# A non-monotonic method for large-scale nonnegative least squares

Dongmin Kim* Suvrit Sra† Inderjit S. Dhillon*

May 26, 2010

**Abstract**

We present a new algorithm for nonnegative least-squares (NNLS). Our algorithm extends the *unconstrained* quadratic optimization algorithm of Barzilai and Borwein (BB) (J. Barzilai and J. M. Borwein; *Two-Point Step Size Gradient Methods.* IMA J. Numerical Analysis; 1988.) to handle nonnegativity constraints. Our extension differs in several basic aspects from other constrained BB variants. The most notable difference is our modified computation of the BB stepsize that takes into account the nonnegativity constraints. We further refine the stepsize computation by introducing a stepsize scaling strategy that, in combination with orthogonal projections onto the nonnegative quadrant, yields an efficient NNLS algorithm. We compare our algorithm with both established convex solvers and a specialized NNLS method: on several synthetic and real-world datasets we observe highly competitive empirical performance.

**Keywords:** Least squares, nonnegativity constraints, large-scale, non-monotonic descent, Barzilai-Borwein stepsize, gradient projection method, NNLS.

## 1 Introduction

The central object of study in this paper is the *nonnegative least-squares (NNLS)* problem:

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad f(\boldsymbol{x}) = \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \quad \text{s.t.} \quad \boldsymbol{x} \geq 0, \tag{1}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ is a design matrix and $\boldsymbol{b} \in \mathbb{R}^m$ a vector of observations. NNLS is a fundamental problem that arises naturally in applications where, in addition to satisfying a least-squares model, the variables must satisfy nonnegativity constraints. These constraints usually stem from physical grounds, e.g., when the variables $x_i$ $(1 \leq i \leq n)$ encode quantities such as frequency counts (data mining [20]), chemical concentrations (chemometrics [10]), and image intensities or photon counts (astronomy [19, 25]; medical imaging [29]).

Despite its apparent simplicity, NNLS can be challenging to solve, especially when dealing with large-scale problems. For such problems often first-order methods are preferred [2, 3], which is also our motivation for developing a scalable first-order method for NNLS. Our method augments the *unconstrained* quadratic optimization algorithm of Barzilai and Borwein (BB) [1] to efficiently handle nonnegativity constraints. But before describing the associated technical details we first review some pertinent background material.

**Background and Motivation** We begin by recalling gradient projection (GP), perhaps the simplest of constrained methods that one could invoke for solving NNLS. Given a domain $\Omega$ of feasible solutions, GP iterates as $\boldsymbol{x}^{k+1} = [\boldsymbol{x}^k - \gamma^k \nabla f(\boldsymbol{x}^k)]_\Omega$ for $k \geq 1$, where $[\cdot]_\Omega$ denotes (orthogonal) projection onto $\Omega$, while $\gamma^k > 0$ is a stepsize [30]. GP seems attractive for NNLS because it is simple both conceptually and implementation wise: the projection $[\boldsymbol{x}]_\Omega$ is trivial, and the stepsize $\gamma^k$ can be computed using a standard

*University of Texas at Austin, Austin, TX, USA
†Max-Planck-Institute for Biological Cybernetics, Tübingen, Germany

linesearch procedure [3]. Unfortunately, GP has some drawbacks that it inherits from its *unconstrained* counterpart steepest descent; e.g., *zig-zagging* or *jamming* that can make convergence very slow [3, 27].

The slow convergence of steepest descent has been addressed by several techniques. Amongst these, beyond the (perhaps best-known) idea of conjugate gradients, the development of Barzilai and Borwein (BB) [1] especially stands out: not only is it computationally efficient [7, 14] but is also convergent despite exhibiting *non-monotonic* descent. Barzilai and Borwein proposed for the *unconstrained* steepest-descent iteration $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k - \gamma^k \nabla f(\boldsymbol{x}^k)$, the following two stepsizes (where $\Delta\boldsymbol{x}^k = \boldsymbol{x}^k - \boldsymbol{x}^{k-1}$ and $\Delta\boldsymbol{f}^k = \nabla f(\boldsymbol{x}^k) - \nabla f(\boldsymbol{x}^{k-1}) = \boldsymbol{A}^T\boldsymbol{A}\Delta\boldsymbol{x}^k$):

$$\gamma^k = \frac{\|\Delta\boldsymbol{x}^k\|^2}{\langle \Delta\boldsymbol{x}^k, \Delta\boldsymbol{f}^k\rangle} = \frac{\|-\gamma^{k-1}\nabla f(\boldsymbol{x}^{k-1})\|^2}{\langle \Delta\boldsymbol{x}^k, \boldsymbol{A}^T\boldsymbol{A}\Delta\boldsymbol{x}^k\rangle} = \frac{\|\nabla f(\boldsymbol{x}^{k-1})\|^2}{\langle \nabla f(\boldsymbol{x}^{k-1}), \boldsymbol{A}^T\boldsymbol{A}\nabla f(\boldsymbol{x}^{k-1})\rangle}, \tag{2a}$$

$$\text{and} \quad \gamma^k = \frac{\langle \Delta\boldsymbol{x}^k, \Delta\boldsymbol{f}^k\rangle}{\|\Delta\boldsymbol{f}^k\|^2} = \frac{\langle \Delta\boldsymbol{x}^k, \boldsymbol{A}^T\boldsymbol{A}\Delta\boldsymbol{x}^k\rangle}{\|\boldsymbol{A}^T\boldsymbol{A}\Delta\boldsymbol{x}^k\|^2} = \frac{\langle \nabla f(\boldsymbol{x}^{k-1}), \boldsymbol{A}^T\boldsymbol{A}\nabla f(\boldsymbol{x})^{k-1}\rangle}{\|\boldsymbol{A}^T\boldsymbol{A}\nabla f(\boldsymbol{x}^{k-1})\|^2}. \tag{2b}$$

Convergence of steepest-descent run with *either of* (2a) and (2b) was first proved for a two variable quadratic problem [1]; later convergence for the general *unconstrained* convex quadratic case was established [28].

The BB stepsizes (2) were found to accelerate steepest descent considerably [1, 14, 28], which makes one think whether they similarly benefit GP. Unfortunately, despite the strong resemblance of GP to steepest-descent, naïvely plugging in the stepsizes (2) into GP does not work. Dai and Fletcher [14] presented a counter-example showing that GP fails to converge when run with BB steps (we reproduce a similar counter-example in Appendix A, Figure 6). Thus it seems that to ensure convergence for *constrained* setups, some form of linesearch is almost inevitable when invoking the BB formulae (or variants thereof). Indeed, this observation is reaffirmed given the numerous methods in the literature that invoke linesearch when using BB steps [5–8, 13–15, 17].

**Our approach.** In light of the above discussion, we explore the possibility of using BB steps in a convergent GP setup without resorting to line search. Consider therefore, the following two typical alternatives to linesearch: (a) a constant stepsize ($\gamma^k \leq 2/L$, where $L$ is the Lipschitz constant of $\nabla f(\boldsymbol{x})$); or (b) stepsizes $\gamma^k$ given by sequence $\{\beta^k\}$ of diminishing scalars (DS) that satisfy[1], e.g.,

$$\text{(i)} \lim_{k\to\infty} \beta^k = 0, \quad \text{and} \quad \text{(ii)} \lim_{k\to\infty} \sum\nolimits_{i=1}^{k} \beta^i = \infty. \tag{3}$$

Alternative (a) cannot be used as the BB steps vary from iteration to iteration. However, Alternative (b) can be combined with BB steps, whereby we may establish convergence under mild conditions (see §2.4 for details). Besides convergence, there are two other reasons that motivate us to invoke diminishing scalars. First they align well with BB steps, since akin to BB methods, descent based on DS is also usually non-monotonic. Second, and more importantly, the use of DS is as yet uninvestigated in the context of BB methods.

The use of DS, however, has its share of difficulties. Although theoretically elegant, DS are not always practical as the diminishing sequence must be carefully selected—for an excellent example on how a wrong choice of the diminishing sequence can be disastrous see [26, pg. 1578]. So to reduce dependence on DS, we propose to *not* invoke the sequence (3) out-of-the-box, but rather use it in a relaxed fashion (see §2.2).

The next difficulty is that even our relaxed use of DS is not sufficient to ensure a highly competitive algorithm. The reason is that even though the DS strategy helps ensure convergence, the actual underlying difficulty posed by BB steps remains unaddressed. In other words, although the clash between projection and the BB steps (see Figure 1) is suppressed by diminishing scalars, it is hardly eliminated. To counter this clash we introduce a crucial modification to the BB steps themselves. Our modification leads to rapid (empirical) convergence. The details on our blending of DS with our modifications to the BB steps are described in Section 2 below, wherein we derive our algorithm following a series of illustrative experiments that also serve to ratify our design choices.

[1]Other choices are also possible, e.g., $\lim_{k\to\infty} \sum_{i=1}^{k} \beta^i = \infty$, and $\lim_{k\to\infty} \sum_{i=1}^{k} (\beta^i)^2 < \infty$.

## 2 Algorithm

We begin our discussion by demonstrating how diminishing scalars combined with BB steps fare in comparison with some known BB approaches. Admittedly, actual performance of all methods can vary significantly in the face of difficulties such as ill-conditioning, so to avoid getting lost in numerical concerns and to illustrate an initial insight into how the different approaches distinguish themselves, we begin our experimentation with a well-conditioned, large, sparse matrix. We remark that the NNLS problem in this section has been carefully chosen to illustrate the major character and deficiency of each method; obviously this does imply any claims about differences with regard to the performance of the methods across all NNLS instances. For example, if the NNLS solution happens to be the solution to an unconstrained least squares problem, all the methods mentioned in this section converge (including *unmodified–unconstrained* BB), despite some being non-convergent otherwise.

