Projected Gradient Methods for Non-negative Matrix Factorization

Chih-Jen Lin
Department of Computer Science
National Taiwan University, Taipei 106, Taiwan
cjlin@csie.ntu.edu.tw

Abstract

Non-negative matrix factorization (NMF) can be formulated as a minimization problem with bound constraints. Although bound-constrained optimization has been studied extensively in both theory and practice, so far no study has formally applied its techniques to NMF. In this paper, we propose two projected gradient methods for NMF, both of which exhibit strong optimization properties. We discuss efficient implementations and demonstrate that one of the proposed methods converges faster than the popular multiplicative update approach. A simple MATLAB code is also provided.

1 Introduction

Non-negative matrix factorization (NMF) (Paatero and Tapper, 1994; Lee and Seung, 1999) is useful for finding representations of non-negative data. Given an $n \times m$ data matrix V with $V_{ij} \geq 0$ and a pre-specified positive integer $r < \min(n, m)$, NMF finds two non-negative matrices $W \in R^{n\times r}$ and $H \in R^{r\times m}$ such that

$$V \approx WH.$$

If each column of V represents an object, NMF approximates it by a linear combination of r "basis" columns in W. NMF has been applied to many areas such as finding basis vectors of images (Lee and Seung, 1999), document clustering (Xu et al., 2003), molecular pattern discovery (Brunet et al., 2004), etc. Donoho and

Stodden (2004) have addressed the theoretical issues associated with the NMF approach.

The conventional approach to find W and H is by minimizing the difference between V and WH:

$$\begin{aligned} \min_{W,H} \quad & f(W,H) \equiv \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\bigl(V_{ij}-(WH)_{ij}\bigr)^2 \\ \text{subject to} \quad & W_{ia} \geq 0, H_{bj} \geq 0, \ \ \forall i,a,b,j. \end{aligned} \tag{1}$$

The inequalities such that variables are upper- and lower-bounded are referred to as bound constraints. Hence, (1) is a standard bound-constrained optimization problem. We also note that

$$\sum_{i=1}^{n}\sum_{j=1}^{m}\bigl(V_{ij}-(WH)_{ij}\bigr)^2 = \|V-WH\|_F^2,$$

where $\|\cdot\|_F$ is the Frobenius norm.

The most popular approach to solve (1) is the multiplicative update algorithm proposed by Lee and Seung (2001). It is simple to implement and often yields good results. At each iteration of this method, the elements of W and H are multiplied by certain factors. As the zero elements are not updated, all the components of W and H are strictly positive for all iterations. This type of strategy is contrary to the traditional bound-constrained optimization methods, which usually allow iterations to have bounded elements (i.e., zero elements in this case). Thus far, no study has formally applied bound-constrained optimization techniques to NMF. This paper investigates such methods in detail. Some earlier NMF studies require all W's column sums to be ones: $\sum_{i=1}^{n} W_{ia} = 1, \forall a = 1,\ldots,r$. The function value does not change because $f(WD, D^{-1}H) = f(W,H)$ for any $r \times r$ positive diagonal matrix D. With the inclusion of such additional constraints, (1) no longer remains a bounded problem. As adding these constraints may complicate the optimization procedures, we do not consider this modification in this study.

Among the existing bound-constrained optimization techniques, the projected gradient method is simple and effective. Though several researchers have used this method for NMF (Hoyer, 2002; Chu et al., 2005; Shepherd, 2004), there is neither a systematic study nor an easy implementation comparable to that of the multiplicative update method. This paper presents a comprehensive study on using projected gradient methods for NMF. Several useful modifications lead to efficient implementations. While the multiplicative update method still lacks convergence results, our proposed methods exhibit strong optimization properties. We experimentally show that one of the proposed methods converges faster than the multiplicative update method. This new method is thus an attractive approach to solve NMF. We also provide a complete MATLAB implementation.

Another popular NMF optimization formula is to minimize the (generalized) Kullback-Leibler divergence between V and WH (Lee and Seung, 1999):

$$\begin{aligned} \min_{W,H} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m}\left(V_{ij}\log\frac{V_{ij}}{(WH)_{ij}} - V_{ij} + (WH)_{ij}\right) \\ \text{subject to} \quad & W_{ia} \geq 0, H_{bj} \geq 0, \forall i, a, b, j. \end{aligned}$$

Strictly speaking, this formula is not a bound-constrained problem, which requires the objective function to be well-defined at any point of the bounded region. The log function is not well-defined if $V_{ij} = 0$ or $(WH)_{ij} = 0$. Hence, we do not consider this formulation in this study.

This paper is organized as follows. Section 2 discusses existing approaches for solving NMF problem (1), and presents several new properties. Section 3 introduces the projected gradient methods for bound-constrained optimization. Section 4 investigates specific but essential modifications for applying the proposed projected gradients methods to NMF. The stopping conditions in an NMF code are discussed in Section 5. Experiments on synthetic and real data sets are presented in Section 6. The discussion and conclusions are presented in Section 7. Appendix B contains the MATLAB code of one of the proposed approaches. All source codes used in this paper are available online at `http://www.csie.ntu.edu.tw/~cjlin/nmf`.

2 Existing Methods and New Properties

There are many existing methods for NMF. Some discussions are in (Paatero, 1999), but bound constraints are not rigorously handled. A more recent and complete survey is by Chu et al. (2005). This section briefly discusses some existing methods and presents several previously unmentioned observations.

To begin, we need certain properties of the NMF problem (1). The gradient of the function $f(W, H)$ consists of two parts:

$$\nabla_W f(W, H) = (WH - V)H^T \text{ and } \nabla_H f(W, H) = W^T(WH - V), \tag{2}$$

which are, respectively, partial derivatives to elements in W and H. From the Karush-Kuhn-Tucker (KKT) optimality condition (e.g., Bertsekas, 1999), (W, H) is a stationary point of (1) if and only if

$$\begin{aligned} & W_{ia} \geq 0, H_{bj} \geq 0, \\ & \nabla_W f(W, H)_{ia} \geq 0, \nabla_H f(W, H)_{bj} \geq 0, \\ & W_{ia} \cdot \nabla_W f(W, H)_{ia} = 0, \text{ and } H_{bj} \cdot \nabla_H f(W, H)_{bj} = 0, \forall i, a, b, j. \end{aligned} \tag{3}$$

Optimization methods for NMF produce a sequence $\{W^k, H^k\}_{k=1}^{\infty}$ of iterations. Problem (1) is non-convex and may have several local minima. A common misunderstanding is that limit points of the sequence are local minima. In fact, most non-convex optimization methods guarantee only the stationarity of the limit points. Such a property is still useful, as any local minimum must be a stationary point.

