A New Projected Quasi-Newton Approach for the Nonnegative Least Squares Problem

Dongmin Kim, Suvrit Sra, and Inderjit S. Dhillon

Department of Computer Sciences
The University of Texas at Austin

Technical Report # TR-06-54

Abstract

Constrained least squares estimation lies at the heart of many applications in fields as diverse as statistics, psychometrics, signal processing, or even machine learning. Nonnegativity requirements on the model variables are amongst the simplest constraints that arise naturally, and the corresponding least-squares problem is called Nonnegative Least Squares or NNLS. In this paper we present a new, efficient, and scalable Quasi-Newton-type method for solving the NNLS problem, improving on several previous approaches and leading to a superlinearly convergent method. We show experimental results comparing our method to well-known methods for solving the NNLS problem. Our method significantly outperforms other methods, especially as the problem size becomes larger.

1 Introduction

A fundamental problem in data modeling is the estimation of a parameterized model for describing the data. For example, imagine that several experimental observations that are linear functions of the underlying parameters, have been made. Given a sufficiently large number of such observations, one can reliably estimate the true underlying parameters. Let the unknown model parameters be denoted by the vector $\boldsymbol{x} = [x_1, \ldots, x_n]$, the different experiments relating $\boldsymbol{x}$ be encoded by the measurement matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and the set of observed values be given by $\boldsymbol{b}$. The aim is to reconstruct a vector $\boldsymbol{x}$ that explains the observed values as well as possible. This requirement may be fulfilled by solving the linear system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},$$

where the system may be either under-determined ($m < n$) or over-determined ($m \geq n$). In the latter case, the technique of least-squares proposes to compute $\boldsymbol{x}$ so that the reconstruction error

$$f(\boldsymbol{x}) = \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \tag{1.1}$$

is minimized, where $\|\cdot\|$ denotes the L_2 norm. However, the estimation is not always that straightforward because in many real-world problems, the underlying parameters represent quantities that can take on only nonnegative values, e.g., amounts of rainfall, chemical concentrations, color intensities, to name a few. In such a case, Problem (1.1) must be modified to include nonnegativity constraints on the model parameters $\boldsymbol{x}$. The resulting problem is called Nonnegative Least Squares (NNLS), and is formulated as

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) = \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \\ \text{subject to} \quad & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned} \tag{1.2}$$

The NNLS problem (1.2) is fairly old and the algorithm of Lawson and Hanson [11] seems to be apparently the first method for solving it (this algorithm is available as the `lsqnonneg` procedure in

MATLAB). In this paper we present a new approach based on a projected Quasi-Newton procedure that outperforms not only `lsqnonneg` but also several other existing approaches. Other obvious variations of NNLS have also been studied, for example, (a) Bounded Least Squares where $\boldsymbol{x}$ is bounded ($\boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}$), (b) Regularized Nonnegative Least Squares that includes a regularization term in the objective function (beneficial for ill-conditioned problems). Our approach generalizes easily to handle both these variants.

1.1 Previous approaches

Over the years, a variety of methods have been applied to tackle the NNLS problem. Many of these approaches are summarized in [6, 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 [6, 15]. However, such transformations prove to be costly for NNLS, and do not seem to yield much advantage, without additional engineering efforts.

The second category of methods includes *active-set methods* [10] that do not involve transforming (1.2) into a corresponding minimum distance problem. These methods are based on the observation that only small subset of constraints are usually active (i.e., satisfied exactly) at the solution. Thus, the overall optimization problem is approximated by solving a series of equality-constrained problems, where the equality constraints form a working set corresponding to a predicted active set. The NNLS algorithm of Lawson and Hanson [11] is an active set method, and was the *de facto* method for solving (1.2) for many years. Recently Bro and Jong [7] modified it and developed a method called Fast-NNLS (FNNLS), which often speeds up the basic algorithm, especially in the presence of multiple right-hand sides by avoiding unnecessary re-computations. However, FNNLS requires the construction of $\boldsymbol{A}^T\boldsymbol{A}$, which can make it prohibitive for large-scale problems. A recent variant of FNNLS appropriately rearranges calculations to achieve further speedups in the presence of multiple right hand sides [16]. However, all of these approaches still depend on $\boldsymbol{A}^T\boldsymbol{A}$, rendering them infeasible for large problems.

The third category of methods includes algorithms based on iterative approaches, which essentially produce a sequence of intermediate solutions that converge to the solution of (1.2). For example, the gradient projection method [14] and some variants have been applied to NNLS [5, 12]. The main advantage of this class of algorithms is that by using information from the projected gradient step along with a good guess of the active set, one can handle multiple active constraints per iteration. In contrast, the active-set method typically deals with only one active constraint at each iteration. Some of the iterative methods are based on the solution of the corresponding linear complementarity problem (LCP)

$$\boldsymbol{\lambda} = \nabla f(\boldsymbol{x}) = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^T\boldsymbol{b} \geq 0, \quad \boldsymbol{x} \geq 0, \quad \boldsymbol{\lambda}^T\boldsymbol{x} = 0, \tag{1.3}$$

which is essentially the set of KKT optimality conditions for a quadratic programming reformulation of (1.2). See §7.7 of [6] for further information.

Other approaches, that do not directly fit any of the above three categories include approaches such as an interior point method [13]. However, for large-scale problems, such methods are usually prohibitive. In addition, some custom designed algorithms such as Principal Block Pivoting [13] have also been developed, though once again with limited success [6].

1.2 Contributions

Our first major contribution in this paper is the derivation of a new algorithm (Section 2) that combines the strengths of gradient projection with a non-diagonal gradient scaling scheme. In contrast to an active set approach, gradient projection enables our method to incorporate multiple active constraints at each iteration. By employing non-diagonal gradient scaling, our method overcomes some of the deficiencies of a projected gradient method such as slow convergence and zigzagging. An important characteristic of our algorithm is that despite the efficiencies that we introduce, it still remains relatively simple in comparison with other optimization-oriented algorithms. The second important contribution comes by way of the implementation efficiencies that we provide (Section 2.2). Despite the long history of the problem and

a diverse pool of algorithms, there is surprisingly little publicly available software for NNLS. One may try to use off-the-shelf optimization packages, but due to their algorithmic complexity they tend to be computationally inefficient for NNLS. In this paper we provide a customized implementation for NNLS that outperforms all methods known to us, while still retaining competitive accuracy. Our algorithm takes advantage of the sparsity of the input data, especially by avoiding explicit computation of $\boldsymbol{A}^T\boldsymbol{A}$, and the corresponding speedups are reported in our experiments in Section 3.

1.3 Summary

The remainder of this paper is organized as follows. Section 2 describes our method for NNLS. Section 2.2 provides a detailed guide to the major implementation issues. In Section 2.3 we provide a proof of convergence for our algorithm from a feasible direction method viewpoint uncovering connections between our approach and previous methods. In Section 3, we compare our method to other major NNLS approaches by showing experimental results across several data sets (both synthetic as well as from real-life applications). Finally in Section 4, we provide our conclusions and discuss future work.

Notation

For easy reference, we summarize our notation here. Bold face lower case letters are used to denote vectors, e.g., $\boldsymbol{x}, \boldsymbol{y}$, while bold face upper case letters denote matrices, e.g., $\boldsymbol{A}$. The single letter $\mathcal{P}$ is used to denote an orthogonal projection onto an appropriate set. Ordinary lower case letters such as s or α denote scalar quantities. $\|\cdot\|$ denotes the Euclidean L_2 norm. We reserve k to denote the iteration count, while i and j represent indices of entries of vectors or matrices.