We experiment with a sparse matrix $\boldsymbol{A}$ of size $12288 \times 4096$ with 16.57% nonzeros. This matrix has full-rank, and its smallest singular value $\sigma_{\min}(\boldsymbol{A}) = 0.0102$; the largest one $\sigma_{\max}(\boldsymbol{A}) = 0.5779$. We simulate a 'true' nonnegative solution $\boldsymbol{x}^*$ to have 26.12% (1070) nonzeros, and to satisfy $\boldsymbol{A}\boldsymbol{x}^* = \boldsymbol{b}$ exactly. Given $\boldsymbol{A}$ and $\boldsymbol{b}$ we solve the associated NNLS problem (1) using the following three algorithms:

1. Ordinary Algorithm (OA) – Gradient projection with an alternating use of BB steps (2);
2. OA + LineSearch (OA+LS) – OA with occasional linesearch that ensures *monotonic* descent;
3. OA + Diminishing Scalar (OA+DS) – OA, but with stepsize $\beta^k\gamma^k$, where a diminishing scalar $\beta^k$ satisfies (3).

All three algorithms were implemented in MATLAB. For OA+LS we invoked Armijo along projection arc linesearch [3] every fifth iteration by using a reference iteration—this idea is similar to that in [14]; the reference iteration helps ensure overall convergence. For OA+DS we experimented with several diminishing sequences and have reported the best results. We ran all three algorithms with a convergence tolerance[2] of $\|\nabla f_+\|_\infty < 10^{-8}$, where $\nabla f_+$ is the *projected-gradient*: (hereafter $\partial_i f(\boldsymbol{x})$ is used as a shorthand for $(\nabla f(\boldsymbol{x}))_i$)

$$[\nabla f_+(\boldsymbol{x})]_i = \begin{cases} \min\{0, \partial_i f(\boldsymbol{x})\}, & \text{if} \quad x_i = 0, \\ \partial_i f(\boldsymbol{x}), & \text{if} \quad x_i > 0. \end{cases} \tag{4}$$

Notice that $\|\nabla f_+(\boldsymbol{x}^*)\|_\infty = 0$ must hold at optimality.

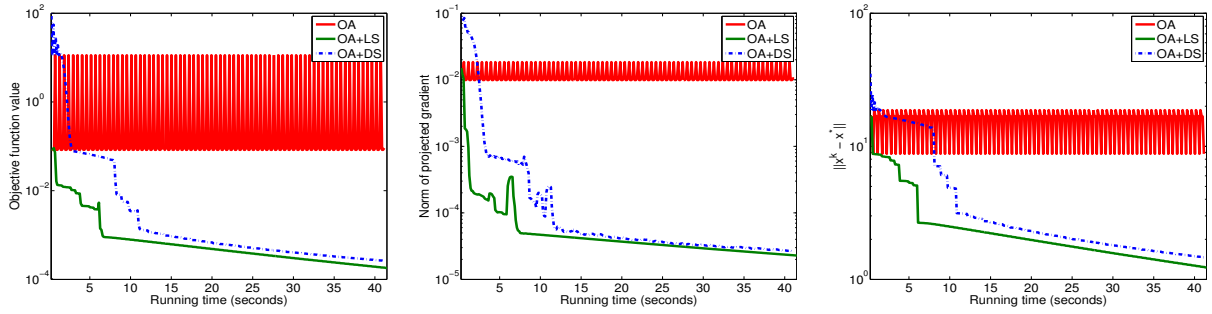


Figure 1: Objective function value (left), norm of projected gradient $\|\nabla f_+(\boldsymbol{x}^k)\|_\infty$ (middle), and true error $\|\boldsymbol{x}^k - \boldsymbol{x}^*\|$ (right) versus running time (in seconds) for OA, OA+LS, and OA+DS.

Figure 1 reports the result of experiment with the three algorithms mentioned above. It plots the objective function value, the norm of the projected-gradient, and the norm of error (i.e., distance to the true solution), against the running time in seconds. As also observed by Dai and Fletcher [14], OA oscillates. From the

[2]This tolerance is already very tight for first-order methods. We will later see that for experiments with real-world data such high tolerances are often not achieved.

figure we see that both OA+LS and OA+DS converge, though, linesearch is seemingly more helpful as OA+LS converges faster than OA+DS. Although OA+LS approaches the convergence tolerance the quickest, one notable behavior of OA+DS is that it rapidly catches up and eventually satisfies the convergence tolerance, even with a rate similar to OA+LS near the solution. The figure suggests that despite being slower than OA+LS, algorithm OA+DS is still a viable candidate. But clearly it is not competitive enough, especially considering that it requires user intervention to tune the diminishing sequence. A new approach is thus called for to improve OA+DS, and the sequel uncovers one.

## 2.1 Subspace BB Steps

To avoid the dominance of diminishing scalars while ensuring convergence, we need to address the deeper underlying problem: the oscillatory behavior exhibited by the *unconstrained* BB method when mixed with projections. When do oscillations (see Figure 6) happen? Recall that the unconstrained BB generates a sequence converging to the unconstrained optimum, so it is the projection step that derails such a sequence by pulling it towards the nonnegative orthant. In ordinary gradient projection, the projection step is a simple device for enforcing feasibility and it does not "break" the algorithm as the stepsize is always chosen to ensure descent. Quite logically, we may also conclude that some restriction to the BB step is needed, especially when the projection actually affects the current iterate.

We identified (the evident fact) that non-monotonicity of the BB step is the source of difficulties, but we also know that at the same time it is the key ingredient for rapid convergence. Thus, to alleviate oscillation without curtailing the non-monotonicity of the BB steps, we once again recall that the unconstrained BB is guaranteed to converge. Now, suppose that oscillations start appearing. If we knew which variables were active, i.e., zero, at the solution, we could reduce the optimization to an unconstrained problem over the *inactive* variables alone. Then we could compute the solution to this reduced problem by restricting the computation of BB steps to the inactive variables only. Note that we can obtain the constrained optimum by simply incorporating the active variables into this unconstrained solution.

This is a key observation behind active set methods, and it proves key in the development of our method too. Specifically, we partition the variables into *active* and *working* sets, carrying out the optimization over the working set alone. In addition, since the gradient is readily available, we exploit it to refine the active set and obtain the *binding* set; both are formalized below.

**Definition 2.1** (Active & binding sets)**.** Given $\boldsymbol{x}$, the *active* set $\mathcal{A}(\boldsymbol{x})$ and the *binding* set $\mathcal{B}(\boldsymbol{x})$ are defined as

$$\mathcal{A}(\boldsymbol{x}) = \big\{ i \;\big|\; x_i = 0 \big\},, \tag{5}$$

$$\mathcal{B}(\boldsymbol{x}) = \big\{ i \;\big|\; x_i = 0,\ \partial_i f(\boldsymbol{x}) > 0 \big\}. \tag{6}$$

The role of the binding set is simple: variables bound at the current iteration are guaranteed to be active at the next. Specifically, let us denote an orthogonal projection onto nonnegative orthant by $[\cdot]_+$; if $i \in \mathcal{B}(\boldsymbol{x}^k)$ and we iterate $\boldsymbol{x}^{k+1} = [\boldsymbol{x}^k - \gamma^k \nabla f(\boldsymbol{x}^k)]_+$ (for $\gamma^k > 0$), then since $x_i^{k+1} = [x_i^k - \gamma^k \partial_i f(\boldsymbol{x}^k)]_+ = 0$, the membership $i \in \mathcal{A}(\boldsymbol{x}^{k+1})$ holds. Therefore, if we know that $i \in \mathcal{B}(\boldsymbol{x}^k)$, we may discard $x_i^k$ from the update. Now, to employ this idea in conjunction with the BB step, we first compute $\mathcal{B}(\boldsymbol{x}^k)$ and then confine the computation of the stepsize to the *subspace* defined by $j \notin \mathcal{B}(\boldsymbol{x}^k)$. Formally, we propose to replace the basic BB steps (2) by the *subspace-BB* steps:

$$\alpha^k = \frac{\|\nabla \tilde{f}^{k-1}\|^2}{\langle \nabla \tilde{f}^{k-1}, \boldsymbol{A}^T \boldsymbol{A} \nabla \tilde{f}^{k-1} \rangle}, \tag{7a}$$

$$\text{or} \qquad \alpha^k = \frac{\langle \nabla \tilde{f}^{k-1}, \boldsymbol{A}^T \boldsymbol{A} \nabla \tilde{f}^{k-1} \rangle}{\|\boldsymbol{A}^T \boldsymbol{A} \nabla \tilde{f}^{k-1}\|^2}. \tag{7b}$$

where $\nabla \tilde{f}^{k-1}$ is defined as $\nabla_i \tilde{f}^{k-1} = \partial_i \nabla f(\boldsymbol{x}^{k-1})$ for $i \notin \mathcal{B}(\boldsymbol{x}^k)$, $\nabla_i \tilde{f}^{k-1} = 0$ otherwise. Notice that $\nabla \tilde{f}^{k-1} = \nabla f_+(\boldsymbol{x}^{k-1})$ only if $\mathcal{B}(\boldsymbol{x}^k) = \mathcal{B}(\boldsymbol{x}^{k-1})$.

Using the subspace-BB steps (7) we now modify OA+DS to obtain the iteration

$$\boldsymbol{x}^{k+1} = [\boldsymbol{x}^k - \beta^k \cdot \alpha^k \nabla f(\boldsymbol{x}^k)]_+, \tag{8}$$

which defines another algorithm, which we call SA+DS (for Subspace Algorithm + DS). To illustrate how SA+DS performs in comparison to OA+DS, with an identical choice of the diminishing scalar sequence, we repeat the experiment of Figure 1, running it this time with SA+DS. Figure 2 compares the objective function, projected-gradient norm, and the error norm achieved by OA+DS to those attained by SA+DS. Since both algorithms are run with the same sequence of diminishing scalars, the vast difference in performance seen in the plots, may be attributed to the subspace-BB steps. Also note that in contrast to all other methods shown so far (Figs. 1, 2), SA+DS manages to satisfy the convergence criterion $\|\nabla f_+\|_\infty < 10^{-8}$ fairly rapidly.