2.1 Multiplicative Update Methods

The most used approach to minimize (1) is a simple multiplicative update method proposed by Lee and Seung (2001):

Algorithm 1 Multiplicative Update

```
1. Initialize $W_{ia}^1 > 0, H_{bj}^1 > 0, \forall i, a, b, j$.
2. For $k = 1, 2, \ldots$
       $W_{ia}^{k+1} = W_{ia}^k \frac{(V(H^k)^T)_{ia}}{(W^k H^k (H^k)^T)_{ia}}, \quad \forall i, a,$   (4)
       $H_{bj}^{k+1} = H_{bj}^k \frac{((W^{k+1})^T V)_{bj}}{((W^{k+1})^T W^{k+1} H^k)_{bj}}, \quad \forall b, j.$   (5)
```

This algorithm is a fixed-point type method: If $(W^k H^k (H^k)^T)_{ia} \neq 0$ and

$W_{ia}^{k+1} = W_{ia}^k > 0$, then

$$(V(H^k)^T)_{ia} = (W^k H^k (H^k)^T)_{ia} \qquad \text{implies} \qquad \nabla_W f(W^k, H^k)_{ia} = 0,$$

which is part of the KKT condition (3). Lee and Seung (2001) have shown that the function value is non-increasing after every update:

$$f(W^{k+1}, H^k) \leq f(W^k, H^k) \qquad \text{and} \qquad f(W^{k+1}, H^{k+1}) \leq f(W^{k+1}, H^k). \tag{6}$$

They claim that the limit of the sequence $\{W^k, H^k\}_{k=1}^{\infty}$ is a stationary point (i.e., a point satisfying the KKT condition (3)). However, Gonzales and Zhang (2005) indicate that this claim is wrong as having (6) may not imply the convergence. Therefore, this multiplicative update method still lacks optimization properties.

To have Algorithm 1 well-defined, one must ensure that denominators in (4) and (5) are strictly positive. Moreover, if $W_{ia}^k = 0$ at the kth iteration, then $W_{ia} = 0$ at all subsequent iterations. Thus, one should keep $W_{ia}^k > 0$ and $H_{bj}^k > 0$, $\forall k$. The following theorem discusses when this property holds:

Theorem 1 *If V has neither zero column nor row, and $W_{ia}^1 > 0$ and $H_{bj}^1 > 0, \forall i, a, b, j$, then*

$$W_{ia}^k > 0 \text{ and } H_{bj}^k > 0, \forall i, a, b, j, \forall k \geq 1. \tag{7}$$

The proof is straightforward, and is in Appendix A.

If V has zero columns or rows, a division by zero may occur. Even if Theorem 1 holds, denominators close to zero may still cause numerical problems. Some studies such as (Piper et al., 2004) have proposed adding a small positive number in the denominators of (4)-(5). We observe numerical difficulties in a few situations and provide more discussions in Section 6.3.

Regarding the computational complexity, $V(H^k)^T$ and $(W^{k+1})^T V$ in (4) and (5) are both $O(nmr)$ operations. One can calculate the denominator in (4) by either

$$(WH)H^T \qquad \text{or} \qquad W(HH^T). \tag{8}$$

The former takes $O(nmr)$ operations, but the latter costs $O(\max(m,n)r^2)$. As $r < \min(m,n)$, the latter is better. Similarly for (5), $(W^T W)H$ should be used.

This discussion indicates the importance of having fewer $O(nmr)$ operations (i.e., WH, W^TV, or VH^T) in any NMF code.

In summary, the overall cost of Algorithm 1 is

$$\#\text{iterations} \times O(nmr).$$

All time complexity analysis in this paper assumes that V, W, and H are implemented as dense matrices.

2.2 Alternating Non-negative Least Squares

From the non-increasing property (6), Algorithm 1 is a special case of a general framework, which alternatively fixes one matrix and improves the other:

$$\begin{aligned}&\text{Find } W^{k+1} \text{ such that } f(W^{k+1}, H^k) \le f(W^k, H^k), \text{ and}\\&\text{Find } H^{k+1} \text{ such that } f(W^{k+1}, H^{k+1}) \le f(W^{k+1}, H^k).\end{aligned}$$

The extreme situation is to obtain the best point (Paatero, 1999; Chu et al., 2005):

Algorithm 2 Alternating non-negative least squares

```
1. Initialize $W^1_{ia} \ge 0, H^1_{bj} \ge 0, \forall i, a, b, j$.
2. For $k = 1, 2, \ldots$
       $W^{k+1} = \arg\min_{W \ge 0} f(W, H^k)$,          (9)
       $H^{k+1} = \arg\min_{H \ge 0} f(W^{k+1}, H)$.      (10)
```

This approach is the "block coordinate descent" method in bound-constrained optimization (Bertsekas, 1999), where sequentially one block of variables is minimized under corresponding constraints and the remaining blocks are fixed. For NMF, we have the simplest case of only two block variables W and H.

We refer to (9) or (10) as a sub-problem in Algorithm 2. When one block of variables is fixed, a sub-problem is indeed the collection of several non-negative least square problems: From (10),

$$H^{k+1}\text{'s } j\text{th column} = \min_{\mathbf{h} \ge \mathbf{0}} \|\mathbf{v} - W^{k+1}\mathbf{h}\|^2, \tag{11}$$

where $\mathbf{v}$ is the jth column of V and $\mathbf{h}$ is a vector variable. Chu et al. (2005) suggest projected Newton's methods (Lawson and Hanson, 1974) to solve each problem (11). Clearly, solving sub-problems (9) and (10) per iteration could be more expensive than the simple update in Algorithm 1. Then Algorithm 2 may be slower even though we expect that it better decreases the function value at each iteration. Efficient methods to solve sub-problems are thus essential. Section 4.1 proposes using project gradient methods and discusses why they are suitable for solving sub-problems in Algorithm 2.

