A Coordinate Gradient Descent Method for ℓ_1-regularized Convex Minimization

Sangwoon Yun [*], and Kim-Chuan Toh [†]

January 30, 2009

Abstract

In applications such as signal processing and statistics, many problems involve finding sparse solutions to under-determined linear systems of equations. These problems can be formulated as a structured nonsmooth optimization problems, i.e., the problem of minimizing ℓ_1-regularized linear least squares problems. In this paper, we propose a block coordinate gradient descent method (abbreviated as CGD) to solve the more general ℓ_1-regularized convex minimization problems, i.e., the problem of minimizing an ℓ_1-regularized convex smooth function. We establish a Q-linear convergence rate for our method when the coordinate block is chosen by a Gauss-Southwell-type rule to ensure sufficient descent. We propose efficient implementations of the CGD method and report numerical results for solving large-scale ℓ_1-regularized linear least squares problems arising in compressed sensing and image deconvolution as well as large-scale ℓ_1-regularized logistic regression problems for feature selection in data classification. Comparison with several state-of-the-art algorithms specifically designed for solving large-scale ℓ_1-regularized linear least squares or logistic regression problems suggests that an efficiently implemented CGD method may outperform these algorithms despite the fact that the CGD method is not specifically designed just to solve these special classes of problems.

Key words. Coordinate gradient descent, Q-linear convergence, ℓ_1-regularization, compressed sensing, image deconvolution, linear least squares, logistic regression, convex optimization

[*]Singapore-MIT Alliance, 4 Engineering Drive 3, Singapore 117576 (smaysw@nus.edu.sg).

[†]Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543 (mattohkc@nus.edu.sg); and Singapore-MIT Alliance, 4 Engineering Drive 3, Singapore 117576. Research supported in part by NUS Academic Research Grant R146-000-076-112.

1 Introduction

Recent interests in signal/image denoising [7, 15, 37] and data mining/classification [26, 27, 30, 34, 40] have focused on the following minimization problem with ℓ_1-regularization:

$$\min_{x\in\Re^n} \; F_\rho(x) \stackrel{\rm def}{=} f(x) + \rho^T|x| \tag{1}$$

where $f : \Re^n \to \Re$ is smooth (i.e., continuously differentiable) and convex, but not necessarily strictly convex, ρ is a given nonnegative weight vector, and $|x|$ denotes obtained from x by taking absolute values. The weighted ℓ_1 term has the desirable property of inducing sparsity in the solution, i.e., few nonzero components. In particular, a special case of (1), with $\rho = \mu e$ where μ is a positive constant and e is a vector of ones, that has attracted much interest in statistical and signal processing contexts is the following ℓ_1-regularized under-determined linear least squares problem:

$$\min_{x\in\Re^n} \; \frac{1}{2}\|Ax - b\|^2 + \mu\|x\|_1, \tag{2}$$

where $A \in \Re^{m\times n}$ is dense, $m < n$ or even $m \ll n$, and $b \in \Re^m$. In applications of interest, the problems are often large and A may not be explicitly given as a matrix but as a linear map.

Several methods have been proposed to solve the ℓ_1-regularized linear least squares problem (2). Homotopy methods that find the full path of solutions, for all nonnegative values of the scalar parameters, have been proposed in [10, 11, 29]. Homotopy methods start with the zero vector and an initial parameter $\mu_0 \geq \|A^T b\|_\infty$ and successively build a sparse solution by adding or removing elements from its active set (the set of indices of zero components). Hence, if the solution of (2) is very sparse, these methods can be very fast even for large-scale problems. Some methods that require only matrix-vector multiplications involving A and A^T have also been proposed in [7, 14, 18, 19, 41]. Chen, Donoho and Saunders [7] proposed a primal-dual interior point method, for which a preconditioned conjugate gradient (PCG) method is used to solve the linear equations at each iteration, requiring only the multiplication by A and A^T. Recently, Kim et al. [19] proposed another interior-point (called l1-ls) method, that uses a different PCG method for computing the search direction at each iteration, to solve large-scale problems. The code is available from `http://www.stanford.edu/~boyd/l1_ls/`. Figueiredo, Nowak and Wright [14] proposed a gradient projection (called GPSR) method for solving the bound constrained quadratic programming reformulation of (2). In order to accelerate convergence, a technique based on Barzilai-Borwein (BB) steps is used. The code is available from `http://www.lx.it.pt/~mtf/GPSR/`. Hale, Yin and Zhang [18] proposed a fixed-point continuation (FPC) method, which is based on operator splitting, to solve the more general problem (1). They established a Q-linear rate of convergence of the method without assuming strict convexity nor uniqueness of solution. The code is available from `http://www.caam.rice.edu/~optimization/L1/fpc/`. During the writing of this paper,

Wright, Nowak and Figueiredo [41] proposed an iterative (called SpaRSA) method, in which a sequence of iterates x^k is generated by solving a subproblem of the following form:

$$x^{k+1} = \arg\min_{y \in \Re^n} \nabla f(x^k)^T (y - x^k) + \frac{\eta^k}{2} \|y - x^k\|^2 + \mu \|y\|_1, \tag{3}$$

where $\eta^k > 0$ is a given parameter, to solve the problem (1). The SpaRSA method was originally proposed for solving a more general problem of (1), for which separable (but not necessarily smooth nor convex) regularization term is used instead of ℓ_1 norm. But the theoretical convergence properties have not been analyzed. This method and the FPC method are closely related to the *iterative shrinkage/thresholding* (IST) methods [8, 12, 13]. The FPC method and the IST methods use the same form of subproblem (3) to generate a sequence of iterates x^k with a more conservative choice of η^k where η^k is fixed at a constant value. Convergence of IST methods was shown in [8].

Besides the ℓ_1-regularized linear least squares problem (2), another important special case of (1), with $\rho = \mu\bar{e}$ where μ is a positive constant and $\bar{e} = (1, \cdots, 1, 0)$, is the ℓ_1-regularized logistic regression problem, which has been proposed as a promising method for feature selection in classification problems. The ℓ_1-regularized logistic regression problem has the following form:

$$\min_{w \in \Re^{n-1}, v \in \Re} \quad \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-(w^T a_i + v b_i))) + \mu \|w\|_1, \tag{4}$$

where $a_i = b_i z_i$ and $(z_i, b_i) \in \Re^{n-1} \times \{-1, 1\}$, $i = 1, ..., m$ are a given set of (observed or training) examples. Recently, several methods have been developed for solving (4). Lee et al. [21] proposed a method that requires the solution of a bound constrained convex quadratic programming (QP) at each iteration. The bound constrained convex QP arises because the objective function is approximated by the second order Taylor expansion at the current point, while the ℓ_1 penalty term is reformulated as linear inequality constraints. Genkin, Lewis and Madigan [16] proposed a method that is based on a (cyclic) coordinate descent method for solving a ridge logistic regression problem and (4). Koh, Kim and Boyd [20] proposed an inexact interior-point (called l1-logreg) method, which uses a PCG method for computing the search direction at each iteration, for solving (4). The code is available from `http://www.stanford.edu/~boyd/l1_logleg/`. The SpaRSA method that is mentioned above can be applied to solve the problem (4).

Recently, Tseng and Yun [39] proposed a block coordinate gradient descent (CGD) method for minimizing the sum of a smooth function and a separable convex function. This method was shown to have a local linear rate convergence under a local Lipschitzian error bound assumption. Numerical results in [39] show the practical efficiency of the CGD method when it is applied to solve small and medium scale ℓ_1-regularized nonlinear least squares problems. The problem (1) is a special case of the problem that is considered in [39]. In this method, a descent direction is computed from the subproblem (3) but with the term $\eta^k \|y - x^k\|^2$ replaced by $(y - x^k)^T H^k (y - x^k)$ where $H^k \succ 0_n$ and $y_j = x_j^k$ for a

certain selected subset of $\{1, ..., n\}$; see Section 2. Hence the CGD method is more general but closely related to the IST, FPC and SpaRSA method when it is applied to solve (1).

In this paper, our main contributions are three-fold. First, we designed and implemented an efficient and robust method (CGD method), which can be viewed as a hybrid of the block coordinate descent with FPC/IST, for solving the convex ℓ_1-minimization problems (1). In particular, we demonstrate its effectiveness on large-scale problems (2) arising from applications such as compressed sensing [4, 5, 9, 37, 38] and image deconvolution [12, 13, 36], and (4) arising from data classification. At each iteration of this method, we replace f in F_ρ by a strictly convex quadratic approximation and apply block coordinate descent to generate a feasible descent direction. Then we perform an inexact line search along this direction and re-iterate. This method is simple, highly parallelizable, and is suited for solving large-scale problems. We choose the coordinate block according to the Gauss-Southwell (GS) type rules; see (12) and (13). We choose the stepsize according to an Armijo type rule for the general problem (1) and (4); see (8) and Section 5. For the ℓ_1-regularized linear least squares problem (2), we choose the stepsize according to a minimization rule; see (10) and Section 4. This proposed method is efficient for (2) especially when A is explicitly stored; see Subsection 4.2. Second, we prove a new Q-linear rate convergence of this method using Gauss-southwell type rules for the convex ℓ_1-minimization (1). Roughly speaking, if f is a convex function with Lipschitz continuous gradient and positive semidefinite Hessian on the level set associated with the initial point of the iterates, then under a bounded scaled Hessian assumption, we strengthen the convergence rate of the iterates from R-linear to Q-linear for the CGD method with GS-q-rule when it is applied to (1); see Subsection 3.2. We also establish the Q-linear rate convergence of the CGD method with GS-r-rule when it is applied to (1). Third, an efficient and robust implementation of the CGD method may outperform algorithms specifically designed just to solve large-scale ℓ_1-regularized linear least squares problems arising in compressed sensing and image deconvolution as well as large-scale ℓ_1-regularized logistic regression problems for feature selection in data classification.

