TACKLING BOX-CONSTRAINED OPTIMIZATION VIA A NEW PROJECTED QUASI-NEWTON APPROACH

DONGMIN KIM*, SUVRIT SRA†, AND INDERJIT S. DHILLON*

Abstract. Numerous scientific applications across a variety of fields depend on box-constrained convex optimization. Box-constrained problems therefore continue to attract research interest. We address box-constrained (strictly convex) problems by deriving two new quasi-Newton algorithms. Our algorithms are positioned between the projected-gradient [J. B. Rosen, *J. SIAM*, 8(1), 1960, pp. 181–217], and projected-Newton [D. P. Bertsekas, *SIAM J. Cont. & Opt.*, 20(2), 1982, pp. 221–246] methods. We also prove their convergence under a simple Armijo step-size rule. We provide experimental results for two particular box-constrained problems: nonnegative least squares (NNLS), and nonnegative Kullback-Leibler (NNKL) minimization. For both NNLS and NNKL our algorithms perform competitively as compared to well-established methods on medium-sized problems; for larger problems our approach frequently outperforms the competition.

Key words. Non-negative least squares, KL-divergence minimization, projected Newton methods, quasi-Newton, box-constrained convex optimization,

1. Introduction. The central object of study in this paper is the *box-constrained* optimization problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}), \quad \text{s.t.} \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \tag{1.1}$$

where $\boldsymbol{l}$ and $\boldsymbol{u}$ are fixed vectors and inequalities are taken componentwise; the function f is assumed to be twice continuously differentiable and *strictly* convex.

Problem (1.1) is a simple constrained optimization problem that can be solved by numerous methods. But general purpose methods might be overkill; indeed, for simple problems they can be computationally expensive or even needlessly complicated. These concerns motivate us to develop algorithms specifically designed for (1.1). Our algorithms are founded upon the *unconstrained* BFGS and L-BFGS procedures, both of which we adapt to our *constrained* setting. This adaptation is the crux of our work; but before we describe it, we would like to position it vis-á-vis other methods.

A basic, and perhaps obvious, approach to solve (1.1) is the well-known projected-gradient method [23]. This method, however, frequently suffers from slow convergence, making it unappealing when medium to high accuracy solutions are desired. In such cases higher-order methods might be more preferable, e.g., LBFGS-B [5][1], TRON [12], or projected-Newton [1] (which could be viewed as a non-trivial second-order extension to projected-gradient). LBFGS-B uses a limited amount of additional memory to approximate the second-order information of the objective function, while TRON uses trust-regions [15], and is usually very efficient for medium scale problems.

LBFGS-B and TRON tackle constraints by explicitly distinguishing between steps that identify active variables and steps that solve the resulting unconstrained subproblems. In comparison, projected-Newton makes the variable-identification step *implicit* by dynamically finding active candidates at every iteration. Bertsekas noted that such an implicit identification of active variables could lead to unstable active sets across iterations, which in turn could adversely affect convergence [1]. Thus, to

*Department of Computer Sciences, University of Texas, Austin, TX 78712-1188, USA (dmkim@cs.utexas.edu, inderjit@cs.utexas.edu).

†Max-Planck Institute for Biological Cybernetics, 72076 Tübingen, Germany (suvrit.sra@tuebingen.mpg.de).

[1] An extension of L-BFGS [18] that allows box-constraints.

stabilize the projected-Newton method he developed a refined procedure that guarantees finite identification of the active set. Although we build on projected-Newton, we avoid its refinements and make important modifications: we use simpler line-search and a different choice of working sets. These changes sacrifice the finite identification property, but at the gain of implementation ease. Moreover, by supporting a limited-memory version, we allow our method to scale well to large problems.

The rest of this paper is organized as follows. In §2.1 we overview our approach; in §2.2 major implementation issues are discussed; and in §3 we prove convergence. In §4 we compare our method to several other major approaches and show numerical results across several synthetic as well as real-life data. Finally, §5 concludes the paper with an outlook to future work.

2. Algorithms and Theory. Recall that the central problem of interest is

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}), \quad \text{subject to} \quad \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \tag{2.1}$$

where $\boldsymbol{l}$ and $\boldsymbol{u}$ are fixed vectors and inequalities are taken componentwise; f is a twice continuously differentiable, strictly convex function.

2.1. Overview of the method. Our algorithm for solving (2.1) is inspired by projected-gradient [23] and projected-Newton [1]. Viewed in the general light of gradient methods [2], our method distinguishes itself in two main ways: (i) its choice of the variables optimized at each iteration; and (ii) the *gradient-scaling* it uses to compute the descent direction. Further details of the resulting iterative method follow.

At each iteration we partition the variables into two groups: *free* and *fixed*. Then, we optimize over only the free variables while holding the fixed ones unchanged. The *fixed* variables are defined as a particular subset of the active variables[2], and this subset is identified using both the gradient and second-order information.

Formally, first at iteration k, we identify the *binding set*

$$I_1^k = \left\{ i \,\middle|\, x_i^k = l_i \wedge \partial_i f(\boldsymbol{x}^k) > 0, \quad \text{or} \quad x_i^k = u_i \wedge \partial_i f(\boldsymbol{x}^k) < 0 \right\}, \tag{2.2}$$

where $\partial_i f(\boldsymbol{x}^k)$ denotes the i-th component of the gradient $\nabla f(\boldsymbol{x}^k)$. The set I_1^k collects variables that satisfy the KKT conditions ($1 \leq i \leq n$):

$$\nabla f(\boldsymbol{x}) - \boldsymbol{\lambda} + \boldsymbol{\mu} = 0, \quad \lambda_i (x_i - l_i) = 0, \quad \mu_i (x_i - u_i) = 0, \quad \text{and} \quad \boldsymbol{\lambda},\ \boldsymbol{\mu} \geq 0,$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are Lagrange multipliers corresponding to the $\boldsymbol{\ell} \leq \boldsymbol{x}$ and $\boldsymbol{x} \leq \boldsymbol{u}$ constraints, respectively. To see why the gradient information makes a difference in (2.2), consider an active variable $x_i^k = l_i$ whose gradient component $\partial_i f(\boldsymbol{x}^k) \leq 0$. For such a variable it may be possible to decrease the objective further, if at the next iteration we allow $x_i^{k+1} > l_i$. Thus, it is better to leave x_i^k free by not including it in (2.2) (a similar observation holds for $x_i^k = u_i$).

Next, we refine the active variables using gradient-scaling information. To that end, let $\boldsymbol{S}^k$ be some non-diagonal positive-definite matrix. Further let $\bar{\boldsymbol{S}}^k$ be the matrix induced by $\boldsymbol{S}^k$ and the variables *not* in the binding set, i.e.,

$$\bar{\boldsymbol{S}}_{ij}^k = \begin{cases} \boldsymbol{S}_{ij}^k, & \text{if } i, j \notin I_1^k, \\ 0, & \text{otherwise.} \end{cases}$$

[2]Components of $\boldsymbol{x}$ that satisfy the constraints with equality.

Then, we define the second index-set

$$I_2^k = \big\{ i \,\big|\, x_i^k = l_i \wedge \big[\bar{\boldsymbol{S}}^{\boldsymbol{k}} \nabla f(\boldsymbol{x}^k)\big]_i > 0, \quad \text{or} \quad x_i^k = u_i \wedge \big[\bar{\boldsymbol{S}}^{\boldsymbol{k}} \nabla f(\boldsymbol{x}^k)\big]_i < 0 \big\}. \tag{2.3}$$

The set I_2^k captures variables left free by I_1^k, that should actually be fixed—e.g., if x_i^k is active but not in the binding set (i.e., $x_i^k = l_i$, but $x_i^k \notin I_1^k$), then as argued before, the objective function can be potentially decreased if we let $x_i^{k+1} > l_i$. In the presence of gradient-scaling, however, this decrease might not happen because even though $x_i^k \notin I_1^k$, the scaled-gradient $\big[\bar{\boldsymbol{S}}^{\boldsymbol{k}} \nabla f(\boldsymbol{x}^k)\big]_i$ is positive. Thus, at the next iteration too we should fix $x_i^{k+1} = l_i$. The actual *fixed-set* is as defined below.

