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. 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, nonsmooth 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

\[
V \approx WH.
\]

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
solve
\begin{equation}
\min f(W, H) = \frac{1}{2} \|V - WH\|_F^2 \text{ s.t. } W \geq 0, \ H \geq 0,
\end{equation}
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:
\begin{equation}
V \approx WH,
\end{equation}
where \( S \in \mathbb{R}^{r,r} \) is a “smoothing” matrix of the form
\begin{equation}
S = (1 - \theta)I + \frac{\theta}{r} J,
\end{equation}
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 \( \theta \) 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 \( \theta \) can control the sparseness of \( W \) and \( H \). The larger \( \theta \) 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
\begin{equation}
\min h_S(W, H) = \frac{1}{2} \|V - WSH\|_F^2 \text{ s.t. } W \geq 0, \ H \geq 0,
\end{equation}
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
\begin{equation}
h_S(W, H) = f(W, SH) = f(WS, H).
\end{equation}

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:
\begin{equation}
W^{k+1} = W^k \ast (V(H^k)^T)/(W^kH^k(H^k)^T),
\end{equation}
\begin{equation}
H^{k+1} = H^k \ast ((W^{k+1})^TV)/((W^{k+1})^TW^{k+1}H^k).
\end{equation}

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:

\[
W^{k+1} = \arg\min_{W \geq 0} f(W, H^k) = \arg\min_{W \geq 0} \frac{1}{2} \| V^T - (H^k)^T W^T \|_F^2,
\]

and

\[
H^{k+1} = \arg\min_{H \geq 0} f(W^{k+1}, H) = \arg\min_{H \geq 0} \frac{1}{2} \| V - W^{k+1} H \|_F^2.
\]

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

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

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

\[
\min_{h \geq 0} \frac{1}{2} \| R_t - w_i h \|_F^2
\]
Alternating Projected Barzilai–Borwein Methods for Nonnegative Matrix Factorization

and

\[(1.13) \min_{\mathbb{W} \geq 0} \frac{1}{2} \| \mathbb{R}_t - \mathbb{w}_t \mathbb{h}_t \|_F^2 \]

for \(t = 1, 2, \ldots, r\), where \(\mathbb{R}_t = \mathbb{V} - \sum_{i \neq t} \mathbb{w}_i \mathbb{h}_i\). 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:

\[(1.14) \quad \mathbb{W}^{k+1} = \arg\min_{\mathbb{W} \geq 0} f(\mathbb{W}, \mathbb{S}\mathbb{H}^k) = \arg\min_{\mathbb{W} \geq 0} \frac{1}{2} \| \mathbb{V}^T - (\mathbb{S}\mathbb{H}^k)^T \mathbb{W} \|^2_F \]

and

\[(1.15) \quad \mathbb{H}^{k+1} = \arg\min_{\mathbb{H} \geq 0} f(\mathbb{W}^k S, \mathbb{H}) = \arg\min_{\mathbb{H} \geq 0} \frac{1}{2} \| \mathbb{V} - (\mathbb{W}^k S) \mathbb{H} \|^2_F.\]

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 \(\mathbb{H}^k\) by \(\mathbb{S}\mathbb{H}^k\) in (1.7) and \(\mathbb{W}^{k+1}\) by \(\mathbb{W}^k S\) in (1.8). The resulting Lee–Seung type multiplicative updates for nsNMF are:

\[(1.16) \quad \mathbb{W}^{k+1} = \mathbb{W}^k \ast ((\mathbb{V}(\mathbb{S}\mathbb{H}^k)^T)/(\mathbb{W}^k \mathbb{S}\mathbb{H}^k(S\mathbb{H}^k)^T)) \]

and

\[(1.17) \quad \mathbb{H}^{k+1} = \mathbb{H}^k \ast ((\mathbb{W}^k S)^T \mathbb{V})/(((\mathbb{W}^k S)^T \mathbb{W}^k S \mathbb{H}^k)).\]

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 \(\mathbb{W}\) and \(\mathbb{H}\) 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

\[ \min_{x \in \mathbb{R}^n} f(x), \]

are iterative methods of the form

\[ x^{k+1} = x^k + \lambda^k d^k, \]

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

\[ \lambda^k_{SD} = \arg\min_{\lambda \in \mathbb{R}} f(x^k + \lambda d^k). \]

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 \( \lambda^k \) from a two–point approximation to the secant equation underlying quasi–Newton methods, which leads to the following choice of \( \lambda^k \):

\[ \lambda^k_{BB} = \frac{s^T s}{y^T s}, \]