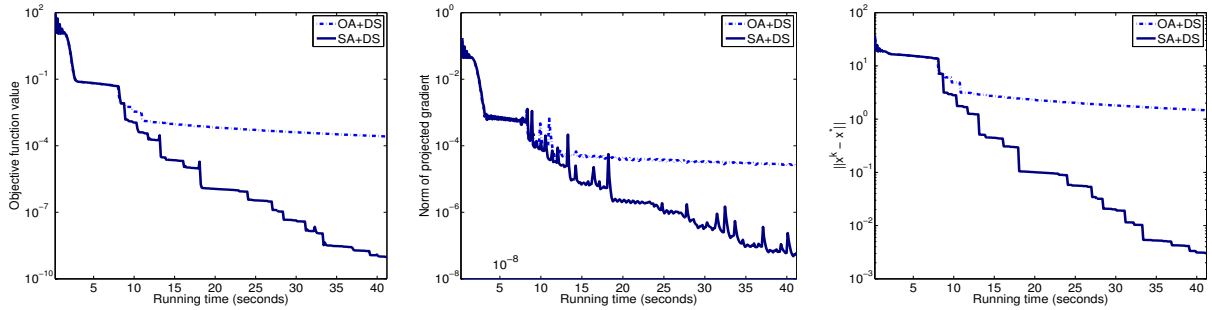


Figure 2: Objective function value (left), projected gradient norm (middle), and true error $\|\boldsymbol{x}^k - \boldsymbol{x}^*\|$ (right) versus running time (in seconds) for OA+DS and SA+DS.

## 2.2 Diminishing optimistically

Having seen the benefit of subspace-BB steps in SA+DS, we now take a closer look at the role of diminishing scalars in SA+DS. As already mentioned, gradient projection methods are in general highly sensitive to the choice of the diminishing sequence, which must be chosen carefully to obtain good empirical performance. Even though Figure 2 shows SA+DS to exhibit good performance, without proper tuning such performance is usually not easy to attain. Figure 3 illustrates this difficulty by showing results of running SA+DS with various choices for the diminishing sequence: one observes that the subspace-BB steps do not help much if the "wrong" diminishing sequence is used.

However, one does notice that the subspace-BB steps help in a robust manner, that is, they consistently improve (convergence speed of SA+DS) as the effect of diminishing sequence weakens. To investigate this behavior further, we run SA+DS with $\beta^k = 1$, i.e., without scaling the subspace-BB steps. While considering the impact of subspace-BB steps without diminishing scalars, one might wonder if linesearch combined with subspace-BB is superior. So we also test a method that combines subspace-BB steps with linesearch (SA+LS). We compare SA+DS run with $\beta^k = 1$ against SA+LS and the best-performing SA+DS (from Figure 3): the results are pleasantly surprising and are shown in Figure 4.

Curiously, Figure 4 suggests that the subspace-BB alone can produce converging iterates and even the lazy linesearch may affect it adversely! In view of Figures 3 and 4, we conclude that either scaling the stepsize via $\beta^k$ or invoking linesearch can adversely affect the convergence speed of SA+DS, whereas even *near constant* $\{\beta^k\}$ seems to retain SA+DS's convergence. This behavior is opposite to that exhibited by the non-subspace method OA+DS, for which the diminishing scalars not only control the convergence speed but also dominate the convergence itself. This contrast (Fig. 4) is a crucial distinguishing feature of SA+DS that also plays a key role toward accelerating SA+DS empirically. Therefore, to retain empirical benefits of subspace steps while still guaranteeing convergence, we propose to relax the application of the diminishing scalar by using an "optimistic" diminishment strategy.

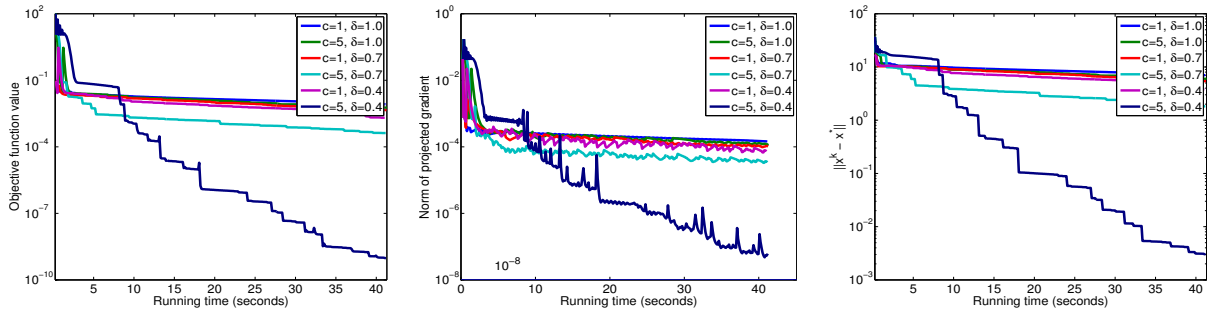


Figure 3: Empirical convergence of the SA+DS with respect to different choices of the diminishing sequence. The choices used were $\beta^k = c/k^\delta$. In this experiment, $c = 5$, and $\delta = 0.4$ eventually led to the fastest convergence. As expected, if $\beta^k$ decays too rapidly, the convergence gets impaired and the algorithm eventually stalls due to limited machine precision. The exact values of the diminishing parameters are not as important as the message that SA+DS's convergence is very sensitive to the diminishing sequence employed.

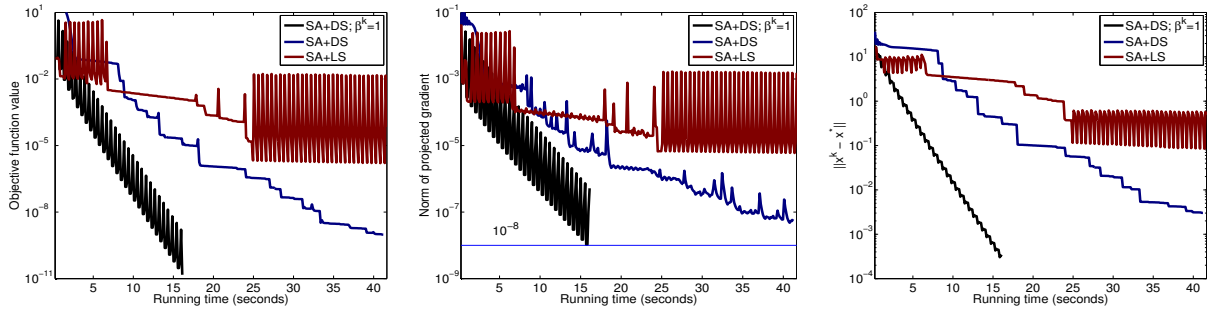


Figure 4: Objective function value (left), projected gradient norm (middle) and true error $\|\boldsymbol{x}^k - \boldsymbol{x}^*\|$ (right) versus running time for SA+DS with $\beta^k = 1$ compared with the best performing instance of SA+DS selected from Figure 2.

Our strategy is as follows. We scale the subspace-BB step (7) with some constant scalar $\beta$ for a fixed number, say $M$, of iterations. Then, we check whether a descent condition is satisfied. If yes, then the method continues for $M$ more iterations with the *same* $\beta$; if no, then we diminish the scaling factor $\beta$. The diminishment is "optimistic" because even when the method fails to satisfy the descent condition that triggers diminishment, we merely shrink $\beta$ once and continue using it for the next $M$ iterations. We remark that superficially this strategy might seem similar to a occasional linesearch, but it is fundamentally different: unlike linesearch it does not enforce monotonicity after failing to descend for a prescribed number of iterations. We formalize this below.

Suppose that the method is at iteration $c$, and then iterates with a constant $\beta^c$ for the next $M$ iterations, so that from the current iterate $\boldsymbol{x}^c$, we compute

$$\boldsymbol{x}^{k+1} = [\boldsymbol{x}^k - \beta^c \cdot \alpha^k \nabla f(\boldsymbol{x}^k)]_+, \tag{9}$$

for $k = c, c+1, \cdots, c+M-1$, where $\alpha^k$ is computed via (7). Now, for $\boldsymbol{x}^{c+M}$, we check the *descent condition*

$$f(\boldsymbol{x}^c) - f(\boldsymbol{x}^{c+M}) \geq \sigma \langle \nabla f(\boldsymbol{x}^c), \boldsymbol{x}^c - \boldsymbol{x}^{c+M} \rangle, \tag{10}$$

for some $\sigma \in (0,1)$. If $\boldsymbol{x}^{c+M}$ passes the test (10), then we reuse $\beta^c$ and set $\beta^{c+M} = \beta^c$; otherwise we diminish $\beta^c$ and set

$$\beta^{c+M} \leftarrow \eta \cdot \beta^c, \tag{11}$$

for some $\eta \in (0,1)$. After adjusting $\beta^{c+M}$ the method repeats the update (9) for another $M$ iterations.

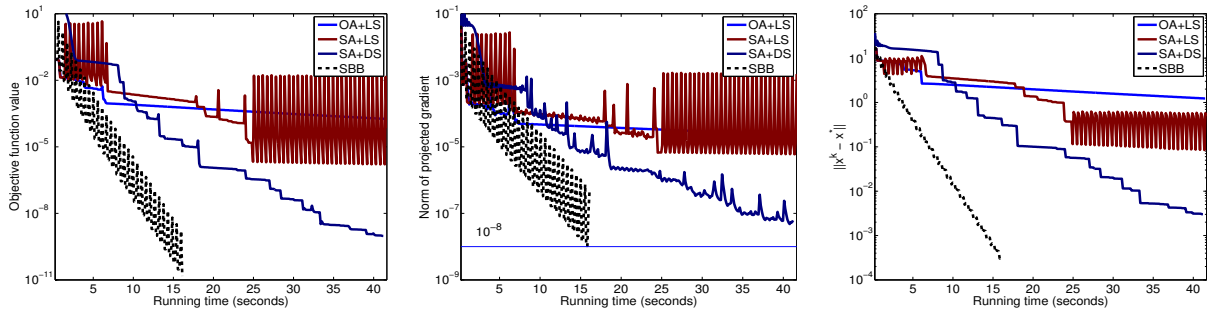


Figure 5: Objective function value (left), projected gradient norm (middle) and true error $\|\boldsymbol{x}^k - x^*\|$ (right) versus running time for all the algorithms. In these plots we have shown SBB using a dashed line to distinguish it from other methods. This figure is essentially a summary of the information shown in Figures 1, 2, 3, and 4.

## 2.3 SBB: Subspace-BB with optiministic diminishment

With the subspace-BB steps and the optimistic diminishment strategy we now have all the ingredients necessary to present our final NNLS algorithm: Subspace BB (SBB). The termination criterion used by our algorithm is approximate satisfaction of the KKT optimality conditions, which for NNLS reduce to checking whether the norm of the projected gradient

$$\max_{i \notin \mathcal{B}(\boldsymbol{x}^k)} \left|\partial_i f(\boldsymbol{x}^k)\right| \leq \epsilon, \quad \text{for given threshold} \quad \epsilon \geq 0. \tag{12}$$

Note that condition (12) is equivalent to (4), which is the condition used for checking convergence by all the variants developed in this paper. Algorithm 1 presents pseudo-code of SBB.