Regarding the convergence of Algorithm 2, one may think that it is a trivial result. For example, Paatero (1999) states that for the alternating non-negative least square approach, no matter how many blocks of variables we have, the convergence is guaranteed. However, this issue deserves some attention. Past convergence analysis for "block coordinate descent" methods requires sub-problems to have unique solutions (Powell, 1973; Bertsekas, 1999), but this property does not hold here: Sub-problems (9) and (10) are convex, but they are not strictly convex. Hence, these sub-problems may have multiple optimal solutions. For example, when H^k is the zero matrix, any W is optimal for (9). Fortunately, for the case of *two blocks*, Grippo and Sciandrone (2000) have shown that this uniqueness condition is not needed. Directly from Corollary 2 of (Grippo and Sciandrone, 2000), we have the following convergence result:

Theorem 2 *Any limit point of the sequence $\{W^k, H^k\}$ generated by Algorithm 2 is a stationary point of (1).*

The remaining issue is whether the sequence $\{W^k, H^k\}$ has at least one limit point (i.e., there is at least one convergent subsequence). In optimization analysis, this property often comes from the boundedness of the feasible region, but our region under constraints $W_{ia} \geq 0$ and $H_{bj} \geq 0$ is unbounded. One can easily add a large upper bound to all variables in (1). As the modification still leads to a bound-constrained problem, Algorithm 2 can be applied and Theorem 2 holds. In contrast, it is unclear how to easily modify the multiplicative update rules if there are upper bounds in (1).

In summary, contrary to Algorithm 1, which still lacks convergence results, Algorithm 2 has nice optimization properties.

2.3 Gradient Approaches

In (Chu et al., 2005, Section 3.3), several gradient-type approaches have been mentioned. In this subsection, we briefly discuss methods that select the step size along the negative gradient direction. By defining

$$W_{ia} = E_{ia}^2 \text{ and } H_{bj} = F_{bj}^2,$$

Chu et al. (2005) reformulate (1) as an unconstrained optimization problem of variables E_{ia} and F_{bj}. Then standard gradient descent methods can be applied. The same authors also mention that Shepherd (2004) uses

$$\begin{aligned} W^{k+1} &= \max(0, W^k - \alpha_k \nabla_W f(W^k, H^k)), \\ H^{k+1} &= \max(0, H^k - \alpha_k \nabla_H f(W^k, H^k)), \end{aligned}$$

where α_k is the step size. This approach is already a projected gradient method. However, in the above references, details are not discussed.

3 Projected Gradient Methods for Bound-constrained Optimization

We consider the following standard form of bound-constrained optimization problems:

$$\begin{aligned} \min_{\mathbf{x} \in R^n} \quad & f(\mathbf{x}) \\ \text{subject to} \quad & l_i \le x_i \le u_i, \quad i = 1, \dots, n, \end{aligned} \tag{12}$$

where $f(\mathbf{x}) : R^n \to R$ is a continuously differentiable function, and $\mathbf{l}$ and $\mathbf{u}$ are lower and upper bounds, respectively. Assume k is the index of iterations. Projected gradient methods update the current solution $\mathbf{x}^k$ to $\mathbf{x}^{k+1}$ by the following rule:

$$\mathbf{x}^{k+1} = P[\mathbf{x}^k - \alpha^k \nabla f(\mathbf{x}^k)],$$

where

$$P[x_i] = \begin{cases} x_i & \text{if } l_i < x_i < u_i, \\ u_i & \text{if } x_i \ge u_i, \\ l_i & \text{if } x_i \le l_i, \end{cases}$$

maps a point back to the bounded feasible region. Variants of projected gradient methods differ on selecting the step size α^k. We consider a simple and effective one called "Armijo rule along the projection arc" in Bertsekas (1999), which originates from Bertsekas (1976). The procedure is illustrated in Algorithm 3.

Algorithm 3 A projected gradient method for bound-constrained optimization

```
1. Given $0 < \beta < 1, 0 < \sigma < 1$. Initialize any feasible $\mathbf{x}^1$.
2. For $k = 1, 2, \ldots$
       $\mathbf{x}^{k+1} = P[\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)]$,
   where $\alpha_k = \beta^{t_k}$, and $t_k$ is the first non-negative integer $t$ for which
       $f(\mathbf{x}^{k+1}) - f(\mathbf{x}^k) \leq \sigma \nabla f(\mathbf{x}^k)^T (\mathbf{x}^{k+1} - \mathbf{x}^k)$.     (13)
```

The condition (13), used in most proofs of projected gradient methods, ensures the sufficient decrease of the function value per iteration. By trying the step sizes $1, \beta, \beta^2, \ldots$, Bertsekas (1976) has proved that $\alpha_k > 0$ satisfying (13) always exists and every limit point of $\{\mathbf{x}^k\}_{k=1}^{\infty}$ is a stationary point of (12). A common choice of σ is 0.01, and we consider $\beta = 1/10$ in this paper. In the experiment section 6.2, we have some discussions about the choice of β.

Searching α_k is the most time consuming operation in Algorithm 3, so one should check as few step sizes as possible. Since α_{k-1} and α_k may be similar, a trick in (Lin and Moré, 1999) uses α_{k-1} as the initial guess and then either increases or decreases it in order to find the largest β^{t_k} satisfying (13). Moreover, with non-negative t_k, Algorithm 4 may be too conservative by restricting $\alpha_k \leq 1$. Sometimes, a larger step more effectively projects variables to bounds at one iteration. Algorithm 4 implements a better initial guess of α at each iteration and allows α to be larger than one.

Algorithm 4 An improved projected gradient method

```
1. Given $0 < \beta < 1, 0 < \sigma < 1$. Initialize any feasible $\mathbf{x}^1$. Set $\alpha_0 = 1$.
2. For $k = 1, 2, \ldots$
    (a) Assign $\alpha_k \leftarrow \alpha_{k-1}$
    (b) If $\alpha_k$ satisfies (13), repeatedly increase it by
            $\alpha_k \leftarrow \alpha_k/\beta$
        until either $\alpha_k$ does not satisfy (13) or $\mathbf{x}(\alpha_k/\beta) = \mathbf{x}(\alpha_k)$.
        Else repeatedly decrease $\alpha_k$ by
            $\alpha_k \leftarrow \alpha_k \cdot \beta$
        until $\alpha_k$ satisfies (13).
    (c) Set
            $\mathbf{x}^{k+1} = P[\mathbf{x}^k - \alpha_k \nabla f(\mathbf{x}^k)]$.
```

The convergence result has been proved in, for example, Calamai and Moré (1987). One may think that finding α with the largest function reduction leads to faster convergence:

$$\alpha_k \equiv \arg\min_{\alpha \geq 0} f\big(P[\mathbf{x}^k - \alpha \nabla f(\mathbf{x}^k)]\big). \tag{14}$$

The convergence of selecting such an α_k is proved in McCormick and Tapia (1972). However, (14) is a piecewise function of α, which is difficult to be minimized.