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
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

\[
\begin{align*}
 f(x_k + \alpha \lambda_{BB}^k d^k) &\leq \max_{0 \leq j \leq \min(k, M)} f(x_{k-j}) + \gamma \alpha \lambda_{BB}^k \nabla f(x_k)^T d^k ,
\end{align*}
\]

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

\[
 x_{k+1} = x_k + \alpha \lambda_{BB}^k d^k.
\]

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:

\[
\begin{align*}
 \min_{x \in \Omega} f(x),
\end{align*}
\]

where \( \Omega \) be a convex subset of \( \mathbb{R}^n \). They call their methods SPG1 and SPG2. The \( k \)th iteration of the SPG2 method is of the form

\[
 x_{k+1} = x_k + \alpha d^k,
\]

where \( d^k \) is the search direction of the form

\[
 d^k = P_\Omega [x_k - \lambda_{BB}^k \nabla f(x_k)] - x_k,
\]

where \( \lambda_{BB}^k \) is the BB stepsize which is calculated using (2.4) and \( P_\Omega \) is the projection operator that projects a vector \( x \in \mathbb{R}^n \) onto the convex region \( \Omega \). 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 \( k \)th iteration of the SPG1 method is of the form

\[
 x_{k+1} = P_\Omega [x_k - \alpha \lambda_{BB}^k \nabla f(x_k)]
\]

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

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

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

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

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 $\mu$ and $\nu$ such that

$$WSHH^TS^T - VH^TS^T = \mu, \quad W \geq 0, \quad \mu \geq 0$$

and

$$S^TW^TWSH - S^TW^TV = \nu, \quad H \geq 0, \quad \nu \geq 0.$$  

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

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

and

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

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

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

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

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

$$\nabla^p_W h_S(W, H) = 0.$$

$$\nabla^p_H h_S(W, H) = 0.$$
and

\( \nabla^P h_S(W, H) = 0. \)

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

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

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

\[
\langle S, T \rangle = \sum_{i,j} S_{ij} T_{ij}
\]

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

The gradient of \(Q\) is of the form:

\[
\nabla Q(X) = A^T AX - A^T B
\]

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

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

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

\[
\nabla^P Q(X) = 0.
\]
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:

\[
\min_{X \geq 0} \frac{1}{2} \| B - AX \|^2_F = \min_{\text{vec}(X) \geq 0} \frac{1}{2} \| \text{vec}(B) - (I \otimes A)\text{vec}(X) \|^2_F,
\]

where \( \otimes \) is the Kronecker product and \( \text{vec}(\cdot) \) 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

\[
\min_{X \geq 0} \frac{1}{2} \| B - AX \|^2_F = \sum_{j=1}^n \min_{X_j \geq 0} \frac{1}{2} \| B_j - AX_j \|^2_F
\]

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 \text{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:

\[
\min_{X \geq 0} \tilde{Q}(X) = -\langle A^T B, X \rangle + \frac{1}{2} \langle X, A^T A X \rangle.
\]

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 nonmonotone 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:
\[ \gamma = 10^{-4}, M = 10, \lambda_{\text{max}} = 10^{20}, \lambda_{\text{min}} = 10^{-20}. \]

Step 1. **Initialization:** Choose the initial approximation \( X^0 \in \mathbb{R}^{r \times n}_+ \) 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
\[ \|\nabla^P \tilde{Q}(X^k)\|_F < \rho, \]
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
\[ \tilde{Q}(P[X^k - \alpha \lambda^k \nabla \tilde{Q}(X^k)]) \leq \max_{0 \leq j \leq \min(k, M)} \tilde{Q}(X^{k-j}) + \gamma \alpha \langle \nabla \tilde{Q}(X^k), D^k \rangle, \]
set
\[ X^{k+1} = P[X^k - \alpha \lambda^k \nabla \tilde{Q}(X^k)] \]
and go to Step 4. Otherwise set
\[ \alpha = \alpha/4 \]
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 \( \langle s^k, y^k \rangle \leq 0 \), set \( \lambda^{k+1} = \lambda_{\text{max}} \), else set
\[ \lambda^{k+1} = \min(\lambda_{\text{max}}, \max(\lambda_{\text{min}}, \frac{\langle s^k, s^k \rangle}{\langle s^k, y^k \rangle})). \]

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

Our second and third NLS solvers PBBNLS2 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.