DEFINITION 2.1 (Fixed set). *Given sets I_1^k and I_2^k identified by (2.2) and (2.3), respectively, the* fixed *set I^k at iteration k is defined as the union $I^k = I_1^k \cup I_2^k$.*

2.1.1. Projected quasi-Newton Step. Having defined the *fixed* set we now present the main projected quasi-Newton step. The projection part is easy and requires computing $\mathcal{P}_\Omega(\boldsymbol{x})$, the orthogonal projection of $\boldsymbol{x}$ onto the set $\Omega = [\boldsymbol{l}, \boldsymbol{u}]$. The quasi-Newton part is slightly more involved as the variables not in the fixed-set I^k influence the gradient-scaling matrix, which is given by

$$\hat{\boldsymbol{S}}_{ij}^k = \begin{cases} \boldsymbol{S}_{ij}^k, & \text{if } i, j \notin I^k, \\ 0, & \text{otherwise.} \end{cases}$$

Finally, our projected quasi-Newton step is given by

$$\boldsymbol{x}^{k+1} = \mathcal{P}_\Omega\big(\boldsymbol{x}^k - \alpha^k \hat{\boldsymbol{S}}^k \nabla f(\boldsymbol{x}^k)\big), \tag{2.4}$$

where $\alpha^k > 0$ is a step-size.

Pseudo-code encapsulating the details described above is presented in Algorithm 1, which we call projected quasi-Newton (PQN). Observe that in PQN an arbitrary positive-definite matrix may be used to initialize $\boldsymbol{S}^0$, and any feasible point to initialize $\boldsymbol{x}^0$. Further notice that we have not yet specified the details of computing the step-size α^k, and of updating the matrix $\boldsymbol{S}^k$. We now turn to these two important steps (and some other implementation issues) in §2.2 below.

2.2. Implementation details. In this section we describe a line-search routine for computing α^k, and two methods for updating the gradient scaling matrix $\boldsymbol{S}^k$.

2.2.1. Line Search. We choose the Armijo step-size rule [2, 6] for the line-search; this rule is simple and usually works well both for unconstrained and constrained problems. The important difference in our implementation of Armijo is that we *restrict* the computation to be over the *free* variables only.

More specifically, to compute the step-size, first choose an initial step $\gamma > 0$ and some scalars τ, $\sigma \in (0, 1)$. Then, determine the smallest non-negative integer t, for which the *sufficient-descent* condition

$$f(\boldsymbol{x}^k) - f\big(\mathcal{P}_\Omega\big[\boldsymbol{x}^k - \gamma\sigma^t \hat{\boldsymbol{S}}^k \nabla f(\boldsymbol{x}^k)\big]\big) \quad \geq \quad \tau\gamma\sigma^t \nabla f(\boldsymbol{x}^k)^T \hat{\boldsymbol{S}}^k \nabla f(\boldsymbol{x}^k), \tag{2.5}$$

is satisfied. The step-size is then set to $\alpha^k = \gamma\sigma^t$. In §3 we prove that with this Armijo-based step-size our algorithm converges to the optimum.

2.2.2. Gradient Scaling. Now we turn to gradient scaling. To circumvent some of the problems associated with using the full (inverse) Hessian, it is usual to approximate it. Popular choices involve iteratively approximating the Hessian

Algorithm 1: Projected Quasi-Newton Framework

Input: Function: $f(\boldsymbol{x})$; vectors $\boldsymbol{l}, \boldsymbol{u} \in \mathbb{R}^n$
Output: $\boldsymbol{x}^* = \operatorname{argmin}_{\boldsymbol{l} \le \boldsymbol{x} \le \boldsymbol{u}} f(\boldsymbol{x})$
Initialize $k \leftarrow 0$, $\boldsymbol{S}^k \leftarrow \boldsymbol{I}$, and $\boldsymbol{l} \le \boldsymbol{x}^k \le \boldsymbol{u}$;
repeat
  Compute the first index set
  $I_1^k = \left\{ i \middle| x_i^k = l_i \wedge \partial_i f(\boldsymbol{x}^k) > 0, \text{ or } x_i^k = u_i \wedge \partial_i f(\boldsymbol{x}^k) < 0 \right\}$;
  Compute the second index set
  $I_2^k = \left\{ i \middle| x_i^k = l_i \wedge \left[\bar{\boldsymbol{S}}^{\boldsymbol{k}} \nabla f(\boldsymbol{x}^k)\right]_i > 0, \text{ or } x_i^k = u_i \wedge \left[\bar{\boldsymbol{S}}^{\boldsymbol{k}} \nabla f(\boldsymbol{x}^k)\right]_i < 0 \right\}$;
  Compute the *fixed* set $I^k = I_1^k \cup I_2^k$;
  if $\forall i, i \in I^k$ **or** $\partial_i f(\boldsymbol{x}^k) = 0$ *for all* $i \notin I^k$ **then**
    $\bar{\boldsymbol{S}}^k = \boldsymbol{I}$;
    Re-compute I^k;
  begin
    Find appropriate value for α^k using line-search;
    $\boldsymbol{x}^{k+1} \leftarrow \mathcal{P}_\Omega \left[\boldsymbol{x}^k - \alpha^k \hat{\boldsymbol{S}}^k \nabla f(\boldsymbol{x}^k)\right]$;
  Update gradient scaling matrix $\boldsymbol{S}^k$ to obtain $\boldsymbol{S}^{k+1}$, if necessary;
  $k \leftarrow k + 1$;
until *Stopping criteria are met*;

using the Powell-Symmetric-Broyden (PSB), Davidson-Fletcher-Powell (DFP), or the Broyden-Fletcher-Goldfarb-Shanno (BFGS) updates; the latter are believed to be the most effective in general [2, 8], so we adapt them for our algorithm.

Suppose $\boldsymbol{H}^k$ is the current approximation to the Hessian, and that vectors $\boldsymbol{g}$ and $\boldsymbol{s}$ are given by the differences

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

Then, the BFGS update makes a rank-two correction to $\boldsymbol{H}^k$, and is given by

$$\boldsymbol{H}^{k+1} = \boldsymbol{H}^k - \frac{\boldsymbol{H}^k \boldsymbol{s}\boldsymbol{s}^T \boldsymbol{H}^k}{\boldsymbol{s}^T \boldsymbol{H}^k \boldsymbol{s}} + \frac{\boldsymbol{g}\boldsymbol{g}^T}{\boldsymbol{s}^T \boldsymbol{g}}. \tag{2.6}$$

Let $\boldsymbol{S}^k$ be the inverse of $\boldsymbol{H}^k$. Applying the Sherman-Morrison-Woodbury formula to (2.6) one obtains the update

$$\boldsymbol{S}^{k+1} = \boldsymbol{S}^k + \left(1 + \frac{\boldsymbol{g}^T \boldsymbol{S}^k \boldsymbol{g}}{\boldsymbol{s}^T \boldsymbol{g}}\right) \frac{\boldsymbol{s}\boldsymbol{s}^T}{\boldsymbol{s}^T \boldsymbol{g}} - \frac{(\boldsymbol{S}^k \boldsymbol{g}\boldsymbol{s}^T + \boldsymbol{s}\boldsymbol{g}^T \boldsymbol{S}^k)}{\boldsymbol{s}^T \boldsymbol{g}}. \tag{2.7}$$

The version of Alg. 1 that uses (2.7) for the gradient-scaling will be called PQN-BFGS.