A major obstacle for minimizing bounded problems is to identify free (i.e., $l_i < x_i < u_i$) and active (i.e., $x_i = l_i$ or u_i) components at the convergent stationary point. Projected gradient methods are considered effective for doing so since they are able to add several active variables at a single iteration. However, once these sets have been (almost) identified, the problem in a sense reduces to an unconstrained one, and the slow convergence of gradient-type methods may occur. We will explain in Section 4.1 that for NMF problems, this issue may not be serious.

4 Projected Gradient Methods for NMF

We apply projected gradient methods to NMF in two situations. The first case solves non-negative least square problems discussed in Section 2.2. The second case directly minimizes (1). Both approaches have convergence properties following from Theorem 2 and Calamai and Moré (1987), respectively. Several modifications specific to NMF will be presented.

4.1 Alternating Non-negative Least Squares Using Projected Gradient Methods

Section 2.2 indicates that Algorithm 2 relies on efficiently solving sub-problems (9) and (10), each of which is a bound-constrained problem. We propose using project gradient methods to solve them.

Sub-problem (10) consists of m independent non-negative least square problems (11), so one could solve them separately, a situation suitable for parallel environments. However, in a serial setting, treating them together is better because of the following reasons:

1. These non-negative least square problems are closely related as they share the same constant matrices V and W^{k+1} in (11).

2. Working on the whole H but not its individual columns implies that all operations are matrix-based. Since finely tuned numerical linear algebra codes have better speed-up on matrix than on vector operations, we can thus save computational time.

For a simpler description of our method, we focus on (10) and rewrite it as

$$\begin{aligned} \min_{H} \quad & \bar{f}(H) \equiv \frac{1}{2}\|V - WH\|_F^2 \\ \text{subject to} \quad & H_{bj} \geq 0, \quad \forall b, j. \end{aligned} \tag{15}$$

Both V and W are constant matrices in (15). If we concatenate H's columns to a vector vec(H), then

$$\begin{aligned}\bar{f}(H) &= \frac{1}{2}\|V - WH\|_F^2 \\ &= \frac{1}{2}\text{vec}(H)^T \begin{bmatrix} W^TW & & \\ & \ddots & \\ & & W^TW \end{bmatrix} \text{vec}(H) + H\text{'s linear terms.}\end{aligned}$$

The Hessian matrix (i.e., second derivative) of $\bar{f}(H)$ is block diagonal, and each block W^TW is an $r \times r$ positive semi-definite matrix. As $W \in R^{n\times r}$ and $r \ll n$, W^TW and the whole Hessian matrix tend to be well-conditioned, a good property for optimization algorithms. Thus, gradient-based methods may converge fast enough. A further investigation of this conjecture is in the experiment section 6.2.

The high cost of solving the two sub-problems (9) and (10) at each iteration is a concern. It is thus essential to analyze the time complexity and find efficient implementations. Each sub-problem requires an iterative procedure, whose iterations are referred to as sub-iterations. When using Algorithm 4 to solve (15), we must maintain the gradient

$$\nabla\bar{f}(H) = W^T(WH - V)$$

at each sub-iteration. Following the discussion near Eq. (8), one should calculate it by $(W^TW)H - W^TV$. Constant matrices W^TW and W^TV can be computed respectively in $O(nr^2)$ and $O(nmr)$ operations before running sub-iterations.

The main computational task per sub-iteration is to find a step size α such that the sufficient decrease condition (13) is satisfied. Assume $\bar{H}$ is the current solution. To check if

$$\tilde{H} \equiv P[\bar{H} - \alpha\nabla\bar{f}(\bar{H})],$$

satisfies (13), calculating $\bar{f}(\tilde{H})$ takes $O(nmr)$ operations. If there are t trials of $\tilde{H}$'s, the computational cost $O(tnmr)$ is prohibitive. We propose the following strategy to reduce the cost: For a quadratic function $f(\mathbf{x})$ and any vector $\mathbf{d}$,

$$f(\mathbf{x} + \mathbf{d}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T\nabla^2 f(\mathbf{x})\mathbf{d}. \tag{16}$$

Hence, for two consecutive iterations $\bar{\mathbf{x}}$ and $\tilde{\mathbf{x}}$, (13) can be written as

$$(1-\sigma)\nabla f(\bar{\mathbf{x}})^T(\tilde{\mathbf{x}} - \bar{\mathbf{x}}) + \frac{1}{2}(\tilde{\mathbf{x}} - \bar{\mathbf{x}})^T\nabla^2 f(\bar{\mathbf{x}})(\tilde{\mathbf{x}} - \bar{\mathbf{x}}) \le 0.$$

Now $\bar{f}(H)$ defined in (15) is quadratic, so (13) becomes

$$(1-\sigma)\langle \nabla \bar{f}(\bar{H}), \tilde{H}-\bar{H}\rangle + \frac{1}{2}\langle \tilde{H}-\bar{H}, (W^TW)(\tilde{H}-\bar{H})\rangle \le 0, \tag{17}$$

where $\langle \cdot, \cdot \rangle$ is the sum of the component-wise product of two matrices. The major operation in (17) is the matrix product $(W^TW)\cdot(\tilde{H}-\bar{H})$, which takes $O(mr^2)$. Thus, the cost $O(tnmr)$ of checking (13) is significantly reduced to $O(tmr^2)$. With the cost $O(nmr)$ for calculating W^TV in the beginning, the complexity of using Algorithm 4 to solve the sub-problem (15) is

$$O(nmr) + \#\text{sub-iterations} \times O(tmr^2),$$

where t is the average number of checking (13) at each sub-iteration.

The pseudo code for optimizing (15) is in Appendix B.2. We can use the same procedure to obtain W^{k+1} by rewriting (10) as a form similar to (15):

$$\bar{f}(W) \equiv \frac{1}{2}\|V^T - H^TW^T\|_F^2,$$

where V^T and H^T are constant matrices.