---

1The code for PBBNLS2 is available at: http://www.math.uconn.edu/~neumann/nmf/PBBNLS2.m
In PBBNLS2, the step size $\alpha$ 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:**

1. Set $D^k = P[X^k - \lambda^k \nabla \tilde{Q}(X^k)] - X^k$ and $\alpha = 1$.
2. If
   
   \begin{equation}
   \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,
   \end{equation}

   set

   \begin{equation}
   X^{k+1} = X^k + \alpha D^k
   \end{equation}

   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

\begin{equation}
\alpha = -\frac{\langle \nabla \tilde{Q}(X^k), D^k \rangle}{\langle D^k, A^T A D^k \rangle}.
\end{equation}

**Algorithm 3.5 (PBBNLS3).**

Step 3. **Line Search:**

1. Set $D^k = P[X^k - \lambda^k \nabla \tilde{Q}(X^k)] - X^k$.
2. If
   
   \begin{equation}
   \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,
   \end{equation}

   set $X^{k+1} = X^k + D^k$ and go to Step 4.

   Otherwise compute $\alpha$ 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.
In the four PBBNLS methods we employ the following strategies to save computational time:

- To avoid repetition, the constant matrices $A^T A$ and $A^T B$ 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:

$$
\tilde{Q}(X^k + \alpha D^k) = -(A^T B, X^k + \alpha D^k) + \frac{1}{2}(X^k + \alpha D^k, A^T A(X^k + \alpha D^k))
$$

(3.18)

In this form, only one calculation of $\langle \nabla \tilde{Q}(X^k), D^k \rangle$ and $\langle D^k, A^T A D^k \rangle$ 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 stepsize: 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 stepizes.

2) In the use of line search: our method uses a nonmonotone line search and we have a scalar steplength $\alpha$ 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 steplenthgs.

### 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, $i = 1, 2, 3, \text{ or } 4$).

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

$$PGN^k < \epsilon \times PGN^0,$$

where $PGN^k = \|\nabla^T_{W} h_S(W^k, H^k), (\nabla^T_{H} h_S(W^k, H^k))\|_F$. stop. Otherwise

Step 3. **Main Iteration:**

3.1. **Update $W$:** Use PBBNLSi to solve

$$\min_{W \geq 0} f(W, SH^k),$$

for $W^{k+1}$.

3.2. **Update $H$:** Use PBBNLSi to solve

$$\min_{H \geq 0} f(W^{k+1}S, H),$$

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 $\rho_W$ and $\rho_H$ be the tolerances for (3.20) and (3.21) by PBBNLSi, respectively. This strategy initially sets

$$\rho_W = \max(0.001, \epsilon)\|\nabla^T_{W} h_S(W^0, H^0), (\nabla^T_{H} h_S(W^0, H^0))\|_F.$$

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

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.
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 $\geq 1000$ 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>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>nit</th>
<th>CPU Time</th>
<th>PGN</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = rand(100,15)</td>
<td>nllsubprob</td>
<td>631</td>
<td>0.390</td>
<td>0.000089</td>
<td>39.341092</td>
</tr>
<tr>
<td>B = rand(100,200)</td>
<td>PBBNLS1</td>
<td>103</td>
<td>0.062</td>
<td>0.000066</td>
<td>39.341092</td>
</tr>
<tr>
<td>X^0 = rand(15,200)</td>
<td>PBBNLS2</td>
<td>93</td>
<td>0.047</td>
<td>0.000098</td>
<td>39.341092</td>
</tr>
<tr>
<td></td>
<td>PBBNLS3</td>
<td>150</td>
<td>0.078</td>
<td>0.000097</td>
<td>39.341092</td>
</tr>
<tr>
<td></td>
<td>PBBNLS4</td>
<td>129</td>
<td>0.047</td>
<td>0.000070</td>
<td>39.341092</td>
</tr>
<tr>
<td>A = rand(300,50)</td>
<td>nllsubprob</td>
<td>$\geq 1000$</td>
<td>7.535</td>
<td>0.023347</td>
<td>107.023210</td>
</tr>
<tr>
<td>B = rand(300,500)</td>
<td>PBBNLS1</td>
<td>196</td>
<td>1.108</td>
<td>0.000080</td>
<td>107.023210</td>
</tr>
<tr>
<td>X^0 = rand(50,500)</td>
<td>PBBNLS2</td>
<td>177</td>
<td>0.952</td>
<td>0.000098</td>
<td>107.023210</td>
</tr>
<tr>
<td></td>
<td>PBBNLS3</td>
<td>210</td>
<td>1.154</td>
<td>0.000099</td>
<td>107.023210</td>
</tr>
<tr>
<td></td>
<td>PBBNLS4</td>
<td>195</td>
<td>0.733</td>
<td>0.000033</td>
<td>107.023210</td>
</tr>
<tr>
<td>A = rand(2000,50)</td>
<td>nllsubprob</td>
<td>$\geq 1000$</td>
<td>7.472</td>
<td>0.321130</td>
<td>288.299174</td>
</tr>
<tr>
<td>B = rand(2000,500)</td>
<td>PBBNLS1</td>
<td>196</td>
<td>1.108</td>
<td>0.000080</td>
<td>288.299174</td>
</tr>
<tr>
<td>X^0 = rand(50,500)</td>
<td>PBBNLS2</td>
<td>177</td>
<td>0.952</td>
<td>0.000098</td>
<td>288.299174</td>
</tr>
<tr>
<td></td>
<td>PBBNLS3</td>
<td>210</td>
<td>1.154</td>
<td>0.000099</td>
<td>288.299174</td>
</tr>
<tr>
<td></td>
<td>PBBNLS4</td>
<td>195</td>
<td>0.733</td>
<td>0.000033</td>
<td>288.299174</td>
</tr>
</tbody>
</table>