Although the update of gradient scaling matrix via BFGS updates is efficient, it can require up to $O(n^2)$ *memory*, which restricts its applicability to large-scale problems. But we can also implement a limited memory BFGS (L-BFGS) [18] version, which we call PQN-LBFGS. Here, instead of storing an actual gradient scaling matrix, a small number of additional vectors (say M) are used to approximate the (inverse)

Hessian. The standard L-BFGS approach is implemented using the following formula

$$\begin{aligned}\boldsymbol{S}^k = &\frac{\boldsymbol{s}_{k-1}^T\boldsymbol{g}_{k-1}}{\boldsymbol{g}_{k-1}^T\boldsymbol{g}_{k-1}}\bar{\boldsymbol{V}}_{k-M}^T\bar{\boldsymbol{V}}_{k-M} + \rho_{k-M}\bar{\boldsymbol{V}}_{k-M+1}^T\boldsymbol{s}_{k-M}\boldsymbol{s}_{k-M}^T\bar{\boldsymbol{V}}_{k-M+1}\\ &+ \rho_{k-M+1}\bar{\boldsymbol{V}}_{k-M+2}^T\boldsymbol{s}_{k-M+1}\boldsymbol{s}_{k-M+1}^T\bar{\boldsymbol{V}}_{k-M+2}\\ &+\cdots\\ &+ \rho_{k-1}\boldsymbol{s}_{k-1}\boldsymbol{s}_{k-1}^T,\end{aligned} \tag{2.8}$$

where $\boldsymbol{S}^0 = \boldsymbol{I}$, $k \geq 1$, the scalars ρ_k, and matrices $\bar{\boldsymbol{V}}_{k-M}$ are defined by

$$\rho_k = 1/(\boldsymbol{s}_k^T\boldsymbol{g}_k), \qquad \bar{\boldsymbol{V}}_{k-M} = [\boldsymbol{V}_{k-M}\cdots\boldsymbol{V}_{k-1}], \quad \text{and} \quad \boldsymbol{V}_k = \boldsymbol{I} - \rho_k\boldsymbol{s}_k\boldsymbol{g}_k^T.$$

At this point the reader might wonder how our method *differs* from the original BFGS and L-BFGS methods. There are two basic differences: First, we are performing constrained optimization. Second, we actually perform the updates (2.7) and (2.8) with respect to the first index set (2.2)—that is, both updates (2.7) and (2.8) result in the matrix $\bar{\boldsymbol{S}}^k$, which is then used in Algorithm 1.

3. Convergence. Let $\{\boldsymbol{x}^k\}$ be a sequence of iterates produced by either the PQN-BFGS or the PQN-LBFGS versions of Algorithm 1. We prove below under mild assumptions that each accumulation point of the sequence $\{\boldsymbol{x}^k\}$ is a stationary point of (2.1). We begin by showing that the sequence $\{f(\boldsymbol{x}^k)\}$ is monotonically decreasing.

LEMMA 3.1 (Descent). *If $\boldsymbol{x}^k$ is not a stationary point of Problem* (2.1)*, then with $\boldsymbol{x}^{k+1}$ given by (2.4), there exists some constant $\hat{\alpha} > 0$ such that*

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k), \quad \textit{for all } \alpha^k \in (0, \hat{\alpha}].$$

Proof. By construction of the *fixed* set I^k, the *free* variables in $\boldsymbol{x}^k$ must satisfy:

$$\begin{aligned}\text{either} \quad & l_i < x_i^k < u_i,\\ \text{or} \quad & x_i = l_i \ \wedge\ \partial_i f(\boldsymbol{x}^k) \leq 0 \wedge\ [\bar{\boldsymbol{S}}^k\nabla f(\boldsymbol{x}^k)]_i \leq 0,\\ \text{or} \quad & x_i = u_i \wedge\ \partial_i f(\boldsymbol{x}^k) \geq 0 \wedge\ [\bar{\boldsymbol{S}}^k\nabla f(\boldsymbol{x}^k)]_i \geq 0.\end{aligned}$$

Note that if $I_1^k \cup I_2^k$ captures all the variables in $\boldsymbol{x}$, Algorithm 1 reduces to $I^k = I_1^k$. If there are still *no* free variables, it means that all KKT conditions are fulfilled; but this implies that $\boldsymbol{x}^k$ is stationary, a contradiction. Thus, for a non-stationary point $\boldsymbol{x}^k$, there exists at least one index i in $\boldsymbol{x}^k$ for which $\partial_i f(\boldsymbol{x}^k) \neq 0$. This $\boldsymbol{x}^k$ yields a descent direction. To see how, let $\boldsymbol{d}^k$ be the direction defined by

$$d_i^k = \begin{cases} -[\hat{\boldsymbol{S}}^k\nabla f(\boldsymbol{x}^k)]_i, & \text{if } i \notin I^k,\\ 0, & \text{otherwise.}\end{cases}$$

Now partition (reordering variables if necessary) all objects with respect to I^k so that

$$\hat{\boldsymbol{S}}^k = \begin{bmatrix}\boldsymbol{T}^k & \boldsymbol{0}\\ \boldsymbol{0} & \boldsymbol{0}\end{bmatrix}, \quad \nabla f(\boldsymbol{x}^k) = \begin{bmatrix}\boldsymbol{y}^k\\ \boldsymbol{z}^k\end{bmatrix}, \quad \text{and} \quad \boldsymbol{d}^k = \begin{bmatrix}\boldsymbol{p}^k\\ \boldsymbol{0}\end{bmatrix} = \begin{bmatrix}-\boldsymbol{T}^k\boldsymbol{y}^k\\ \boldsymbol{0}\end{bmatrix}. \tag{3.1}$$

In (3.1) the components that constitute $\boldsymbol{z}^k$ belong to I^k, while those that constitute $\boldsymbol{y}^k, \boldsymbol{p}^k$ and $\boldsymbol{T}^k$ do not. Given this partitioning we see that

$$\langle\nabla f(\boldsymbol{x}^k), \boldsymbol{d}^k\rangle = \langle\boldsymbol{y}^k, \boldsymbol{p}^k\rangle = -\langle\boldsymbol{y}^k, \boldsymbol{T}^k\boldsymbol{y}^k\rangle < 0,$$

where the final inequality follows because $\boldsymbol{T}^k$ is positive-definite (as it is a principal submatrix of the positive definite matrix $\boldsymbol{S}^k$). Thus, $\boldsymbol{d}^k$ is a descent direction.

Now we only need to verify that descent along $\boldsymbol{d}^k$ upholds feasibility. It is easy to check that there exists an $\bar{\alpha} > 0$ such that for $i \notin I^k$,

$$l_i \leq \mathcal{P}_\Omega \left[x_i^k + \alpha^k d_i^k\right] = x_i^k + \alpha^k d_i^k \leq u_i, \quad \text{for all} \;\; \alpha^k \leq \bar{\alpha}.$$

Noting that $\boldsymbol{d}^k$ is zero for fixed variables, we may equivalently write

$$\boldsymbol{l} \leq \mathcal{P}_\Omega \left[\boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k\right] = \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k \leq \boldsymbol{u}, \quad \text{for all} \;\; \alpha^k \leq \bar{\alpha}. \tag{3.2}$$

Thus, we can finally conclude that there exists some $\hat{\alpha} \leq \bar{\alpha}$ such that

$$f(\boldsymbol{x}^{k+1}) < f(\boldsymbol{x}^k), \quad \text{for all} \;\; \alpha^k \in (0, \hat{\alpha}]. \qquad \square$$

To prove the convergence of an iterative algorithm one generally needs to show that the sequence of iterates $\{\boldsymbol{x}^k\}$ has an accumulation point. Since the existence of limit points inevitably depends on the given problem instance, it is frequently assumed as a given. For our problem, accumulation points certainly exist as we are dealing with a continuous function over a compact domain.

To prove the main theorem, we consider the set $\Omega = [\boldsymbol{l}, \boldsymbol{u}]$ as the domain of Problem 2.1 and assume the following condition:

ASSUMPTION 1. *The eigenvalues of the gradient scaling matrix $\boldsymbol{T}^k$ (for all k) lie in the interval $[m, M]$ where $0 < m \leq M < \infty$.*

This assumption is frequently found in convergence proofs of iterative methods, especially when proving the so-called *gradient related condition*, which, in turn helps prove convergence for several iterative descent methods (as it ensures that the search direction remains well-defined throughout the iterations).

LEMMA 3.2 (Gradient Related Condition). *Let $\{\boldsymbol{x}^k\}$ be a sequence generated by (2.4). Then, for any subsequence $\{\boldsymbol{x}^k\}_{k\in\mathcal{K}}$ that converges to a non-stationary point, the corresponding subsequence $\{\boldsymbol{d}^k\}_{k\in\mathcal{K}}$ is bounded and satisfies*

$$\limsup_{k\to\infty, k\in\mathcal{K}} \|\boldsymbol{x}^{k+1} - \boldsymbol{x}^k\| < \infty \tag{3.3}$$

$$\limsup_{k\to\infty, k\in\mathcal{K}} \nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k < 0. \tag{3.4}$$

Proof. Since $\{\boldsymbol{x}^k\}_{k\in\mathcal{K}}$ is a converging sequence over a compact domain, $\boldsymbol{x}^{k+1} - \boldsymbol{x}^k$ is also bounded for all k, therefore (3.3) holds. Since $\boldsymbol{d}^k = [-\boldsymbol{T}^k \boldsymbol{y}^k; \boldsymbol{0}]$ and $\boldsymbol{T}^k$ is positive-definite, under Assumption 1, the gradient-related condition (3.4) follows immediately [2, Chapter 1]. □

Now we are ready to prove the main convergence theorem. Although using Lemma 3.2, it is possible to invoke a general convergence proof for the feasible direction method (e.g. [2]), for completeness we outline a short proof.