The overall cost to solve (1) is

$$\#\text{iterations} \times \left(O(nmr) + \#\text{sub-iterations} \times O(tmr^2 + tnr^2)\right). \tag{18}$$

At each iteration, there are two $O(nmr)$ operations: $V(H^k)^T$ and $(W^{k+1})^TV$, the same as those in the multiplicative update method. If t and #sub-iterations are small, this method is efficient.

To reduce the number of sub-iterations, a simple but useful technique is to warm start the solution procedure of each sub-problem. At final iterations (W^k, H^k)'s are all similar, so W^k is an effective initial point for solving (9).

4.2 Directly Applying Projected Gradients to NMF

We may directly apply Algorithm 4 to minimize (1). Similar to solving the non-negative least square problems in Section 4.1, the most expensive operation is checking the sufficient decrease condition (13). From the current solution $(\bar{W}, \bar{H})$, we simultaneously update both matrices to $(\tilde{W}, \tilde{H})$:

$$(\tilde{W}, \tilde{H}) \equiv P\left[(\bar{W}, \bar{H}) - \alpha\left(\nabla_W f(\bar{W}, \bar{H}), \nabla_H f(\bar{W}, \bar{H})\right)\right].$$

As $f(W,H)$ is not a quadratic function, (16) does not hold. Hence the trick (17) cannot be applied to save the computational time. Then, calculating $f(\tilde{W},\tilde{H}) = \frac{1}{2}\|V - \tilde{W}\tilde{H}\|_F^2$ takes $O(nmr)$ operations. The total computational cost is

$$\#\text{iterations} \times O(tnmr),$$

where t is the average number of condition (13) checked per iteration.

Given any random initial (W^1, H^1), if $\|V - W^1H^1\|_F^2 > \|V\|_F^2$, very often after the first iteration $(W^2, H^2) = (0,0)$ causes the algorithm to stop. The solution $(0,0)$ is a useless stationary point of (1). A simple remedy is to find a new initial point $(W^1, \bar{H}^1)$ such that $f(W^1, \bar{H}^1) < f(0,0)$. By solving $\bar{H}^1 = \arg\min_{H \geq 0} f(W^1, H)$ using the procedure described in Section 4.1, we have

$$\|V - W^1\bar{H}^1\|_F^2 \leq \|V - W^1 \cdot 0\|_F^2 = \|V\|_F^2.$$

The strict inequality generally holds, so $f(W^1, \bar{H}^1) < f(0,0)$.

5 Stopping Conditions

In all algorithms mentioned so far, we did not specify when the procedure should stop. Several implementations of the multiplicative update method (e.g., Hoyer, 2004) have an infinite loop, which must be interrupted by users after a time or iteration limit. Some researchers (e.g., Brunet, 2004) check the difference between recent iterations. If the difference is small enough, then the procedure stops. However, such a stopping condition does not reveal whether a solution is close to a stationary point or not. In addition to a time or iteration limit, standard conditions to check the stationarity should also be included in NMF software. Moreover, in alternating least squares, each sub-problem involves an optimization procedure, which needs a stopping condition as well.

In bound-constrained optimization, a common condition to check if a point $\mathbf{x}^k$ is close to a stationary point is the following (Lin and Moré, 1999):

$$\|\nabla^P f(\mathbf{x}^k)\| \leq \epsilon \|\nabla f(\mathbf{x}^1)\|, \tag{19}$$

where $\nabla^P f(\mathbf{x}^k)$ is the *projected gradient* defined as

$$\nabla^P f(\mathbf{x})_i \equiv \begin{cases} \nabla f(\mathbf{x})_i & \text{if } l_i < x_i < u_i, \\ \min(0, \nabla f(\mathbf{x})_i) & \text{if } x_i = l_i, \\ \max(0, \nabla f(\mathbf{x})_i) & \text{if } x_i = u_i. \end{cases} \tag{20}$$

This condition follows from an equivalent form of the KKT condition for bounded problems: $l_i \leq x_i \leq u_i, \forall i$, and

$$\|\nabla^P f(\mathbf{x})\| = \mathbf{0}.$$

For NMF, (19) becomes

$$\|\nabla^P f(W^k, H^k)\|_F \leq \epsilon \|\nabla f(W^1, H^1)\|_F. \tag{21}$$

For alternating least squares, each sub-problem (9) or (10) requires a stopping condition as well. Ideally, the condition for them should be related to the "global" one for (1), but a suitable condition is not obvious. For example, we cannot use the same stopping tolerance in (21) for sub-problems. A user may specify $\epsilon = 0$ and terminate the code after a certain time or iteration limit. Then the same $\epsilon = 0$ in solving the first sub-problem will cause Algorithm 2 to keep running at the first iteration. We thus use the following stopping conditions for sub-problems. The returned matrices W^{k+1} and H^{k+1} from the iterative procedures of solving the sub-problem (9) and (10) should respectively satisfy

$$\begin{aligned} &\|\nabla^P_W f(W^{k+1}, H^k)\|_F \leq \bar{\epsilon}_W, \text{ and} \\ &\|\nabla^P_H f(W^{k+1}, H^{k+1})\|_F \leq \bar{\epsilon}_H, \end{aligned}$$

where we set

$$\bar{\epsilon}_W = \bar{\epsilon}_H \equiv \max(10^{-3}, \epsilon) \left\|\nabla f(W^1, H^1)\right\|_F$$

in the beginning and ϵ is the tolerance in (21). If the projected gradient method for solving (9) stops without any iterations, we decrease the stopping tolerance by

$$\bar{\epsilon}_W \leftarrow \bar{\epsilon}_W / 10. \tag{22}$$

For the sub-problem (10), $\bar{\epsilon}_H$ is reduced in a similar way.

Table 1: Results of running synthetic data sets (from small to large) under various stopping tolerances. We present the average time (in seconds), number of iterations, and objective values of using 30 initial points. Approaches with the smallest time or objective values are in bold type. Note that when $\epsilon = 10^{-5}$, mult and pgrad often exceed the iteration limit of 8,000.

(a) $m = 25, r = 5, n = 125$.

	Time				#iterations				Objective values			
ϵ	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
mult	0.10	1.22	2.60		698	4651	7639		**390.4**	389.3	389.3	
alspgrad	**0.03**	**0.10**	**0.45**	**0.97**	6	26	100	203	412.9	392.8	389.2	**389.1**
pgrad	0.05	0.24	0.68		53	351	1082		401.6	389.9	389.1	
lsqnonneg	6.32	27.76	57.57		23	96	198		391.1	**389.1**	**389.0**	

(b) $m = 50, r = 10, n = 250$.