In Section 2, we briefly describe the block coordinate gradient descent method. We also describe the rules, that ensure the convergence, of choosing the coordinate block. In Section 3, we summarize the known results on the global convergence and the local linear rate convergence of the CGD method using the Gauss-southwell type rules to choose the coordinate block and present a new Q-linear rate convergence of this method. In Section 4, we describe an efficient and robust implementation of the CGD method for solving (2) and report our numerical experience when A are Gaussian matrices or partial discrete cosine transform (DCT) matrices for compressed sensing experiments, and $A = RW$, where R is a matrix representation of the blur operation and W represents the inverse orthogonal wavelet transform with Haar wavelets, for image deconvolution experiments. For compressed sensing experiments, we compare the CGD method with l1-ls [19], GPSR [14] and FPC [18]. For image deconvolution experiments, we compare the CGD method with IST [13], GPSR and FPC. Our comparison suggests that the CGD method can be effective in practice. In Section 5, we describe an efficient implementation of the CGD method for solving (4) and report

our numerical results. We compare the CGD method with l1-logreg [20] and SpaRSA [41]. Our comparison suggests that the CGD method can be effective in practice for large-scale problems. We discuss conclusions and extensions in Section 6.

In our notation, $\Re^n$ denotes the space of n-dimensional real column vectors, ^{T} denotes transpose. For any $x \in \Re^n$, x_j denotes the jth component of x, $x_{\mathcal{J}}$ denotes the subvector of x comprising x_j, $j \in \mathcal{J}$, and $\|x\|_p = \left(\sum_{j=1}^n |x_j|^p\right)^{1/p}$ for $1 \le p < \infty$ and $\|x\|_\infty = \max_j |x_j|$. For simplicity, we write $\|x\| = \|x\|_2$. For any nonempty $\mathcal{J} \subseteq \mathcal{N} \stackrel{\text{def}}{=} \{1, \ldots, n\}$, $|\mathcal{J}|$ denotes the cardinality of $\mathcal{J}$. For any symmetric matrices $H, D \in \Re^{n\times n}$, we write $H \succeq D$ (respectively, $H \succ D$) to mean that $H - D$ is positive semidefinite (respectively, positive definite). We use $H_{\mathcal{J}\bar{\mathcal{J}}} = [H_{ij}]_{(i\in\mathcal{J},\ j\in\bar{\mathcal{J}})}$ to denote the submatrix of H indexed by $\mathcal{J}$ and $\bar{\mathcal{J}}$. The minimum and maximum eigenvalues of H are denoted by $\lambda_{\min}(H)$ and $\lambda_{\max}(H)$, respectively. $\|H\|_2 = \sqrt{\lambda_{\max}(H^T H)}$. For simplicity, we write $\|H\| = \|H\|_2$. The identity matrix is denoted by I and the matrix of zero entries is denoted by 0_n. e is the vector of all ones. Unless otherwise specified, $\{x^k\}$ denotes the sequence $x^0, x^1, \ldots$.

2 Block Coordinate Gradient Descent Method

In this section, we briefly describe the block coordinate gradient descent method.

At a current point $x \in \Re^n$, we choose a nonempty subset $\mathcal{J} \subseteq \mathcal{N}$ and a symmetric matrix $H \succ 0_n$ (approximating the Hessian $\nabla^2 f(x)$), and move x along the direction $d = d^H(x; \mathcal{J})$, where

$$d^H(x;\mathcal{J}) \ \stackrel{\text{def}}{=}\ \arg\min_{d\in\Re^n} \left\{ \nabla f(x)^T d + \frac{1}{2} d^T H d + \rho^T |x + d| \ \mid\ d_j = 0\ \forall j \notin \mathcal{J} \right\}. \tag{5}$$

Here $d^H(x; \mathcal{J})$ depends on H through the submatrix $H_{\mathcal{J}\mathcal{J}}$ only.

Using the convexity of $\rho^T|x|$, we have the following lemma showing that d is a descent direction at x whenever $d \neq 0$.

Lemma 2.1 *For any $x \in \Re^n$, nonempty $\mathcal{J} \subseteq \mathcal{N}$ and $H \succ 0_n$, let $d = d^H(x; \mathcal{J})$ and $g = \nabla f(x)$. Then*

$$F_\rho(x + \alpha d) \le F_\rho(x) + \alpha \left(g^T d + \rho^T |x + d| - \rho^T |x| \right) + o(\alpha) \quad \forall \alpha \in (0, 1], \tag{6}$$

$$g^T d + \rho^T |x + d| - \rho^T |x| \le -d^T H d. \tag{7}$$

Proof. See the proof of [39, Lemma 1] with $c = 1$ and $P(x) = \rho^T|x|$. ∎

We next choose a stepsize $\alpha > 0$ so that $x' = x + \alpha d$ achieves sufficient descent, and re-iterate. We now describe formally the block coordinate gradient descent (abbreviated as CGD) method.

CGD method:

Choose $x^0 \in \Re^n$. For $k = 0, 1, 2, ...$, generate x^{k+1} from x^k according to the iteration:

1. Choose an $H^k \succ 0_n$ and a nonempty set $\mathcal{J}^k \subseteq \mathcal{N}$.

2. Solve (5) with $x = x^k,\ \mathcal{J} = \mathcal{J}^k,\ H = H^k$ to obtain $d^k = d^{H^k}(x^k; \mathcal{J}^k)$.

3. Set $x^{k+1} = x^k + \alpha^k d^k$, with $\alpha^k > 0$.

The following adaptation of the Armijo rule, based on Lemma 2.1, is simple and seems effective from both theoretical and practical standpoints.

Armijo rule:

Choose $\alpha^k_{\text{init}} > 0$ and let α^k be the largest element of $\{\alpha^k_{\text{init}}\beta^j\}_{j=0,1,...}$ satisfying

$$F_\rho(x^k + \alpha^k d^k) \leq F_\rho(x^k) + \alpha^k \sigma \Delta^k, \tag{8}$$

where $0 < \beta < 1$, $0 < \sigma < 1$, $0 \leq \gamma < 1$, and

$$\Delta^k \stackrel{\text{def}}{=} \nabla f(x^k)^T d^k + \gamma {d^k}^T H^k d^k + \rho^T |x^k + d^k| - \rho^T |x^k|. \tag{9}$$

Since $H^k \succ 0_n$ and $0 \leq \gamma < 1$, we see from Lemma 2.1 that $\Delta^k \leq (\gamma - 1){d^k}^T H^k d^k < 0$, whenever $d^k \neq 0$, and

$$F_\rho(x^k + \alpha d^k) \leq F_\rho(x^k) + \alpha \Delta^k + o(\alpha) \quad \forall \alpha \in (0, 1].$$

Since $0 < \sigma < 1$, this shows that α^k given by the Armijo rule is well defined and positive. This rule requires only function evaluations, and by choosing α^k_{init} based on the previous stepsize α^{k-1}, the number of function evaluations can be kept small in practice. Notice that Δ^k increases with γ. Thus, larger stepsizes will be accepted if we choose either σ near 0 or γ near 1. The minimization rule:

$$\alpha^k \in \arg\min\{F_\rho(x^k + \alpha d^k) \mid \alpha \geq 0\} \tag{10}$$

or the limited minimization rule:

$$\alpha^k \in \arg\min\{F_\rho(x^k + \alpha d^k) \mid 0 \leq \alpha \leq s\}, \tag{11}$$

where $0 < s < \infty$, can be used instead of the Armijo rule if the minimization is relatively inexpensive, such as for the problem (2). The latter stepsize rule yields a larger descent than the Armijo rule with $\alpha^k_{\text{init}} = s$. We will use the minimization rule in our numerical experiments for the ℓ_1-regularized linear least squares problems (2) in Section 4.

For convergence, the index subset $\mathcal{J}^k$ must be chosen judiciously. We will see in Lemma 3.1 that $x \in \Re^n$ is a stationary point of F_ρ if and only if $d^H(x; \mathcal{N}) = 0$. Thus, $\|d^H(x; \mathcal{N})\|_\infty$ acts as a scaled "residual" function (with scaling matrix H), measuring how close x comes

to being stationary for F_ρ. Moreover, if H is diagonal, then the separability of $\rho^T|x|$ means that $d_j^H(x;\mathcal{N})$, the jth components of $d^H(x;\mathcal{N})$, depends on x_j and ρ_j only and is easily computable. Accordingly, we choose $\mathcal{J}^k$ to satisfy

$$\|d^{D^k}(x^k;\mathcal{J}^k)\|_\infty \geq \upsilon \|d^{D^k}(x^k;\mathcal{N})\|_\infty, \tag{12}$$

where $0 < \upsilon \leq 1$ and $D^k \succ 0_n$ is diagonal (e.g., $D^k = \eta^k I$ with $\eta^k > 0$ or $D^k = \mathrm{diag}(H^k)$). Following [39], we will call (12) the *Gauss-Southwell-r* rule. Notice that $\mathcal{J}^k = \mathcal{N}$ is a valid choice.

For any $x \in \Re^n$, nonempty $\mathcal{J} \subseteq \mathcal{N}$, and $H \succ 0_n$, define $q^H(x;\mathcal{J})$ to be the difference between the optimal objective value of (5) and $\rho^T|x|$, i.e.,

$$q^H(x;\mathcal{J}) \stackrel{\text{def}}{=} \left(\nabla f(x)^T d + \frac{1}{2} d^T H d + \rho^T|x+d|\right)_{d=d^H(x;\mathcal{J})} - \rho^T|x|.$$

Thus $q^H(x;\mathcal{J})$ estimates the descent in F_ρ from x to $x + d^H(x;\mathcal{J})$. We have from (7) in Lemma 2.1 that $q^H(x;\mathcal{J}) \leq -\frac{1}{2} d^H(x;\mathcal{J})^T H d^H(x;\mathcal{J}) \leq 0$, so that $q^H(x;\mathcal{N}) = 0$ if and only if $d^H(x;\mathcal{N}) = 0$. Thus, like $\|d^H(x;\mathcal{N})\|_\infty$, $-q^H(x;\mathcal{N})$ acts as a "residual" function, measuring how close x comes to being stationary for F_ρ. Since $\rho^T|x|$ is separable, if H is diagonal, then $q^H(x;\mathcal{J})$ is separable in the sense that $q^H(x;\mathcal{J}) = \sum_{j\in\mathcal{J}} q^H(x;j)$. Accordingly, we choose $\mathcal{J}^k$ to satisfy

$$q^{D^k}(x^k;\mathcal{J}^k) \leq \upsilon\ q^{D^k}(x^k;\mathcal{N}), \tag{13}$$

where $0 < \upsilon \leq 1$, $D^k \succ 0_n$ is diagonal. Following [39], we will call (13) the *Gauss-Southwell-q* rule. Again, $\mathcal{J}^k = \mathcal{N}$ is a valid choice.