We observe from Table 4.1 that the PBBNLS1, PBBNLS2, and PBBNLS3 methods perform similarly. Each of them is significantly faster than the nllsubprob 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 \textit{nlssubprob} method is similar to the one illustrated in Table 4.1.

As the APBBi methods and Lin’s method use PBBNLSi or \textit{nlssubprob} to solve the Subproblems (3.20) and (3.21) alternately, we expect the APBBi methods to be faster 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 \textit{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:

\begin{equation}
\| \nabla_C W h_S(W^k, H^k), (\nabla_C H h_S(W^k, H^k))^T \|_F < \epsilon \cdot PGN^0,
\end{equation}

where $\epsilon$ 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 test problems and the initial pair $(W^0, H^0)$ were generated. The value CPU TIME denotes the CPU time used when the termination criterion (4.1) was met. The notation $\geq 600$ 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 $\| \nabla_C W h_S(W^k, H^{k-1}), (\nabla_C H h_S(W^k, H^k))^T \|_F$, 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 CPU TIME.

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>iter</th>
<th>iter</th>
<th>CPU Time</th>
<th>PGN</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V = \text{rand}(100,50)$</td>
<td>Lin</td>
<td>359</td>
<td>6117</td>
<td>1.248</td>
<td>0.00028</td>
<td>15.925779</td>
</tr>
<tr>
<td></td>
<td>APBB1</td>
<td>646</td>
<td>9818</td>
<td>1.716</td>
<td>0.000108</td>
<td>15.927577</td>
</tr>
<tr>
<td>$W^0 = \text{rand}(100,10)$</td>
<td>APBB2</td>
<td>439</td>
<td>5961</td>
<td>1.014</td>
<td>0.000207</td>
<td>15.954012</td>
</tr>
<tr>
<td>$H^0 = \text{rand}(10,50)$</td>
<td>APBB3</td>
<td>475</td>
<td>5660</td>
<td>1.076</td>
<td>0.000278</td>
<td>15.934005</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>562</td>
<td>12256</td>
<td>1.591</td>
<td>0.000181</td>
<td>15.910972</td>
</tr>
<tr>
<td>$V = \text{rand}(100,200)$</td>
<td>Lin</td>
<td>576</td>
<td>15996</td>
<td>6.708</td>
<td>0.001299</td>
<td>33.885342</td>
</tr>
<tr>
<td>$W^0 = \text{rand}(100,15)$</td>
<td>APBB1</td>
<td>375</td>
<td>6455</td>
<td>2.683</td>
<td>0.000905</td>
<td>33.901354</td>
</tr>
<tr>
<td>$H^0 = \text{rand}(15,200)$</td>
<td>APBB2</td>
<td>631</td>
<td>10561</td>
<td>4.274</td>
<td>0.001</td>
<td>33.890555</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>499</td>
<td>8784</td>
<td>3.588</td>
<td>0.00157</td>
<td>33.902192</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>430</td>
<td>13751</td>
<td>3.775</td>
<td>0.001386</td>
<td>33.883501</td>
</tr>
<tr>
<td>$V = \text{rand}(300,500)$</td>
<td>Lin</td>
<td>1150</td>
<td>85874</td>
<td>302.736</td>
<td>0.03136</td>
<td>94.856419</td>
</tr>
<tr>
<td>$W^0 = \text{rand}(300,40)$</td>
<td>APBB1</td>
<td>101</td>
<td>2156</td>
<td>7.878</td>
<td>0.020744</td>
<td>77.815999</td>
</tr>
<tr>
<td>$H^0 = \text{rand}(40,500)$</td>
<td>APBB2</td>
<td>110</td>
<td>2429</td>
<td>8.362</td>
<td>0.028711</td>
<td>77.82096</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>103</td>
<td>2133</td>
<td>7.722</td>
<td>0.028945</td>
<td>77.822524</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>118</td>
<td>7770</td>
<td>16.084</td>
<td>0.031105</td>
<td>77.830637</td>
</tr>
<tr>
<td>$V = \text{rand}(1000,100)$</td>
<td>Lin</td>
<td>2651</td>
<td>81560</td>
<td>310.351</td>
<td>0.010645</td>
<td>161.578261</td>
</tr>
<tr>
<td>$W^0 = \text{rand}(1000,20)$</td>
<td>APBB1</td>
<td>543</td>
<td>7613</td>
<td>17.94</td>
<td>0.009247</td>
<td>77.815999</td>
</tr>
<tr>
<td>$H^0 = \text{rand}(20,100)$</td>
<td>APBB2</td>
<td>488</td>
<td>6953</td>
<td>15.444</td>
<td>0.010538</td>