**Algorithm:** SBB

Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until the stopping criteria* (12) *met* **do**
  $\hat{\boldsymbol{x}}^0 \leftarrow \boldsymbol{x}^{i-1}$ and $\hat{\boldsymbol{x}}^1 \leftarrow \boldsymbol{x}^i$;
  **for** $j = 1, \cdots, M$ **do** /* Subspace BB */
    Compute $\alpha^j$ using (7a) and (7b) alternatively;
    $\hat{\boldsymbol{x}}^{j+1} \leftarrow [\hat{\boldsymbol{x}}^j - \beta^i \cdot \alpha^j \nabla f(\hat{\boldsymbol{x}}^j)]_+$;
  **if** $\hat{\boldsymbol{x}}^M$ *satisfies* (10) **then**
    $\boldsymbol{x}^{i+1} \leftarrow \hat{\boldsymbol{x}}^M$, and $\beta^{i+1} \leftarrow \beta^i$ ;
  **else** /* Diminish Optimistically */
    $\beta^{i+1} \leftarrow \eta\beta^i$, where $\eta \in (0, 1)$;

**Algorithm 1:** The Subspace BB algorithm (SBB).

Figure 5 summarizes the best performing methods from Figures 1 to 4, while highlighting that SBB outperforms all other variants.

## 2.4 Convergence Analysis

In this section we analyze some theoretical properties of SBB. First, we establish convergence under the assumption that $f$ is strictly convex (or equivalently that $\boldsymbol{A}^T\boldsymbol{A}$ has full-rank). Later, with an additional mild assumption, we show that the proof easily extends to the case where $\boldsymbol{A}^T\boldsymbol{A}$ is rank-deficient. Finally, we briefly discuss properties such as convergence rate and the identification of active variables.

We remind the reader that when proving convergence of iterative optimization routines, one often assumes Lipschitz continuity of the objective function. The objective function for NNLS is only *locally* Lipschitz

continuous, i.e., there exists a constant $L$, such that

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L\|\boldsymbol{x} - \boldsymbol{y}\|,\ \forall \boldsymbol{x}, \boldsymbol{y} \in \Omega, \quad \text{or equivalently,} \quad \|\nabla f(\boldsymbol{x})\| \leq L,\ \forall \boldsymbol{x} \in \Omega,$$

where $\Omega$ is an appropriate compact set.

Even though the domain of NNLS ($\boldsymbol{x} \geq 0$) does not define such a compact set, we can essentially view the domain to be compact. To see why, let $\boldsymbol{x}^u$ and $\boldsymbol{x}^*$ denote the *unconstrained* least-squares solution and the NNLS solution, respectively. Let $\boldsymbol{x}^p = [\boldsymbol{x}^u]_+$ be the projection of $\boldsymbol{x}^u$ onto the nonnegative orthant. Then, the following inequalities are immediate

$$\|\boldsymbol{A}\boldsymbol{x}^u - \boldsymbol{b}\| \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\| \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^p - \boldsymbol{b}\|.$$

Using these inequalities we can derive a simple upper bound $U$ on $\|\boldsymbol{x}^*\|$ as follows:

$$\begin{aligned} & \|\boldsymbol{A}\boldsymbol{x}^*\| - \|\boldsymbol{b}\| \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^* - \boldsymbol{b}\| \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^p - \boldsymbol{b}\|, \\ \text{hence} \quad & \sigma_{\min}(\boldsymbol{A}) \cdot \|\boldsymbol{x}^*\| \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^*\| \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^p - \boldsymbol{b}\| + \|\boldsymbol{b}\|, \\ & \|\boldsymbol{x}^*\| \quad \leq \quad \frac{\|\boldsymbol{A}\boldsymbol{x}^p - \boldsymbol{b}\| + \|\boldsymbol{b}\|}{\sigma_{\min}(\boldsymbol{A})} \quad = \quad U, \end{aligned} \tag{13}$$

where $\sigma_{\min}(\boldsymbol{A}) > 0$ denotes the smallest singular value of $\boldsymbol{A}$. Thus, the domain of NNLS can be effectively restricted to $\Omega = \{\boldsymbol{x} : 0 \leq \boldsymbol{x} \leq U\}$, and we may safely consider the NNLS objective to be Lipschitz continuous. Finally, to ensure that the iterates remain in $\Omega$, we can modify the projection $[\boldsymbol{x}]_+$ so that no element $x^i$ grows larger than $U$. We will assume this upper bound implicitly in the discussion below (also in Algorithm 1), and avoid mentioning it for simplicity of presentation.[3]

For clarity in our proofs, we introduce some additional notation. Let $M$ be the fixed number of subspace-BB iterations, so that we check the descent condition (10) only once every $M$ iterations. We index these $M$-th iterates with $\mathcal{I} = \{1,\ M+1,\ 2M+1,\ 3M+1,\ \cdots\}$, and then consider the sequence $\{\boldsymbol{x}^r\},\ r \in \mathcal{I}$ generated by SBB. Let $\boldsymbol{x}^*$ denote the optimal solution to the problem. We prove that $\{\boldsymbol{x}^r\} \to \boldsymbol{x}^*$ as $r \to \infty$.

Suppose that the diminishment step (11) is triggered only a finite number of times. Then, there exists a sufficiently large $K$ such that

$$f(\boldsymbol{x}^r) - f(\boldsymbol{x}^{r+1}) \geq \sigma\langle \nabla f(\boldsymbol{x}^r), \boldsymbol{x}^r - \boldsymbol{x}^{r+1}\rangle,$$

for all $r \in \mathcal{I},\ r \geq K$. In this case, we may view the sequence $\{\boldsymbol{x}^r\}$ as if it were generated by an ordinary gradient projection scheme, whereby convergence of $\{\boldsymbol{x}^r\}$ follows from [3]. Therefore, to prove the convergence of the entire algorithm, it is sufficient to discuss the case where (11) is invoked infinitely often.

Given an infinite sequence $\{\boldsymbol{x}^r\}$, there is a corresponding sequence $\{\beta^r\}$ which by construction is diminishing. Recall that the diminishing sequence with unbounded sum condition (3) ensures convergence [3]. We show that multiplying $\{\beta^r\}$ with the subspace-BB steps preserves the condition (3), and hence inherits the convergence guarantees.

**Proposition 2.2.** *In Algorithm 1, the stepsize $\beta^r \cdot \alpha^r$ satisfies*

$$\textit{(i)}\ \lim_{r\to\infty} \beta^r \cdot \alpha^r = 0, \quad \textit{and} \quad \textit{(ii)}\ \lim_{r\to\infty} \sum\nolimits_{i=1}^{r} \beta^i \cdot \alpha^i = \infty.$$

*Proof.* From the definition of the subspace-BB (7), simple algebra shows that that

$$\frac{\|\nabla \tilde{f}^{k-1}\|^2}{\langle \nabla \tilde{f}^{k-1}, \boldsymbol{A}^T\boldsymbol{A}\nabla \tilde{f}^{k-1}\rangle} = \frac{\langle \boldsymbol{y}_1, \boldsymbol{y}_1\rangle}{\langle \boldsymbol{y}_1, \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{y}_1\rangle}, \qquad \text{and} \qquad \frac{\langle \nabla \tilde{f}^{k-1}, \boldsymbol{A}^T\boldsymbol{A}\nabla \tilde{f}^{k-1}\rangle}{\|\boldsymbol{A}^T\boldsymbol{A}\nabla \tilde{f}^{k-1}\|^2} = \frac{\langle \boldsymbol{y}_2, \boldsymbol{y}_2\rangle}{\langle \boldsymbol{y}_2, \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{y}_2\rangle},$$

[3]Note that in practice we need not compute $U$ at all, and without compromising the theoretical guarantees we can replace it by the maximum value permitted by the machine, provided that the solution $x^*$ is representable without overflow.

where $\boldsymbol{y}_1 = \nabla \tilde{f}^{k-1}$ and $\boldsymbol{y}_2 = \left(\boldsymbol{A}^T\boldsymbol{A}\right)^{1/2} \nabla \tilde{f}^{k-1}$. Since $\boldsymbol{A}^T\boldsymbol{A}$ is a positive definite matrix, for all $\boldsymbol{y} \neq \boldsymbol{0}$, its Rayleigh quotient satisfies

$$0 < \lambda_{\min}(\boldsymbol{A}^T\boldsymbol{A}) \leq \frac{\langle \boldsymbol{y}, \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{y}\rangle}{\langle \boldsymbol{y}, \boldsymbol{y}\rangle} \leq \lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A}),$$

where $\lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})$ and $\lambda_{\min}(\boldsymbol{A}^T\boldsymbol{A})$ denote the largest and the smallest eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$, respectively. Now, we can see that at any given iteration $r$, the subspace-BB step $\alpha^r$ satisfies,

$$\frac{1}{\lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})} \;\leq\; \alpha^r \;\leq\; \frac{1}{\lambda_{\min}(\boldsymbol{A}^T\boldsymbol{A})}. \tag{14}$$

Since $\lim_{r\to\infty} \beta^r = 0$ by construction, we can show that Condition (i) of (3) also holds for $\beta^r\alpha^r$, since

$$\lim_{r\to\infty} \beta^r \cdot \alpha^r \;\leq\; \frac{1}{\lambda_{\min}(\boldsymbol{A}^T\boldsymbol{A})} \cdot \lim_{r\to\infty} \beta^r = 0.$$

Similarly, we also obtain Condition (ii) of (3), since

$$\lim_{r\to\infty} \sum_{i=1}^{r} \beta^i \cdot \alpha^i \;\geq\; \frac{1}{\lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})} \cdot \lim_{r\to\infty} \sum_{i=1}^{r} \beta^i = \infty. \qquad \square$$

Using this proposition we now state the main convergence theorem. The proof is essentially that of gradient descent with diminishing stepsizes; we adapt it by showing some additional properties of $\{\boldsymbol{x}^r\}$.