2 Algorithms and Theory

In this section we derive our algorithm for the NNLS problem, which seeks to

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & f(\boldsymbol{x}) = \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2, \\ \text{subject to} \quad & \boldsymbol{x} \geq \boldsymbol{0}. \end{aligned} \tag{2.1}$$

Broadly speaking, our algorithm generalizes a projected gradient approach [5] to achieve its simplicity and efficiency. The gradient information helps to identify the set of active variables more efficiently, which impacts performance positively—if the constraints active at the final solution are known in advance, then the original problem can be solved by simply optimizing the objective function in an unconstrained manner over only the variables that correspond to the inactive constraints. However, despite this capability to identify the active set, a projected gradient approach is not enough for an efficient and accurate solution, since just like the steepest descent method, it also suffers from deficiencies such as slow convergence and zigzagging. Fortunately, at least for *unconstrained* optimization problems, it is known that the use of non-diagonal positive-definite gradient scaling matrices alleviates such problems.

For solving the NNLS Problem (2.1), we extend the idea of non-diagonal gradient scaling to *constrained* minimization problems. It turns out that due to the simplicity of the nonnegativity constraints, this scaling approach remains feasible. We describe our approach is greater detail below.

2.1 Overview of our method

We derive an iterative algorithm for solving (2.1). At each iteration, our algorithm partitions the optimization variables (entries of the vector $\boldsymbol{x}$) into two groups, namely, the *free* and the *fixed* variables. Then, we consider the problem as an equality-constrained one, wherein the *fixed* variables must remain active even for the next iteration. This particular equality-constrained problem is solved by simply optimizing over the *free* variables alone.

The *fixed* set of variables corresponds to those indexed by the set

$$I^k = \Big\{ i \big| x_i^k = 0,\, [\nabla f(\boldsymbol{x}^k)]_i > 0 \Big\}. \tag{2.2}$$

Note that the index set I^k in (2.2) is a subset of the entire active set due to the additional positivity requirement on the gradient. More specifically, I^k contains variables that satisfy the KKT conditions (1.3) at the current iteration, which makes them more likely to be active at the final solution. In contrast, for a component $x_j^k = 0$, for which $[\nabla f(\boldsymbol{x}^k)]_j \leq 0$, it may be possible to optimize the objective further by making $x_j^{k+1} > 0$ and $[\nabla f(\boldsymbol{x}^{k+1})]_j = 0$ at the next iteration, thereby violating the notion of a *fixed* variable.

Let us denote *free* variables by $\boldsymbol{y}$ and fixed variables by $\boldsymbol{z}$. Without loss of generality, we can assume that $\boldsymbol{x}^k$ and $\nabla f(\boldsymbol{x}^k)$ are partitioned as

$$\boldsymbol{x}^k = \begin{bmatrix} \boldsymbol{y}^k \\ \boldsymbol{z}^k \end{bmatrix}, \quad \nabla f(\boldsymbol{x}^k) = \begin{bmatrix} \nabla f(\boldsymbol{y}^k) \\ \nabla f(\boldsymbol{z}^k) \end{bmatrix}, \quad \text{where } y_i^k \notin I^k \text{and } z_i^k \in I^k.$$

Then, the equality-constrained (over the fixed variables) version of (2.1) at iteration k can be written as

$$\begin{aligned} \underset{\boldsymbol{y}}{\text{minimize}} \quad & \tfrac{1}{2} \big\| \boldsymbol{A}[\boldsymbol{y}; \boldsymbol{z}^k] - \boldsymbol{b} \big\|^2, \\ \text{subject to} \quad & \boldsymbol{y} \geq \boldsymbol{0}, \quad \boldsymbol{z}^k = \boldsymbol{0}, \end{aligned} \tag{2.3}$$

or equivalently, as a problem over the *free* variables,

$$\begin{aligned} \underset{\boldsymbol{y}}{\text{minimize}} \quad & g^k(\boldsymbol{y}) = \tfrac{1}{2} \|\bar{\boldsymbol{A}}\boldsymbol{y} - \boldsymbol{b}\|^2, \\ \text{subject to} \quad & \boldsymbol{y} \geq \boldsymbol{0}, \end{aligned} \tag{2.4}$$

where $\bar{\boldsymbol{A}}$ consists of the appropriate parts of $\boldsymbol{A}$ corresponding to the *free* variables.

Let $\tilde{\boldsymbol{y}}$ denote the solution[1] to (2.4). Then, we obtain the update

$$\boldsymbol{x}^{k+1} = \begin{bmatrix} \tilde{\boldsymbol{y}} \\ \boldsymbol{z}^{k+1} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{y}} \\ \boldsymbol{z}^{k} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{y}} \\ \boldsymbol{0} \end{bmatrix}, \tag{2.5}$$

for $\boldsymbol{x}$, wherein the latter two equalities used the fact that $\boldsymbol{z}^{k+1}$ is fixed (and thus, equal to zero).

Now we show how to compute $\tilde{\boldsymbol{y}}$, noting that the key difference between our approach and the projected gradient method lies in this computation. We make use of a non-diagonal gradient scaling matrix $\boldsymbol{S}^k$, which approximates the Hessian $\nabla^2 f = \boldsymbol{A}^T\boldsymbol{A}$ at each iteration. The use of such non-diagonal gradient scaling matrices accelerates the rate of convergence, and it is central to the success of the popular Quasi-Newton method. We remark that the "non-diagonality" helps by pointing out that the ordinary projected gradient method may be viewed as using $\boldsymbol{S}^k = \boldsymbol{I}$ at each iteration.

We compute the vector $\tilde{\boldsymbol{y}}$ by using the following update

$$\tilde{\boldsymbol{y}} = \boldsymbol{y}^k + \alpha\big(\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k\big), \tag{2.6}$$

where $\alpha, \beta \geq 0$ are certain parameters, and the function $\gamma(\beta; \boldsymbol{y})$ is defined as (for $\beta \geq 0$) the projection

$$\gamma^k(\beta; \boldsymbol{y}^k) = \mathcal{P}\big(\boldsymbol{y}^k - \beta \bar{\boldsymbol{S}}^k \nabla f(\boldsymbol{y}^k)\big), \tag{2.7}$$

where $\bar{\boldsymbol{S}}^k$ is an appropriate principal submatrix of $\boldsymbol{S}^k$, and $\mathcal{P}(\cdot)$ denotes the orthogonal projection onto $\mathbb{R}^n_+$. Update (2.7) is similar to the Generalized Cauchy Point (GCP) from the projected gradient method. Note that in (2.7) any positive definite matrix $\boldsymbol{S}^k$ is permitted, whereby one can recover the basic projected gradient method of Bierlaire et al. [5] by setting $\boldsymbol{S}^k = \boldsymbol{I}$ and $\alpha = 1$.

The overall method that encapsulates all the details mentioned above is presented as Algorithm 1, which we name `PQN-NNLS` (for Projected Quasi-Newton NNLS).

In Algorithm 1 any positive definite matrix may be used to initialize $\boldsymbol{S}^0$. Further, any feasible point is permitted as an initialization for $\boldsymbol{x}^0$. However, note that the actual implementation details of Steps 3.1 and 4 are left unspecified. We describe implementation details for these two important sub-steps in Section 2.2 below.

[1] Problem (1.2) will have a unique solution if $g^k(\boldsymbol{y})$ is strictly-convex, i.e., $\bar{A}$ is of full-rank. Otherwise it can have multiple global minima. Even then, we can just use any of the global minima.