<td>77.82096</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>425</td>
<td>5252</td>
<td>13.088</td>
<td>0.005955</td>
<td>77.822524</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>351</td>
<td>10229</td>
<td>16.224</td>
<td>0.005741</td>
<td>77.830637</td>
</tr>
<tr>
<td>$V = \text{abs(randn(300,300))}$</td>
<td>Lin</td>
<td>177</td>
<td>7729</td>
<td>10.624</td>
<td>0.032641</td>
<td>161.578261</td>
</tr>
<tr>
<td>$W^0 = \text{abs(randn(300,200))}$</td>
<td>APBB1</td>
<td>183</td>
<td>3350</td>
<td>4.508</td>
<td>0.016747</td>
<td>77.815999</td>
</tr>
<tr>
<td>$H^0 = \text{abs(randn(20,300))}$</td>
<td>APBB2</td>
<td>176</td>
<td>3215</td>
<td>4.15</td>
<td>0.017171</td>
<td>77.815999</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>194</td>
<td>2996</td>
<td>4.118</td>
<td>0.024201</td>
<td>77.815999</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>209</td>
<td>6351</td>
<td>6.068</td>
<td>0.032873</td>
<td>77.815999</td>
</tr>
<tr>
<td>$V = \text{abs(randn(300,500))}$</td>
<td>Lin</td>
<td>934</td>
<td>13155</td>
<td>62.182</td>
<td>0.127067</td>
<td>199.116691</td>
</tr>
<tr>
<td>$W^0 = \text{abs(randn(300,40))}$</td>
<td>APBB1</td>
<td>646</td>
<td>7874</td>
<td>36.379</td>
<td>0.138395</td>
<td>199.168363</td>
</tr>
<tr>
<td>$H^0 = \text{abs(randn(40,500))}$</td>
<td>APBB2</td>
<td>577</td>
<td>8068</td>
<td>33.353</td>
<td>0.08876</td>
<td>199.069136</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>792</td>
<td>9768</td>
<td>43.805</td>
<td>0.100362</td>
<td>199.123688</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>438</td>
<td>14787</td>
<td>35.256</td>
<td>0.075076</td>
<td>199.163277</td>
</tr>
<tr>
<td>$V = \text{abs(randn(1000,100))}$</td>
<td>Lin</td>
<td>1438</td>
<td>24112</td>
<td>64.99</td>
<td>0.043276</td>
<td>162.820691</td>
</tr>
<tr>
<td>$W^0 = \text{abs(randn(1000,20))}$</td>
<td>APBB1</td>
<td>235</td>
<td>2822</td>
<td>7.254</td>
<td>0.044061</td>
<td>162.845427</td>
</tr>
<tr>
<td>$H^0 = \text{abs(randn(20,100))}$</td>
<td>APBB2</td>
<td>247</td>
<td>2929</td>
<td>7.176</td>
<td>0.036261</td>
<td>162.845179</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>258</td>
<td>2809</td>
<td>7.472</td>
<td>0.046555</td>
<td>162.841468</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>247</td>
<td>5193</td>
<td>9.516</td>
<td>0.043423</td>
<td>162.901371</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>iter</th>
<th>niter</th>
<th>CPUTime</th>
<th>PGN</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>V = rand(2000,1000)</td>
<td>Lin</td>
<td>542</td>
<td>25420</td>
<td>≥ 600</td>
<td>0.293784</td>
<td>388.404591</td>
</tr>
<tr>
<td>W₀ = rand(2000,40)</td>
<td>APBB1</td>
<td>63</td>
<td>1789</td>
<td>48.36</td>
<td>0.169287</td>
<td>388.753789</td>
</tr>
<tr>
<td>H₀ = rand(40,1000)</td>
<td>APBB2</td>
<td>60</td>
<td>1732</td>
<td>43.93</td>
<td>0.176915</td>
<td>388.788783</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>70</td>
<td>1708</td>
<td>45.989</td>
<td>0.215956</td>
<td>388.755657</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>337</td>
<td>46797</td>
<td>≥ 600</td>
<td>1.625634</td>
<td>388.389849</td>
</tr>
<tr>
<td>V = rand(3000,2000)</td>
<td>Lin</td>
<td>40</td>
<td>3960</td>
<td>≥ 600</td>
<td>21.122063</td>
<td>665.880519</td>
</tr>
<tr>
<td>W₀ = rand(3000,100)</td>
<td>APBB1</td>
<td>95</td>
<td>2049</td>
<td>367.148</td>
<td>1.648503</td>
<td>665.212812</td>
</tr>
<tr>
<td>H₀ = rand(100,2000)</td>
<td>APBB2</td>
<td>80</td>
<td>1936</td>
<td>308.664</td>
<td>1.67074</td>
<td>665.233771</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>94</td>
<td>1993</td>
<td>343.53</td>
<td>1.623925</td>