3 Convergence Analysis

In this section, we study the convergence of the CGD method as applied to our general ℓ_1-regularized convex minimization problem (1).

Formally, we say that $x \in \Re^n$ is *a stationary point* of F_ρ if $F_\rho'(x;d) \geq 0$ for all $d \in \Re^n$. The following lemma gives an alternative characterization of stationarity.

Lemma 3.1 *For any $H \succ 0_n$, $x \in \Re^n$ is a stationary point of F_ρ if and only if $d^H(x;\mathcal{N}) = 0$.*

Proof. See the proof of [39, Lemma 2] with $c = 1$ and $P(x) = \rho^T|x|$. ∎

3.1 Known Convergence Results

In this subsection, we summarize the results of the global convergence and local linear rate convergence of the CGD method; see [39] for details.

Lemma 3.2 *For any $H \succ 0_n$ and nonempty $\mathcal{J} \subseteq \mathcal{N}$, let $d = d^H(x; \mathcal{J})$. If f satisfies*

$$\|\nabla f(y) - \nabla f(z)\| \leq L\|y - z\| \quad \forall y,\, z \in \Re^n, \tag{14}$$

for some $L \geq 0$, and $H \succeq \underline{\zeta} I$, where $\underline{\zeta} > 0$, then the descent condition

$$F_\rho(x + \alpha d) - F_\rho(x) \leq \sigma\alpha\Delta,$$

where $\Delta = \nabla f(x)^T d + \gamma d^T H d + \rho^T|x+d| - \rho^T|x|$ with $\gamma \in [0, 1)$, is satisfied for any $\sigma \in (0, 1)$ whenever $0 \leq \alpha \leq \min\{1, 2\underline{\zeta}(1 - \sigma + \sigma\gamma)/L\}$.

Proof. See the proof of [39, Lemma 3.4 (b)] with $c = 1$ and $P(x) = \rho^T|x|$. ∎

Assumption 1 *$\bar{\zeta} I \succeq H^k \succeq \underline{\zeta} I$ for all k, where $0 < \underline{\zeta} \leq \bar{\zeta}$.*

The following proposition states the global convergence properties of the CGD method when it is applied to the problem (1). For its proof, we refer to that of [39, Theorem 1 (a), (b), (c), (d), (f)] with $c = 1$ and $P(x) = \rho^T|x|$.

Proposition 3.1 *Let $\{x^k\}$, $\{d^k\}$, $\{H^k\}$ be the sequences generated by the CGD method under Assumption 1, where $\{\alpha^k\}$ is chosen by the Armijo rule with $\inf_k \alpha^k_{\rm init} > 0$. Then the following results hold.*

(a) *$\{F_\rho(x^k)\}$ is nonincreasing and Δ^k given by (9) satisfies*

$$F_\rho(x^{k+1}) - F_\rho(x^k) \leq \sigma\alpha^k\Delta^k \leq 0 \quad \forall k.$$

(b) *If $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-r rule (12) and $\bar{\delta} I \succeq D^k \succeq \underline{\delta} I$ for all k, where $0 < \underline{\delta} \leq \bar{\delta}$, then every cluster point of $\{x^k\}$ is a stationary point of F_ρ.*

(c) *If $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-q rule (13) and $\bar{\delta} I \succeq D^k \succeq \underline{\delta} I$ for all k, where $0 < \underline{\delta} \leq \bar{\delta}$, then every cluster point of $\{x^k\}$ is a stationary point of F_ρ.*

(d) *If ∇f is Lipschitz continuous on $\Re^n$ satisfying (14), then $\inf_k \alpha^k > 0$. Furthermore, if $\lim_{k\to\infty} F_\rho(x^k) > -\infty$ also, then $\{\Delta^k\} \to 0$ and $\{d^k\} \to 0$.*

Proposition 3.1 readily extends to any stepsize rule that yields a larger descent than the Armijo rule at each iteration.

Corollary 3.1 *Proposition 3.1 still holds if in the CGD method the iterates are instead updated by $x^{k+1} = x^k + \tilde{\alpha}^k d^k$, where $\tilde{\alpha}^k \geq 0$ satisfies $F_\rho(x^k + \tilde{\alpha}^k d^k) \leq F_\rho(x^k + \alpha^k d^k)$ for $k = 0, 1, \ldots$ and $\{\alpha^k\}$ is chosen by the Armijo rule with $\inf_k \alpha^k_{\rm init} > 0$.*

Proof. It is readily seen using $F_\rho(x^{k+1}) \leq F_\rho(x^k + \alpha^k d^k)$ that Proposition 3.1(a) holds. The proofs of Proposition 3.1(b)–(d) remain unchanged. ∎

For example, $\tilde{\alpha}^k$ may be generated by the minimization rule (10) or by the limited minimization rule (11).

In what follows, X^* denotes the set of stationary points of F_ρ and

$$\mathrm{dist}(x, X^*) = \min_{x^* \in X^*} \|x - x^*\| \qquad \forall x \in \Re^n.$$

Assumption 2 **(a)** *$X^* \neq \emptyset$ and, for any $\eta \geq \min_x F_\rho(x)$, there exist scalars $\varrho > 0$ and $\epsilon > 0$ such that*

$$\mathrm{dist}(x, X^*) \leq \varrho \|d^I(x; \mathcal{N})\| \quad \textit{whenever} \quad F_\rho(x) \leq \eta,\ \|d^I(x; \mathcal{N})\| \leq \epsilon.$$

(b) *There exists a scalar $\delta > 0$ such that*

$$\|x - y\| \geq \delta \quad \textit{whenever} \quad x,\ y \in X^*,\ F_\rho(x) \neq F_\rho(y).$$

Assumption 2(a) is a local Lipschitzian error bound assumption, saying that the distance from x to X^* is locally in the order of the norm of the residual at x; see [24] and references therein. Assumption 2(b) says that the isocost surfaces of F_ρ restricted to the solution set X^* are "properly separated." Assumption 2(b) holds automatically if f is convex or f is quadratic; see [24, 39] for further discussions. Upon applying [39, Theorem 4] to the problem (1), we obtain the following sufficient conditions for Assumption 2(a) to hold.

Proposition 3.2 *Suppose that $X^* \neq \emptyset$ and any of the following conditions hold.*

C1 *f is strongly convex and ∇f is Lipschitz continuous on $\Re^n$.*

C2 *f is quadratic.*

C3 *$f(x) = g(Ex) + q^T x$ for all $x \in \Re^n$, where $E \in \Re^{m \times n}$, $q \in \Re^n$, and g is a strongly convex differentiable function on $\Re^m$ with ∇g Lipschitz continuous on $\Re^m$.*

C4 *$f(x) = \max_{y \in Y}\{(Ex)^T y - g(y)\} + q^T x$ for all $x \in \Re^n$, where Y is a polyhedral set in $\Re^m$, $E \in \Re^{m \times n}$, $q \in \Re^n$, and g is a strongly convex differentiable function on $\Re^m$ with ∇g Lipschitz continuous on $\Re^m$.*

Then Assumption 2(a) holds.

The next proposition states, under Assumptions 1, 2 and (14), the linear rate of convergence of the CGD method using the Gauss-Southwell-q rule to choose $\{\mathcal{J}^k\}$. For the proof of this proposition, we refer to that of [39, Theorem 3] with $c = 1$ and $P(x) = \rho^T |x|$.

Proposition 3.3 *Assume that ∇f is Lipschitz continuous on $\Re^n$ satisfying* (14). *Let $\{x^k\}$, $\{H^k\}$, $\{d^k\}$ be the sequences generated by the CGD method satisfying Assumption 1, where $\{\mathcal{J}^k\}$ is chosen by Gauss-Southwell-q rule* (13) *and $\bar{\delta} I \succeq D^k \succeq \underline{\delta} I$ for all k, with $0 < \underline{\delta} \leq \bar{\delta}$. If F_ρ satisfies Assumption 2 and $\{\alpha^k\}$ is chosen by the Armijo rule with $\sup_k \alpha^k_{\rm init} \leq 1$ and $\inf_k \alpha^k_{\rm init} > 0$, then either $\{F_\rho(x^k)\} \downarrow -\infty$ or $\{F_\rho(x^k)\}$ converges at least Q-linearly and $\{x^k\}$ converges at least R-linearly.*

Similar to Corollary 3.1, Proposition 3.3 readily extends to any stepsize rule that yields a uniformly bounded stepsize and a larger descent than the Armijo rule at each iteration. An example is the limited minimization rule (11).

Corollary 3.2 *Proposition 3.3 still holds if in the CGD method the iterates are instead updated by $x^{k+1} = x^k + \tilde{\alpha}^k d^k$, where $\tilde{\alpha}^k \geq 0$ satisfies $\sup_k \tilde{\alpha}^k \leq 1$ and $F_\rho(x^k + \tilde{\alpha}^k d^k) \leq F_\rho(x^k + \alpha^k d^k)$ for $k = 0, 1, \ldots$ and $\{\alpha^k\}$ is chosen by the Armijo rule with $\sup_k \alpha^k_{\rm init} \leq 1$ and $\inf_k \alpha^k_{\rm init} > 0$.*

Remark 1 *As explained in the end of [39, Section 5], the Lipschitz continuity assumption on ∇f in Propositions 3.1 (d) and 3.3 can be relaxed to ∇f being Lipschitz continuous on $(X^0 + \epsilon B)$ for some $\epsilon > 0$, where B denotes the unit Euclidean ball in $\Re^n$ and $X^0 = \{x \mid F_\rho(x) \leq F_\rho(x^0)\}$.*

3.2 New Convergence Results

In this subsection, by using the relation between the descent direction and the mappings that are extensions of the corresponding mappings defined in [18], we can strengthen the convergence rate of $\{x^k\}$ from R-linear to Q-linear for the CGD method using the Gauss-Southwell-q rule to choose $\{\mathcal{J}^k\}$ and establish the Q-linear convergence rate of $\{x^k\}$ for the CGD method using the Gauss-Southwell-r rule to choose $\{\mathcal{J}^k\}$.