Algorithm 1 PQN-NNLS

Input: $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \quad \boldsymbol{b} \in \mathbb{R}^m$
Output: $\boldsymbol{x}^* \geq 0$ such that $\boldsymbol{x}^* = \operatorname{argmin} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2$.
{Initialization}
$\boldsymbol{x}^0 \in \mathbb{R}^n_+$, $\boldsymbol{S}^0 \leftarrow \boldsymbol{I}$, and $k \leftarrow 0$
repeat

1. Compute fixed variable set $I^k = \left\{ i \middle| x_i^k = 0,\ [\nabla f(\boldsymbol{x}^k)]_i > 0 \right\}$.
2. Partition $\boldsymbol{x}^k = [\boldsymbol{y}^k;\ \boldsymbol{z}^k]$, where $y_i^k \notin I^k$ and $z_i^k \in I^k$.
3. Solve equality-constrained subproblem:
   3.1. Find appropriate values for α^k and β^k.
   3.2. $\gamma^k(\beta^k; \boldsymbol{y}^k) \leftarrow \mathcal{P}\big(\boldsymbol{y}^k - \beta^k \bar{\boldsymbol{S}}^k \nabla f(\boldsymbol{y}^k)\big)$.
   3.3. $\tilde{\boldsymbol{y}} \leftarrow \boldsymbol{y}^k + \alpha^k(\gamma^k(\beta^k; \boldsymbol{y}^k) - \boldsymbol{y}^k)$.
4. Update gradient scaling matrix $\boldsymbol{S}^k$ to obtain $\boldsymbol{S}^{k+1}$
5. Update $\boldsymbol{x}^{k+1} \leftarrow [\tilde{\boldsymbol{y}}; \boldsymbol{z}^k]$.
6. $k \leftarrow k + 1$

until Stopping criteria are met.

2.2 Implementation details

In this section we describe particular methods for performing line-search (Step 3.1) and gradient scaling (Step 4) for our `PQN-NNLS` algorithm.

2.2.1 Line Search

First we look at how the line-search step (Step 3.1) of Algorithm 1 can be implemented. Naturally, the parameters β and α are crucial to the update (2.6), and subsequently to the convergence speed of the overall algorithm.

One of the most popular methods is the limited minimization rule, which computes an appropriate α for a pre-specified (fixed) value of β by performing the following single-variable minimization

$$\alpha^k \ = \ \underset{\alpha \in [0,1]}{\operatorname{argmin}} \quad g^k\big(\boldsymbol{y}^k + \alpha(\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k)\big). \tag{2.8}$$

In general, the above line search for constrained optimization (2.8) is difficult to carry out. However, for our case, since the nonnegativity constraints are particularly simple, we can derive an analytic solution. We exploit the fact that the objective function in (2.8) is a continuous and quadratic univariate function of α. Leting $\boldsymbol{d} = (\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k)$ and setting the derivative $\frac{dg^k}{d\alpha} = 0$, we have

$$\frac{dg^k}{d\alpha} = \boldsymbol{d}^T \nabla g^k \left(\boldsymbol{y}^k + \alpha \boldsymbol{d}\right) = \boldsymbol{d}^T \bar{\boldsymbol{A}}^T \bar{\boldsymbol{A}} \boldsymbol{y}^k + \alpha \|\bar{\boldsymbol{A}}\boldsymbol{d}\|^2 - \boldsymbol{d}^T \bar{\boldsymbol{A}}^T \boldsymbol{b} = 0,$$

$$\text{which yields} \quad \alpha = \frac{\boldsymbol{d}^T\big(\bar{\boldsymbol{A}}^T \boldsymbol{b} - \bar{\boldsymbol{A}}^T \bar{\boldsymbol{A}} \boldsymbol{y}^k\big)}{\|\bar{\boldsymbol{A}}\boldsymbol{d}\|^2}.$$

With the above α, we can choose the current step-size to be $\alpha^k = \operatorname{mid}(0, \alpha, 1)$.

Although the limited minimization rule above yields an analytic solution for a fixed value of β, sometimes the convergence speed of the overall algorithm can be sensitive to this pre-specified *inner* step-length β. Finding a good value of β could itself be time-consuming. Hence, we also investigate an alternative method to determine a proper step-size β at each iteration.

One choice can be the Armijo step-size rule [9], which is simple and usually works well both for unconstrained and constrained problems. Like the limited minimization rule, it can be applied to compute α with the pre-specified β. Bertsekas [2] proposed a variation of Armijo-like rule, called the Armijo along projection arc (APA) rule. Interestingly, unlike the above methods, APA fixes α in (2.6) and computes

an appropriate β for the step-size at each iteration. In our implementation of APA, we fix $\alpha = 1$ in (2.6) which leads the following changes to the update

$$\begin{aligned}\tilde{\boldsymbol{y}} &= \boldsymbol{y}^k + 1 \cdot \big(\gamma^k(\beta^k; \boldsymbol{y}^k) - \boldsymbol{y}^k\big), \quad \beta^k \geq 0, \\ &= \mathcal{P}\big(\boldsymbol{y}^k - \beta^k \boldsymbol{S}^k \nabla f(\boldsymbol{y}^k)\big). \end{aligned} \tag{2.9}$$

Here, the computation of β^k takes the following form. Let m be the smallest non-negative integer for which

$$g^k(\boldsymbol{y}^k) - g^k(\gamma^k(s^m\sigma; \boldsymbol{y}^k)) \geq \tau \nabla g^k(\boldsymbol{y}^k)^T \big(\boldsymbol{y}^k - \gamma^k(s^m\sigma; \boldsymbol{y}^k)\big), \tag{2.10}$$

where $\sigma > 0, s \in (0,1)$ and $\tau \in (0, \frac{1}{2})$ are fixed scalars. The resulting step-size $\beta = s^m\sigma$. We provide both limited minimization as well APA based step-size choices in our implementation. Furthermore, in Section 2.3 we prove that these two step-size choices are well-founded by proving that our algorithm converges to the optimum with either of these step-sizes.

2.2.2 Gradient Scaling

Now we look at how the gradient scaling step, namely Step 4 of Algorithm 1, can be implemented. In a traditional Newton method the inverse of the Hessian of the objective function serves as the gradient scaling matrix. However, a fundamental drawback of using the inverse Hessian for large-scale problems is cost of computing it. For the NNLS objective function, this amounts to $O(n^3)$ in general, which can make the overall procedure unacceptably expensive. A further drawback, especially for large-scale problems is that the Hessian or its inverse can be ill-conditioned. This not only causes numerical difficulties, but can also lead to a poor rate of convergence.

To circumvent some of the problems associated with the use of the full Hessian, researchers have used the idea of approximating it in various ways. Such an approximation leads to what is known as the Quasi-Newton or Variable Metric method—a popular and well-established method for non-linear programming. Some common techniques for iteratively approximating the Hessian are the Powell-Symmetric-Broyden (PSB), Davidon-Fletcher-Powell (DFP), and the Broyden-Fletcher-Goldfarb-Shanno (BFGS) updates, where the latter is believed to be the most effective in general [4, 10].

The BFGS Update. Suppose $\boldsymbol{H}^k$ is the current approximation to the Hessian. The BFGS update adds a rank-two correction to $\boldsymbol{H}^k$ to obtain

$$\boldsymbol{H}^{k+1} = \boldsymbol{H}^k - \frac{\boldsymbol{H}^k \boldsymbol{u}\boldsymbol{u}^T \boldsymbol{H}^k}{\boldsymbol{u}^T \boldsymbol{H}^k \boldsymbol{u}} + \frac{\boldsymbol{w}\boldsymbol{w}^T}{\boldsymbol{u}^T \boldsymbol{w}}, \tag{2.11}$$

where $\boldsymbol{w}$ and $\boldsymbol{u}$ are defined as

$$\boldsymbol{w} = \nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k), \quad \text{and} \quad \boldsymbol{u} = \boldsymbol{x}^{k+1} - \boldsymbol{x}^k.$$

Let $\boldsymbol{S}^k$ denote the inverse of $\boldsymbol{H}^k$. Then, by applying the Sherman-Morrison-Woodbury formula, update (2.11) amounts to the following,

$$\boldsymbol{S}^{k+1} = \boldsymbol{S}^k + \left(1 + \frac{\boldsymbol{w}^T \boldsymbol{S}^k \boldsymbol{w}}{\boldsymbol{u}^T \boldsymbol{w}}\right) \frac{\boldsymbol{u}\boldsymbol{u}^T}{\boldsymbol{u}^T \boldsymbol{w}} - \frac{(\boldsymbol{S}^k \boldsymbol{w}\boldsymbol{u}^T + \boldsymbol{u}\boldsymbol{w}^T \boldsymbol{S}^k)}{\boldsymbol{u}^T \boldsymbol{w}}. \tag{2.12}$$