THEOREM 3.3 (Convergence). *Let $\{\boldsymbol{x}^k\}$ be a sequence of points generated by (2.4). Then every limit point of $\{\boldsymbol{x}^k\}$ is a stationary point of Problem (2.1).*

Proof. The proof is by contradiction. Since $\{f(\boldsymbol{x}^k)\}$ is monotonically decreasing (Lemma 3.1), the domain $\Omega = [\boldsymbol{l}, \boldsymbol{u}]$ is compact, and f is continuous, we have

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

Further, the sufficient-descent condition (2.5) implies that

$$f(\boldsymbol{x}^k) - f(\boldsymbol{x}^{k+1}) \geq -\tau\alpha^k \nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k, \qquad \tau \in (0,1), \tag{3.5}$$

whereby

$$\{\alpha^k \nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k\} \to 0. \tag{3.6}$$

Assume that a subsequence $\{\boldsymbol{x}^k\}_{k\in\mathcal{K}}$ converges to a non-stationary point $\tilde{\boldsymbol{x}}$. From (3.4) and (3.6), we obtain $\{\alpha^k\}_{k\in\mathcal{K}} \to 0$. Since $\boldsymbol{d}^k$ is gradient related, this implies that we have some index $k_1 \geq 0$ that satisfies (3.2) such that

$$\mathcal{P}_\Omega\left[\boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k\right] = \boldsymbol{x}^k + \alpha^k \boldsymbol{d}^k, \quad \text{for all } \ k \in \mathcal{K},\ k \geq k_1.$$

Further, the Armijo rule (2.5) implies that there exists an index k_2 satisfying

$$f(\boldsymbol{x}^k) - f\big(\mathcal{P}_\Omega\big[\boldsymbol{x}^k + \frac{\alpha^k}{\gamma}\boldsymbol{d}^k\big]\big) = f(\boldsymbol{x}^k) - f\big(\boldsymbol{x}^k + \frac{\alpha^k}{\gamma}\boldsymbol{d}^k\big) < -\tau\frac{\alpha^k}{\gamma}\nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k,$$

or equivalently,

$$\frac{f(\boldsymbol{x}^k) - f\big(\boldsymbol{x}^k + \breve{\alpha}^k \boldsymbol{d}^k\big)}{\breve{\alpha}^k} < -\tau \nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k, \tag{3.7}$$

for all $k \in \mathcal{K},\ k \geq k_2$ where $\breve{\alpha}^k = \alpha^k/\gamma$.

Now let $\tilde{k} = \max\{k_1, k_2\}$. Since $\{\boldsymbol{d}^k\}_{k\in\mathcal{K}}$ is bounded it can further be shown that there exists a subsequence $\{\boldsymbol{d}^k\}_{k\in\tilde{\mathcal{K}},\tilde{\mathcal{K}}\subset\mathcal{K}}$ of $\{\boldsymbol{d}^k\}_{k\in\mathcal{K}}$ such that $\{\boldsymbol{d}^k\}_{k\in\tilde{\mathcal{K}}} \to \tilde{\boldsymbol{d}}$ where $\tilde{\boldsymbol{d}}$ is some finite vector. On the other hand, by applying the mean value theorem to (3.7), it can be shown that there exists some $\{\tilde{\alpha}^k\}_{k\in\tilde{\mathcal{K}},\tilde{\mathcal{K}}\subset\mathcal{K}}$ such that

$$-\nabla f(\boldsymbol{x}^k + \tilde{\alpha}^k \boldsymbol{d}^k)^T \boldsymbol{d}^k < -\tau \nabla f(\boldsymbol{x}^k)^T \boldsymbol{d}^k,$$

where $\tilde{\alpha}^k \in [0, \bar{\alpha}^k],\ \forall k \in \tilde{\mathcal{K}},\ k \geq \tilde{k}$. By taking limits on both sides we obtain

$$-\nabla f(\tilde{\boldsymbol{x}})^T \tilde{\boldsymbol{d}} \leq -\tau \nabla f(\tilde{\boldsymbol{x}})^T \tilde{\boldsymbol{d}}, \quad \text{or} \quad 0 \leq (1-\tau)\nabla f(\tilde{\boldsymbol{x}})^T \tilde{\boldsymbol{d}}, \qquad \tau \in (0,1),$$

from which it follows that $\nabla f(\tilde{\boldsymbol{x}})^T \tilde{\boldsymbol{d}} \geq 0$, contradicting that $\{\boldsymbol{d}^k\}$ satisfies (3.4). □

4. Applications & Numerical Results. In this section we report numerical results on two fundamental and important examples of (1.1). Performance results are shown for different software fed with both synthetic and real-world data. The experiments reveal that for several datasets, our methods are competitive to well-established algorithms; in fact, as problem size increases our algorithms often outperform the competition significantly.

4.1. Non-negative Least Squares. The first specific problem to which we specialize our method is *nonnegative least-squares* (NNLS):

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad \tfrac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2, \quad \text{s.t.} \quad \boldsymbol{x} \geq 0, \tag{4.1}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. To fit (4.1) in our framework, we further assume that $\boldsymbol{A}^T\boldsymbol{A}$ is positive-definite whereby the objective function is strictly convex. We remark that (L-)BFGS ensures the positive definiteness of the gradient scaling matrix $\boldsymbol{S}^k$ hence Assumption 1 is always satisfied. Note that even though (4.1) is written

with only lower bounds, we can easily guarantee that the sequence of iterates $\{\boldsymbol{x}^k\}$ generated by our algorithm is bounded. To see how, we first show that the initial level set $L_0 = \{\boldsymbol{x} \,|\, \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \leq \|\boldsymbol{A}\boldsymbol{x}^0 - \boldsymbol{b}\|_2\}$ is bounded; then since each iteration enforces descent, the sequence $\{\boldsymbol{x}^k\}$ must be bounded as it also belongs to this level set.

To show that L_0 is bounded, consider for any $\boldsymbol{x} \in L_0$, the inequalities

$$\|\boldsymbol{A}\boldsymbol{x}\|_2 - \|\boldsymbol{b}\|_2 \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \quad \leq \quad \|\boldsymbol{A}\boldsymbol{x}^0 - \boldsymbol{b}\|_2.$$

These inequalities yield a bound on $\|\boldsymbol{x}\|_2$ (which implies L_0 is bounded), because

$$\begin{array}{rcccl} \sigma_{\min}(\boldsymbol{A}) \cdot \|\boldsymbol{x}\|_2 & \leq & \|\boldsymbol{A}\boldsymbol{x}\|_2 & \leq & \|\boldsymbol{A}\boldsymbol{x}^0 - \boldsymbol{b}\|_2 + \|\boldsymbol{b}\|_2, \\ \|\boldsymbol{x}\|_2 & \leq & \sigma_{\min}^{-1}(\boldsymbol{A}) \cdot (\|\boldsymbol{A}\boldsymbol{x}^0 - \boldsymbol{b}\|_2 + \|\boldsymbol{b}\|_2) & = & U < \infty, \end{array} \tag{4.2}$$

where $\sigma_{\min}(\boldsymbol{A}) > 0$ is the smallest singular value of $\boldsymbol{A}$.

NNLS is a classical problem in scientific computing [11]; applications include image restoration [17], vehicle dynamics control [25], and data fitting [14]. Given its wide applicability many specialized algorithms have been developed for NNLS, e.g., methods based on active sets such as the classic Lawson-Hanson method [11], or its efficient modification Fast-NNLS (FNNLS) [4]. Not surprisingly, some constrained optimization methods have also been applied to solve NNLS [3, 13]. It is interesting to note that for large scale problems these specialized algorithms are outperformed by modern methods such as TRON, LBFGS-B, or the methods of this paper.