**Theorem 2.3** (Convergence)**.** *Let the objective function $f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$ be strictly convex, $\{\boldsymbol{x}^r\}$ be a sequence generated by Algorithm 1, and $\boldsymbol{x}^* = \operatorname{argmin}_{\boldsymbol{x}} f(\boldsymbol{x})$. Then $\{f(\boldsymbol{x}^r)\} \to f(\boldsymbol{x}^*)$ and $\boldsymbol{x}^r \to \boldsymbol{x}^*$.*

*Proof.* Consider the update (9) of Algorithm 1, we can rewrite it as

$$\boldsymbol{x}^{r+1} = \boldsymbol{x}^r + \beta^r \cdot \alpha^r \boldsymbol{d}^r, \tag{15}$$

where the descent direction $\boldsymbol{d}^r$ satisfies

$$d_i^r = \begin{cases} 0, & \text{if } i \in \mathcal{B}(\boldsymbol{x}^r), \\ -\min\left\{\frac{x_i^r}{\beta^r\cdot\alpha^r},\; \partial_i f(\boldsymbol{x}^r)\right\}, & \text{if } x_i^r > 0,\; \partial_i f(\boldsymbol{x}^r) > 0, \\ -\partial_i f(\boldsymbol{x}^r), & \text{otherwise.} \end{cases} \tag{16}$$

With (16), and since there exists at least one $\partial_i f(\boldsymbol{x}^r) > 0$ unless $\boldsymbol{x}^r = \boldsymbol{x}^*$, we can conclude that there exists a constant $c_1 > 0$ such that

$$-\langle \nabla f(\boldsymbol{x}^r), \boldsymbol{d}^r\rangle \;\geq\; \sum_{i\notin\mathcal{B}(\boldsymbol{x}^r)} m_i^2 \;\geq\; c_1\|\nabla f(\boldsymbol{x}^r)\|^2 \;>\; 0, \tag{17}$$

where $m_i = \min\left\{\frac{x_i^r}{\beta^r\cdot\alpha^r},\; \partial_i f(\boldsymbol{x}^r)\right\}$. Similarly, it can also be shown that there exists $c_2 > 0$ such that

$$\|\boldsymbol{d}^r\|^2 \leq c_2\|\nabla f(\boldsymbol{x}^r)\|^2. \tag{18}$$

Using Proposition 2.2 with inequalities (17) and (18), the proof is immediate from Proposition 1.2.4 in [3]. $\square$

To extend the above proof for rank-deficient $\boldsymbol{A}^T\boldsymbol{A}$, notice that ultimately our convergence proof rests on that of ordinary gradient projection with linesearch or diminishing scalars. Neither of these two requires $f$ to be strictly convex. The only difficulty that arises is in (13) and (14), where we compute the values $1/\sigma_{\min}(\boldsymbol{A})$ or $1/\lambda_{\min}(\boldsymbol{A}^T\boldsymbol{A})$. It can be shown that for the convex quadratic program

$$\text{minimize} \qquad \tfrac{1}{2}\boldsymbol{x}^T\boldsymbol{H}\boldsymbol{x} - \boldsymbol{c}^T\boldsymbol{x},$$

if one assumes $\boldsymbol{c}$ to be in the range of $\boldsymbol{H}$, then the BB step (2) is bounded above by $1/\lambda^+_{\min}(\boldsymbol{H})$, where $\lambda^+_{\min}$ denotes the smallest *positive* eigenvalue of $\boldsymbol{H}$ [15]. For our problem, we can equate this assumption on $\boldsymbol{c}$ to the assumption that $\boldsymbol{A}^T\boldsymbol{b}$ lies in the range of $\boldsymbol{A}^T\boldsymbol{A}$. Consequently, we can modify (13) and (14) to

$$\|\boldsymbol{x}^*\| \;\leq\; \frac{\|\boldsymbol{A}\boldsymbol{x}^p - \boldsymbol{b}\| + \|\boldsymbol{b}\|}{\sigma^+_{\min}(\boldsymbol{A})} \;=\; U, \qquad \text{and} \qquad \alpha^r \;\leq\; \frac{1}{\lambda^+_{\min}(\boldsymbol{A}^T\boldsymbol{A})},$$

where $\sigma^+_{\min}$ and $\lambda^+_{\min}$ denote the smallest *positive* singular- and eigenvalues respectively.[4]

Finally, to close our analysis we point out that since SBB may be viewed as a gradient projection method, it inherits properties such as the convergence rate [3] and identification of the active variables [11].

# 3 Related Work

Before moving on to numerical results, it is fitting to briefly review related work. Over the years, a variety of methods have been applied to solve NNLS. Several of these approaches are summarized in [9, Chapter 5]. The main approaches can be roughly divided into three categories.

The first category includes methods that were developed for solving linear inequality constrained least squares problems by transforming them into corresponding least distance problems [9, 31]. However, such transformations prove to be too costly for NNLS and do not yield much advantage, unless significant additional engineering efforts are made.

The second and more successful category of methods includes *active-set methods* [16] that do not involve transforming (1) into a corresponding minimum distance problem. Active set methods typically deal with one constraint per iteration, and the overall optimization problem is approximated by solving a series of equality-constrained problems; the equality constraints form the current active set that is then incrementally updated to construct the final active set. The famous NNLS algorithm of Lawson & Hanson [21] is an active set method, and has been the *de facto* method for solving (1) for many years. In fact, MATLAB continues to ship `lsqnonneg`, an implementation of the original Lawson-Hanson NNLS algorithm [21]. Bro and Jong [10] modified the latter algorithm and developed a method called Fast-NNLS (FNNLS) that is often faster than the Lawson-Hanson algorithm. The rationale behind FNNLS is simple as it accepts $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{b}$ instead of $\boldsymbol{A}$ and $\boldsymbol{b}$, thus taking advantage of the reduced dimensionality of $\boldsymbol{A}^T\boldsymbol{A}$ when $m \gg n$ for $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. However, constructing $\boldsymbol{A}^T\boldsymbol{A}$ is expensive, which makes the method prohibitive for large-scale problems, i.e., when both $m$ and $n$ are large.

The third category of methods includes algorithms based on more general iterative approaches which produce a sequence of intermediate solutions that converge to the solution of (1). For example, the gradient projection method [30] and some variants have been applied to NNLS [4, 24]. The main advantage of this class of algorithms is that by using information from the projected gradient step to obtain a good approximation of the final active set, one can handle multiple active constraints per iteration. However, the projected gradient approach frequently suffers from slow convergence (*zig-zagging*), a difficulty potentially alleviated by more sophisticated methods such as LBFGS-B [12] or TRON [22].

The method proposed in this paper belongs to the third category. Observe that since NNLS problem is one of the simplest constrained optimization problems, any modern constrained optimization technique can be applied to solve it. However, generic off-the-shelf approaches frequently fail to exploit the inherent advantages arising from the simplicity of the problem, resulting in unnecessary computational and implementation overheads. In the following section, we illustrate the computational performance of our method by comparing it with established optimization software, thereby providing empirical support to our claim.

We point out that there exist other BB-based methods that could also be applied to NNLS. However, in contrast to our optimistic diminishment approach, for guaranteeing convergence these approaches either employ linesearch procedures [5–8, 13, 17] or introduce an explicit active-variable identification step to utilize

[4] To run our method for rank-deficient $\boldsymbol{A}^T\boldsymbol{A}$ in practice, we do not need to compute $\sigma^+_{\min}(\boldsymbol{A})$ nor $\lambda^+_{\min}(\boldsymbol{A}^T\boldsymbol{A})$, as it is sufficient to place an arbitrary large upper bound $\alpha_U$ and a small positive lower-bound $\alpha_L$ on the subspace-BB computation (7). The diminishing $\{\beta^r\}$ safeguards from the stepsizes becoming too large or too small, thereby eventually ensuring convergence.

the *unconstrained* version of stepsizes (2) [15]. SPG [6] belongs to the group of methods employing a nonmonotonic linesearch, and it is one of the methods against which we compare not only because it provides a publicly available implementation, but also because it is highly competitive.

Recently, Hager and Zhang [18] developed an active set algorithm (ASA) that has two phases, namely, a nonmonotone gradient projection step and an unconstrained optimization step. It utilizes the BB step in the gradient projection step and restricts its computation in a way similar to that developed in this paper. In our notation, their modified BB steps can be written as

$$\frac{\|\Delta \boldsymbol{x}^k\|^2}{\langle \Delta \boldsymbol{x}^k, \nabla {f_+}^k - \nabla {f_+}^{k-1} \rangle}, \qquad \text{and} \qquad \frac{\langle \Delta \boldsymbol{x}^k, \nabla {f_+}^k - \nabla {f_+}^{k-1} \rangle}{\|\nabla {f_+}^k - \nabla {f_+}^{k-1}\|^2}.$$

ASA is reported to be competitive to LBFGS-B in a certain domain of their evaluation metric (See Figure 6.4 of [18], for $\tau \leq 1$, i.e., more stringent convergence criteria). Hence we do not report numerical results against ASA and compare against LBFGS-B.

# 4 Numerical Results

In this section, we compare the performance of our SBB method with the following methods: FNNLS [10], LBFGS-B [12], SPG [6]. We ran all experiments on a Linux machine, with an Intel Xeon 2.0GHz CPU and 16GB memory. We used MATLAB to interface algorithms and its multi-threading option was turned off to prevent skewing of results due to multiple cores. We note that the underlying implementations of LBFGS-B and SPG are in FORTRAN, while FNNLS and SBB have purely MATLAB implementations.

We begin with experimentation on synthetic data. In our first two sets of experiments, we simulate NNLS problems and assess the computational efficiency of various methods in a "clean" setup: for each dataset, we generate a nonnegative matrix $\boldsymbol{A}$ and compute an *unconstrained* observation vector $\boldsymbol{b}_t$ with a pre-determined sparse nonnegative solution $x^*$, i.e., we first compute $\boldsymbol{b}_t \leftarrow \boldsymbol{A}\boldsymbol{x}^*$. Although it is possible to form a NNLS problem with these $\boldsymbol{A}$ and $\boldsymbol{b}_t$, we further refine the generated $\boldsymbol{b}_t$ to avoid non-degeneracy at the solution. Specifically, given $\boldsymbol{A}, \boldsymbol{b}_t$ and $\boldsymbol{x}^*$, we identify $\mathcal{A}(\boldsymbol{x}^*)$, then generate an augmenting vector $\boldsymbol{y}$ such that $y_i > 0$ for $i \in \mathcal{A}(\boldsymbol{x}^*)$. Finally, a *constrained* observation vector $\boldsymbol{b}$ is produced by solving the system of equation $\boldsymbol{A}^T\boldsymbol{b} = \boldsymbol{A}^T\boldsymbol{b}_t - \boldsymbol{y}$. Notice that $\boldsymbol{x}^*$ is still a solution for the newly generated NNLS problem with $\boldsymbol{A}$ and $\boldsymbol{b}$, and it is "clean" as it satisfies the KKT complementarity conditions *strictly*.