Now we use the fact that

$$\boldsymbol{w} = \nabla f(\boldsymbol{x}^{k+1}) - \nabla f(\boldsymbol{x}^k) = \boldsymbol{A}^T \boldsymbol{A}(\boldsymbol{x}^{k+1} - \boldsymbol{x}^k) = \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{u},$$

to rewrite (2.12) as

$$\boldsymbol{S}^{k+1} \leftarrow \boldsymbol{S}^k + \left(1 + \frac{\boldsymbol{u}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{S}^k \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{u}}{\boldsymbol{u}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{u}}\right) \frac{\boldsymbol{u}\boldsymbol{u}^T}{\boldsymbol{u}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{u}} - \frac{(\boldsymbol{S}^k \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{u}\boldsymbol{u}^T + \boldsymbol{u}\boldsymbol{u}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{S}^k)}{\boldsymbol{u}^T \boldsymbol{A}^T \boldsymbol{A} \boldsymbol{u}}. \tag{2.13}$$

Remark: Observe that update (2.13) can be implemented without explicitly computing $\boldsymbol{A}^T\boldsymbol{A}$, because of the matrix-vector products such as $\boldsymbol{A}\boldsymbol{u}$. However, with progressing iterations the matrix $\boldsymbol{S}^k$ becomes denser, and must be stored.

2.3 Convergence

We now prove that when $\boldsymbol{A}$ is of full-rank, then Algorithm 1 converges to the global-minimum of (2.1). Throughout our proof, we assume that either the Armijo along projection arc (APA) or the limited-minimization rule (LM) are used for computing the step-size.

Before we can state our convergence theorem, we need to prove a few supporting lemmas. Our proof is structured as follows. First we show that the updates to $\boldsymbol{x}$ via APA or Limited Minimization (LM) ensure a monotonic descent in the objective function value (Lemma 1 for APA, and Lemma 2 for LM). Then, we show that the resulting sequence of iterates $\{\boldsymbol{x}^k\}$ has a limit-point (Lemma 3). Finally, Theorem 1 shows that any limit point of the sequence $\{\boldsymbol{x}^k\}$ is also a stationary or KKT point of (1.2), provided that a certain gradient related condition is satisfied (Lemma 4), thereby concluding the proof. In addition to convergence, we show optimality of the achieved iterate in Theorem 2, which shows that our algorithm stops if and only if the current iterate is an optimum.

Lemma 1 (Descent with APA)**.** *If $\boldsymbol{x}^k$ is not a stationary point of Problem* (2.1)*, then there exists some constant $\bar{\beta}$ such that*

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k), \quad \forall \beta \in (0, \bar{\beta}],$$

where $\boldsymbol{x}^{k+1}$ is given by (2.5), i.e., $\boldsymbol{x}^{k+1} = [\tilde{\boldsymbol{y}};\ \boldsymbol{0}]$ and $\tilde{\boldsymbol{y}}$ is given by (2.9)*.*

Proof. By the construction of I^k, all components of $\boldsymbol{y}^k$ satisfy:

$$\text{either} \qquad \boldsymbol{y}_i^k \neq 0 \qquad \text{or} \qquad [\nabla f(\boldsymbol{y}^k)]_i \leq 0.$$

Furthermore, since $\boldsymbol{x}^k$ is not a stationary point, there exists at least one entry in $\boldsymbol{y}^k$ such that

$$[\nabla f(\boldsymbol{y}^k)]_i \neq 0.$$

Thus letting $\boldsymbol{d} = -\bar{\boldsymbol{S}}^k \nabla f(\boldsymbol{y}^k)$, we see that

$$\nabla f(\boldsymbol{y}^k)^T \boldsymbol{d} = -\nabla f(\boldsymbol{y}^k)^T \bar{\boldsymbol{S}}^k \nabla f(\boldsymbol{y}) < 0,$$

since $\bar{\boldsymbol{S}}^{\boldsymbol{k}}$ is a principal submatrix of the positive definite matrix $\boldsymbol{S}^k$, and is therefore itself positive definite. This establishes the fact that $\boldsymbol{d}$ is a feasible descent direction and we can rewrite (2.7) as

$$\gamma^k(\beta; \boldsymbol{y}^k) = \mathcal{P}[\boldsymbol{y} + \beta \boldsymbol{d}],$$

and consider partitioning the *free* variables into two disjoint sets of indices such that

$$I_1 = \{i | y_i^k > 0 \text{ or } (y_i^k = 0 \text{ and } d_i \geq 0)\},$$
$$I_2 = \{i | y_i^k = 0 \text{ and } d_i < 0\}.$$

It is easy to see that there exists $\beta_1 > 0$ such that $\forall i \in I_1$,

$$y_i^k + \beta d_i \geq 0, \quad \forall \beta \leq \beta_1.$$

Let us define a new search direction $\bar{\boldsymbol{d}}$,

$$\bar{d}_i = \begin{cases} d_i, & i \in I_1, \\ 0, & \text{otherwise.} \end{cases}$$

Then we have

$$\gamma^k(\beta; \boldsymbol{y}^k) = \boldsymbol{y}^k + \beta \bar{\boldsymbol{d}}, \quad \forall \beta \in (0, \beta_1].$$

Since $\left[\nabla f(\boldsymbol{y}^k)\right]_i \leq 0$ and $d_i < 0$ for $i \in I_2$, we get

$$\sum_{i \in I_2} \left[\nabla f(\boldsymbol{y}^k)\right]_i \cdot d_i \geq 0.$$

Now we can conclude that

$$\nabla f(\boldsymbol{y}^k)^T \bar{\boldsymbol{d}} = \sum_{i \in I_1} \left[\nabla f(\boldsymbol{y}^k)\right]_i \cdot d_i \leq \sum_{i \in \{I_1 \cup I_2\}} \left[\nabla f(\boldsymbol{y}^k)\right]_i \cdot d_i = \nabla f(\boldsymbol{y}^k)^T \boldsymbol{d} < 0.$$

Hence $\bar{\boldsymbol{d}}$ is also a feasible descent direction. Therefore, letting $\alpha = 1$ as in (2.9), we get

$$\tilde{\boldsymbol{y}} = \gamma^k(\beta; \boldsymbol{y}^k) = \boldsymbol{y}^k + \beta \bar{\boldsymbol{d}},$$

and there exists $\bar{\beta} \in (0, \beta_1]$ such that

$$g^k(\tilde{\boldsymbol{y}}) < g^k(\boldsymbol{y}^k), \quad \forall \beta \in (0, \bar{\beta}]$$

where g^k is as in (2.4). From (2.5), since $\boldsymbol{z}^k$ remains fixed in $\boldsymbol{x}^{k+1}$, we can finally conclude that

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k), \quad \forall \beta \in (0, \bar{\beta}]. \qquad \square$$

Lemma 2 shows the limited minimization rule also leads to monotonic descent.