<td>665.20267</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>10</td>
<td>10020</td>
<td>≥ 600</td>
<td>32322.8139</td>
<td>711.14501</td>
</tr>
<tr>
<td>V = rand(2000,5000)</td>
<td>Lin</td>
<td>45</td>
<td>1925</td>
<td>32.48</td>
<td>7.742485</td>
<td>1394.828534</td>
</tr>
<tr>
<td>W₀ = rand(2000,100)</td>
<td>APBB1</td>
<td>92</td>
<td>2049</td>
<td>367.148</td>
<td>1.648503</td>
<td>665.212812</td>
</tr>
<tr>
<td>H₀ = rand(100,5000)</td>
<td>APBB2</td>
<td>80</td>
<td>1936</td>
<td>308.664</td>
<td>1.67074</td>
<td>665.233771</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>94</td>
<td>1993</td>
<td>343.53</td>
<td>1.623925</td>
<td>665.20267</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>10</td>
<td>10020</td>
<td>≥ 600</td>
<td>32322.8139</td>
<td>711.14501</td>
</tr>
<tr>
<td>V = csc(rand(50,50))</td>
<td>Lin</td>
<td>6195</td>
<td>198110</td>
<td>26.255</td>
<td>0.00181</td>
<td>718.456033</td>
</tr>
<tr>
<td>W₀ = csc(rand(50,10))</td>
<td>APBB1</td>
<td>2669</td>
<td>32219</td>
<td>4.103</td>
<td>0.001332</td>
<td>718.456035</td>
</tr>
<tr>
<td>H₀ = csc(rand(10,50))</td>
<td>APBB2</td>
<td>30</td>
<td>1267</td>
<td>457.395</td>
<td>9.070003</td>
<td>839.962863</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>32</td>
<td>1470</td>
<td>535.473</td>
<td>6.558566</td>
<td>839.754882</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>11</td>
<td>3458</td>
<td>≥ 600</td>
<td>834.139074</td>
<td>898.640966</td>
</tr>
<tr>
<td>V = csc(rand(100,50))</td>
<td>Lin</td>
<td>22093</td>
<td>111794</td>
<td>243.801</td>
<td>0.003822</td>
<td>1165.453591</td>
</tr>
<tr>
<td>W₀ = csc(rand(100,10))</td>
<td>APBB1</td>
<td>21991</td>
<td>135041</td>
<td>26.115</td>
<td>0.002869</td>
<td>1165.453605</td>
</tr>
<tr>
<td>H₀ = csc(rand(10,50))</td>
<td>APBB2</td>
<td>22410</td>
<td>136280</td>
<td>24.898</td>
<td>0.002869</td>
<td>1165.453605</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>22401</td>
<td>118545</td>
<td>23.338</td>
<td>0.001773</td>
<td>1165.453605</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>21483</td>
<td>172171</td>
<td>22.729</td>
<td>0.002435</td>
<td>1165.453604</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>iter</th>
<th>niter</th>
<th>CPUTime</th>
<th>PGN</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>V = csc(rand(50,50))</td>
<td>Lin</td>
<td>6195</td>
<td>198110</td>
<td>26.255</td>
<td>0.00181</td>
<td>718.456033</td>
</tr>
<tr>
<td>W₀ = csc(rand(50,10))</td>
<td>APBB1</td>
<td>2669</td>
<td>32219</td>
<td>4.103</td>
<td>0.001332</td>
<td>718.456035</td>
</tr>
<tr>
<td>H₀ = csc(rand(10,50))</td>
<td>APBB2</td>
<td>30</td>
<td>1267</td>
<td>457.395</td>
<td>9.070003</td>
<td>839.962863</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>32</td>
<td>1470</td>
<td>535.473</td>
<td>6.558566</td>
<td>839.754882</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>11</td>
<td>3458</td>
<td>≥ 600</td>
<td>834.139074</td>
<td>898.640966</td>
</tr>
<tr>
<td>V = csc(rand(100,50))</td>
<td>Lin</td>
<td>22093</td>
<td>111794</td>
<td>243.801</td>
<td>0.003822</td>
<td>1165.453591</td>
</tr>
<tr>
<td>W₀ = csc(rand(100,10))</td>
<td>APBB1</td>
<td>21991</td>
<td>135041</td>
<td>26.115</td>
<td>0.002869</td>
<td>1165.453605</td>
</tr>
<tr>
<td>H₀ = csc(rand(10,50))</td>
<td>APBB2</td>
<td>22410</td>
<td>136280</td>
<td>24.898</td>
<td>0.002869</td>
<td>1165.453605</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>22401</td>
<td>118545</td>
<td>23.338</td>
<td>0.001773</td>
<td>1165.453605</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>21483</td>
<td>172171</td>
<td>22.729</td>
<td>0.002435</td>
<td>1165.453604</td>
</tr>
</tbody>
</table>