There exist several different algorithms for solving the NNLS problem. Of these, some are designed expressly for NNLS, while others are for general differentiable convex minimization. Naturally, owing to the large variety of optimization methods and software available, we cannot hope to be exhaustive in our comparisons. We solved the NNLS problem using several different software, and then picked the ones that performed the best. Specifically, we show results on the following implementations:

- TRON – a trust region Newton-type method for bound constrained optimization [12]. The underlying implementation was in FORTRAN. For convenience we used the MTRON [19] MATLAB interface.
- LBFGS-B – a limited memory BFGS algorithm extended to handle bound constraints [5]. Again, the underlying implementation was in FORTRAN, and we invoked it via a MATLAB interface.
- PQN-BFGS – our projected quasi-Newton implementation with the full BFGS update; written entirely in MATLAB (i.e., no MEX or C-level interface is used),
- PQN-LBFGS – our projected L-BFGS implementation; written entirely in MATLAB.

We also compare our method with FNNLS, which is Bro and Jong's improved implementation of the Lawson-Hanson NNLS procedure [4]. FNNLS was obtained as a MATLAB implementation. We remark that the classic NNLS algorithm of Lawson and Hanson [11], which is still available as the function `lsqnonneg` in MATLAB, was far too slow for all our experiments (taking several hours to make progress when other methods were taking seconds), and hence we have not reported any results on it. For all the subsequent experiments, whenever possible, we use $\|\boldsymbol{g}(\boldsymbol{x})\|_\infty$ to decide when to terminate; here g_i is defined as

$$g_i(\boldsymbol{x}^k) = \begin{cases} \partial_i f(\boldsymbol{x}^k), & \text{if } i \notin I^k, \\ 0, & \text{otherwise.} \end{cases}$$

We also note that each method also includes a different set of stopping criteria; we leave the other criteria at their default setting.

We first show experiments on synthetic data. These experiments compare in a controlled setting our method against others. These experiments obviously do not reflect the complete situation, because the behavior of the different algorithms can vary substantially on real-world data. Nevertheless, results on synthetic data *do* provide a basic performance comparison that helps place our method relative to competing approaches. We use the following three data sets:

- P1: dense random (uniform) nonnegative matrices with varying sizes,
- P2: medium sized sparse random matrices, and
- P3: large sparse random matrices with varying sparsity.

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

TABLE 4.1
The size and condition number of the test matrices $\boldsymbol{A}$ in dense data set P1.

We generated the test matrices $\boldsymbol{A}$ and vectors $\boldsymbol{b}$ corresponding to these datasets using MATLAB's `rand` and `sprand` functions. Table 4.1 shows the number of rows and columns in $\boldsymbol{A}$ for dataset P1, along with the associated condition numbers $\kappa(\boldsymbol{A})$. The matrices for problem set P2 were of size 12000×6400, and for P3 they were of size 65536×50000. We varied the sparsity in both P2 and P3. Table 4.2 lists the matrices in these datasets. Note that by *sparsity* we mean the ratio of zero entries to the total number of entries in the given matrix.

Matrix	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6
Sparsity	0.996	0.994	0.992	0.99	0.988	0.900
Matrix	P3-1	P3-2	P3-3	P3-4	P3-5	P3-6
Sparsity	0.998	0.997	0.996	0.995	0.994	0.990

TABLE 4.2
The sparsity of the test matrices $\boldsymbol{A}$ in problem sets P2 and P3.

Table 4.3 shows the running times of TRON, FNNLS, PQN-BFGS, LBFGS-B, and PQN-LBFGS for the matrices in problem set P1. 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 one-time computations while reporting results for FNNLS. On some mid-sized matrices, e.g., 2800×2000 (P1-1) and 3600×2400 (P1-2), TRON and FNNLS remain relatively competitive to others. However, with increasing matrix sizes, the running time of these methods increases drastically while that of LBFGS-B and PQN-LBFGS changes only linearly. Consequently, for the largest matrix in the problem set (P1-5), PQN-LBFGS outperforms others significantly, as revealed by Table 4.3.

In Table 4.4 we show running time comparisons analogous to those in Table 4.3, except that the matrices used were sparse. As shown in the table, LBFGS-B and PQN-LBFGS generally outperform others for sparse matrices, and the difference becomes starker with decreasing sparsity. However, we also observe that sparsity benefits the TRON software, as it is claimed to take advantage of sparse problems [12].

[1] A suffix (F) denotes that the underlying implementation is in FORTRAN, while (M) denotes MATLAB. All reported results are averages over 10 runs to decrease variability.

Method[1]	P1-1	P1-2	P1-3	P1-4	P1-5
TRON(F)	55.60s	8.32s	142.93s	204.09s	290.77s
$f(x)$	4.45e+04	1.67e+04	5.56e+04	7.64e+04	1.02e+05
$\|\boldsymbol{g}(x)\|_\infty$	1.17e-05	2.27e-04	1.69e-04	2.80e-05	4.85e-05
FNNLS(M)	9.56s	16.93s	26.89s	38.94s	60.22s
$f(x)$	4.45e+04	1.67e+04	5.46e+04	7.64e+04	1.02e+05
$\|\boldsymbol{g}(x)\|_\infty$	**2.64e-14**	**5.63e-15**	**2.64e-14**	**1.50e-14**	**2.01e-14**
PQN-BFGS(M)	71.92s	18.31s	158.56s	120.17s	112.12s
$f(x)$	4.45e+04	1.67e+04	5.56e+04	7.64e+04	1.02e+05
$\|\boldsymbol{g}(x)\|_\infty$	*0.0018*	8.49e-04	*0.0022*	*0.0055*	*0.0011*
LBFGS-B(F)	17.78s	**3.71s**	33.91s	26.41s	27.39s
$f(x)$	4.45e+04	1.67e+04	5.56e+04	7.64e+04	1.02e+05
$\|\boldsymbol{g}(x)\|_\infty$	9.76e-04	9.95e-04	9.28e-04	9.23e-04	9.91e-04
PQN-LBFGS(M)	**8.77s**	5.06s	**9.24s**	**18.05s**	**17.75s**
$f(x)$	4.45e+04	1.67e+04	5.46e+04	7.64e+04	1.02e+05
$\|\boldsymbol{g}(x)\|_\infty$	7.40e-04	7.28e-04	8.38e-04	3.72e-04	5.98e-04

TABLE 4.3

NNLS experiments on dataset P1. For all algorithms (except FNNLS), we used $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-3}$ as the main stopping criterion. In addition, we stop each method after 10,000 iterations or 20,000 seconds. Each method retains its own default stopping criteria, e.g., PQN-BFGS, LBFGS-B, and PQN-LBFGS terminate iterations when their internal line-search fails. Finally we ran FNNLS with no custom parameters as it does not support the stopping criterion $\|\boldsymbol{g}(x)\|_\infty$. The best values are shown in bold, and the italicized entries represent abnormal terminations.

Method[1]	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6
TRON(F)	153.31s	2.06e+03s	2.68e+03s	3.58e+03s	4.66e+03s	1.68e+04
$f(\boldsymbol{x})$	1.11e+09	1.45e+09	1.51e+08	2.06e+09	1.30e+10	1.56e+12
$\|\boldsymbol{g}(x)\|_\infty$	*0.077*	*0.050*	*0.174*	**2.75e-04**	0.0053	*0.371*
PQN-BFGS(M)	48.93s	105.61s	144.27s	186.95s	202.21s	601.78s
$f(\boldsymbol{x})$	1.11e+09	1.45e+09	1.51e+08	2.06e+09	1.30e+10	1.56e+12
$\|\boldsymbol{g}(x)\|_\infty$	0.0092	**0.0079**	0.0092	0.0088	0.0084	**0.0098**
LBFGS-B(F)	0.55s	0.85s	1.35s	1.89s	2.53s	43.50s
$f(\boldsymbol{x})$	1.11e+09	1.45e+09	1.51e+08	2.06e+09	1.30e+10	1.56e+12
$\|\boldsymbol{g}(x)\|_\infty$	0.0086	0.0088	0.0097	0.0088	0.0091	*1.15*
PQN-LBFGS(M)	**0.22s**	**0.37s**	**0.49s**	**0.75s**	**0.84s**	**18.34s**
$f(\boldsymbol{x})$	1.11e+09	1.45e+09	1.51e+08	2.06e+09	1.30e+10	1.56e+12
$\|\boldsymbol{g}(x)\|_\infty$	**0.0087**	0.0099	**0.0083**	0.0085	**0.0046**	*0.88*

TABLE 4.4

NNLS experiments on dataset P2. For all algorithms, we used $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-2}$ as the main stopping criterion, and as for P1, other method-specific stopping criteria are set to default. FNNLS does not scale on this dataset.