First, we define two mappings $s_\nu : \Re^n \to \Re$ and $h_D : \Re^n \to \Re$ as follows:

$$h_D(x) \stackrel{\rm def}{=} x - D^{-1}\nabla f(x),$$
$$s_\nu(x) \stackrel{\rm def}{=} {\rm sgn}(x) \odot \max\{|x| - \nu, 0\},$$

where $D \succ 0_n$ is diagonal and $\nu \in \Re^n$ is a vector whose components are all nonnegative, $\odot$ denotes the component-wise product, i.e., $(x \odot y)_i = x_i y_i$, and sgn is the signum function defined by

$${\rm sgn}(t) \stackrel{\rm def}{=} \begin{cases} +1 & \text{if } t > 0; \\ 0 & \text{if } t = 0; \\ -1 & \text{if } t < 0. \end{cases}$$

It is easy to prove that $s_\nu(y)$, where $\nu = D^{-1}\rho$, is the unique solution of

$$\min_{z \in \Re^n} \frac{1}{2}(z - y)^T D(z - y) + \rho^T |z|,$$

for any $y \in \Re^n$. Also, for any diagonal $D \succ 0_n$, we have

$$\begin{aligned}
& \min_{d\in\Re^n} \left\{ \nabla f(x)^T d + \frac{1}{2} d^T D d + \rho^T |x+d| \right\} \\
= & \sum_{i=1}^n \min_{d_i\in\Re} \left\{ \nabla f(x)_i d_i + \frac{1}{2} d_i D_{ii} d_i + \rho_i |x_i + d_i| \right\} \\
= & \sum_{i=1}^n D_{ii} \min_{d_i\in\Re} \left\{ \frac{1}{2} \left(d_i + \frac{1}{D_{ii}} \nabla f(x)_i \right)^2 + \frac{\rho_i}{D_{ii}} |x_i + d_i| - \frac{1}{2} \left(\frac{\nabla f(x)_i}{D_{ii}} \right)^2 \right\} \\
= & \sum_{i=1}^n D_{ii} \min_{u_i\in\Re} \left\{ \frac{1}{2} \left(u_i - \left(x_i - \frac{1}{D_{ii}} \nabla f(x)_i \right) \right)^2 + \frac{\rho_i}{D_{ii}} |u_i| - \frac{1}{2} \left(\frac{\nabla f(x)_i}{D_{ii}} \right)^2 \right\} \\
= & \min_{u\in\Re^n} \frac{1}{2} \left(u - (x - D^{-1}\nabla f(x)) \right)^T D \left(u - (x - D^{-1}\nabla f(x)) \right) + \rho^T |u| \\
& - \frac{1}{2} \nabla f(x)^T D^{-1} \nabla f(x).
\end{aligned}$$

This implies that

$$d^D(x; \mathcal{N}) = s_{D^{-1}\rho}(h_D(x)) - x. \tag{15}$$

By Lemma 3.1, if $x^* \in X^*$ then, for any diagonal $D \succ 0_n$, $d^D(x^*; \mathcal{N}) = 0$ and so

$$s_{D^{-1}\rho}(h_D(x^*)) = x^*. \tag{16}$$

The next lemma shows some component-wise properties of s_ν. This result will be used to prove Lemma 3.4, Theorem 3.1, and Corollary 3.3.

Lemma 3.3 *The mapping $s_\nu(\cdot)$ is component-wise non-expansive, i.e., for any $y,\ z\ \in \Re^n$,*

$$|(s_\nu(y))_i - (s_\nu(z))_i| \le |y_i - z_i|, \quad \forall i.$$

Also we have, for each index i,

$$(s_\nu(y))_i \neq 0 = (s_\nu(z))_i \ \Rightarrow |y_i| > \nu_i,\ |z_i| \le \nu_i,\ |(s_\nu(y))_i - (s_\nu(z))_i| \le |y_i - z_i| - (\nu_i - |z_i|).$$

Proof. See the proof of [18, Lemma 3.2]. ∎

The following assumption, which states that f is a convex function with Lipschitz continuous gradient and positive semidefinite Hessian on the level set associated with the initial point of the iterates, is needed to establish the Q-linear rate of convergence of the CGD method.

Assumption 3 **(a)** *$X^* \neq \emptyset$ and, for $X^0_\epsilon \stackrel{\text{def}}{=} \{x \mid F_\rho(x) \le F_\rho(x^0)\} + \epsilon B$ with some $\epsilon > 0$, $f \in C^2(X^0_\epsilon)$, $\nabla^2 f(x) \succeq 0$ for all $x \in X^0_\epsilon$.*

(b) *∇f is Lipschitz continuous on the set X_ϵ^0 with Lipschitz constant L_0.*

Note that Assumption 3 holds for a convex quadratic f that is bounded below.

From the mean-value theorem, we recall that for any $y,\ z \in X_\epsilon^0$

$$\nabla f(y) - \nabla f(z) = \left(\int_0^1 \nabla^2 f(z + t(y-z))dt\right)(y-z) \stackrel{\text{def}}{=} \Psi(y,z)(y-z). \tag{17}$$

We first define, for $x \in X_\epsilon^0,\ x^* \in X^*$

$$Q^H(x;x^*) = I - H^{-1}\Psi(x,x^*),$$

where $H \succ 0_n$.

Given $x^* \in X^*$, we define the following partitions of indices in $\mathcal{N}$:

$$P^* \stackrel{\text{def}}{=} \{i : |g_i^*| < \rho_i\} \text{ and } \bar{P}^* \stackrel{\text{def}}{=} \{i : |g_i^*| = \rho_i\}.$$

where $g^* = \nabla f(x^*)$. By the following optimality condition:

$$x^* \in X^* \iff g_i^* \begin{cases} = -\rho_i & \text{if } x_i^* > 0 \\ \in [-\rho_i, \rho_i] & \text{if } x_i^* = 0 \\ = \rho_i & \text{if } x_i^* < 0, \end{cases}$$

we obtain that, for any $x^* \in X^*$,

$$P^* \cup \bar{P}^* = \mathcal{N},\ \text{supp}(x^*) \subseteq \bar{P}^*,\ \text{and}\ x_i^* = 0,\ \forall i \in P^*,$$

where $\text{supp}(x) \stackrel{\text{def}}{=} \{j \in \mathcal{N} \mid x_j \neq 0\}$.

The next lemma shows that we obtain finite convergence for components in P^*. It will be used to prove Theorem 3.1.

Lemma 3.4 *Suppose f is a function that satisfies Assumption* 3. *Let $\{x^k\}$, $\{H^k\}$, $\{d^k\}$ be the sequences generated by the CGD method satisfying Assumption* 1 *with $\underline{\zeta} \geq L_0$, where $\{\alpha^k\}$ is chosen by the Armijo rule with $0 < \sigma \leq 1/2$ and $\alpha_{\text{init}}^k = 1$ for all k, $H^k = D^k$ is diagonal for all k, and $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-r rule* (12) *or the Gauss-Southwell-q rule* (13). *Then $\{x^k\}$ converges to some $x^* \in X^*$. In addition, if*

$$\|Q^{H^k}(x^k;x^*)\| \leq 1 \tag{18}$$

is satisfied for all k, then for all but finitely many iterations, we have

$$x_i^k = x_i^* = 0,\ \forall i \in P^*.$$

Proof. By Assumption 3, $\lim_{k\to\infty} F_\rho(x^k) > -\infty$ and ∇f is Lipschitz continuous on X^0_ϵ. Hence Lemma 3.2 with $\underline{\zeta} \geq L_0$, $\sigma \leq 1/2$ and $\alpha^k_{\rm init} = 1\ \forall k$ imply that $\alpha^k = 1$ for all k. Thus

$$x^{k+1} = x^k + d^k. \tag{19}$$

By Proposition 3.1(d) (see Remark 1), $\{\Delta^k\} \to 0$ and $\{d^k\} \to 0$. Next we will show that

$$\{d^{H^k}(x^k;\mathcal{N})\} \to 0. \tag{20}$$

Case (1): If $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-r rule (12). By (12) with $D^k = H^k$, $\|d^k\|_\infty \geq \upsilon \|d^{H^k}(x^k;\mathcal{N})\|_\infty$, where $0 < \upsilon \leq 1$. This together with $\{d^k\} \to 0$ yields (20).

Case (2): If $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-q rule (13). Since $q^{H^k}(x^k;\mathcal{J}^k) = \Delta^k + (\frac{1}{2} - \gamma)(d^k)^T H^k d^k \leq 0$, the boundedness of $\{H^k\}$, $\{\Delta^k\} \to 0$, and $\{d^k\} \to 0$ imply that $\{q^{H^k}(x^k;\mathcal{J}^k)\} \to 0$. This and (13) with $D^k = H^k$ yield

$$\{q^{H^k}(x^k;\mathcal{N})\} \to 0. \tag{21}$$

By $H^k \succeq \underline{\zeta} I$ and (D) in Lemma 2.1 with $H = H^k$ and $\mathcal{J} = \mathcal{N}$,

$$q^{H^k}(x^k;\mathcal{N}) \leq -\frac{1}{2}(d^{H^k}(x^k;\mathcal{N}))^T H^k d^{H^k}(x^k;\mathcal{N}) \leq -\frac{\underline{\zeta}}{2}\|d^{H^k}(x^k;\mathcal{N})\|^2.$$

Hence this together with (21) yields (20).

Since $\{d^k\} \to 0$, there is an $x^* \in \Re^n$ such that

$$\lim_{k\to\infty} x^k = x^*. \tag{22}$$

This together with (20) implies that $d^{\bar{H}}(x^*;\mathcal{N}) = 0$ where $\bar{H}$ is any cluster point of $\{H^k\}$. Since $H^k \succeq \underline{\zeta} I$ for all k, $\bar{H} \succ 0_n$. Therefore, by Lemma 3.1, $x^* \in X^*$.