	Time				#iterations				Objective values			
ϵ	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
mult	0.16	14.73	21.53		349	6508	8000		**1562.1**	**1545.7**	1545.6	
alspgrad	**0.03**	**0.13**	**0.99**	**5.51**	4	14	76	352	1866.6	1597.1	1547.8	**1543.5**
pgrad	0.38	3.17	10.29		47	1331	4686		1789.4	1558.4	**1545.5**	

(c) $m = 100, r = 20, n = 500$.

	Time				#iterations				Objective values			
ϵ	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-3}	10^{-4}	10^{-5}	10^{-6}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
mult	0.41	8.28	175.55		170	2687	8000		**6535.2**	**6355.7**	**6342.3**	
alspgrad	**0.02**	**0.21**	**1.09**	**10.02**	2	8	31	234	9198.7	6958.6	6436.7	**6332.9**
pgrad	0.60	2.88	35.20		2	200	3061		8141.1	6838.7	6375.0	

6 Experiments

We compare four methods discussed in this paper and refer to them in the following way:

1. mult: the multiplicative update method described in Section 2.1.
2. alspgrad: alternating non-negative least squares using the projected gradient method for each sub-problem (Section 4.1).
3. pgrad: a direct use of the projected gradient method on (1) (Section 4.2).
4. lsqnonneg: Using MATLAB command lsqnonneg to solve m problems (11) in the alternating least squares framework.

All implementations are in MATLAB (`http://www.mathworks.com`). We conduct experiments on an Intel Xeon 2.8GHz computer. Results of using synthetic and real data are presented in the following subsections. All source codes for experiments are available online at `http://www.csie.ntu.edu.tw/~cjlin/nmf`.

6.1 Synthetic Data

We consider three problem sizes:

$$(m, r, n) = (25, 5, 25),\ (50, 10, 250),\ \text{and}\ (100, 20, 500).$$

The matrix V is randomly generated by the normal distribution (mean 0 and standard deviation 1)

$$V_{ij} = |N(0, 1)|.$$

The initial (W^1, H^1) is constructed in the same way, and all four methods share the same initial point. These methods may converge to different points due to the non-convexity of the NMF problem (1). To have a fair comparison, for the same V we try 30 different initial (W^1, H^1), and report the average results. As synthetic data may not resemble practical problems, we leave detailed analysis of the proposed algorithms in Section 6.2, which considers real data.

We set ϵ in (21) to be 10^{-3}, 10^{-4}, 10^{-5}, and 10^{-6} in order to investigate the convergence speed to a stationary point. We also impose a time limit of 1,000 seconds and a maximal number of 8,000 iterations on each method. As lsqnonneg takes long computing time, we run it only on the smallest data. Due to the slow convergence of mult and pgrad, for $\epsilon = 10^{-6}$ we run only alspgrad.

Results of average time, number of iterations, and objective values are in Tables 1(a)-1(c). For small problems, Table 1(a) shows that all four methods give similar objective values as $\epsilon \to 0$. The method lsqnonneg is rather slow, a result supporting our argument in Section 4.1 that a matrix-based approach is better than a vector-based one. For larger problems, when $\epsilon = 10^{-5}$, mult and pgrad often exceed the maximal number of iterations. Clearly, mult quickly decreases the objective value in the beginning, but slows down in the end. In contrast, alspgrad has the fastest final convergence. For the two larger problems, it gives the smallest objective value under $\epsilon = 10^{-6}$, but takes less time than that by mult under $\epsilon = 10^{-5}$. Due to the poor performance of pgrad and lsqnonneg, subsequently we focus on comparing mult and alspgrad.

6.2 Image Data

We consider three image problems used in Hoyer (2004):

1. CBCL face image database.
   `http://cbcl.mit.edu/cbcl/software-datasets/FaceData2.html`.
2. ORL face image database.
   `http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html`.
3. Natural image data set (Hoyer, 2002).

All settings are the same as those in Hoyer (2004). We compare objective values and projected-gradient norms of mult and alspgrad after running 25 and 50 seconds. Table 2 presents average results of using 30 random initial points. For all three problems, alspgrad gives smaller objective values. While mult may quickly lower the objective value in the beginning, alspgrad catches up very soon and has faster final convergence. Results here are consistent with the findings in Section 6.1. Regarding the projected-gradient norms, those by alspgrad are much smaller. Hence, solutions by alspgrad are closer to stationary points.

To further illustrate the slow final convergence of mult, Figure 1 checks the relation between the running time and the objective value. The CBCL set with the first of the 30 initial (W^1, H^1) is used. The figure clearly demonstrates that mult very slowly decreases the objective value at final iterations.

The number of sub-iterations for solving (9) and (10) in alspgrad is an important issue. First, it is related to the time complexity analysis. Second, Section 4.1 conjectures that the number should be small as W^TW and HH^T are generally well-conditioned. Table 3 presents the number of sub-iterations and the condition numbers of W^TW and HH^T. Compared to gradient-based methods in other scenarios, the number of sub-iterations is relatively small. Another projected gradient method pgrad discussed in Table 1 easily takes hundreds or thousands of iterations. For condition numbers, both CBCL and Natural sets have $r < n < m$, so HH^T

Table 2: Image data: Average objective values and projected-gradient norms of using 30 initial points under specified time limits. Smaller values are in bold type.

Problem		CBCL		ORL		Natural	
Size (n r m)		361 49	2,429	10,304 25	400	288 72	10,000
Time limit (in seconds)		25	50	25	50	25	50
Objective Value	mult	963.81	914.09	16.14	14.31	**370797.31**	353709.28
	alspgrad	**923.77**	**870.18**	**14.34**	**13.82**	377167.27	**352355.64**
$\|\nabla f^P(W,H)\|_F$	mult	488.72	327.28	19.37	9.30	54534.09	21985.99
	alspgrad	**230.67**	**142.13**	**4.82**	**4.33**	**19357.03**	**4974.84**

tends to be better conditioned than W^TW. ORL has the opposite as $r < m < n$. All condition numbers are small, and this result confirms our earlier conjecture. For ORL, cond$(W^TW) >$ cond(HH^T), but the number of sub-iterations on solving W is more. One possible reason is the different stopping tolerances for solving (9) and (10).