As per the above description we generated NNLS problems of varying size (P1) and sparsity (P2)—the associated matrices are summarized in Table 1.

| **P-1** | P1-1 | P1-2 | P1-3 | P1-4 | P1-5 | P1-6 |
|---|---|---|---|---|---|---|
| Rows | 600 | 1,200 | 2,400 | 4,800 | 9,600 | 19,200 |
| Columns | 400 | 800 | 1,600 | 3,200 | 6,400 | 12,800 |
| #active | 300 | 594 | 1,181 | 2,369 | 4,738 | 9,464 |
| $\|\nabla f_+(\boldsymbol{x}^*)\|_\infty$ | 7.84e-12 | 5.82e-11 | 2.93e-10 | 2.19e-09 | 1.11e-08 | 6.70e-08 |
| **P-2** | P2-1 | P2-2 | P2-3 | P2-4 | P2-5 | P2-6 |
| #nnz | 1,225,734 | 2,445,519 | 3,659,062 | 4,866,734 | 6,068,117 | 7,263,457 |
| #active | 7,122 | 7,115 | 7101 | 7122 | 7095 | 7137 |
| $\|\nabla f_+(\boldsymbol{x}^*)\|_\infty$ | 2.89e-12 | 1.21e-11 | 2.38e-11 | 1.19e-10 | 1.59e-10 | 1.74e-10 |

Table 1: The size of dense and uniformly random nonnegative matrices $\boldsymbol{A}$ in data set P1 (above), and #nnz, i.e., number of nonzeros of the matrices (of size $25600 \times 9600$) in data set P2 (below). #active and $\|\nabla f_+(\boldsymbol{x}^*)\|_\infty$ denote the number of active variables in each $\boldsymbol{x}^*$ and the infinity norm of the projected gradient at $\boldsymbol{x}^*$ respectively.

After synthetic experiments, we also experiment on six datasets drawn from real world applications. We obtained the first two datasets, namely, 'ash958' and 'well1850' from the MatrixMarket[1]— they arise from problems solving least squares, and we impose nonnegativity constraints on the solution to form NNLS

[1] http://math.nist.gov/MatrixMarket

problems. We obtained the remaining four datasets, 'real-sim', 'mnist', 'news20', and 'webspam' from the collection of LIBSVM datasets[2]. On these four datasets one can interpret the use of NNLS as a regression method for predicting class labels.

| Matrix | ash958 | well1850 | real-sim | mnist | news20 | webspam |
|---|---|---|---|---|---|---|
| Rows | 958 | 1,850 | 72,309 | 60,000 | 19,996 | 350,000 |
| Columns | 292 | 712 | 20,958 | 780 | 1,355,191 | 254 |
| #nnz | 1,916 | 8,755 | 3,709,083 | 8,994,156 | 9,097,916 | 29,796,333 |

Table 2: The size and sparsity of real-world datasets. #nnz denotes the number of non-zero entries in the data matrix.

For each dataset, all methods were started at the same initial point $\boldsymbol{x}^0 = \mathbf{0}$.

| Method | | P1-1 | P1-2 | P1-3 | P1-4 | P1-5 | P1-6 |
|---|---|---|---|---|---|---|---|
| FNNLS | Time(sec.) | 0.20 | **0.31** | **4.69** | **40.41** | 15,526 | - |
| | $f(\boldsymbol{x})$ | 3.21e+06 | 2.87e+07 | 3.12e+08 | 6.27e+09 | 8.72e+10 | - |
| | #active | **300** | **594** | **1,181** | **2,369** | **4,738** | - |
| LBFGS-B | Time(sec.) | 0.41 | 5.15 | 39.18 | 207.24 | 1,516.4 | 8,830.9 |
| | $f(\boldsymbol{x})$ | 3.21e+06 | 2.87e+07 | 3.12e+08 | 6.27e+09 | 8.72e+10 | 1.07e+12 |
| | $\#f$ | 502 | 968 | 1,481 | 2,261 | 3,913 | 6,638 |
| | $\#\nabla f$ | 502 | 968 | 1,481 | 2,261 | 3,913 | 6,638 |
| | #active | 300 | 264 | 39 | 1,545 | 1,930 | 3,039 |
| | $\|\nabla f_+\|_\infty$ | 1.53e-05 | 1.07e-02 | 2.66e-02 | 2.72e-01 | 5.04 | 22.29 |
| SPG | Time(sec.) | 1.78 | 28.21 | 283.83 | 1,045.4 | 3,936.5 | 11,896 |
| | $f(\boldsymbol{x})$ | 3.21e+06 | 2.87e+07 | 3.12e+08 | 6.27e+09 | 8.72e+10 | 1.07e+12 |
| | $\#f$ | 3,605 | 4,712 | 10,000 | 10,000 | 10,000 | 10,000 |
| | $\#\nabla f$ | 2,303 | 2,892 | 6,130 | 5,984 | 5615 | 5390 |
| | #active | 298 | 594 | 42 | 150 | 2084 | 8 |
| | $\|\nabla f_+\|_\infty$ | 9.92e-07 | 6.61e-07 | 1.02 | 2.66 | 16.19 | 1.04 |
| SBB | Time(sec.) | **0.16** | 3.36 | 18.10 | 75.69 | **290.73** | **1,073.2** |
| | $f(\boldsymbol{x})$ | 3.21e+06 | 2.87e+07 | 3.12e+08 | 6.27e+09 | 8.72e+10 | 1.07e+12 |
| | $\#f$ | **2** | **2** | **2** | **3** | **4** | **4** |
| | $\#\nabla f$ | **285** | **285** | **238** | **372** | **443** | **452** |
| | #active | **300** | **594** | **1,181** | 2,368 | 4,737 | 9,464 |
| | $\|\nabla f_+\|_\infty$ | 6.89e-07 | 5.68e-07 | 8.34e-07 | 7.64e-07 | 9.60e-07 | 3.23e-07 |

Table 3: NNLS experiments on dataset P1. We used $\|\nabla f_+\|_\infty \leq 10^{-6}$ and $\#f \leq 10^4$ as the stopping criteria. In the above experiments, LBFGS-B terminates with its own condition on "linesearch" before it achieves the given $\|\nabla f_+\|_\infty$ value. From P1-3 to P1-6, we could not obtain comparable $\|\nabla f_+\|_\infty$ values from SPG within $10^5$ seconds, thereby we forced it to terminate after a fixed number of iterations.

Table 3 shows the running times for FNNLS, LBFGS-B, SPG, and SBB for the matrices in problem set P1. For small to mid-sized problems, e.g., P1-1 through P1-5, we observe that FNNLS is very competitive with other methods, though its performance starts to deteriorate rapidly with increasing matrix sizes, and it eventually fails to scale on P1-6. LBFGS-B, SPG and SBB generally scale better.

In Table 4 we show running time comparisons on dataset P2. As shown in the table, SBB generally outperforms other methods, and the difference becomes starker with increasing number of nonzeros (i.e., decreasing sparsity) in the matrix. Also note that the performance of FNNLS is substantially degraded in comparison to dense problems (dataset P1). We attribute this to its computation of $\boldsymbol{A}^T\boldsymbol{A}$. In other words, FNNLS inputs $\boldsymbol{A}^T\boldsymbol{A}$, and though this effectively reduces problem size when $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $m \gg n$, it is more likely to be dense even when $\boldsymbol{A}$ is highly sparse; so it may actually increase the amount of computation required to solve an equivalent problem.

[2]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets

| Method | | P2-1 | P2-2 | P2-3 | P2-4 | P2-5 | P2-6 |
|---|---|---|---|---|---|---|---|
| FNNLS | Time(sec.) | 1866.7 | 2745.6 | 2963.1 | 2911.9 | 2845.1 | 2799.8 |
| | $f(\boldsymbol{x})$ | 1.93e+09 | 7.06e+09 | 8.87e+09 | 1.16e+11 | 1.57e+11 | 2.11e+11 |
| | #active | **7,122** | **7,115** | **7,101** | **7,122** | **7,095** | **7,137** |
| LBFGS-B | Time(sec.) | **2.01** | **4.78** | 13.05 | 16.13 | 19.81 | 26.18 |
| | $f(\boldsymbol{x})$ | 1.93e+09 | 7.06e+09 | 8.87e+09 | 1.16e+11 | 1.57e+11 | 2.11e+11 |
| | $\#f$ | 100 | 138 | 276 | 258 | 211 | 238 |
| | $\#\nabla f$ | 100 | 138 | 276 | 258 | 211 | 238 |
| | #active | 6,850 | 361 | 4,368 | 9 | 2,452 | 2,482 |
| | $\|\nabla f_+\|_\infty$ | 3.3e-03 | 3.04e-02 | 3.25e-02 | 1.72e-01 | 2.28e-01 | 2.18e-01 |
| SPG | Time(sec.) | 3.19 | 45.70 | 71.73 | 592.64 | 590.10 | 523.18 |
| | $f(\boldsymbol{x})$ | 1.93e+09 | 7.06e+09 | 8.87e+09 | 1.16e+11 | 1.57e+11 | 2.11e+11 |
| | $\#f$ | 142 | 1165 | 1259 | 10,000 | 10,000 | 10,000 |
| | $\#\nabla f$ | 141 | 1,087 | 1,021 | 5,858 | 1,262 | 1,337 |
| | #active | 6,975 | 6,612 | 7,070 | 7,121 | 3,476 | 5,303 |
| | $\|\nabla f_+\|_\infty$ | 5.81e-06 | 9.91e-06 | 6.02e-06 | 7.31e-05 | 5.02e-04 | 4.42e-04 |
| SBB | Time(sec.) | 2.02 | 5.01 | **7.07** | **14.36** | **18.01** | **22.88** |
| | $f(\boldsymbol{x})$ | 1.93e+09 | 7.06e+09 | 8.87e+09 | 1.16e+11 | 1.57e+11 | 2.11e+11 |
| | $\#f$ | **0** | **1** | **0** | **1** | **1** | **1** |
| | $\#\nabla f$ | **76** | **102** | **98** | **151** | **153** | **163** |
| | #active | **7,122** | 7,114 | 7,096 | 7,121 | 7,092 | 7,136 |
| | $\|\nabla f_+\|_\infty$ | 3.31e-06 | 4.24e-06 | 9.63e-06 | 6.62e-06 | 8.40e-06 | 5.66e-06 |