Method[1]	P3-1	P3-2	P3-3	P3-4	P3-5	P3-6
LBFGS-B(F)	0.27s	0.47s	0.48s	0.74s	0.94s	2.01s
$f(\boldsymbol{x})$	5.24e+07	3.46e+07	1.82e+09	1.18e+09	3.60e+09	1.13e+09
$\|\boldsymbol{g}(x)\|_\infty$	0.010	*0.014*	0.0057	*0.012*	0.0099	0.0095
PQN-LBFGS(M)	**0.077s**	**0.13s**	**0.18s**	**0.26s**	**0.38s**	**0.68s**
$f(\boldsymbol{x})$	5.24e+07	3.46e+07	1.82e+09	1.18e+09	3.60e+09	1.13e+09
$\|\boldsymbol{g}(x)\|_\infty$	**0.008**	**0.0032**	**0.0051**	**0.0073**	**0.0054**	**0.008**

TABLE 4.5

NNLS experiments on dataset P3. The main stopping criterion is $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-2}$ for both algorithms, other settings are the same as the previous experiments. FNNLS,TRON, and PQN-BFGS do not scale on this dataset.

Our next set of experiments is on dataset P3, and the result is shown in Table 4.5. Note that only LBFGS-B and PQN-LBFGS scale to this problem set. Although both methods perform well across all the problems, PQN-LBFGS is seen to have an edge

Matrix	ash958	well1850	bcspwr10	e40r5000	conf6.0-8000
Size	958×292	1850×712	5300×5300	17281×17281	49152×49152
Sparsity	0.9932	0.9934	0.9992	0.9981	0.9992

TABLE 4.6
The size and sparsity of some real-world data sets from the MatrixMarket.

Method[1]	ash958	well1850	bcspwr10	e40r5000	conf6.0-8000
TRON(F)	0.0065s	**0.011s**	0.32s	1.71e+03s	21.70s
$f(\boldsymbol{x})$	6.01e+03	4.15e-07	3.66e-07	**1.26e-06**	3.22e+04
$\|\boldsymbol{g}(x)\|_\infty$	9.77e-15	3.33e-10	5.10e-04	5.86e-04	**1.08e-06**
FNNLS(M)	0.013s	0.088s	7.03s	-	-
$f(\boldsymbol{x})$	6.01e+03	**2.50e-30**	**6.99e-29**	-	-
$\|\boldsymbol{g}(x)\|_\infty$	7.99e-15	5.62e-16	3.89e-15	-	-
PQN-BFGS(M)	0.007s	0.71s	240.84s	1.67e+04s	-
$f(\boldsymbol{x})$	6.01e+03	1.27e-07	2.23e-07	0.0066	-
$\|\boldsymbol{g}(x)\|_\infty$	5.51e-05	6.80e-05	8.92e-05	*0.044*	-
LBFGS-B(F)	**0.0044s**	0.064s	1.79s	17.36s	3.82s
$f(\boldsymbol{x})$	6.01e+03	7.60e-07	5.72e-07	0.020	3.22e+04
$\|\boldsymbol{g}(x)\|_\infty$	8.91e-05	9.16e-05	9.99e-05	*0.027*	7.96e-05
PQN-LBFGS(M)	0.0097s	0.029s	**0.26s**	**15.40s**	**1.90s**
$f(\boldsymbol{x})$	6.01e+03	6.85e-07	1.48e-07	0.0091	3.22e+04
$\|\boldsymbol{g}(x)\|_\infty$	3.37e-05	7.31e-05	9.61e-05	*0.009*	2.68e-05

TABLE 4.7
NNLS experiments on real-world datasets. For all algorithms, we used $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-4}$ as the main stopping criterion with other method-specific default stopping criteria. FNNLS does not scale on e40r5000 and conf6.0-8000, PQN-BFGS also fails to scale on conf6.0-8000.

on LBFGS-B as it runs almost twice as fast while attaining similar objective function values, and better satisfaction of convergence tolerance.

Finally, we experiment with five datasets drawn from real-world applications (see Table 4.6). The datasets were obtained from the MatrixMarket[2], and they arise from problems solving least squares (ash958, well1850) and linear systems (bcspwr10, conf6.0-8000). We impose non-negativity on the solutions to obtain NNLS problems with these matrices.

Table 4.7 shows results on these real-world datasets; we observe somewhat different behaviors of the various methods. For example, TRON is competitive overall, except on *conf6.0-8000*, where it is overwhelmed by LBFGS-B and PQN-LBFGS. PQN-LBFGS resembles (as expected) characteristics of LBFGS-B, while consistently delivering competitive results across all ranges.

4.2. KL-Divergence Minimization. Our next example of (1.1) is *nonnegative Kullback-Leibler divergence* minimization (NNKL):

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}) = \sum_i b_i \log \frac{b_i}{[\boldsymbol{A}\boldsymbol{x}]_i} - b_i + [\boldsymbol{A}\boldsymbol{x}]_i, \quad \text{s.t.} \quad \boldsymbol{x} \geq 0, \tag{4.3}$$

where $\boldsymbol{A} \in \mathbb{R}_+^{m\times n}$ $(m \leq n)$ is a full-rank matrix and $\boldsymbol{b} \in \mathbb{R}_+^m$. We note here a minor technical point: to guarantee that the Hessian has bounded eigenvalues, we actually need to ensure that at iteration k of the algorithm, $[\boldsymbol{A}\boldsymbol{x}^k]_i \geq \delta > 0$. This bound can be achieved for example by modifying the constraint to be $\boldsymbol{x} \geq \epsilon > 0$. However, in practice one can usually run the algorithm with the formulation (4.3) itself. Since

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

$\boldsymbol{A}$ is of full-rank, simple calculations show that $\lim_{\|\boldsymbol{x}\|\to\infty} f(\boldsymbol{x})/\|\boldsymbol{x}\| > 0$. Therefore, $f(\boldsymbol{x})$ has bounded sublevel sets, and (4.3) has a solution [22, Theorem 1.9]. Thus, in particular, if we select $\boldsymbol{x}^0 = \mathbf{1}$, the all-ones vector, we know that the initial sublevel set $L_0 = \{\boldsymbol{x} : f(\boldsymbol{x}) \leq f(\boldsymbol{x}^0)\}$ is bounded, ensuring that the sequence of iterates $\{\boldsymbol{x}^k\}$ generated by our algorithm is bounded as it performs descent on each iteration.

NNKL arises as a core problem in several applications, e.g., positron emission tomography [7], astronomical image restoration [26], and signal processing [24]. In contrast to (4.1), the most popular algorithms for (4.3) have been based on the Expectation Maximization (EM) framework [26]. Two of the most popular EM algorithms for NNKL have been maximum-likelihood EM and ordered-subsets EM [9], where the latter and its variants are pervasive in medical imaging. Several other related algorithms for NNKL are summarized in [7, 21]. It turns out that for NNKL too, optimization methods like TRON, LBFGS-B, or our methods frequently (though not always) perform better than the essentially gradient based EM methods.

We report below results for the NNKL problem (4.3). Our first set of results are again with synthetic data. For these experiments we simply reuse the random matrices described in Section 4.1. We do not report the results of TRON as it takes an inordinately long time in comparison to the other methods.

The empirical results shown in Tables 4.8, 4.9 and 4.10 are roughly in agreement with the behavior observed for NNLS, but with a few notable exceptions. For example, PQN-BFGS exhibits its competence against other methods for the small problem P1-1. This competence can be accounted for by observing the Hessian of the KL-divergence function. Unlike NNLS problems where the Hessian is a constant, a method like TRON is constantly required to compute a varying Hessian across iterations, whereas PQN-BFGS updates its Hessian approximation for both NNLS and NNKL at virtually the same cost. This efficiency combined with a more accurate Hessian results in a greater convergence speed of PQN-BFGS as compared to the other methods. However, the size of the given problem still plays a critical role and an increased scale even makes some of the methods infeasible to test.

