.339669	10736.711903
$P_{17}, m = 20$	HALS/RRI	1596		0.952	0.000163	0.000454
$n = 50, r = 5$	APBB2	565	9100	0.577	0.000163	0.000420
$P_{18}, m = 100$	HALS/RRI	2570		5.538	0.000971	0.002506
$n = 50, r = 10$	APBB2	947	22529	3.432	0.000949	0.001882
$P_{19}, m = 50$	HALS/RRI	4324		9.656	0.000878	0.002886
$n = 100, r = 10$	APBB2	957	27701	4.165	0.000868	0.002168
$P_{20}, m = 100$	HALS/RRI	5008		46.535	0.004903	0.007428
$n = 200, r = 20$	APBB2	927	32966	17.893	0.004786	0.009190
$P_{21}, m = 300$	HALS/RRI	2340		100.043	3.766695	4.606791
$n = 500, r = 30$	APBB2	920	41285	100.075	0.746586	2.755773
$P_{22}, m = 300$	HALS/RRI	1044		100.075	42.645934	25.055011
$n = 500, r = 50$	APBB2	374	21068	100.184	1.750924	1.555805
$P_{23}, m = 2000$	HALS/RRI	155		100.745	1004.622468	244.083901
$n = 100, r = 40$	APBB2	110	5340	100.496	49.765120	69.009367

we can see that the APBB2 method has a comparable performance with the HALS/RRI method for small or medium scale problems and the APBB2 method becomes faster than the HALS/RRI method as the the size of the NMF problem increases. We also observe that the APBB2 method outperforms the HALS/RRI method on large-scale problems.

A natural question is, can the APBB2, Lin, and HALS/RRI methods produce a sparse nonnegative factorization if the matrix V has sparse nonnegative factorizations? To answer this, we tested the APBB2, Lin, and HALS/RRI methods on the CBCL face database (see

ETNA
Kent State University
http://etna.math.kent.edu

[35]). The CBCL database consists of 2429 facial images of dimensions 19×19. The matrix V representing this database has 361 rows and 2429 columns. In our experiments, we used $r = 49$.

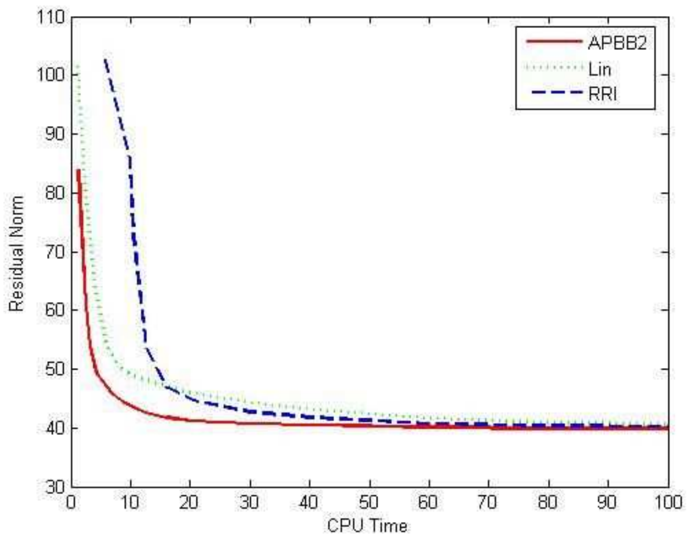
As shown by Lee and Seung [22], the data matrix V of the CBCL database has sparse nonnegative factorizations when $r = 49$. Moreover, it has been observed that the Lee–Seung algorithm (1.7) and (1.8) can produce a sparse factorization if it is allowed to run a sufficient amount of time.

We report our experiments of the APBB2, Lin, and HALS/RRI methods on the CBCL database in Figures 4.9, 4.10, and 4.11. In Figure 4.9, we plot the residual norm $RN = \|V - WH\|_F$ versus CPU time for each algorithm using a range of CPU time between 0 and 100 seconds. In Figure 4.10 and Figure 4.11, we plot the sparseness of W of H versus CPU time respectively. We used the sparseness measurement given in (4.2), where A_{ij} is considered a zero if $|A_{ij}| < 10^{-6}$.

We can see from Figures 4.10 and 4.11 that all the three algorithms can generate a sparse nonnegative factorization in this example. We comment that the huge sparseness in W and H given by the HALS/RRI method at early stages is due to the fact that this method generates some zero vectors h_t or w_t initially. The large residual norm of the HALS/RRI method at early stages confirms this. The rank-deficient problem of the HALS/RRI method is remedied after the strategy of Ho [14, page 72] is in full play.