By (16), we obtain that, for any $x^* \in X^*$ and H^k,

$$s_{(H^k)^{-1}\rho}(h_{H^k}(x^*)) = x^*. \tag{23}$$

This together with letting $h(\cdot) = h_{H^k}(\cdot)$, and the mean value theorem yields

$$\begin{aligned} h(x^k) - h(x^*) &= x^k - x^* - (H^k)^{-1}(\nabla f(x^k) - \nabla f(x^*)) \\ &= (I - (H^k)^{-1}\Psi(x^k, x^*))(x^k - x^*) \ = \ Q^{H^k}(x^k; x^*)(x^k - x^*). \end{aligned}$$

Thus

$$\|h(x^k) - h(x^*)\| \ \leq \ \|Q^{H^k}(x^k; x^*)\|\|x^k - x^*\| \ \leq \ \|x^k - x^*\|. \tag{24}$$

Fix any $k \geq 0$. Letting $\mathcal{J} = \mathcal{J}^k$, $s(\cdot) = s_{(H^k)^{-1}\rho}(\cdot)$, $h(\cdot) = h_{H^k}(\cdot)$, $s_{\mathcal{J}}(\cdot) = (s(\cdot))_{\mathcal{J}}$, and $h_{\mathcal{J}}(\cdot) = (h(\cdot))_{\mathcal{J}}$, we have from (15) with $D = H^k$, (19), and (23) that,

$$\begin{aligned}
|x_j^{k+1} - x_j^*| &= |x_j^k + d_j^{H^k}(x^k; \mathcal{J}) - x_j^*| \\
&\leq |x_j^k + d_j^{H^k}(x^k; \mathcal{N}) - x_j^*| + |d_j^{H^k}(x^k; \mathcal{N}) - d_j^{H^k}(x^k; \mathcal{J})| \\
&= |s_j(h(x^k)) - s_j(h(x^*))| + |d_j^{H^k}(x^k; \mathcal{J}^c)| \\
&\leq |h_j(x^k) - h_j(x^*)| + |d_j^{H^k}(x^k; \mathcal{J}^c)| \quad \forall\, j \in \mathcal{N}, \qquad (25)
\end{aligned}$$

where the fourth step uses Lemma 3.3.

Let $g^* = \nabla f(x^*)$ and $\omega = \min\{(\rho_i - |g_i^*|)/\bar{\zeta} \mid i \in P^*\}$. Consider any $i \in P^*$. Note that $x_i^* = 0$. By (20) and (22), there is a sufficiently large integer k_0 such that $\|x^k - x^*\| \leq \omega/3$ and $\|d^{H^k}(x^k; \mathcal{N})\| \leq \omega/3$ for all $k \geq k_0$. We will show that if $x_i^t = 0$ for some $t \geq k_0$ then $x_i^{t+1} = 0$.

Case (1): If $i \notin \mathcal{J}^t$, then it is obvious that $x_i^{t+1} = 0$.

Case (2): If $i \in \mathcal{J}^t$, then we will prove that $x_i^{t+1} = 0$ by contradiction. Suppose $x_i^{t+1} \neq 0$. We have from (15) with $D = H^k$, (19), and (23) that

$$\begin{aligned}
|x_i^{t+1} - x_i^*| \;=\; |x_i^t + d_i^t - x_i^*| &= |s_i(h(x^t)) - s_i(h(x^*))| \\
&\leq |h_i(x^t) - h_i(x^*)| - (\rho_i - |g_i^*|)/H_{ii}^t \\
&\leq |h_i(x^t) - h_i(x^*)| - (\rho_i - |g_i^*|)/\bar{\zeta} \\
&\leq |h_i(x^t) - h_i(x^*)| - \omega,
\end{aligned}$$

where the first inequality uses Lemma 3.3; the second inequality uses $\rho_i - |g_i^*| > 0$ and $H_{ii}^t \leq \bar{\zeta}$. Thus by using (25), we have

$$|x_i^{t+1} - x_i^*| \;\leq\; |h_i(x^t) - h_i(x^*)| - \omega + |d_i^{H^t}(x^t; \mathcal{J}^c)|. \qquad (26)$$

This together with (24) and (25) implies that,

$$\begin{aligned}
&\|x^{t+1} - x^*\|^2 \\
&\leq \|h(x^t) - h(x^*)\|^2 - \omega^2 + \|d^{H^t}(x^t; \mathcal{J}^c)\|^2 + 2\|h(x^t) - h(x^*)\|\|d^{H^t}(x^t; \mathcal{J}^c)\| \\
&\leq \|x^t - x^*\|^2 - \omega^2 + \|d^{H^t}(x^t; \mathcal{J}^c)\|^2 + 2\|x^t - x^*\|\|d^{H^t}(x^t; \mathcal{J}^c)\| \\
&\leq \left(\|x^t - x^*\| + \|d^{H^t}(x^t; \mathcal{J}^c)\|\right)^2 - \omega^2 \\
&\leq -\frac{5}{9}\omega^2 \;<\; 0, \qquad (27)
\end{aligned}$$

which is a contradiction. Hence $x_i^{t+1} = 0$.

This together with using induction on t implies that if $x_i^t = 0$ for some $t \geq k_0$ then $x_i^k = 0$ for all $k \geq t$. We will show that there is an integer k_i such that $x_i^k = 0$ for all $k \geq k_i$. The latter implies that the number of iterations where $x_i^k \neq 0$ for some $i \in P^*$ must be finite.

If $x_i^{k_0} = 0$, then, by the above argument, $x_i^k = 0$ for all $k \geq k_0$. If $x_i^{k_0} \neq 0$, then, since $x_i^k \to 0$, there is an integer $l \geq k_0$ such that $x_i^l = x_i^{k_0}$ and $i \in \mathcal{J}^l$. We will show $x_i^{l+1} = 0$ by contradiction. Suppose $x_i^{l+1} \neq 0$. Then, by (27) with $t = l$, it is a contradiction. Hence $x_i^{l+1} = 0$. Since $l \geq k_0$, $x_i^k = 0$ for all $k \geq l + 1$. ∎

The following theorem establishes, under Assumptions 1 and 3, the Q-linear convergence of the CGD method using either the Gauss-Southwell-r rule or Gauss-Southwell-q rule to choose $\{\mathcal{J}^k\}$.

Theorem 3.1 *Suppose f is a function that satisfies Assumption 3. Let $\{x^k\}$, $\{H^k\}$, $\{d^k\}$ be the sequences generated by the CGD method satisfying Assumption 1 with $\underline{\zeta} \geq L_0$, where $\{\alpha^k\}$ is chosen by the Armijo rule with $0 < \sigma \leq 1/2$ and $\alpha_{\text{init}}^k = 1$ for all k, $H^k = D^k$ is diagonal for all k, $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-r rule (12) or the Gauss-Southwell-q rule (13), $\bar{P}^* \subseteq \mathcal{J}^k$ for all sufficiently large k. If (18) is satisfied, and*

$$\limsup_{k\to\infty} \|\left(Q^{H^k}(x^k; x^*)\right)_{\bar{P}^*\bar{P}^*}\| \;<\; 1, \tag{28}$$

then $\{x^k\}$ converges at least Q-linearly to some $x^* \in X^*$.

Proof. For this proof, we let $Q^k_{\bar{P}^*\bar{P}^*} = \left(Q^{H^k}(x^k; x^*)\right)_{\bar{P}^*\bar{P}^*}$. By Lemma 3.4 and assumption on $\mathcal{J}^k$, there is an index $\bar{k} > 0$ and an $x^* \in X^*$ such that, for all $k \geq \bar{k}$, $x_i^k = x_i^* = 0 \;\forall i \in P^*$ and $\bar{P}^* \subseteq \mathcal{J}^k$. Since $x_i^k = x_i^* = 0 \;\forall i \in P^*$, the mean-value theorem yields

$$\begin{aligned} h_{\bar{P}^*}(x^k) - h_{\bar{P}^*}(x^*) \;&=\; x^k_{\bar{P}^*} - x^*_{\bar{P}^*} - (H^k)^{-1}\left(\nabla f(x^k) - \nabla f(x^*)\right)_{\bar{P}^*} \\ &=\; \left(I - (H^k)^{-1}\Psi(x^k, x^*)\right)_{\bar{P}^*\bar{P}^*}(x^k_{\bar{P}^*} - x^*_{\bar{P}^*}) \;=\; Q^k_{\bar{P}^*\bar{P}^*}(x^k_{\bar{P}^*} - x^*_{\bar{P}^*}). \end{aligned} \tag{29}$$

Fix any $k \geq \bar{k}$. Letting $s(\cdot) = s_{(H^k)^{-1}\rho}(\cdot)$, $h(\cdot) = h_{(H^k)^{-1}}(\cdot)$, $s_{\bar{P}^*}(\cdot) = (s(\cdot))_{\bar{P}^*}$, and $h_{\bar{P}^*}(\cdot) = (h(\cdot))_{\bar{P}^*}$, we have from (15) with $D = H^k$, (19) and (23) that

$$\begin{aligned} \|x^{k+1} - x^*\| \;&=\; \|x^k_{\bar{P}^*} + d^k_{\bar{P}^*} - x^*_{\bar{P}^*}\| \\ &=\; \|s_{\bar{P}^*}(h(x^k)) - s_{\bar{P}^*}(h(x^*))\| \\ &\leq\; \|h_{\bar{P}^*}(x^k) - h_{\bar{P}^*}(x^*)\| \\ &\leq\; \|Q^k_{\bar{P}^*\bar{P}^*}\|\|x^k_{\bar{P}^*} - x^*_{\bar{P}^*}\|, \end{aligned}$$

where $d^k = d^{H^k}(x^k; \mathcal{J}^k)$ and the third step uses Lemma 3.3; the last step uses (29). Hence

$$\limsup_{k\to\infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} \;\leq\; \limsup_{k\to\infty} \|Q^k_{\bar{P}^*\bar{P}^*}\| \;<\; 1.$$

■

Under Assumption 3 (a), if $H^k = \tau I \ \forall k$ with τ chosen such that $\tau > \widehat{\lambda}_{\max}/2 > 0$, where

$$\widehat{\lambda}_{\max} \stackrel{\text{def}}{=} \sup\{\lambda_{\max}(\nabla^2 f(x)) \ : \ x \in X^0_\epsilon\},$$

then, since $0 \preceq \Psi(x^k, x^*) \preceq \widehat{\lambda}_{\max} I$, we have $\|Q^{H^k}(x^k; x^*)\| \leq 1 \quad \forall k$. In addition, if

$$\lambda^{\bar{P}^*}_{\min} \stackrel{\text{def}}{=} \lambda_{\min}\left(\left(\nabla^2 f(x^*)\right)_{\bar{P}^*\bar{P}^*}\right) \ > \ 0, \tag{30}$$

then (28) is satisfied.