The 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 \( \| V - W^* H^* \|_F \) 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.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Algorithm</th>
<th>iter</th>
<th>niter</th>
<th>CPUPTime</th>
<th>PGN</th>
<th>RN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exactly Factorable V = rand(20,5)*rand(5,50) W0 = rand(20,5) H0 = rand(5,50)</td>
<td>Lin</td>
<td>761</td>
<td>143759</td>
<td>9.781</td>
<td>0.00002</td>
<td>0.000146</td>
</tr>
<tr>
<td></td>
<td>APBB1</td>
<td>837</td>
<td>17865</td>
<td>1.232</td>
<td>0.000014</td>
<td>0.0001395</td>
</tr>
<tr>
<td></td>
<td>APBB2</td>
<td>869</td>
<td>18722</td>
<td>1.17</td>
<td>0.000014</td>
<td>0.0001395</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>1045</td>
<td>20461</td>
<td>1.295</td>
<td>0.000011</td>
<td>0.0001514</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>215</td>
<td>55065</td>
<td>2.387</td>
<td>0.000013</td>
<td>0.000385</td>
</tr>
<tr>
<td>Exactly Factorable V = rand(100,10)*rand(10,50) W0 = rand(100,10) H0 = rand(10,50)</td>
<td>Lin</td>
<td>837</td>
<td>115740</td>
<td>15.709</td>
<td>0.000078</td>
<td>0.000573</td>
</tr>
<tr>
<td></td>
<td>APBB1</td>
<td>524</td>
<td>13180</td>
<td>2.122</td>
<td>0.000076</td>
<td>0.005784</td>
</tr>
<tr>
<td></td>
<td>APBB2</td>
<td>560</td>
<td>13633</td>
<td>1.981</td>
<td>0.000075</td>
<td>0.005586</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>597</td>
<td>14917</td>
<td>2.246</td>
<td>0.000075</td>
<td>0.00545</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>304</td>
<td>76860</td>
<td>6.958</td>
<td>0.000082</td>
<td>0.000877</td>
</tr>
<tr>
<td>Exactly Factorable V = rand(100,15)*rand(15,200) W0 = rand(100,15) H0 = rand(15,200)</td>
<td>Lin</td>
<td>2735</td>
<td>243572</td>
<td>122.586</td>
<td>0.000366</td>
<td>0.003565</td>
</tr>
<tr>
<td></td>
<td>APBB1</td>
<td>1926</td>
<td>54640</td>
<td>23.837</td>
<td>0.000371</td>
<td>0.04548</td>
</tr>
<tr>
<td></td>
<td>APBB2</td>
<td>2217</td>
<td>65976</td>
<td>25.381</td>
<td>0.000357</td>
<td>0.045799</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>1879</td>
<td>49998</td>
<td>19.906</td>
<td>0.000333</td>
<td>0.045776</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>2065</td>
<td>279023</td>
<td>75.972</td>
<td>0.000368</td>
<td>0.016646</td>
</tr>
<tr>
<td>Exactly Factorable V = rand(200,30)*rand(30,500) W0 = rand(200,30) H0 = rand(30,500)</td>
<td>Lin</td>
<td>1790</td>
<td>320943</td>
<td>600</td>
<td>0.032549</td>
<td>0.34627</td>
</tr>
<tr>
<td></td>
<td>APBB1</td>
<td>1687</td>
<td>67167</td>
<td>148.763</td>
<td>0.00209</td>
<td>0.24903</td>
</tr>
<tr>
<td></td>
<td>APBB2</td>
<td>1640</td>
<td>64987</td>
<td>125.971</td>
<td>0.002076</td>
<td>0.250934</td>
</tr>
<tr>
<td></td>
<td>APBB3</td>
<td>1985</td>
<td>108521</td>
<td>208.87</td>
<td>0.002295</td>
<td>0.257472</td>
</tr>
<tr>
<td></td>
<td>APBB4</td>
<td>682</td>
<td>632573</td>
<td>600</td>
<td>1.283378</td>
<td>0.409974</td>
</tr>
</tbody>
</table>