Lemma 2 (Descent with LM)**.** *If $\boldsymbol{x}^k$ is not a stationary point of Problem* (2.1)*, then for sufficiently small fixed β, there exists some constant $\bar{\alpha}$ such that*

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k), \quad \forall \alpha \in (0, \bar{\alpha}],$$

where $\boldsymbol{x}^{k+1}$ and $\tilde{\boldsymbol{y}}$ are given by (2.5)*,* (2.6)*, and* (2.7)*.*

Proof. Note that each g^k in (2.4) is convex. Since $\boldsymbol{y}^k$ and $\gamma^k(\beta; \boldsymbol{y}^k)$ are feasible, by using the convexity of g^k, it follows that

$$g^k(\boldsymbol{y}^k) + \nabla f(\boldsymbol{y}^k)^T \left(\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k\right) \leq g^k\left(\gamma^k(\beta; \boldsymbol{y}^k)\right).$$

From Lemma 1 we know that

$$g^k\left(\gamma^k(\beta; \boldsymbol{y}^k)\right) < g(\boldsymbol{y}^k),$$

for a sufficiently small β, which implies

$$\nabla f(\boldsymbol{y}^k)^T \left(\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k\right) \leq g^k\left(\gamma^k(\beta; \boldsymbol{y}^k)\right) - g(\boldsymbol{y}^k) < 0.$$

Consequently, $(\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k)$ is a feasible descent direction, whereby there exists some $\bar{\alpha} > 0$ such that

$$g^k\big(\boldsymbol{y}^k + \alpha(\gamma^k(\beta; \boldsymbol{y}^k) - \boldsymbol{y}^k)\big) < g^k(\boldsymbol{y}^k), \quad \forall \alpha \in (0, \bar{\alpha}].$$

As in Lemma 1, since $\boldsymbol{z}^k$ remains fixed in $\boldsymbol{x}^{k+1}$, we conclude that

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k), \quad \forall \alpha \in (0, \bar{\alpha}]. \qquad \square$$

In general, to prove the convergence of an iterative algorithm, one needs to show that the sequence of iterates $\{\boldsymbol{x}^k\}$ generated by the algorithm converges to some limit point. Since the existence of such limit points inevitably depends on the given problem instance, it is frequently assumed as a given. However, for the NNLS problem we can show the existence of such a limit point. In the following lemma, we show that our algorithm is guaranteed to have a limit point in the problem domain. That is, the limit point lies in the feasible set.

Lemma 3 (Limit-point)**.** *Let $\{\boldsymbol{x}^k\}$ be a sequence of points generated by* (2.5) *with either APA or LM. Then, this sequence has a limit $\boldsymbol{x}^*$. Furthermore, $\boldsymbol{x}^* \geq 0$.*

Proof. Assume that we start the iteration at $\boldsymbol{x}^0$ where $f(\boldsymbol{x}^0) = M$. By Lemma 1 and Lemma 2, $\{f(\boldsymbol{x}^k)\}$ is a monotonically decreasing sequence, whereby $\boldsymbol{x}^0$ is a maximizer of f over the M-level set of f. If a convex quadratic function f is bounded above, its M-level set is also bounded. Denote this M-level set by $\mathbb{X}$. Then we can choose $\boldsymbol{u} \in \mathbb{X}$ such that

$$\|\boldsymbol{u}\|_2 \geq \|\boldsymbol{x}\|_2, \forall \boldsymbol{x} \in \mathbb{X}.$$

Thus, $\{\boldsymbol{x}^k\}$ is bounded as $0 \leq \|\boldsymbol{x}^k\|_2 \leq \|\boldsymbol{u}\|_2$ for all k, and since $\mathbb{X}$ is compact, this sequence has a limit-point $\boldsymbol{x}^*$. Furthermore, since $\mathbb{X}$ is a compact subset of the feasible region and it contains every iterate of the algorithm, it also contains the limit-point $\boldsymbol{x}^*$. $\square$

For the subsequent lemmas, we consider the set $\mathbb{X}$ as the domain of Problem 2.1. Lemma 4 below proves a certain gradient related condition that is essential to the proof of convergence. This condition is frequently found in convergence proofs of iterative methods, and it essentially ensures that the step-sizes are "big" enough to guarantee convergence.

Lemma 4 (Gradient Related Condition)**.** *Let $\{\boldsymbol{x}^k\}$ be a sequence generated by* (2.5)*. Then, for any subsequence $\{\boldsymbol{x}^k\}_{k\in\mathcal{K}}$ that converges to a non-stationary point,*

$$\limsup_{k\to\infty,k\in\mathcal{K}} \|\boldsymbol{x}^{k+1}-\boldsymbol{x}^k\| < \infty, \tag{2.14}$$

$$\limsup_{k\to\infty,k\in\mathcal{K}} \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) < 0. \tag{2.15}$$

Proof. Lemma 3 tells us that $\{\boldsymbol{x}^k\}_{k\in\mathcal{K}}$ is a converging sequence over a compact domain, hence $\boldsymbol{x}^{k+1}-\boldsymbol{x}^k$ is also bounded for all k, therefore (2.14) holds. To show (2.15), consider a Taylor-series expansion of f at $\boldsymbol{x}^k$,

$$f(\boldsymbol{x}^{k+1}) = f(\boldsymbol{x}^k) + \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) + \tfrac{1}{2}(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k)^T\nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k). \tag{2.16}$$

Since f is quadratic (2.16) holds with equality for all k. Furthermore, since $\boldsymbol{A}$ has full-rank, $\nabla^2 f(\boldsymbol{x}^k) = \boldsymbol{A}^T\boldsymbol{A}$ is positive definite, whereby the third term on the right hand side is positive if and only if $\boldsymbol{x}^{k+1} \neq \boldsymbol{x}^k$. Combining this with either Lemma 1 or Lemma 2, we obtain

$$f(\boldsymbol{x}^{k+1}) - f(\boldsymbol{x}^k) = \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) + \epsilon_k < 0,$$

where

$$\epsilon_k = \tfrac{1}{2}(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k)^T\nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k). \tag{2.17}$$

Equivalently,

$$\nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) < -\epsilon_k, \quad \forall k \geq 0.$$

By taking limits, we obtain

$$\limsup_{k\to\infty,k\in\mathcal{K}} \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) < -\lim_{k\to\infty,k\in\mathcal{K}} \epsilon_k. \tag{2.18}$$

Note that Lemma 1 or Lemma 2 ensure that $\boldsymbol{x}^{k+1} \neq \boldsymbol{x}^k$ for all $k \in \mathcal{K}$ since $\boldsymbol{x}^k$ is non-stationary. We can see that ϵ_k in (2.17) is positive and the right-hand side of (2.18) is negative, hence (2.15) holds. □

Finally we are ready to prove the main convergence theorem. We remark that by Lemma 4, for the LM case, it is possible to invoke a general convergence proof for the feasible direction method. However, the following theorem provides a convergence proof for both APA and LS in a somewhat simplified way by exploiting properties of our quadratic objective function.

Theorem 1 (Convergence)**.** *Let $\{\boldsymbol{x}^k\}$ be a sequence of points generated by* (2.5)*. Then every limit point of $\{\boldsymbol{x}^k\}$ is a stationary point of Problem* (2.1)*, and hence optimal since the original problem is strictly convex.*

Proof. Our proof is by contradiction. Assume that $\{\boldsymbol{x}^k\}$ converges to a non-stationary point $\tilde{\boldsymbol{x}}$. We show that this assumption violates the gradient-condition of Lemma 4. We consider any step-size selection rule that satisfies the descent Lemma 1. By rearranging (2.16),

$$f(\boldsymbol{x}^k) - f(\boldsymbol{x}^{k+1}) = -\nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) - \epsilon_k > 0, \tag{2.19}$$

where $\epsilon_k = \frac{1}{2}(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k)^T\nabla^2 f(\boldsymbol{x}^k)(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k)$, as before. Taking both sides of (2.19) to the limit we have,

$$\lim_{k\to\infty}\big(f(\boldsymbol{x}^k) - f(\boldsymbol{x}^{k+1})\big) = -\lim_{k\to\infty} \nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k) - \lim_{k\to\infty}\epsilon_k > 0. \tag{2.20}$$