We also observe from these three figures that the APBB2 method can obtain fairly good residual norm and sparseness in W and H using less CPU time than both the Lin and HALS/RRI methods.

FIGURE 4.9. *Residual Norm versus CPU Time for HALS/RRI, Lin and APBB2 using* $\epsilon = 10^{-7}$.



5. Final remarks . We have proposed four algorithms for solving the nonsmooth nonnegative matrix factorization (nsNMF) problems. Each of our algorithms alternately solves a nonnegative linear least squares subproblem in matrix form using a projected Barzilai–Borwein method with a nonmonotone line search or no line search. These methods can also

FIGURE 4.10. *Sparseness of W versus CPU Time for HALS/RRI, Lin and APBB2 using $\epsilon = 10^{-7}$.*

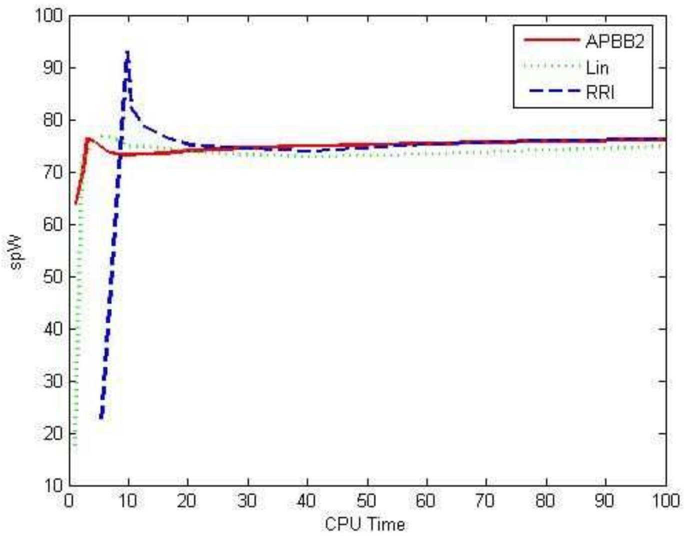
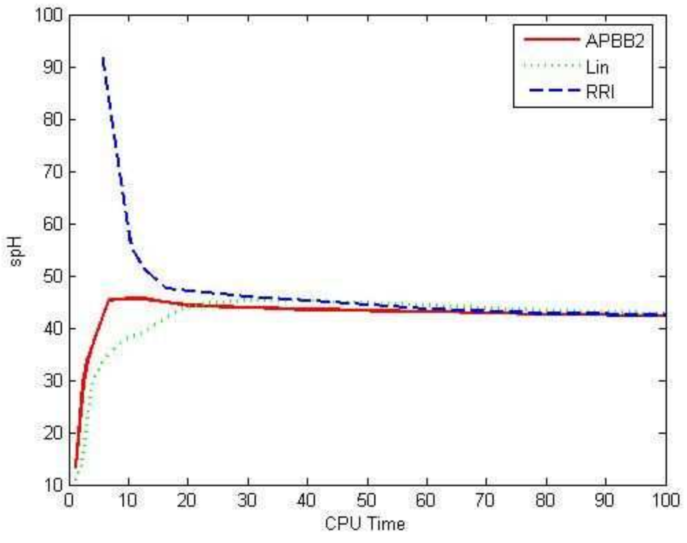


FIGURE 4.11. *Sparseness of H versus CPU Time for HALS/RRI, Lin and APBB2 using $\epsilon = 10^{-7}$.*



be used to solve the original NMF problem by setting the smoothing parameter $\theta = 0$ in nsNMF. We have tested and compared our algorithms with the projected gradient method of Lin on a variety of randomly generated NMF problems. Our numerical results show that three of our algorithms, namely, APBB1, APBB2, and APBB3, are significantly faster than Lin's algorithm for large-scale, difficult, or exactly factorable NMF problems in terms of CPU time used. We have also tested and compared our APBB2 method with the multiplicative algorithm of Lee and Seung and Lin's algorithm for solving the nsNMF problem that resulted from the ORL face database using both $\theta = 0$ and $\theta = 0.7$. The experiments show that when $\theta = 0.7$ is used, the APBB2 method can produce sparse basis images and reconstructed images which are comparable to the ones by the Lin and Lee–Seung methods but in considerably less time. They also show that the APBB2 method can reconstruct better quality images and obtain sparser basis images than the methods of Lee–Seung and Lin when each method is allowed to run for a short period of time. We have also tested the APBB2 method and the HALS/RRI method and the comparison shows that the APBB2 method can outperform the HALS/RRI method on large-scale problems. These numerical tests show that the APBBi ($i = 1, 2, 3$) methods, especially APBB1 and APBB2, are suitable for solving large-scale NMF or nsNMF problems and in particular, if only a short period of computational time is available.