Method[1]	P1-1	P1-2	P1-3	P1-4	P1-5
PQN-BFGS(M)	**29.2s**	51.9s	67.4s	77.3s	84.4s
$f(\boldsymbol{x})$	5.47e-08	5.87e-08	1.75e-08	2.87e-08	1.59e-08
$\|\boldsymbol{g}(x)\|_\infty$	7.83e-06	6.82e-06	9.76e-06	7.69e-06	8.85e-06
LBFGS-B(F)	30.63s	**15.57s**	62.67s	72.61s	78.54s
$f(\boldsymbol{x})$	**1.50e-09**	**6.48e-11**	**6.95e-11**	**2.06e-08**	**7.62e-12**
$\|\boldsymbol{g}(x)\|_\infty$	6.14e-06	3.12e-07	2.95e-07	*1.17e-05*	1.58e-07
PQN-LBFGS(M)	39.90s	35.46s	**42.54s**	**62.58s**	**72.36s**
$f(\boldsymbol{x})$	2.73e-08	3.97e-08	4.07e-08	2.17e-08	1.85e-08
$\|\boldsymbol{g}(x)\|_\infty$	7.79e-06	8.59e-06	9.85e-06	6.36e-06	7.03e-06

TABLE 4.8

NNKL experiments on dataset P1. For all algorithms, we used $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-5}$ as the main stopping criterion.

4.3. Image Deblurring examples. In Figure 4.1 we show examples of image deblurring, where the images shown were blurred using various blur kernels, and deblurred using LBFGS-B and PQN-LBFGS. The latter usually outperforms LBFGS-B, and we show some examples where the performance difference is significant. Objective function values are also shown to permit a quantitative comparison.

We compare the progress of the algorithms with respect to time by running both of them for the *same amount* of time. We illustrate the comparison by plotting the

Method[1]	P2-1	P2-2	P2-3	P2-4	P2-5	P2-6
PQN-BFGS(M)	500.36s	500.53s	62.53s	69.78s	71.85s	501.24s
$f(\boldsymbol{x})$	4.11e-11	1.48e-10	4.87e-12	3.95e-12	3.03e-12	1.04e-11
$\|\boldsymbol{g}(x)\|_\infty$	*2.93e-07*	*4.36e-07*	9.42e-08	8.94e-08	8.85e-08	*3.26e-07*
LBFGS-B(F)	**2.23s**	4.01s	**3.41s**	7.09s	5.55s	**61.86s**
$f(\boldsymbol{x})$	**6.44e-13**	1.16e-12	1.17e-12	**6.26e-13**	**2.24e-12**	6.54e-12
$\|\boldsymbol{g}(x)\|_\infty$	*1.91e-07*	*1.08e-07*	*2.24e-07*	4.00e-08	7.37e-08	*1.90e-07*
PQN-LBFGS(M)	10.40s	**3.62s**	5.86s	**6.54s**	**4.25s**	135.9s
$f(\boldsymbol{x})$	1.13e-12	**8.10e-13**	**6.18e-13**	9.55e-13	3.35e-12	**6.14e-13**
$\|\boldsymbol{g}(x)\|_\infty$	5.87e-08	8.26e-08	4.91e-08	*1.06e-07*	9.49e-08	5.42e-08

TABLE 4.9

NNKL experiments on dataset P2. For all algorithms, we used $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-7}$ *or a maximum running time of 500 seconds as the stopping criterion.*

Method[1]	P3-1	P3-2	P3-3	P3-4	P3-5	P3-6
LBFGS-B(F)	148.15s	466.92s	389.76s	490.99s	837.23s	2354.23s
$f(\boldsymbol{x})$	**1.19e-11**	**3.69e-12**	**4.60e-11**	**3.58e-12**	**3.64e-11**	**5.43e-11**
$\|\boldsymbol{g}(x)\|_\infty$	7.86e-08	*1.22e-07*	*1.82e-07*	*1.20e-07*	*2.82e-07*	*1.70e-07*
PQN-LBFGS(M)	**128.54s**	**297.40s**	**242.67s**	**364.82s**	**491.50s**	**1169.36s**
$f(\boldsymbol{x})$	3.21e-11	2.78e-11	2.64e-11	3.39e-11	4.51e-11	6.85e-11
$\|\boldsymbol{g}(x)\|_\infty$	9.32e-08	9.50e-08	9.99e-08	9.83e-08	7.60e-08	*2.40e-07*

TABLE 4.10

NNKL experiments on dataset P3. For both algorithms, we used $\|\boldsymbol{g}(x)\|_\infty \leq 10^{-7}$ *as the main stopping criterion.*

objective function value against the number of iterations. Naturally, the number of iterations differs for both algorithms; but we adjust this number so that both PQN-LBFGS and LBFGS-B run for approximately the same time. For Figures 4.2(a) and 4.2(b), an iteration of PQN-LBFGS took roughly the same time as an iteration of LBFGS-B, while for Figures 4.2(c) and 4.2(d), LBFGS-B could run fewer iterations for the same time (or equivalently, needed more time to run the same number of iterations). Note that we did not run both algorithms to completion, as that usually led to overfitting of noise and visually less appealing results.

The moon image was obtained from [10]; the cell image is taken from the software of [16]; the brain image is from the PET Sorteo database [20], and the image of Haumea is a rendition obtained from Wikipedia.

5. Discussion and Future work. In this paper we have proposed an algorithmic framework for solving box-constrained convex optimization problems using quasi-Newton methods. We presented two implementations, namely PQN-BFGS and PQN-LBFGS, based on the BFGS and limited-memory BFGS updates, respectively. Our methods are simple like the popular projected gradient method, and they overcome deficiencies such as slow convergence. We reported empirical results of our method applied to the NNLS and the NNKL problems, showing that our MATLAB software delivers uncompromising performance both in terms of running time and accuracy, especially for larger scale problems.

We emphasize that the simplicity and ease of implementation of our methods are two strong points that favor their wide applicability. It remains, however, a subject of future research to make further algorithmic improvements.

Acknowledgments. We acknowledge support of NSF grants CCF-0431257 and CCF-0728879. SS also acknowledges support of a Max-Planck research grant. We thank James Nagy for providing us with his MATLAB code, from which we extracted

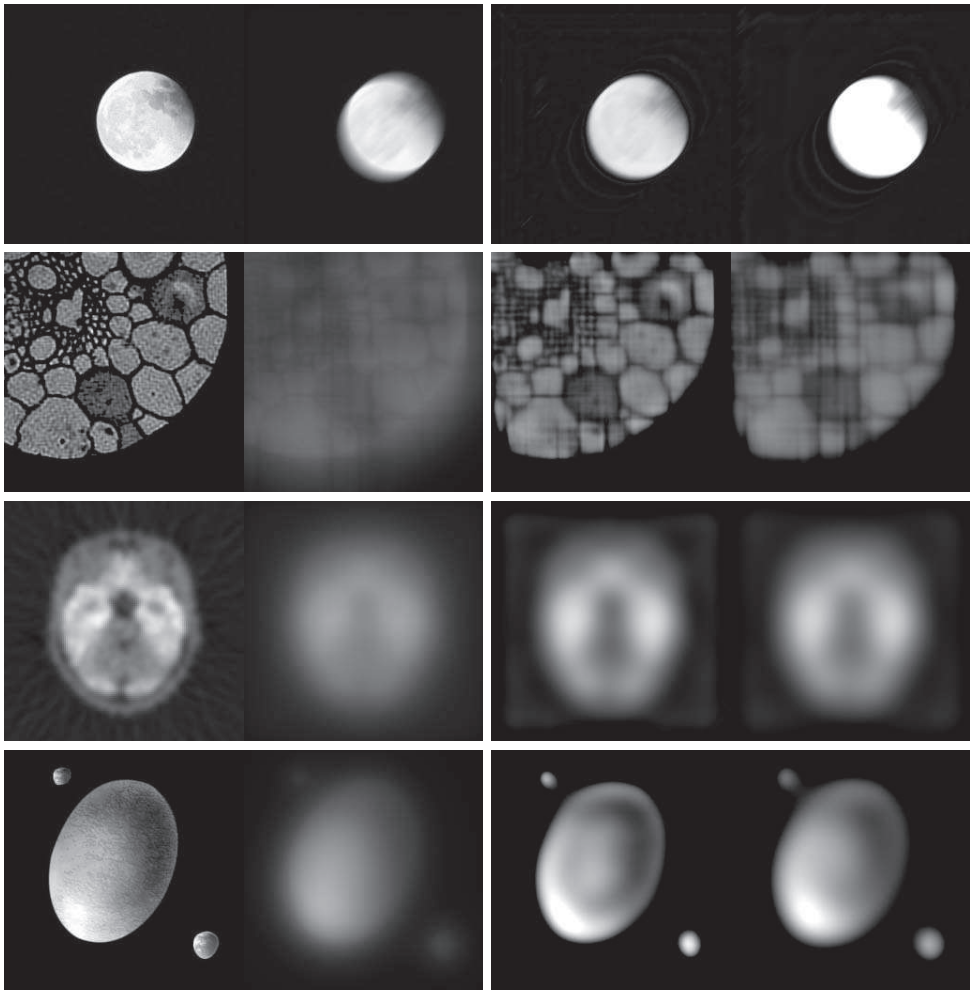

FIG. 4.1. *Deblurring example using PQN-LBFGS and LBFGS-B, from top to bottom, a blurred image of the moon, a cell, a brain, and Haumea (a dwarf planet).*

some useful subroutines.