Since f is continuous over $\mathbb{X}$, from Lemmas 1,2 and 3, we see

$$\lim_{k\to\infty} f(\boldsymbol{x}^k) = f(\tilde{\boldsymbol{x}}).$$

Consequently,

$$\lim_{k\to\infty}\big(f(\boldsymbol{x}^k)-f(\boldsymbol{x}^{k+1})\big)=0, \tag{2.21}$$

which implies

$$\lim_{k\to\infty}\|\boldsymbol{x}^k-\boldsymbol{x}^{k+1}\|=0.$$

Since $\nabla^2 f(\boldsymbol{x}^k)=\boldsymbol{A}^T\boldsymbol{A}$ is a constant regardless of $\boldsymbol{x}^k$, and $\epsilon_k=\frac{1}{2}\|\boldsymbol{A}(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k)\|^2$,

$$L\cdot\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^k\|^2\geq\tfrac{1}{2}\|\boldsymbol{A}\|^2\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^k\|^2\geq\epsilon_k>0,\quad\forall k\geq 0$$

where L is any constant larger than $\frac{1}{2}\|\boldsymbol{A}\|^2$. On the other hand, we have

$$\lim_{k\to\infty}L\cdot\|\boldsymbol{x}^{k+1}-\boldsymbol{x}^k\|^2=0,$$

due to Lemma 4, which implies

$$\lim_{k\to\infty}\epsilon_k=0. \tag{2.22}$$

Combining the results (2.21) and (2.22) with (2.20) we obtain

$$\lim_{k\to\infty}\nabla f(\boldsymbol{x}^k)^T(\boldsymbol{x}^{k+1}-\boldsymbol{x}^k)=0,$$

which contradicts (2.15) in Lemma 4. Hence, the original assumption was false and $\tilde{\boldsymbol{x}}$ must be a stationary, hence optimal, point. □

Since the active set method identifies one active constraint at each iteration, it is guaranteed to identify all active constrains at the final solution after a finite number of iterations, though it could be a prohibitively large number of iterations. Our algorithm naturally incorporates multiple active constraints at each iteration, and we proved that the sequence of iterates $\{\boldsymbol{x}^k\}$ converges to a stationary point $\boldsymbol{x}^*$.

In Theorem 1 we showed that the algorithm is guaranteed to decrease the objective if the current iterate is not a stationary point. The next theorem describes the behavior of our algorithm at the optimum, proving that it stops if and only if the current iterate is a stationary point (hence an optimum) of the problem.

Theorem 2 (Optimality with APA)**.**

$$\boldsymbol{x}^*=\begin{bmatrix}\boldsymbol{y}^*\\ \boldsymbol{z}^*\end{bmatrix}=\begin{bmatrix}\mathcal{P}\big[\boldsymbol{y}^*-\beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)\big]\\ \boldsymbol{0}\end{bmatrix},\quad\forall\beta>0, \tag{2.23}$$

if and only if $\boldsymbol{x}^*$ is a stationary point of Problem (2.1)*.*

Proof. Since the problem is strictly convex (because we assumed $\boldsymbol{A}$ to be of full-rank), this stationary point will also be the minimum.

To show that $\boldsymbol{x}^*$ as given by (2.23) is a stationary point, it is enough to show that $\boldsymbol{x}^*$ satisfies the KKT conditions

$$\boldsymbol{x}^*\geq 0,\quad\nabla f(\boldsymbol{x}^*)\geq 0\text{ and }\nabla f(\boldsymbol{x}^*)^T\boldsymbol{x}^*=0.$$

It is obvious that $\boldsymbol{x}^*\geq 0$. To see that $\nabla f(\boldsymbol{x}^*)\geq 0$, consider $\boldsymbol{y}^*$, so that

$$\boldsymbol{y}^*=\mathcal{P}\big[\boldsymbol{y}^*-\beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)\big],\quad\forall\beta>0,$$

which implies

$$\beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)=\boldsymbol{0},\quad\forall\beta>0. \tag{2.24}$$

Thus, $\nabla f(\boldsymbol{y}^*)=\boldsymbol{0}$. Combining this with $\nabla f(\boldsymbol{z}^*)>0$ (by construction), we obtain $\nabla f(\boldsymbol{x}^*)\geq 0$. Finally,

$$\nabla f(\boldsymbol{x}^*)^T\boldsymbol{x}^*\quad=\quad\nabla f(\boldsymbol{y}^*)^T\boldsymbol{y}^*+\nabla f(\boldsymbol{z}^*)^T\boldsymbol{z}^*\quad=\quad\boldsymbol{0}^T\boldsymbol{y}^*+\nabla f(\boldsymbol{z}^*)^T\boldsymbol{0}=0.$$

Conversely, suppose that $\boldsymbol{x}^*$ satisfies KKT conditions. Then,

$$0 \;=\; \nabla f(\boldsymbol{x}^*)^T\boldsymbol{x}^* \quad = \quad \nabla f(\boldsymbol{y}^*)^T\boldsymbol{y}^* + \nabla f(\boldsymbol{z}^*)^T\boldsymbol{z}^* \quad = \quad \nabla f(\boldsymbol{y}^*)^T\boldsymbol{y}^*.$$

Thus, if $\boldsymbol{y}_i^* \neq 0$, then $\nabla f(\boldsymbol{y}^*)_i = 0$. On the other hand by the construction of $\boldsymbol{y}^*$, if $\boldsymbol{y}_i^*$, then $\nabla f(\boldsymbol{y}^*)_i = 0$. Hence, $\nabla f(\boldsymbol{y}^*) = 0$. Therefore,

$$\begin{bmatrix}\mathcal{P}\big[\boldsymbol{y}^* - \beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)\big] \\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}\mathcal{P}\big[\boldsymbol{y}^*\big] \\ \mathbf{0}\end{bmatrix} = \begin{bmatrix}\boldsymbol{y}^* \\ \boldsymbol{z}^*\end{bmatrix} = \boldsymbol{x}^*, \quad \forall\beta > 0. \qquad \square$$

Corollary 1 (Optimality wih LM)**.**

$$\boldsymbol{x}^* = \begin{bmatrix}\boldsymbol{y}^* \\ \boldsymbol{z}^*\end{bmatrix} = \begin{bmatrix}\boldsymbol{y}^* + \alpha\big(\mathcal{P}\big[\boldsymbol{y}^* - \beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)\big] - \boldsymbol{y}^*\big) \\ \mathbf{0}\end{bmatrix}, \quad \forall\alpha > 0, \tag{2.25}$$

if and only if $\boldsymbol{x}^*$ *is a stationary point of Problem* (2.1)*.*

Proof. From (2.25) we have

$$\boldsymbol{y}^* = \boldsymbol{y}^* + \alpha\big(\mathcal{P}\big[\boldsymbol{y}^* - \beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)\big] - \boldsymbol{y}^*\big),$$

which implies that for some fixed $\beta > 0$,

$$\boldsymbol{y}^* = \mathcal{P}\big[\boldsymbol{y}^* - \beta\bar{\boldsymbol{S}}^*\nabla f(\boldsymbol{y}^*)\big].$$

Now, we can invoke the proof of Theorem 2. $\square$

Remarks: If $\boldsymbol{A}$ is not of full-rank, our method can still be shown to converge to a global minimum of (2.1). Naturally, this minimum need not be unique.

2.4 Relationship to Other Methods

It is also interesting to explore the connection between our approach and other methods. We derived our method by appropriately modifying the general gradient projection method. The works of Bierlaire et al. [5] and Bertsekas [3] are particularly relevant. The former utilizes the projection step to enforce the feasibility of solution, while the latter employs a Newton-type method in addition to the projection for *simply constrained* problems (e.g., nonnegativity, boundedness of solution, and linear constraints).