So far we have emphasized the computational efficiency of NMF algorithms. If high accuracy rather than computational time is a priority, we can run an APBBi ($i = 1, 2, 3$) method by setting tolerance $\epsilon = 0$ and using relatively large values of maximum allowed CPU time and maximum allowed number of iterations to terminate the algorithm. Alternatively, taking the APBB2 method as an example, this can be done in the MATLAB code APBB2.m by choosing $tol = 0$, replacing the command line

```
    tolW = max(0.001,tol)*initgrad_norm;  tolH = tolW;
```

with the following

```
    tolW=10ˆ-8; tolH=tolW;
```

and using suitable MaxTime and MaxIter values. We comment that 10^{-8} is only a reference which can be replaced by smaller values. This implementation solves the NLS subproblems (3.20) and (3.21) very accurately from the beginning.

An interesting question is: How well our APBBi ($i = 1, 2, 3$) methods perform when compared to an algorithm resulting from the projected Barzilai–Borwein approach of Zdunek and Cichocki [34] (It is called GPRS–BB in [33]) to solve the NLS subproblem (3.1). We coded the GPRS-BB method in MATLAB and incorporated it in the ANLS framework as we did for PBBNLSi methods. Our preliminary numerical tests on some randomly generated medium size NMF problems show that the APBBi methods are considerably faster in terms of CPU time used. More experiments are needed to make a comprehensive comparison.

Another interesting question is to extend the projected BB idea to other variants of NMF problems, such as the symmetric–NMF (see [8]) and semi-NMF (see [9]). Our preliminary results show that the projected BB approach is very promising when applied to these two classes of NMF problems.

Acknowledgments. The authors wish to express their gratitude to the referees for their very helpful comments and suggestions.

REFERENCES

[1] J. BARZILAI AND J.M. BORWEIN, *Two-point step size gradient method*, IMA J. Numer. Anal., 8, (1988), pp. 141–148.
[2] M. BERRY, M. BROWNE, A. LANGVILLE, P. PAUCA, AND R. J. PLEMMONS, *Algorithms and Applications for Approximate Nonnegative Matrix Factorization*, Comput. Statist. Data Anal., 52 (2007), pp. 155–173.
[3] E.G. BIRGIN, J.M. MARTÍNEZ, AND M. RAYDAN, *Nonmonotone spectral projected gradient methods on convex sets*, SIAM J. Optim., 10 (2000), pp. 1196–1211.

[4] D. P. BERTSEKAS, *Projected Newton methods for optimization problems with simple constraints*, SIAM J. Control Optim., 20 (1982), pp. 221–246.

[5] A. CICHOCKI, R. ZDUNEK, S. AMARI, *Nonnegative matrix and tensor factorization*, Signal Processing Magazine, IEEE, 25 (2008), pp. 142–145.

[6] ———, *Hierarchical ALS algorithms for nonnegative matrix and 3D tensor factorization*, in ICA07, London, UK, September 9-12, Lecture Notes in Comput. Sci., vol. 4666, Springer, Heidelberg, 2007, pp. 169-176.

[7] Y. DAI AND L. LIAO, *R-Linear convergence of the Barzilai-Borwein gradient method*, IMA J. Numer. Anal., 22 (2002), pp. 1–10.

[8] C. DING, X. HE, AND H. SIMON, *On the equivalence of non-negative matrix factorization and spectral clustering*, Proc. SIAM Int'l Conf. Data Mining (SDM'05), eds. H. Kargupta and J. Srivastava, 2005, pp. 606-610.

[9] C. DING, T. LI, AND M. I. JORDAN, *Convex and semi-nonnegative matrix factorizations*, Technical Report 60428, Lawrence Berkeley National Laboratory, 2006.

[10] R. FLETCHER, *On the Barzilai-Borwein method*, in Optimization and Control with Applications (Appl. Optim.), 96, L. Qi, K. Teo, and X. Yang eds., Springer, New York, 2005, pp. 235–256.

[11] L. GRIPPO, F. LAMPARIELLO, AND S. LUCIDI, *A nonmonotone line search technique for Newton's method*, SIAM J. Numer. Anal., 23 (1986), pp. 707–716.

[12] L. GRIPPO AND M. SCIANDRONE, *On the convergence of the block nonlinear gauss-seidel method under convex constraints*, Oper. Res., 26 (2000), pp. 127-136.