The next corollary establishes, under Assumption 3 (a), the Q-linear convergence of the CGD method with $H^k = \tau I$ with $\tau \geq \widehat{\lambda}_{\max} > 0$ for all k.

Corollary 3.3 *Suppose f is a function that satisfies Assumption* 3 (a). *Let $\{x^k\}$, $\{H^k\}$, $\{d^k\}$ be the sequences generated by the CGD method, where $\{\alpha^k\}$ is chosen by the Armijo rule with $0 < \sigma \leq 1/2$ and $\alpha^k_{\text{init}} = 1$ for all k, $H^k = D^k = \tau I$ with $0 < \widehat{\lambda}_{\max} \leq \tau \leq \bar{\zeta}$ for all k, $\{\mathcal{J}^k\}$ is chosen by the Gauss-Southwell-r rule* (12) *or the Gauss-Southwell-q rule* (13), *$\bar{P}^* \subseteq \mathcal{J}^k$ for all sufficiently large k. If* (30) *is satisfied, then $\{x^k\}$ converges at least Q-linearly to some $x^* \in X^*$, and*

$$\limsup_{k\to\infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} \ \leq \ \limsup_{k\to\infty} \|Q^{H^k}(x^k; x^*)_{\bar{P}^*\bar{P}^*}\| \ \leq \ \max\left\{|1 - \frac{\widehat{\lambda}_{\max}}{\tau}|, \, |1 - \frac{\lambda^{\bar{P}^*}_{\min}}{\tau}|\right\} < 1. \tag{31}$$

Proof. By (17), $\|\nabla f(y) - \nabla f(z)\| \leq \widehat{\lambda}_{\max}\|y - z\|$ for all $y, z \in X^0_\epsilon$. Hence f satisfies Assumption 3 (b) with $L_0 = \widehat{\lambda}_{\max}$. Then, by Lemma 3.4 with $H^k = \tau I$, and the continuity of the Hessian of f, without loss of generality, we can assume that there is an index $\bar{k} > 0$ and an $x^* \in X^*$ such that, for all $k \geq \bar{k}$, $x^k_i = x^*_i = 0 \ \forall i \in P^*$ and the spectrum of $(\nabla^2 f(x^k))_{\bar{P}^*\bar{P}^*}$ falls in the interval $[\lambda^{\bar{P}^*}_{\min} - \epsilon, \widehat{\lambda}_{\max}]$ for an arbitrary $\epsilon > 0$ and $P^* \subseteq \mathcal{J}^k$. Fix any $k \geq \bar{k}$. Since $(\lambda^{\bar{P}^*}_{\min} - \epsilon)I \preceq (\nabla^2 f(x^k))_{\bar{P}^*\bar{P}^*} \preceq \widehat{\lambda}_{\max} I$, we have

$$(1 - \frac{\widehat{\lambda}_{\max}}{\tau})I \preceq (Q^{H^k}(x^k; x^*))_{\bar{P}^*\bar{P}^*} = I - \tau^{-1}(\Psi(x^k, x^*))_{\bar{P}^*\bar{P}^*} \preceq (1 - \frac{\lambda^{\bar{P}^*}_{\min} - \epsilon}{\tau})I.$$

Thus

$$\limsup_{k\to\infty} \|(Q^{H^k}(x^k; x^*))_{\bar{P}^*\bar{P}^*}\| \leq \max\{|1 - \widehat{\lambda}_{\max}/\tau|, |1 - \lambda^{\bar{P}^*}_{\min}/\tau|\}.$$

The required result (31) then follows from Theorem 3.1. ■

Remark 2 *In Lemma 3.4 and Theorem 3.1, we assume $H^k \succeq \underline{\zeta} I$ for all k with $\underline{\zeta} \geq L_0$, where L_0 is Lipschitz constant on X^0_ϵ, and $0 < \sigma \leq 1/2$ to ensure the stepsize $\alpha^k = 1$. If the*

Lipschitz constant L_0 is unknown, we can still ensure that $\alpha^k = 1$ by adaptively scaling H^k when generating d^k, analogous to the Armijo rule along the projection arc for constrained smooth optimization [2, page 236]. In particular, we choose s^k to be the largest element of $\{s\beta^j\}_{j=0,1,\ldots}$ $(s > 0)$ such that

$$d^k = d^{H^k/s^k}(x^k; \mathcal{J}^k)$$

satisfies the Armijo descent condition (8) with $\alpha^k = 1$. This adaptive scaling strategy is more expensive computationally since d^k needs to be recomputed each time s^k is changed.

Remark 3 *In Lemma 3.4 and Theorem 3.1, the assumption on $\underline{\zeta}$ and on σ can be replaced by $\underline{\zeta} > L_0/2$ and $0 < \sigma \leq (1 - L_0/(2\underline{\zeta}))$ to ensure the stepsize $\alpha^k = 1$. If we choose $H^k = \tau I$ for some constant $\tau > 0$ and $\mathcal{J}^k = \mathcal{N}$ for all k, then the CGD method is similar to the IST method (and fixed point method [18]). If we assume that $0 < \sigma \leq (1 - \widehat{\lambda}_{\max}/(2\tau))$ and $H^k = \tau I$ with $\tau > \widehat{\lambda}_{\max}/2$ in Corollary 3.3, then Corollary 3.3 still holds. Moreover, if $\tau = (\widehat{\lambda}_{\max} + \lambda^{\bar{P}^*}_{\min})/2$, then (31) is replaced by*

$$\limsup_{k\to\infty} \frac{\|x^{k+1} - x^*\|}{\|x^k - x^*\|} \leq \frac{\widehat{\lambda}_{\max}/\lambda^{\bar{P}^*}_{\min} - 1}{\widehat{\lambda}_{\max}/\lambda^{\bar{P}^*}_{\min} + 1}$$

which is similar to those of [18, Theorem 4.3]. Hence the CGD method may be viewed as a hybrid of the block coordinate descent and IST (or fixed point) method when $H^k = \tau I$ with some constant $\tau > 0$.

Remark 4 *In Lemma 3.4, we only need $\alpha^k = 1$ whenever $i \in \mathcal{J}^k$ for some $i \in P^*$ with $x^k_i \neq 0$ to prove the finite convergence of all the indexes in P^*. Hence the assumption on $\underline{\zeta} \geq L_0$ and on $\{\alpha^k\}$ being chosen by Armijo rule with $\sigma \leq 1/2$ can be relaxed by the following assumption:*
$\{\alpha^k\}$ is chosen by Armijo rule and $\alpha^k = 1$ whenever $i \in \mathcal{J}^k$ for some $i \in P^*$ with $x^k_i \neq 0$.

4 Numerical Experience I (ℓ_1-regularized linear least squares problem)

In this section, we describe the implementation of the CGD method and report our numerical results on three types of problems of the form (2). In particular, we report the comparison of the CGD method with state-of-the-art algorithms, namely l1-ls [19], IST [13], GPSR [14] and FPC [18]. All runs are performed on an Intel Xeon 3.80GHz, running Linux and MATLAB (Version 7.6). Throughout the experiments, we choose the initial iterate to be $x^0 = 0$.

4.1 Implementation of the CGD method

We have implemented the CGD method in MATLAB to solve a large-scale ℓ_1-regularized linear least squares problem (2). In our implementation of the CGD method, we choose a diagonal positive matrix

$$H^k = \theta^k I. \tag{32}$$

Initially, we set $\theta^0 = \|Au\|^2$ where u is a random unit vector. Then it is updated as follows:

$$\theta^{k+1} = \begin{cases} \max\{\theta^k/\alpha^k, 1\} & \text{if } \alpha^k > 10 \\ \min\{\theta^k/\alpha^k, 1\} & \text{if } \alpha^k < 10^{-1} \\ \theta^k & \text{otherwise.} \end{cases}$$

The above updating formulas for θ^k are guided by the observation that smaller θ^k results in a smaller stepsize α^k, while a larger θ^k has the opposite effect. Thus if α^k is large, we decrease θ^k and if α^k is small, we increase θ^k. The thresholds 10 and 10^{-1} were found after some experimentation. We tested the alternative choice of $H^k = I$ and

$$H^k = \text{diag}\left[\min\{\max\{(A^TA)_{jj}, 10^{-10}\}, 10^{10}\}\right]_{j=1,\ldots,n}, \tag{33}$$

but its overall performance was worse. The number of iterations to meet a given stopping tolerance is almost same for using both (32) and (33). But the CPU time for evaluating (32) is less than that for evaluating (33), especially when the problem is large-scale. The advantage of choosing H^k diagonal is that d^k has a closed form solution. We choose the index subset $\mathcal{J}^k$ by either (i) the Gauss-Southwell-r rule (12) with $D^k = H^k$,

$$\mathcal{J}^k = \left\{j \mid |d_j^{D^k}(x^k;\mathcal{N})| \geq \upsilon^k \|d^{D^k}(x^k;\mathcal{N})\|_\infty\right\}, \quad \upsilon^{k+1} = \begin{cases} \max\{10^{-2}, 0.8\upsilon^k\} & \text{if } \alpha^k > 10 \\ \max\{10^{-2}, 0.9\upsilon^k\} & \text{if } 1 < \alpha^k \leq 10 \\ \max\{10^{-2}, 0.98\upsilon^k\} & \text{if } 0.5 < \alpha^k \leq 1 \\ \min\{0.2, 2\upsilon^k\} & \text{if } \alpha^k < 10^{-1} \\ \upsilon^k & \text{otherwise} \end{cases}$$