2.4.1 Gradient Projection Approaches

Consider the following update from the gradient projection method.

$$\phi^k(\beta; \boldsymbol{x}^k) = \mathcal{P}\big(\boldsymbol{x}^k - \beta\nabla f(\boldsymbol{x}^k)\big),$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha\big(\phi^k(\beta; \boldsymbol{x}^k) - \boldsymbol{x}^k\big), \quad \alpha \geq 0. \tag{2.26}$$

The key differences between this gradient projection method and our approach are: (a) (2.26) does not use non-diagonal gradient scaling, and (b) it does not use a fixed (or free) variable set. Clearly, our updates (2.7) and (2.6) reduce to (2.26), by fixing the gradient scaling matrix $\boldsymbol{S}^k = \boldsymbol{I}$ at each iteration. In contrast to our approach (2.26) updates the entire solution vector $\boldsymbol{x}^k$ instead of using only the *free* variables. Viewed as a reduction of our method, this *anonymous* update is possible due to its special choice of the gradient scaling matrix, namely, the identity matrix. The underlying rationale is as follows. Suppose we are computing the *fixed* set at each iteration of the gradient projection method. Then at iteration k we have some $i \in I^k$. The i-th element in the next iterate $\boldsymbol{x}^{k+1}$ would then be computed as

$$\begin{aligned}[\phi(\beta^k; \boldsymbol{x}^k)]_i &= \mathcal{P}\left(x_i^k - \beta^k[\nabla f(\boldsymbol{x}^k)]_i\right) \\ &= \mathcal{P}\left(-\beta^k[\nabla f(x^k)]_i\right) \\ &= 0, \quad \forall\beta^k > 0, \\ \text{so that} \quad x_i^{k+1} &= x_i^k + \alpha\big([\phi^k(\beta^k; \boldsymbol{x}^k)]_i - x_i^k\big) \\ &= 0, \quad \alpha \geq 0.\end{aligned}$$

Therefore, the x_i^k are automatically *fixed* in our context.

We remark that with $\alpha = 1$ in (2.26), the gradient projection method presented above subsumes the method of Bierlaire et al. [5], which is a projected gradient method for NNLS; so does our approach.

2.4.2 Bertsekas Newton-type method

The work of Bertsekas [3] is even more relevant to our algorithm. If we confine our attention to the NNLS problem, then Bertsekas' method can be considered as an immediate generalization of the projected gradient method for NNLS [5]. Like the projected gradient method, Bertsekas also updates the entire solution vector $\boldsymbol{x}^k$. That is,

$$\boldsymbol{x}^{k+1} = \mathcal{P}\big(\boldsymbol{x}^k - \beta^k \boldsymbol{S}^k \nabla f(\boldsymbol{x}^k)\big), \quad k = 0, 1, \cdots$$

where β^k is determined by the APA rule [2]. After the computation of $\boldsymbol{x}^k$, Bertsekas' method also identifies a subset of the active constraints, namely,

$$I^+ = \Big\{ i \big| 0 \le x_i^k \le \epsilon_k, \quad [\nabla f(\boldsymbol{x}^k)]_i > 0 \Big\},$$

where

$$\epsilon_k = \min\left\{\epsilon,\ \left\|\boldsymbol{x}_k - \mathrm{P}[\boldsymbol{x}^k - \boldsymbol{M}\nabla f(\boldsymbol{x}^k)]\right\|\right\}$$

for some pre-specified small $\epsilon > 0$ and a diagonal positive-definite matrix $\boldsymbol{M}$. In his method, the gradient scaling $\boldsymbol{S}^k$ is reset to the inverse of the Hessian $(\nabla^2 f(\boldsymbol{x}^k))^{-1}$ at every iteration, followed by the modification

$$\boldsymbol{S}^k(i,j) = \boldsymbol{S}^k(j,i) = 0, \quad \forall i \in I^+, \quad j = 1, 2, \cdots, n, \quad j \ne i.$$

The only difference between the set I^+ and our *fixed* set I^k is that the former has a *relaxed* condition on x_i^k (our method uses $x_i^k = 0$ for $i \in I^k$), which is crucial for proving the identification property of his method. As shown above, we can avoid such a *relaxation* while solving NNLS. Similar to the case of the projected gradient method, assume that we have some $i \in I^+$ at iteration k, then

$$\begin{aligned} x_i^{k+1} &= \mathcal{P}\big(x_i^k - \beta^k \boldsymbol{S}^k(i,:)\nabla f(\boldsymbol{x}^k)\big) \\ &= \mathcal{P}\big(x_i^k - \beta^k \boldsymbol{S}^k(i,i)\nabla f(x^k)_i\big) \\ &= \big(-\beta^k \boldsymbol{S}^k(i,i)\nabla f(x^k)_i\big) \\ &= 0, \quad \forall \beta > 0. \end{aligned}$$

Again, the x_i^k are *fixed* in our context, and under the assumption that we set $\alpha = 1$ and $\boldsymbol{S}^k = (\nabla^2 f(\boldsymbol{x}^k))^{-1}$, our method is equivalent to performing optimization only upon the *free* variables not belonging to I^+ .

3 Numerical Results

In this section, we present numerical results illustrating the performance of our algorithm on both synthetic (§3.1) as well as real-world data (§3.2). We implemented our algorithm in MATLAB (`PQN-NNLS`). The other implementations that we test against are

1. `lsqnonneg`—MATLAB's implementation of the original Lawson and Hanson active set method [11].
2. `fnnls`—Bro and Jong's [1997] improved version of the Lawson-Hanson procedure in MATLAB.

3.1 Experiments with Synthetic Data

Our first set of experiments is with synthetic data. These experiments are targeted at assessing the performance of our method mainly by showing that it is faster than the other approaches. These experiments involve the following two problem sets:

- P1: random dense matrices with varying sizes, and
- P2: large random sparse matrices with varying sparsity.

Test matrices $\boldsymbol{A}$ and vectors $\boldsymbol{b}$ for problem set P1 are fully dense and are generated using MATLAB's `rand` function. Table 1 shows the number of rows m and columns n in $\boldsymbol{A}$ along with the condition number $\kappa(\boldsymbol{A})$ of $\boldsymbol{A}$. The length of $\boldsymbol{b}$ equals the number of rows m.

Matrix	P1-1	P1-2	P1-3	P1-4	P1-5
m	2800	3600	4400	5200	6000
n	2000	2400	2800	3200	3600
$\kappa(\boldsymbol{A})$	494	459	450	452	460

Table 1: The size and condition number of the test matrices $\boldsymbol{A}$ in problem set P1.

For problem set P2, we fix the size of $\boldsymbol{A}$ to be 12000×6400 and vary the sparsity using MATLAB's `sprand` function. For each $\boldsymbol{A}$, however, the corresponding $\boldsymbol{b}$ remains fully dense. Table 2 enlists the matrices in this set. Note that by *sparsity* we mean the ratio of zero entries to the number of all entries in the given matrix; MATLAB's `sprand` takes an argument *density*, which is nothing but $1 - sparsity$.

Matrix	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6
Sparsity	0.998	0.996	0.994	0.992	0.99	0.988

Table 2: The sparsity of the test matrices $\boldsymbol{A}$ (12000×6400) in problem set P2.

Table 3 shows the running times of MATLAB's `lsqnonneg`, `fnnls`, and our `PQN-NNLS` method for the matrices in problem set P1. The experiments were performed on an Intel Xeon 3.2GHz Linux machine with 8GB RAM. The CPU times reported were measured using the `cputime` function of MATLAB and the objective function values achieved by the various programs agreed to 6 significant digits. The timing results that we report were averaged over 20 different random runs. That is, for each problem in the set we generated a matrix of the specified size, and then ran the three methods on it 20 times to report the average running time values. We remark that `fnnls` requires the inputs in the form of $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{b}$, hence we included the time for these computations while reporting results for `fnnls`. On some mid-sized matrices, e.g., 2800×2000 (P1-1) and 3600×2400 (P1-2), `fnnls` and `PQN-NNLS` were competitive with each other. However, with increasing matrix sizes, the running time of `fnnls` increased drastically while that of `PQN-NNLS` altered only linearly. Consequently, for the largest matrix in the problem set (P1-5), `PQN-NNLS` outperformed `fnnls` significantly as can be seen in Table 3. Observe that `lsqnonneg` was certainly inferior to the other two methods, and we were unable to obtain results with it in a reasonable amount of time.

Method	P1-1	P1-2	P1-3	P1-4	P1-5
`lsqnonneg`	hours	hours	hours	hours	hours
`fnnls`	49.36	144.90	424.54	2097.8	3075.9
`PQN-NNLS`	86.84	164.83	202.03	229.59	294.02

Table 3: Comparison between MATLAB implementations: `lsqnonneg`, `fnnls`, and `PQN-NNLS` in terms of elapsed CPU time (seconds) with varying matrix sizes (Problem set P1).