In the implementation of alspgrad, there is a parameter β, which is the rate of reducing the step size to satisfy the sufficient decrease condition (13). It must be between 0 and 1, and for the above experiments, we use $\beta = 0.1$. One may wonder the effect of using other β. Clearly, a smaller β more aggressively reduce the step size, but it may also cause a step size that in the end is too small. We consider the CBCL set with the first of the 30 initial (W^1, H^1) (i.e., the setting to generate Figure 1), and check the effect of using $\beta = 0.5$ and 0.1. In both cases, alspgrad works well, but the one using 0.1 slightly more quickly reduces the function value. Therefore, $\beta = 0.5$ causes too many checks for the sufficient decrease condition (13). The cost per iteration is thus higher.

6.3 Text Data

NMF is useful for document clustering, so we next consider a text set RCV1 (Lewis et al., 2004). This set is an archive of manually categorized newswire stories from Reuters Ltd. The collection has been fully pre-processed, including removing stop words, stemming, and transforming into vector space models. Each vector, cosine normalized, contains features of logged TF-IDF (term frequency, inverse document

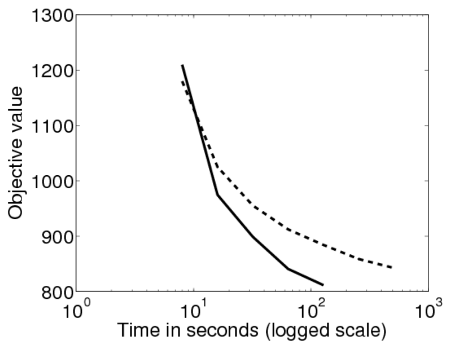

Figure 1: Time (seconds in log scale) vs. objective values for mult (dashed line) and alspgrad (solid line).

Table 3: Number of sub-iterations and condition numbers in solving (9) and (10) of alspgrad. For sub-iterations, we calculate (total sub-iterations)/(total iterations) under each initial point, and report the average of 30 values. For condition numbers, we find the median at all iterations, and report the average. Note that HH^T (W^TW) corresponds to the Hessian of minimizing W (H).

Problem	CBCL		ORL		Natural	
Time limit (in seconds)	25	50	25	50	25	50
W: # sub-iterations	34.51	47.81	9.93	11.27	21.94	27.54
cond(HH^T)	224.88	231.33	76.44	71.75	93.88	103.64
H: # sub-iterations	11.93	18.15	6.84	7.70	3.13	4.39
cond(W^TW)	150.89	124.27	478.35	129.00	38.49	17.19

frequency). Training/testing splits have been defined. We remove documents in the training set which are associated with more than one class and obtain a set of 15,933 instances in 101 classes. We further remove classes which have less than five documents. Using $r = 3$ and 6, we then randomly select r classes of documents to construct the $n \times m$ matrix V, where n is the number of the vocabulary set and m is the number of documents. Some words never appear in the selected documents and cause zero rows in V. We remove them before experiments. The parameter r is the number of clusters that we intend to assign documents to. Results of running mult and alspgrad by 25 and 50 seconds are in Table 4. We again have that alspgrad gives smaller objective values. In addition, projected-gradient norms of alspgrad are smaller.

In Section 2.1, we mention that mult is well-defined if Theorem 1 holds. Now

Table 4: Text data: Average objective values and projected-gradient norms of using 30 initial points under specified time limits. Smaller values are bold-faced. Due to the unbalanced class distribution, interestingly the random selection of six classes results in less documents (i.e., m) than that of selecting three classes.

Size (n r m)		$5,412$ 3 $1,588$		$5,737$ 6 $1,401$	
Time limit (in seconds)		25	50	25	50
Objective Value	mult	710.160	710.135	595.245	594.869
	alspgrad	**710.128**	**710.128**	**594.631**	**594.520**
$\|\nabla f^P(W,H)\|_F$	mult	4.646	1.963	13.633	11.268
	alspgrad	**0.016**	**0.000**	**2.250**	**0.328**

V is a sparse matrix with many zero elements since words appeared in a document are only a small subset of the whole vocabulary set. Thus, some columns of V are close to zero vectors, and for a few situations, numerical difficulties occur. In contrast, we do not face such problems for projected gradient methods.

7 Discussion and Conclusions

We discuss some future issues and draw conclusions.

7.1 Future Issues

As resulting W and H usually have many zero components, NMF is said to produce a sparse representation of the data. To achieve better sparseness, some studies such as (Hoyer, 2002; Piper et al., 2004) add penalty terms to the NMF objective function:

$$\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{m}\big(V_{ij}-(WH)_{ij}\big)^2+\alpha\|W\|_F^2+\beta\|H\|_F^2, \tag{23}$$

where α and β are positive numbers. Besides the Frobenius norm, which is quadratic, we can also use a linear penalty function

$$\alpha\sum_{i,a}W_{ia}+\beta\sum_{b,j}H_{bj}. \tag{24}$$

Our proposed methods can be used for such formulations. As penalty parameters α and β only indirectly control the sparseness, Hoyer (2004) proposes a scheme to directly specify the desired sparsity. It is interesting to investigate how to incorporate projected gradient methods in such frameworks.

7.2 Conclusions

This paper proposes two projected gradient methods for NMF. The one solving least square sub-problems in Algorithm 2 leads to faster convergence than the popular multiplicative update method. Its success is due to our following findings:

1. Sub-problems in Algorithm 2 for NMF generally have well-conditioned Hessian matrices (i.e., second derivative) due to the property $r \ll \min(n, m)$. Hence, projected gradients converge quickly, although they use only the first order information.

2. The cost of selecting step sizes in the projected gradient method for the sub-problem (15) is significantly reduced by some reformulations which again use the property $r \ll \min(n, m)$.

Therefore, taking special NMF properties is crucial when applying an optimization method to NMF.

Roughly speaking, optimization methods are between the following two extreme situations:

$$\begin{array}{lcl} \text{Low cost per iteration;} & & \text{High cost per iteration;} \\ \text{slow convergence.} & \longleftrightarrow & \text{fast convergence.} \end{array}$$

For example, Newton's methods are expensive per iteration but have very fast final convergence. Approaches with low cost per iteration usually decrease the objective value more quickly in the beginning, a nice property enjoyed by the multiplicative update method for NMF. Based on our analysis, we feel that the multiplicative update is very close to the first extreme. The proposed method of alternating least squares using projected gradients tends to be more in between. With faster convergence and strong optimization properties, it is an attractive approach for NMF.