References.

[1] DIMITRI P. BERTSEKAS, *Projected Newton Methods for Optimization Problems with Simple Constraints*, SIAM Jounal on Control and Optimization, 20 (1982), pp. 221–246.

[2] DIMITRI. P. BERTSEKAS, *Nonlinear Programming*, Athena Scientific, second ed., 1999.

[3] 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 (1991), pp. 111–143.

[4] RASMUS BRO AND SIJMEN DE JONG, *A Fast Non-negativity-constrained Least Squares Algorithm*, Journal of Chemometrics, 11 (1997), pp. 393–401.

[5] R. BYRD, P. LU, J. NOCEDAL, AND C. ZHU, *A Limited Memory Algorithm*

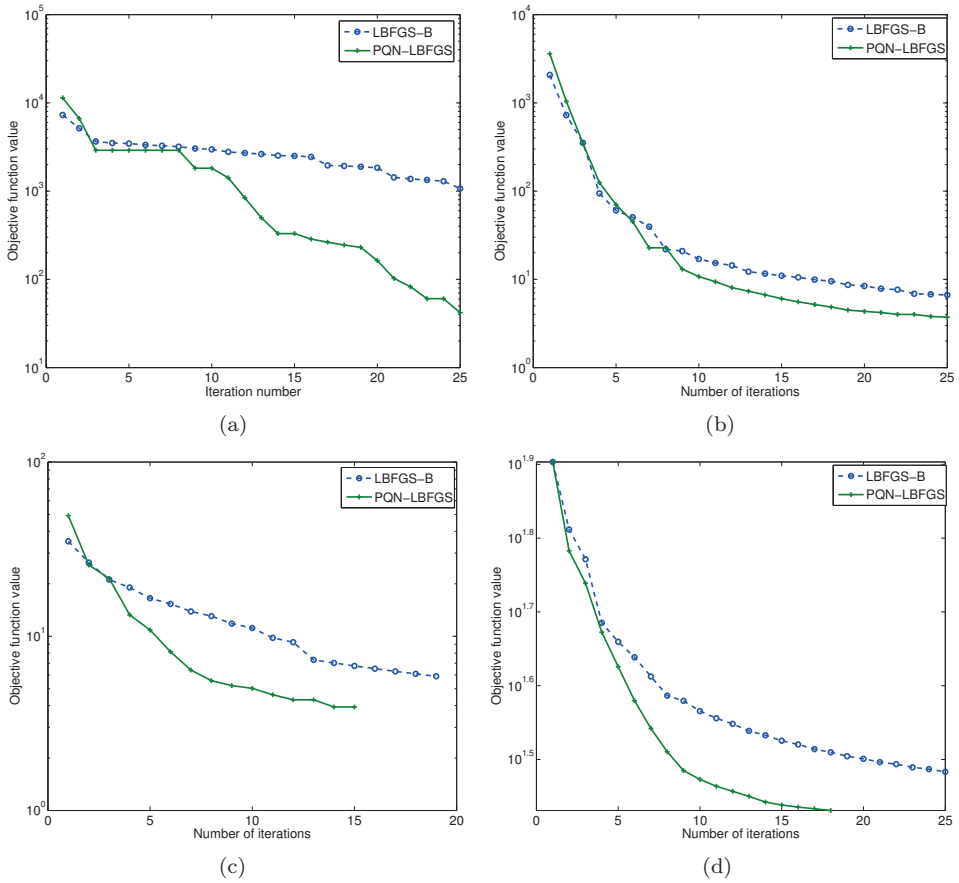

FIG. 4.2. *Comparison of* PQN-LBFGS *against* LBFGS-B *on the examples: (a) the moon, (b) a cell, (c) a brain and (d) Haumea. The objective function values (y-axes) are plotted with respect to the number of iterations (x-axes).*

for Bound Constrained Optimization, SIAM Journal on Scientific Computing, 16 (1995), pp. 1190–1208.

[6] J.C. DUNN, *Global and Asymptotic Convergence Rate Estimates for a Class of Projected Gradient Processes*, SIAM Jounal on Control and Optimization, 19 (1981), pp. 368–400.

[7] J. FESSLER, *Image reconstruction: Algorithms and Analysis*, 2008. Book preprint.

[8] PHILIP E. GILL, WALTER MURRAY, AND MARGARET H. WRIGHT, *Practical Optimization*, Academic Press, 1981.

[9] H. M. HUDSON AND R. S. LARKIN, *Accelerated image reconstruction using ordered subsets of projection data*, IEEE Tran. Med. Imag., 13 (1994), pp. 601–609.

[10] B. KATZUNG. http://www.astronomy-images.com, 2003.

[11] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice–Hall, 1974.

[12] CHIH-JEN LIN AND JORGE J. MORÉ, *Newton's Method for Large Bound-Constrained Optimization Problems*, SIAM Journal on Optimization, 9 (1999),

pp. 1100–1127.
[13] M. Merritt and Y. Zhang, *Interior-point gradient method for large-scale totally nonnegative least squares problems*, Journal of Optimization Theory and Applications, 126 (2005), pp. 191–202.
[14] Nicolas Molinari, Jean-Franqois Durand, and Robert Sabatier, *Bounded Optimal Knots for Regression Splines*, Computational Statistics and Data Analysis, 45 (2004), pp. 159–178.
[15] Jorge J. Moré and D. C. Sorensen, *Computing a Trust Region Step*, SIAM Journal on Scientific and Statistical Computing, 3 (1983), pp. 553–572.
[16] J. Nagy, 2008. Personal Communication.
[17] J. Nagy and Z. Strakos, *Enforcing nonnegativity in image reconstruction algorithms*, Mathematical Modeling, Estimation, and Imaging, 4121 (2000), pp. 182–190.
[18] J. Nocedal, *Updating Quasi-Newton Matrices with Limited Storage*, Mathematics of Computation, 95 (1980), pp. 339–353.
[19] C. Ortner, *MTRON: Matlab wrapper for TRON*. Mathworks file exchange, 2007.
[20] A. Reilhac, *PET-SORTEO*. http://sorteo.cermep.fr/home.php, 2008.
[21] R. M. Rewitt and S. Matej, *Overview of methods for image reconstruction from projections in emission computed tomography*, Proc. IEEE, 91 (2003), pp. 1588–1611.
[22] R. T. Rockafellar and R. J.-B. Wets, *Variational Analysis*, Springer, 1997.
[23] J.B. Rosen, *The Gradient Projection Method for Nonlinear Programming. Part I. Linear Constraints*, Journal of the Society for Industrial and Applied Mathematics, 8 (1960), pp. 181–217.
[24] Lawrence K. Saul, Fei Sha, and Daniel D. Lee, *Statistical signal processing with nonnegativity constraints*, in Proceedings of EuroSpeech 2003, vol. 2, 2003, pp. 1001–1004.
[25] Brad Schofield, *Vehicle Dynamics Control for Rollover Prevention*, PhD thesis, Lund University, 2006.
[26] Y. Vardi and D. Lee, *From Image Deblurring to Optimal Investment Portfolios: Maximum Likelihood Solutions for Positive Linear Problems*, Journal of the Royal Statistical Society:Series B, 55 (1993), pp. 569–612.