In Table 4 we show running time comparisons analogous to those in Table 3, except that the matrices used were sparse. As shown in the table, `PQN-NNLS` generally outperforms `fnnls` for sparse matrices, and the difference becomes starker with increasing sparsity. For example, the running time of `PQN-NNLS` on the sparsest matrix (Problem P2-1) is approximately 600 times shorter than that of `fnnls`!

Method	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6
`lsqnonneg`	N/A	N/A	N/A	N/A	N/A	N/A
`fnnls`	66790	56148	43840	11695	12332	11525
`PQN-NNLS`	96.37	154.71	159.03	224.18	230.73	282.02

Table 4: Comparison between MATLAB implementations: `lsqnonneg`, `fnnls`, and `PQN-NNLS` in terms of elapsed CPU time (seconds) with varying matrix sizes (Problem set P2).

3.2 Experiments with Real-World Data

Our next set of experiments is on data drawn from real world applications. Through these experiments we demonstrate two things. First, that our method runs much faster than the competing approaches, and second, that the objective function values achieved by it are also competitive.

We show results below on two data sets that we chose to be representative of typical real-world datasets, in the sense that: i) they echo the results observed on synthetic data, thereby validating our experiments with synthetic data, and ii) the characteristics of these datasets are general enough to represent a class of similar matrices from real world applications. The problem P3 is a mid-scale proplem which is sparser than the problem P2-3 but denser than P2-2, and the problem P4 is large enough compared to the problem set P2. Both datasets can be downloaded from the University of Florida Sparse Matrix Collection where one can see even larger and sparser matrices, and as our experiments indicate, our method will perform even better on these data.

The datasets that we use for this set of experiments are:

- P3 : LPnetlib/lp_pilot87 dataset. A NETLIB problem that has size 2030×6680 and 74949 nonzero entries (approximately 0.9945 sparsity).
- P4 : Gset/G60 dataset. 7000×7000 random graph generated by *rudy*, an automatic graph generator. Approximately 0.035% of the upper triangle entries are nonzero (0.99965 sparsity). [2]

Method	Elapsed CPU Time		Objective Function Value	
	P3	P4	P3	P4
`lsqnonneg`	hours	hours	N/A	N/A
`fnnls`	4.26×10^4	7.27×10^3	1.28×10^{-10}	1.62×10^2
`PQN-NNLS`	3.93×10^4	2.04×10^2	3.72×10^{-10}	1.62×10^2

Table 5: Comparison between MATLAB implementations: `lsqnonneg`, `fnnls`, and `PQN-NNLS` in terms of elapsed CPU time (seconds) and final objective function value on Problem sets P3 and P4.

As before, we were not able to run `lsqnonneg` for Problems P3 and P4. For a mid-scale sparse matrix, e.g. LPnetlib/lp_pilot87, `fnnls` and `PQN-NNLS` are competitive with each other, both in terms of the elapsed CPU time and the final objective function value. However, as observed in the experiments on artificial matrices, `PQN-NNLS` starts excelling as the problem size becomes larger (and sparser). From Table 5 we can see that `PQN-NNLS` runs approximately 35 times than `fnnls`, while both algorithms agree to 8 digits in the objective function value (only 3 digits shown for brevity).

4 Discussion and Future work

In this paper we proposed a projection method with non-diagonal gradient scaling to solve the nonnegative least squares problem. Our method (`PQN-NNLS`) avoids the pre-computation of $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}^T\boldsymbol{b}$, which is required for the state of the art active set method `fnnls` but prohibitive for large-scale problems. It also

[2] Both LPnetlib/lp_pilot87 and Gset/G60 matrices can be downloaded from the University of Florida Sparse Matrix Collection (http://www.cise.ufl.edu/research/sparse/matrices/list_by_nnz.html), id 525 and 656 respectively.

retains the simplicity of the popular projected gradient method, while overcoming deficiencies such as slow convergence rate. Our experiments showed that our method outperforms other standard approaches to solving the NNLS problem, especially for larger scale problems.

4.1 Extensions

Although we presented our method with only nonnegativity constraints in this report, with a few minor changes we can extend it to handle bound constraints such as $\boldsymbol{a} \preceq \boldsymbol{x} \preceq \boldsymbol{b}$. Specifically, if we define a projection $\mathrm{P}^*(\cdot)$ and an index set $I^*(\boldsymbol{x})$ as follows

$$\mathrm{P}^*(x_i) = \begin{cases} a_i & : \quad x_i \leq a_i \\ x_i & : \quad a_i < x_i < b_i \\ b_i & : \quad b_i \leq x_i \end{cases} \tag{4.1}$$

$$I^*(\boldsymbol{x}) = \left\{ i \middle| x^i = a^i, \frac{\partial f(\boldsymbol{x})}{\partial x^i} > 0 \text{ or } x^i = b^i, \frac{\partial f(\boldsymbol{x})}{\partial x^i} < 0 \right\} \tag{4.2}$$

then, we can obtain the desired method by substituting above definitions for $\mathrm{P}(\cdot)$ and $I(\boldsymbol{x})$ respectively.

Another interesting extension considers the case with multiple right-hand sides. That is, solving the problem

$$\begin{aligned} \underset{\boldsymbol{x}}{\text{minimize}} \quad & \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}_i\|^2 \\ \text{subject to} \quad & \boldsymbol{x} \geq 0, \end{aligned}$$

where $\boldsymbol{b}_i$ for $1 \leq i \leq N$ are varied. Naturally, since $\boldsymbol{A}$ remains fixed, the first simple attempt just solves the problem for $\boldsymbol{b}_1$, and then retains the approximation to the Hessian, and solves the subsequent problems without having to recompute this approximation. However, more work is needed to determine if this approach can be improved.

4.2 Future Work

In addition to the least squares to measure the proximity of two vectors, other convex distortion measures such as the Kullback-Leibler divergence can be used as objective functions. Since the algoritmic framework of `PQN-NNLS` remains same for any convex objective functions, it is natural to extend our method to such problems. It is also an interesting problem to incorporate linearly constrained least-squares problems without breaking the simplicity of algorithm (by passing to the dual problem). Further algorithmic improvements to the crucial line-search steps of our algorithm are needed to fully automate the solution without requiring any parameter selection or tuning on part of the user remain the subject of future research.

References

[1] P. L. De Angelis and G. Toraldo. On the Identification Property of a Projected Gradient Method. *SIAM Journal on Numerical Analysis*, 30(5):1483–1497, 1993.

[2] Dimitri P. Bertsekas. On the Goldstein-Levitin-Poljak Gradient Projection Method. *IEEE Transcations on Automatic Control*, 21(2):174–184, 1976.

[3] Dimitri P. Bertsekas. Projected Newton Methods for Optimization Problems with Simple Constraints. *SIAM Jounal on Control and Optimization*, 20(2):221–246, 1982.

[4] Dimitri. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[5] 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.

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

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

[8] Jason Cantarella and Michael Piatek. Tsnnls: A Solver for Large Sparse Least Squares Problems with Non-negative Variables. *ArXiv Computer Science e-prints*, 2004.

[9] J.C. Dunn. Global and Asymptotic Convergence Rate Estimates for a Class of Projected Gradient Processes. *SIAM Jounal on Control and Optimization*, 19:368–400, 1981.

[10] Philip E. Gill, Walter Murray, and Margaret H. Wright. *Practical Optimization*. Academic Press, 1981.

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

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

[13] Luis F. Portugal, Joaquim J. Judice, and Luis N. Vicente. A Comparison of Block Pivoting and Interior-point Algorithms for Linear Least Squares Problems with Nonnegative Variables. *Mathematics of Computation*, 63(208):625–643, 1994.

[14] 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.

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

[16] M. H. van Benthem and M. R. Keenan. Fast Algorithm for the Solution of Large-scale Non-negativity-constrained Least Squares Problems. *Journal of Chemometrics*, 18:441–450, 2004.