(initially $\upsilon^0 = 0.9$) or (ii) the Gauss-Southwell-q rule (13) with $D^k = H^k$,

$$\mathcal{J}^k = \left\{j \mid q^{D^k}(x^k;j) \leq \upsilon^k \min_i q^{D^k}(x^k;i)\right\}, \quad \upsilon^{k+1} = \begin{cases} \max\{10^{-2}, 0.8\upsilon^k\} & \text{if } \alpha^k > 10 \\ \max\{10^{-2}, 0.9\upsilon^k\} & \text{if } 1 < \alpha^k \leq 10 \\ \max\{10^{-2}, 0.98\upsilon^k\} & \text{if } 0.5 < \alpha^k \leq 1 \\ \min\{0.2, 2\upsilon^k\} & \text{if } \alpha^k < 10^{-1} \\ \upsilon^k & \text{otherwise} \end{cases}$$

(initially $\upsilon^0 = 0.5$). The above updating formulas for υ^k in (i) and (ii) are guided by the observation that smaller υ^k results in more coordinates being updated but a smaller stepsize α^k is obtained, while a larger υ^k has the opposite effect. Thus if α^k is large, we decrease υ^k and if α^k is small, we increase υ^k. The thresholds 10^{-2}, 10^{-1}, 0.2, 0.5, 1, 10 and the scaling constants 0.8, 0.9, 0.98, 2 were found after some experimentation.

Since the problem (2) is quadratic, the stepsize α^k is chosen by the minimization rule (10) instead of the Armijo rule (8). Hence α^k is the solution of the following minimization problem:

$$\min_{\alpha>0} \quad \frac{1}{2}\|(b - Ax^k) - \alpha Ad^k\|^2 + \mu\|x^k + \alpha d^k\|_1.$$

This problem can be reformulated as follows:

$$\min_{\alpha>0} \quad \Pi(\alpha) \stackrel{\text{def}}{=} \frac{a_1}{2}\alpha^2 - a_2\alpha + \mu \sum_{j\in\mathcal{J}^k} |x_j^k + \alpha d_j^k| + a_3, \tag{34}$$

where $a_1 = \|Ad^k\|^2$, $a_2 = (Ad^k)^T(b - Ax^k)$, and $a_3 = \frac{1}{2}\|b - Ax^k\|^2$. For simplicity, we will only discuss the efficient computation of α^k for the harder case where $a_1 > 0$. Under the assumption that $a_1 > 0$, the function Π is a strictly convex piecewise quadratic function. Consider the set of positive break-points of Π, which is given by $\{-x_j^k/d_j^k \mid d_j^k \neq 0,\ x_j^k/d_j^k < 0, j \in \mathcal{J}^k\}$. Since Π is a strictly convex function and $\alpha^k > 0$, there is at least one positive break-point whose objective value is less than the objective value at 0. Let the positive break-points be denoted by $0 < s_1 < s_2 < \cdots < s_l < \infty$ ($l \leq |\mathcal{J}^k|$). If there is a break-point $s_j \in \{s_1, ..., s_{l-1}\}$ such that $\Pi(s_j) \leq \Pi(s_{j+1})$, then the optimal solution α^k is in the interval $[s_{i-1}, s_i]$ or $[s_i, s_{i+1}]$ (if $i = 1$ then $s_0 = 0$) where s_i is the smallest break-point in $\{s_1, ..., s_{l-1}\}$ satisfying $\Pi(s_i) \leq \Pi(s_{i+1})$. Otherwise, the optimal solution α^k is in the interval $[s_{l-1}, s_l]$ or $[s_l, \infty)$. Once we find the two intervals, it is easy to obtain an optimal solution of minimizing Π on each interval. Then the stepsize α^k is the one whose corresponding objective value is smaller. In order to find the smallest break-point s_i satisfying $\Pi(s_i) \leq \Pi(s_{i+1})$ or no such a break-point, we should recursively compare $\Pi(s_i)$ with $\Pi(s_{i+1})$ from $i = 1$. But α^{k-1} and α^k may be the same or differ by a moderate factor. Hence it might save computing time to solve (34) by starting to compare the function value at the largest break-point that is less than or equal to α^{k-1} with the function value at the smallest break-point that is greater than α^{k-1}. We briefly explain the procedure to find two intervals by using previous stepsize α^{k-1}. First, we partition the set of the break-points $\{s_1, ..., s_{l-1}\}$ into three subsets `sortl` $= \{s_i | s_i < \alpha^{k-1}\}$, `stepf` $= \{s_i | s_i = \alpha^{k-1}\}$, and `sortg` $= \{s_i | s_i > \alpha^{k-1}\}$. Then we compare the objective value at the largest point of the subset `sortl` with the objective value at the smallest point of the subset `sortg` and the objective value at the previous stepsize α^{k-1}. If the objective value at the smallest point of the subset `sortg` is less than the other two, then we recursively compare the objective value of the break-points in the subset `sortg` to find two intervals. If the objective value at the largest point of the subset `sortl` is less than the other two, then we recursively compare the objective value of the break-points in the subset `sortl` to find two intervals. If the objective value at the previous stepsize α^{k-1} is less than the other two, then α^k is on the closed interval between the largest point of the subset `sortl` and α^{k-1} or on the closed interval between α^{k-1} and the smallest point of the subset `sortg`.

It is difficult to decide when to terminate the method to get an approximate solution which is of sufficiently high quality. A standard criterion that has been suggested previously

for ℓ_1-regularized nonlinear least squares problems [39] is

$$\|H^k d^{H^k}(x^k;\mathcal{N})\|_\infty \leq \text{Tol}, \tag{35}$$

where Tol is a small real number. Here we scale $d^{H^k}(x^k;\mathcal{N})$ by H^k to reduce its sensitivity to H^k. The criterion (35) was motivated by the fact that the left-hand side is zero if and only if x^k is optimal; see Lemma 3.1.

The alternative criterion that was recently proposed in [18] is

$$\frac{\|x^{k+1} - x^k\|}{\max\{\|x^k\|, 1\}} < \text{Tol}.$$

Unless otherwise noted, we use (35) with $\text{Tol} = 10^{-3}$.

In order to accelerate the convergence, we adopt the continuation strategy that is used in [18]. We briefly describe this strategy. If the problem (2) is to be solved with $\mu = \bar{\mu}$, we propose solving a sequence of problems (2) defined by a decreasing sequence $\{\mu^0, \mu^1, \ldots, \mu^\ell = \mu\}$ with a given finite positive integer ℓ. When a new problem, associated with μ^{j+1}, is to be solved, the approximate solution for the current problem with $\mu = \mu^j$ is used as the starting point. Our computational experience indicates that the performance of this continuation strategy is generally superior to that of directly applying the CGD method with the specified value μ.

4.2 Analysis of Computational Cost

In this subsection, we give a comment on the computational complexity of the CGD method when it is applied to the problem (2). We analyze the cost of each iteration of the CGD method. The main computational cost per iteration is a small number of inner products, vector-scalar multiplications and vector additions, each requiring n floating-point operations, plus one sorting which requires $O(n \ln n)$ floating-point operations (it is for the worst case. Since sorting is needed to find the stepsize, the cost is $O(|\mathcal{J}| \ln |\mathcal{J}|)$ for each iteration) and a small number of matrix-vector multiplications. For matrix-vector multiplications, there are three cases in our numerical experiences. In the first case, the mn elements of A are explicitly stored. Hence matrix-vector multiplications involving A^T cost $O(mn)$ operations which are the same as the computational cost of other state-of-the-art algorithms compared with the CGD method in this Section. But, for matrix-vector multiplications involving A, we only needs to evaluate Ad, where d is a descent direction, once at each iteration. Since the number of nonzero components of d is $|\mathcal{J}|$ which is usually small, the matrix-vector multiplication Ad reduces to $A_{\mathcal{J}} d_{\mathcal{J}}$. Hence, in our implementation of the CGD method, we use the MATLAB function "sparse" and so $O(m|\mathcal{J}|)$ operations are required at each iteration. The small support of a descent direction is an advantage to save CPU time over other state-of-the-art algorithms for matrix-vector multiplications involving A. In the second case, A is identified by listing the rows of the $n \times n$ transform matrix (discrete cosine transform). Hence the mn elements of A can not be explicitly stored. But, by

using fast transform operators, such as the DCT, matrix-vector multiplications involving A and A^T cost $O(n \ln n)$ operations (in our MATLAB (version 7.6), we don't have the MATLAB function DCT. Hence we use FFT to get the same transform. But, in order to get the same transform matrix as DCT, we increase the dimension n to $4n$ by adding zero components to the vector that we want to transform. The code is as follows:

```
xtmp = zeros(4*n,1);
xtmp(2:2:2*n) = x;
xtmp(2*n+2:4*n) = xtmp(2*n:-1:2);
ytmp = real(fft(xtmp));
y = (1/sqrt(2*n))*ytmp(1:n);  y(1) = y(1)/sqrt(2);
```

So the constant in $O(\cdot)$ is greater than that for using DCT). In the third case, $A = RW$ where R is an $n \times n$ matrix representation of the blur operation and W represents the inverse orthogonal wavelet transform with Haar wavelets. By using fast transform operators, such as the FFT, matrix-vector multiplications involving A and A^T cost $O(n \ln n)$ operations. But, for the second and third cases, we lose the advantage of "sparse" multiplication which is used for the first case.