Table 4: NNLS experiments on dataset P2. We used $\|\nabla f_+\|_\infty \leq 10^{-5}$ and $\#f \leq 10^4$ as the stopping criteria. For LBFGS-B and SPG, the troubles that appeared on dataset P1 persisted.

| Method | | ash958 | well1850 | real-sim | mnist | news20 | webspam |
|---|---|---|---|---|---|---|---|
| FNNLS | Time(sec.) | 0.0365 | 0.134 | 3.00e+04 | **15.3** | - | - |
| | $f(\boldsymbol{x})$ | **5.70e-30** | **3.80e-30** | 1.10e+03 | 1.58e+03 | - | - |
| LBFGS-B | Time(sec.) | 2.14 | 9.45 | 44.7 | 1.11e+03 | 559 | 343 |
| | $f(\boldsymbol{x})$ | 3.18e-17 | 8.87e-15 | 1.10e+03 | 1.58e+03 | 39.09 | 1.44e+04 |
| | $\#f$ | 32 | 184 | 538 | 8,562 | 935 | 763 |
| | $\#\nabla f$ | **32** | 184 | 538 | **8,562** | 935 | 763 |
| | $\|\nabla f_+\|_\infty$ | 9.26e-09 | 8.82e-09 | 4.88e-02 | 1.55e-01 | 0.69e+00 | 4.62e-02 |
| SPG | Time(sec.) | 0.087 | 0.115 | 45.4 | 3.20e+04 | 1.04e+04 | 1.34e+03 |
| | $f(\boldsymbol{x})$ | 4.01e-17 | 1.18e-14 | 1.10e+03 | 1.58e+03 | **2.04e+01** | 1.44e+04 |
| | $\#f$ | 38 | 283 | 797 | 196,386 | 118,036 | 2,641 |
| | $\#\nabla f$ | 38 | 248 | 582 | 116,004 | 2,693 | 1,869 |
| | $\|\nabla f_+\|_\infty$ | 8.43e-09 | 9.58e-09 | 4.90e-02 | 4.99e-02 | 4.52e-02 | 4.16e-02 |
| SBB | Time(sec.) | **0.010** | **0.0844** | **29.0** | 2.03e+03 | **104** | **209** |
| | $f(\boldsymbol{x})$ | 2.28e-17 | 2.38e-15 | 1.10e+03 | 1.58e+03 | 3.21e+01 | 1.44e+04 |
| | $\#f$ | **0** | **1** | **2** | **124** | **3** | **2** |
| | $\#\nabla f$ | 37 | **153** | **238** | 12,453 | **327** | **291** |
| | $\|\nabla f_+\|_\infty$ | 6.71e-09 | 6.53e-09 | 5.00e-02 | 5.00e-02 | 4.09e-02 | 4.54e-02 |

Table 5: NNLS experiments on real-world datasets. For LBFGS-B, SPG, and SBB, we used $\|\nabla f_+\|_\infty$ as the stopping criterion. We remark that FNNLS does not provide a comparable stopping criterion hence we test it under its default setting. For the datasets *ash958* and *well1850*, we set $\|\nabla f_+\|_\infty \leq 10^{-8}$ since we have the true solutions where $\nabla f_+$ vanishes. For the remaining four large-scale problems, we relax the condition to $\|\nabla f_+\|_\infty \leq 5 \times 10^{-2}$ to obtain solutions of medium accuracy. We mainly report the elapsed running time and the objective function value from each method. For LBFGS-B, SPG, and SBB we additionally report the number of objective function computations ($\#f$), the number of gradient evaluations ($\#\nabla f$), and the approximate optimality condition ($\|\nabla f_+\|_\infty$) at the final iteration. FNNLS does not scale for 'news20' and 'webspam' in this experiment due to its high memory consumption, while LBFGS-B terminates with its own linesearch condition on 'mnist' and 'news20'.

Our last set of results (Table 5) is on real-world datasets. FNNLS as before works well for small matrices, but rapidly becomes impractical for larger ones. It seems that the problem of using $\boldsymbol{A}^T\boldsymbol{A}$ appears even

more prominently for real-world problems where highly sparse matrices $\boldsymbol{A}$ with $m \leq n$ are not uncommon. For example, in comparison to other methods, FNNLS is highly penalized for 'real-sim' of $72,309 \times 20,958$, but has significant advantage for 'mnist' of $60,000 \times 780$, though eventually failing to scale for 'news20' and 'webspam' where $m \ll n$. From the table we observe once again that LBFGS-B, SPG and SBB are the competitors for the large and very-large problems, i.e., news20 and webspam. We see that SBB delivers competitive performance on real-world examples, along with accurate (and often better) solutions compared to its competitors in terms of $\|\nabla f_+\|_\infty$.

# 5 Conclusion and Discussion

In this paper we have presented a new non-monotonic algorithm for solving the nonnegative least squares (NNLS) problem. Our algorithm is based on the unconstrained Barzilai-Borwein method [1], whose simplicity it manages to retain. Moreover, despite retaining simplicity, we also showed that our method converges without a potentially expensive dependence on linesearch. We reported numerical results of our method applied to synthetic and real-world datasets, showing that our MATLAB implementation [5] performs competitively across all the range of problems both in terms of running time and accuracy—often outperforming other established algorithms such as LBFGS-B and SPG, especially for very large-scale problems.

# Acknowledgments

DK, SS and ID acknowledge support of NSF grants CCF-0431257 and CCF-0728879.

# References

[1] J. Barzilai and J. M. Borwein. Two-Point Step Size Gradient Methods. *IMA Journal of Numerical Analysis*, 8(1):141–148, 1988.

[2] A. Ben-Tal and A. Nemirovski. Non-euclidean restricted memory level method for large-scale convex optimization. *Math. Prog. A*, 102:407–456, 2005.

[3] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[4] M. Bierlaire, Ph. L. Toint, and D. Tuyttens. On Iterative Algorithms for Linear Least Squares Problems with Bound Constraints. *Linear Algebra and its Applications*, 143:111–143, 1991.

[5] E. G. Birgin, J. M. Martínez, and M. Raydan. Nonmonotone Spectral Projected Gradient Methods on Convex Sets. *SIAM Journal on Optimization*, 10(4):1196–1211, 2000.

[6] E. G. Birgin, J. M. Martínez, and M. Raydan. Algorithm 813: SPG - Software for Convex-constrained Optimization. *ACM Transactions on Mathematical Software*, 27:340–349, 2001.

[7] E. G. Birgin, J. M. Martínez, and M. Raydan. Large-Scale Active-Set Box-constrained Optimization Method with Spectral Projected Gradients. *Computational Optimization and Applications*, 23:101–125, 2002.

[8] E. G. Birgin, J. M. Martínez, and M. Raydan. Inexact Spectral Projected Gradient Methods on Convex Sets. *IMA Journal of Numerical Analysis*, 23:539–559, 2003.

[9] Åke Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.

[10] Rasmus Bro and Sijmen De Jong. A Fast Non-negativity-constrained Least Squares Algorithm. *Journal of Chemometrics*, 11(5):393–401, 1997.

[5] Our software can be downloaded from *http://www.cs.utexas.edu/users/dmkim/software/proj_toolbox.tar.bz2*

[11] James V. Burke and Jorge J. Moré. On the Identification of Active Constraints. *SIAM Journal on Numerical Analysis*, 25(5):1197–1211, 1988.

[12] R. Byrd, P. Lu, J. Nocedal, and C. Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[13] W. La Cruz and M. Raydan. Nonmonotone Spectral Methods for Large-Scale Nonlinear Systems. *Optimization Methods and Software*, 18:583–599, 2003.

[14] Y. H. Dai and R. Fletcher. Projected Barzilai-Borwein Methods for Large-scale Box-constrained Quadratic Programming. *Numerische Mathematik*, 100(1):21–47, 2005.

[15] A. Friedlander, J. M. Martínez, and M. Raydan. A New Method for Large-scale Box Constrained Convex Quadratic Minimization Problems. *Optimization Methods and Software*, 5:55–74, 1995.

[16] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[17] L. Grippo and M. Sciandrone. Nonmonotone Globalization Techniques for the Barzilai-Borwein Gradient Method. *Computational Optimization and Applications*, 23:143–169, 2002.

[18] W. W. Hager and H. Zhang. A New Active Set Algorithm for Box Constrained Optimization. *SIAM J. on Optimization*, 17(2):526–557, 2006.

[19] M. Hirsch, S. Sra, , B. Schölkopf, and S. Harmeling. Efficient filter flow for space-variant multiframe blind deconvolution. In *CVPR*, Jun. 2010.

[20] D. Kim, S. Sra, and I. S. Dhillon. Fast Newton-type methods for the least squares nonnegative matrix approximation problem. In *SIAM DM*, 2007.

[21] C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. Prentice–Hall, 1974.

[22] C.-J. Lin and J. J. Moré. Newton's Method for Large Bound-Constrained Optimization Problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[23] D. G. Luenberger and Y. Ye. *Linear and Nonlinear Programming*. Springer, third edition, 2008.

[24] J. J. Moré and G. Toraldo. On The Solution of Large Quadratic Programming Problems with Bound Constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[25] J. Nagy and Z. Strakos. Enforcing nonnegativity in image reconstruction algorithms. *Mathematical Modeling, Estimation, and Imaging*, 4121:182–190, 2000.

[26] A. Nemirovski, A. Juditsky, G. Land, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM J. Optim.*, 19(4):1574–1609, 2009.

[27] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[28] M. Raydan. On the Barzilai and Borwein Choice of the Steplength for the Gradient Method. *IMA Journal on Numerical Analysis*, 13:321–326, 1993.