Acknowledgments

I thank Marco Sciandrone for pointing out the convergence of two-block coordinate descent methods in (Grippo and Sciandrone, 2000) and helpful comments.

References

Dimitri P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21:174–184, 1976.

Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA 02178-9998, second edition, 1999.

Jean-Philippe Brunet. `http://www.broad.mit.edu/mpr/publications/projects/NMF/nmf.m`, 2004.

Jean-Philippe Brunet, Pablo Tamayo, Todd R. Golub, and Jill P. Mesirov. Metagenes and molecular pattern discovery using matrix factorization. *Proceedings of the National Academy of Science*, 101(12):4164–4169, 2004.

Paul H. Calamai and Jorge J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39:93–116, 1987.

M. Chu, F. Diele, R. Plemmons, and S. Ragni. Optimality, computation and interpretation of nonnegative matrix factorizations. 2005. URL `http://www.wfu.edu/~plemmons/papers/chu_ple.pdf`.

David Donoho and Victoria Stodden. When does non-negative matrix factorization give a correct decomposition into parts? In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16.* MIT Press, Cambridge, MA, 2004.

Edward F. Gonzales and Yin Zhang. Accelerating the Lee-Seung algorithm for non-negative matrix factorization. Technical report, Department of Computational and Applied Mathematics, Rice University, 2005.

L. Grippo and M. Sciandrone. On the convergence of the block nonlinear Gauss-Seidel method under convex constraints. *Operations Research Letters*, 26:127–136, 2000.

Patrik O. Hoyer. Non-negative sparse coding. In *Proceedings of IEEE Workshop on Neural Networks for Signal Processing*, pages 557–565, 2002.

Patrik O. Hoyer. Non-negative matrix factorization with sparseness constraints. *Journal of Machine Learning Research*, 5:1457–1469, 2004.

C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974.

Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401:788–791, 1999.

Daniel D. Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 556–562. MIT Press, 2001.

David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. RCV1: A new benchmark collection for text categorization research. *Journal of Machine Learning Research*, 5:361–397, 2004.

Chih-Jen Lin and Jorge J. Moré. Newton's method for large-scale bound constrained problems. *SIAM Journal on Optimization*, 9:1100–1127, 1999.

G. P. McCormick and R. A. Tapia. The gradient projection method under mild differentiability conditions. *SIAM Journal on Control*, 10:93–98, 1972.

Pentti Paatero. The multilinear engine—A table-driven, least squares program for solving multilinear problems, including the n-way parallel factor analysis model. *Journal of Computational and Graphical Statistics*, 8(4):854–888, 1999.

Pentti Paatero and Unto Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error. *Environmetrics*, 5:111–126, 1994.

Jon Piper, Paul Pauca, Robert Plemmons, and Maile Giffin. Object characterization from spectral data using nonnegative factorization and information theory. In *Proceedings of AMOS Technical Conference*, 2004.

Michael J. D. Powell. On search directions for minimization. *Mathematical Programming*, 4:193–201, 1973.

Simon Shepherd. `http://www.simonshepherd.supanet.com/nnmf.htm`, 2004.

Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference*, pages 267–273. ACM Press, 2003. doi: http://doi.acm.org/10.1145/860435.860485.

A Proof of Theorem 1

When $k = 1$, (7) holds by the assumption of this theorem. Using induction, if (7) is correct at k, then at $(k + 1)$, clearly denominators of (4) and (5) are strictly positive. Moreover, as V has neither zero column nor row, both numerators are strictly positive as well. Thus, (7) holds at $(k + 1)$, and the proof is complete.

B MATLAB Code

B.1 Main Code for alspgrad (Alternating Non-negative Least Squares Using Projected Gradients)

```
function [W,H] = nmf(V,Winit,Hinit,tol,timelimit,maxiter)

% NMF by alternative non-negative least squares using projected gradients
% Author: Chih-Jen Lin, National Taiwan University

% W,H: output solution
% Winit,Hinit: initial solution
% tol: tolerance for a relative stopping condition
% timelimit, maxiter: limit of time and iterations

W = Winit; H = Hinit; initt = cputime;

gradW = W*(H*H') - V*H'; gradH = (W'*W)*H - W'*V;
initgrad = norm([gradW; gradH'],'fro');
```

```
fprintf('Init gradient norm %f\n', initgrad);
tolW = max(0.001,tol)*initgrad; tolH = tolW;

for iter=1:maxiter,
  % stopping condition
  projnorm = norm([gradW(gradW<0 | W>0); gradH(gradH<0 | H>0)]);
  if projnorm < tol*initgrad | cputime-initt > timelimit,
    break;
  end

  [W,gradW,iterW] = nlssubprob(V',H',W',tolW,1000); W = W'; gradW = gradW';
  if iterW==1,
    tolW = 0.1 * tolW;
  end

  [H,gradH,iterH] = nlssubprob(V,W,H,tolH,1000);
  if iterH==1,
    tolH = 0.1 * tolH;
  end

  if rem(iter,10)==0, fprintf('.'); end
end
fprintf('\nIter = %d Final proj-grad norm %f\n', iter, projnorm);
```

B.2 Solving the Sub-problem (15) by the Projected Gradient Algorithm 4

```
function [H,grad,iter] = nlssubprob(V,W,Hinit,tol,maxiter)

% H, grad: output solution and gradient
% iter: #iterations used
% V, W: constant matrices
% Hinit: initial solution
% tol: stopping tolerance
% maxiter: limit of iterations

H = Hinit; WtV = W'*V; WtW = W'*W;

alpha = 1; beta = 0.1;
for iter=1:maxiter,
  grad = WtW*H - WtV;
  projgrad = norm(grad(grad < 0 | H >0));
  if projgrad < tol,
    break
  end

  % search step size
  for inner_iter=1:20,
    Hn = max(H - alpha*grad, 0); d = Hn-H;
    gradd=sum(sum(grad.*d)); dQd = sum(sum((WtW*d).*d));
    suff_decr = 0.99*gradd + 0.5*dQd < 0;
    if inner_iter==1,
      decr_alpha = ~suff_decr; Hp = H;
    end
    if decr_alpha,
      if suff_decr,
        H = Hn; break;
      else
        alpha = alpha * beta;
      end
    else
      if ~suff_decr | Hp == Hn,
        H = Hp; break;
      else
        alpha = alpha/beta; Hp = Hn;
      end
    end
  end
end

if iter==maxiter,
  fprintf('Max iter in nlssubprob\n');
end
```