[13] J. HAN, L. HAN, M. NEUMANN, AND U. PRASAD, *On the rate of convergence of the Image Space Reconstruction Algorithm*, Oper. Matrices, 3 (2009), pp. 41-58.

[14] N. D. HO, *Nonnegative Matrix Factorization Algorithms and Applications*, Ph.D. thesis, FSA/INMA - Département d'ingénierie mathématique, Université Catholique de Louvain, 2008.

[15] N. D. HO, P. VAN DOOREN, AND V.D. BLONDEL, *Descent methods for nonnegative matrix factorization*, CESAME, Université Catholique de Louvain, 2008.

[16] P. O. HOYER, *Nonnegative sparse coding*, in Proc. of the 2002 12th IEEE Workshop on Neural Networks for Signal Processing, Martigny, Switzerland, 2002, pp. 557-565.

[17] ———, *Nonnegative matrix factorization with sparseness constraint*, J. of Mach. Learn. Res., 5 (2004), pp. 1457 - 1469.

[18] C. T. KELLEY, *Iterative Methods for Optimization*, SIAM, Philadelphia, 1999.

[19] H. KIM AND H. PARK, *Non-negative matrix factorization based on alternating non-negative constrained least squares and active set method*, SIAM J. Matrix Anal. Appl., 30 (2008), pp. 713-730.

[20] D. KIM, S. SRA, AND I.S. DHILLON, *Fast Newton type methods for the least square nonnegative matrix approximation problem*, Proc. SIAM Int'l Conf. Data Mining (SDM'07), eds. S. Parthasarathy and Bing Liu, 2007, pp. 343–354.

[21] D. D. LEE AND H. S. SEUNG, *Unsupervised learning by convex and conic coding*, Advances in Neural Information Processing Systems, 9 (1997), pp. 515–521.

[22] ———, *Learning the parts of the objects by non-negative matrix factorization*, Nature, 401 (1999), pp. 788–791.

[23] ———, *Algorithms for non-negative matrix factorization*, Advances in Neural Information Processing Systems, Proceedings of the 2000 Conference, NIPS, MIT Press, 2001, pp. 556–562.

[24] S. LI, X. HOU, AND H. ZHANG, *Learning spatially localized, parts-based representation*, in Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition, 2001.

[25] C. J. LIN, *Projected gradient methods for non-negative matrix factorization*, Neural Comput., 19 (2007), pp. 2756-2779.

[26] ———, *On the convergence of multiplicative update algorithms for nonnegative matrix factorization*, IEEE Trans. Neural Networks, 18 (2007), pp. 1589-1596.

[27] J. NOCEDAL AND S. WRIGHT, *Numerical Optimization*, Springer, New York, 1999.

[28] P. PAATERO, *Least squares formulation of robust non-negative factor analysis*, Chemometr. Intell. Lab., 37 (1987), pp. 23–35.

[29] P. PAATERO AND U. TAPPER, *Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values*, Environmetrics, 5 (1994), pp. 111–126.

[30] A. PASCUAL-MONTANO, J.M. CARAZO, K. KOCHI, D. LEHMANN, AND R. D. PASCUAL-MARQUI, *Nonsmooth nonnegative matrix factorization (nsNMF)*, IEEE Trans. Pattern Anal. Mach. Intell., 28 (2006), pp. 403–415.

[31] M. RAYDAN, *On the Barzilai- Borwein choice of steplength for the gradient method*, IMA J. Numer. Anal., 13 (1993), pp. 321–326.

[32] ———, *The Barzilai and Borwein gradient method for the large-scale unconstrained minimization problem*, SIAM J. Optim., 7 (1997), pp. 26–33.

[33] R. ZDUNEK AND A. CICHOCKI, *Nonnegative matrix factorization with Quasi-Newton optimization*, in Proc. ICAISC 8th International Conference, Zakopane, Poland, June 25-29, eds. J. G. Carbonell and J.

Siekmann, Lecture Notes in Comput. Sci., vol. 4029, Springer, Heidelberg, 2006, pp. 870–879.
[34] ———, *Application of selected projected gradient algorithms to nonnegative matrix factorization*, Comput. Intell. Neurosci., to appear.
[35] CBCL Face Database. Center for Biological and Computational Learning at MIT and MIT, 2000. Available at: `http://cbcl.mit.edu/software-datasets/FaceData2.html`
[36] ORL Database of Faces. AT&T Laboratories, Cambridge, 1999. Available at: `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`.