value (for example, when each converges to a global minimizer of the NMF problem), they can still have different final $\|V - W^k H^k\|_F$ 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 $\|V - W^k H^k\|_F$ 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 $56 \times 46$. 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
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 $\theta$ 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,
Alternating Projected Barzilai–Borwein Methods for Nonnegative Matrix Factorization

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

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

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

where $A_{ij}$ is considered a zero entry if $|A_{ij}| < 10^{-6}$ in our experiment. From this table, we
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 4.4.** 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)
Alternating Projected Barzilai–Borwein Methods for Nonnegative Matrix Factorization

**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).

![Reconstructed images](image1)

**Table 4.6**

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

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>Algorithm</th>
<th>PGN</th>
<th>RN</th>
<th>Sparseness in W</th>
<th>Sparseness in H</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>APBB2</td>
<td>0.021793</td>
<td>2.189993</td>
<td>39</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Lin</td>
<td>0.021746</td>
<td>2.211365</td>
<td>40</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>Lee-Seung</td>
<td>10.838066</td>
<td>2.247042</td>
<td>34</td>
<td>20</td>
</tr>
<tr>
<td>0.7</td>
<td>APBB2</td>
<td>0.002024</td>
<td>2.859032</td>
<td>81</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>Lin</td>
<td>0.022953</td>
<td>2.877591</td>
<td>81</td>
<td>55</td>
</tr>
<tr>
<td></td>
<td>Lee-Seung</td>
<td>16.703386</td>
<td>2.928300</td>
<td>60</td>
<td>32</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>PGN</th>
<th>RN</th>
<th>Sparseness in W</th>
<th>Sparseness in H</th>
</tr>
</thead>
<tbody>
<tr>
<td>APBB2</td>
<td>0.023983</td>
<td>2.979195</td>
<td>82</td>
<td>50</td>
</tr>
<tr>
<td>Lin</td>
<td>2.296475</td>
<td>3.497314</td>
<td>71</td>
<td>32</td>
</tr>
<tr>
<td>Lee–Seung</td>
<td>10.894495</td>
<td>4.698601</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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:

$$\|\left[ \nabla_W^T h_S(W^k, H^k), (\nabla_H^T h_S(W^k, H^k))^T \right] \|_F < \epsilon \cdot PGN^0.$$  

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 $P_{12}$–$P_{16}$):
  $$V = \text{csc(rand}(m, n)), W_0 = \text{rand}(m, r), H_0 = \text{rand}(r, n).$$

- **Group 3** (Problems $P_{17}$–$P_{23}$):
  $$V = \text{rand}(m, r) \ast \text{rand}(r, n), W_0 = \text{rand}(m, r), H_0 = \text{rand}(r, n).$$

The numerical results of these experiments are reported in Table 4.8. From this table
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 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...
The CBCL database consists of 2429 facial images of dimensions $19 \times 19$. 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.

5. Final remarks. We have proposed four algorithms for solving the nonsmooth nonnegative matrix factorization (nNMF) 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
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 APBB$i$ ($i = 1, 2, 3$) 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 APBB$i$ ($i = 1, 2, 3$) 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

$$tolW = \max(0.001, tol) \times \text{initgrad_norm}; \quad tolH = tolW;$$

with the following

$$tolW = 10^{-8}; \quad tolH = tolW;$$

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 APBB$i$ ($i = 1, 2, 3$) 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 PBBNLS$i$ methods. Our preliminary numerical tests on some randomly generated medium size NMF problems show that the APBB$i$ 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


Alternating Projected Barzilai–Borwein Methods for Nonnegative Matrix Factorization