[29] A. J. Reader and H. Zaidi. Advances in PET Image Reconstruction. *PET Clinics*, 2(2):173–190, 2007.

[30] J. B. Rosen. The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8(1):181–217, 1960.

[31] K. Schittkowski. The Numerical Solution of Constrained Linear Least-Squares Problems. *IMA Journal of Numerical Analysis*, 3:11–36, 1983.

# A Ordinary Algorithm (OA) Counterexample

We illustrate in Figure 6 that plugging in (2) naively into GP does not work. Our illustration is inspired by the instructive counterexample of Dai and Fletcher [14].

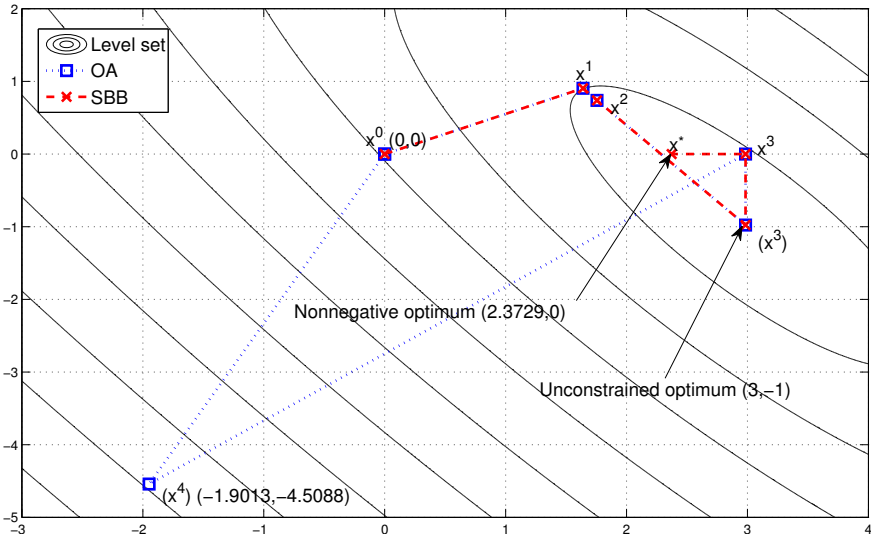


Figure 6: Comparison between OA and SBB. The figure shows a 2D counterexample where OA fails to converge. Given $\boldsymbol{A} = \begin{bmatrix} 0.8147 & 0.1270 \\ 0.9058 & 0.9134 \end{bmatrix}$ and $\boldsymbol{b} = \begin{bmatrix} 2.3172 \\ 1.8040 \end{bmatrix}$, we start both OA and SBB at the same initial point $\boldsymbol{x}^0 = [0\ 0]^T$. At iteration $k$, each method first computes an intermediate point $(\boldsymbol{x}^k)$; this point is then projected onto $\mathbb{R}^2_+$ to obtain a feasible iterate $\boldsymbol{x}^k$. Both methods generate the same iterate sequence $\boldsymbol{x}^0, \boldsymbol{x}^1, \boldsymbol{x}^2, (\boldsymbol{x}^3), \boldsymbol{x}^3$ for the first 3 iterations. OA starts behaving differently at the $4^{th}$ iteration where it converges to the optimal solution $\boldsymbol{x}^*$. OA, in contrast, generates $(\boldsymbol{x}^4)$, which upon subsequent projection brings it back to the initial point $\boldsymbol{x}^0$, leading OA to cycle indefinitely without converging.

| Iterate | $\boldsymbol{x}^0$ | $\boldsymbol{x}^1$ | $\boldsymbol{x}^2$ | $(\boldsymbol{x}^3)$ |
|---|---|---|---|---|
| Fixed set | {} | {} | {} | {} |
| OA | $[0, 0]$ | $[1.7934, 0.6893]$ | $[1.8971, 0.5405]$ | $[2.9779, -0.9683]$ |
| SBB | $[0, 0]$ | $[1.7934, 0.6893]$ | $[1.8971, 0.5405]$ | $[2.9779, -0.9683]$ |
| **Iterate** | $\boldsymbol{x}^3$ | $(\boldsymbol{x}^4)$ | $\boldsymbol{x}^4$ | |
| Fixed set | {2} | {2} | | |
| OA | $[2.9779, 0]$ | $[-1.9013, -4.5088]$ | $[0, 0] = \boldsymbol{x}^0$ | |
| SBB | $[2.9779, 0]$ | $[2.3729, 0] = \boldsymbol{x}^*$ | | |

Table 6: Coordinates of the iterates in Figure 6.

# B The various BB based algorithms

**Original BB steps**

$$\gamma^k = \frac{\|\nabla f(\boldsymbol{x}^{k-1})\|^2}{\langle \nabla f(\boldsymbol{x}^{k-1}), \boldsymbol{A}^T\boldsymbol{A}\nabla f(\boldsymbol{x}^{k-1})\rangle}, \quad \text{and} \quad \gamma^k = \frac{\langle \nabla f(\boldsymbol{x}^{k-1}), \boldsymbol{A}^T\boldsymbol{A}\nabla f(\boldsymbol{x})^{k-1}\rangle}{\|\boldsymbol{A}^T\boldsymbol{A}\nabla f(\boldsymbol{x}^{k-1})\|^2}. \tag{19}$$

**Subspace BB steps**

$$\alpha^k = \frac{\|\nabla \tilde{f}^{k-1}\|^2}{\langle \nabla \tilde{f}^{k-1}, \boldsymbol{A}^T\boldsymbol{A}\nabla \tilde{f}^{k-1}\rangle}, \quad \text{and} \quad \alpha^k = \frac{\langle \nabla \tilde{f}^{k-1}, \boldsymbol{A}^T\boldsymbol{A}\nabla \tilde{f}^{k-1}\rangle}{\|\boldsymbol{A}^T\boldsymbol{A}\nabla \tilde{f}^{k-1}\|^2}. \tag{20}$$

where $\nabla \tilde{f}^{k-1}$ is defined as $\nabla_i \tilde{f}^{k-1} = \partial_i \nabla f(\boldsymbol{x}^{k-1})$ for $i \notin \mathcal{B}(\boldsymbol{x}^k)$, $\nabla_i \tilde{f}^{k-1} = 0$

## B.1 Pseudocode for the variants

**Algorithm:** OA
Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until stopping criteria* (12) *met* **do**
  Compute $\gamma^i$ using (19);
  $\boldsymbol{x}^{i+1} \leftarrow [\boldsymbol{x}^i - \gamma^i \nabla f(\boldsymbol{x}^i)]_+$

**Algorithm 2:** Basic Algorithm (OA).

**Algorithm:** OA+DS
Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until stopping criteria* (12) *met* **do**
  Compute $\gamma^i$ using (19)
  $\boldsymbol{x}^{i+1} \leftarrow [\boldsymbol{x}^i - \beta^i \cdot \gamma^i \nabla f(\boldsymbol{x}^i)]_+$;
  $\beta^{i+1} \leftarrow \eta\beta^i$, where $\eta \in (0, 1)$;

**Algorithm 3:** OA + diminishing scalar (OA+DS).

**Algorithm:** OA+LS
Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until stopping criteria* (12) *met* **do**
  $\hat{\boldsymbol{x}}^0 \leftarrow \boldsymbol{x}^{i-1}$ and $\hat{\boldsymbol{x}}^1 \leftarrow \boldsymbol{x}^i$;
  **for** $j = 1, \cdots, M$ **do**
    Compute $\gamma^j$ using (19)
    $\hat{\boldsymbol{x}}^{j+1} \leftarrow [\hat{\boldsymbol{x}}^j - \gamma^j \nabla f(\hat{\boldsymbol{x}}^j)]_+$
  **repeat** /* linesearch */
    Compute $\boldsymbol{x}^{i+1} \leftarrow [\hat{\boldsymbol{x}}^M - \tau \nabla f(\hat{\boldsymbol{x}}^M)]_+$;
    Update $\tau$;
  **until** $\boldsymbol{x}^{i+1}$ *and* $\hat{\boldsymbol{x}}^M$ *satisfy* (10);

**Algorithm 4:** OA + linesearch (OA+LS).

Variants of the Ordinary Algorithm: OA.

---

**Algorithm:** SA
Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until stopping criteria* (12) *met* **do**
  Compute $\alpha^i$ using (20);
  $\boldsymbol{x}^{i+1} \leftarrow [\boldsymbol{x}^i - \alpha^i \nabla f(\boldsymbol{x}^i)]_+$

**Algorithm 5:** Subspace-BB only (SA+DS; $\beta^k = 1$).

**Algorithm:** SA+DS
Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until stopping criteria* (12) *met* **do**
  Compute $\alpha^i$ using (20)
  $\boldsymbol{x}^{i+1} \leftarrow [\boldsymbol{x}^i - \beta^i \cdot \alpha^i \nabla f(\boldsymbol{x}^i)]_+$;
  $\beta^{i+1} \leftarrow \eta\beta^i$, where $\eta \in (0, 1)$;

**Algorithm 6:** SA + diminishing scalar (SA+DS).

**Algorithm:** SA+LS
Given $\boldsymbol{x}^0$ and $\boldsymbol{x}^1$;
**for** $i = 1, \cdots$ *until stopping criteria* (12) *met* **do**
  $\hat{\boldsymbol{x}}^0 \leftarrow \boldsymbol{x}^{i-1}$ and $\hat{\boldsymbol{x}}^1 \leftarrow \boldsymbol{x}^i$;
  **for** $j = 1, \cdots, M$ **do**
    Compute $\alpha^j$ using (20)
    $\hat{\boldsymbol{x}}^{j+1} \leftarrow [\hat{\boldsymbol{x}}^j - \alpha^j \nabla f(\hat{\boldsymbol{x}}^j)]_+$
  **repeat** /* linesearch */
    Compute $\boldsymbol{x}^{i+1} \leftarrow [\hat{\boldsymbol{x}}^M - \tau \nabla f(\hat{\boldsymbol{x}}^M)]_+$;
    Update $\tau$;
  **until** $\boldsymbol{x}^{i+1}$ *and* $\hat{\boldsymbol{x}}^M$ *satisfy* (10);

**Algorithm 7:** SA + linesearch (SA+LS).

Subspace-BB step based algorithms.