4.3 Compressed Sensing

In this subsection, we report the performance of the CGD method using either the Gauss-Southwell-r (CGD-GS-r) rule or the Gauss-Southwell-q (CGD-GS-q) rule with the continuation strategy and compare it with the performances of l1-ls, GPSR-BB (in what follows, GPSR-BB denotes the nonmonotone version using BB step) and FPC-BB (in what follows, FPC-BB denotes the version using BB step) in two compressed sensing scenarios. In our first experiment we considered a typical compressed sensing scenario similar to the one presented in [14], where the goal is to reconstruct a length-n sparse signal from m observations with $m < n$. In this experiment, the $m \times n$ matrix A was obtained by first filling it with independent samples of the standard Gaussian distribution and then orthonormalizing the rows. For this experiment, $n = 4096$, $m = 1024$, the original signal x contained 160 randomly placed ± 1 spikes, or $n = 8192$, $m = 2048$, the original signal x contained 320 randomly placed ± 1 spikes. The measurements Ax were corrupted by noise ξ,

$$b = Ax + \xi,$$

where ξ is Gaussian white noise with variance $(0.01\|Ax\|)^2$. If $\mu \geq \|A^T b\|_\infty$, then the unique minimum of (2) is the zero vector; see [19] for details. Hence the parameter μ is chosen as

$$\mu = c\|A^T b\|_\infty,$$

where $0 < c < 1$. For the continuation strategy, we choose the initial $\mu^0 = 0.01\|A^T b\|_\infty$ and update $\mu^{k+1} = \max\{0.25\mu^k, \mu\}$ whenever the following criterion is satisfied:

$$\frac{\|H^k d^{H^k}(x^k; \mathcal{N})\|_\infty}{\max\{1, \|x^k\|_\infty\}} \leq \text{Tol}, \tag{36}$$

where $\text{Tol} = \max\{10^{\lfloor log(\mu^k) \rfloor}, 10^{-3})\}$ and $\lfloor \cdot \rfloor$ is a floor function. In our experiments, this updating rule works better than the rule with (36) and $\text{Tol} = 10^{-3}$ which is similar to the updating rule used in [18].

Tables 1 and 2 report the number of iterations, the number of nonzero components (nnz(x)) in the final solution found, the final objective value, the CPU time (in seconds), and the relative error $\|x - x_s\|/\|x_s\|$, where x_s is the original signal, of the first 2 ones out of 10 random instances. We note that we use MATLAB function "nnz" to obtain the number of nonzero components of the final solution found except for l1-ls algorithm (for l1-ls, the number is always n). Hence each component for the final solution x of l1-ls is considered as a nonzero component when its absolute value is greater than $0.001\|x\|_\infty$. Tables 1 and 2 also report the mean values of the number of iterations, the number of nonzero components, the CPU time, and the relative error of 10 random instances. To perform this comparison, we first ran the l1-ls algorithm and then each of the other algorithms until each reached the same value of the objective function reached by l1-ls. For each n, three different values of c were chosen to track the sensitivity to μ. CGD outperforms l1-ls, GPSR-BB, and FPC-BB for all random instances in terms of CPU time. Based on the average CPU time, CGD is at least 11.4 times faster than l1-ls, 2.5 times faster than GPSR-BB, and 1.8 times faster than FPC-BB. When μ is large ($\mu = 0.05\|A^T b\|_\infty$), CGD is at least 18.2 times faster than l1-ls and 3.3 times faster than FPC-BB. When $\mu = 0.01\|A^T b\|_\infty$, CGD is at least 4.5 times faster than GPSR-BB. l1-ls is robust to μ and produces accurate reconstructions in terms of error, but is much slower than GPSR-BB, FPC-BB and CGD. CGD and FPC-BB are also robust to μ. In contrast, when μ is small, GPSR-BB reaches its maximum iterations and so is no longer able to recover the original signal as mentioned in [18]. This could be due to the use of BB steps.

In our second experiment we considered another compressed sensing scenario which is similar to the one in [18]. In this case, the $m \times n$ matrix A was a partial DCT matrix whose m rows were chosen at random (uniformly) from the $n \times n$ discrete cosine transform (DCT) matrix (as we mentioned in Subsection 4.2, we used FFT to get A). In this experiment, the choices of m, n, the original signal x and the noise corruption were same as those of the first experiment. Tables 3 and 4 report the same results as Tables 1 and 2 do. We performed this comparison the same way we did for the first experiment. Two different values of c were chosen to track the sensitivity to μ. Based on the average CPU time, CGD outperforms l1-ls, GPSR-BB, and FPC-BB for all cases–except when $\mu = 0.005\|A^T b\|_\infty$, for which CGD is 1.1 times slower than FPC-BB (this could be due to lose the advantage of "sparse" multiplication as indicated in Subsection 4.2). CGD is at least 5.6 times faster than l1-ls and 2.1 times faster than GPSR-BB. When $m = 1024$, $n = 4096$, $\mu = 0.01\|A^T b\|_\infty$,

CGD is at least 6.5 times faster than l1-ls, 2.1 times faster than GPSR-BB, 1.1 times faster than FPC-BB. As we observed for the first experiment, l1-ls, CGD and FPC-BB are robust to μ but GPSR-BB is not.

In the next experiment we considered the compressed sensing scenario in the first experiment to observe how well the CGD method performs on different types of original solutions with have both large, median, and small nonzero components. For this experiment, the $m \times n$ matrix A was obtained by the same way as did for the first experiment with $n = 4096$ and $m = 1024$, but the original signal x contained 50 randomly placed ± 0.1 spikes, 60 randomly placed ± 1 spikes, and 50 randomly placed ± 10 spikes. Table 5 reports the same results as Table 1 does. We performed this comparison the same way we did for the first experiment. By comparing the results in Tables 1 and 5, the ratio ((mean iteration of Table 1 - mean iteration of Table 5)/(mean iteration of Table 1), for CGD: 0.42-0.77, for GPSR-BB: 0.33-0.55, for FPC-BB: 0.43-0.46, no reduction for l1-ls) of reduction of the number of iterations for each algorithm shows that CGD works better than other algorithms on this experiment. Based on the average CPU time, CGD is at least 11.4 times faster than l1-ls, 2.5 times faster than GPSR-BB, and 1.8 times faster than FPC-BB. We note that CGD first chose the coordinates with large magnitudes as expected. Table 6 reports the mean values of relative errors $\|x - x_s\|/\|x_s\|$ (denoted as error) and relative errors of $\|x_{\mathcal{L}} - (x_s)_{\mathcal{L}}\|/\|x_s\|$ (denoted as error-L), where x_s is the original signal and $\mathcal{L}$ denotes the set of coordinates of components with large magnitude, of 10 random instances. In addition, the second column of Table 6 presents the mean values of relative errors of l1-ls with stopping tolerance 10^{-8} (more accurate final solution of the problem of the type (2)). From Table 6, we see that CGD has better reconstructions for the problems with large magnitude (the reconstruction of CGD is as good as that of l1-ls in terms of errors).

4.4 Image Deconvolution

In this subsection, our goal is to compare the speed of the proposed CGD method using the Gauss-Southwell-r rule or the Gauss-Southwell-q rule with the continuation strategy with that of IST, GPSR-BB and FPC-BB in image deconvolution. Following [14], we considered three standard benchmark problems summarized in Table 7, all based on the well-known Cameraman image [12, 13]. The description of $A = RW$ is found in the end of Subsection 4.3. In order to perform the comparison, we first ran the IST and then each of the other algorithms until each reached the same value of the objective function reached by IST. Tables 8 reports the number of iterations, the number of nonzero components (nnz(x)) in the final solution found, the final objective value, the CPU time (in seconds), and the relative error $\|x - x_s\|_F/\|x_s\|_F$, where x_s is the original solution and $\|\cdot\|_F$ is the Frobenius norm, of 3 problems. From Table 8, we see that CGD is at least 1.6 times faster than IST and almost at least 3.8 times faster than FPC-BB, but almost 2 times slower than GPSR-BB for all cases.

5 Numerical Experience II (ℓ_1-regularized logistic regression problem)

In this section, we describe the implementation of the CGD method and report our numerical experience on randomly generated problems and three benchmark problems of the form (4). In particular, we report the comparison with l1-logreg [20], which is written in C, and SpaRSA (the MATLAB code follows the algorithm presented in [41]). All runs are performed on an Intel Xeon 3.80GHz, running Linux and MATLAB (Version 7.3) and C codes are compiled using gcc compiler (Version 3.4.5). Throughout the experiments, we choose the initial iterate to be $(w^0, v^0) = (0, 0)$.

5.1 Implementation of the CGD method

We have implemented the CGD method in MATLAB to solve a large-scale ℓ_1-regularized logistic regression problem (4). In our implementation of the CGD method, we choose a diagonal Hessian approximation

$$H^k = \text{diag}\left[\min\{\max\{\nabla^2 f(x^k)_{jj}, 10^{-10}\}, 10^{10}\}\right]_{j=1,\ldots,n}, \tag{37}$$

where $x^k = ((w^k)^T, v^k)^T$. We tested the alternative choice of $H^k = \eta I$ for some fixed constant η including 1, which does not require the evaluation of the diagonal part of Hessian, but its overall performance was much worse. For example, for CGD-GS-q with the rcv1 problem, it takes more 100000 iterations to meet a given stopping criterion when the alternative choice $H^k = I$ is used, whereas it takes 151 iterations when (37) is used. In order to speed up the computation of the Hessian diagonal, we compute it in C language with interface to MATLAB through the mex-function. We choose the index subset $\mathcal{J}^k$ by either (i) the Gauss-Southwell-r rule (12) with $D^k = H^k$,

$$\mathcal{J}^k = \left\{ j \mid |d_j^{D^k}(x^k; \mathcal{N})| \geq v^k \|d^{D^k}(x^k; \mathcal{N})\|_\infty \right\},$$

or (ii) the Gauss-Southwell-q rule (13) with $D^k = H^k$,

$$\mathcal{J}^k = \left\{ j \mid q^{D^k}(x^k; j) \leq v^k \min_i q^{D^k}(x^k; i) \right\}.$$

We update $v^k = \max\{0.05, 0.95v^k\}$ at iteration k whenever $k = 0 \pmod{20}$ or $k < 10$ (initially $v^0 = 0.9$). The thresholds 0.05 and the scaling constant 0.95 were found after some experimentation. The stepsize α^k is chosen by the Armijo rule (8) with

$$\sigma = 0.1, \quad \beta = 0.5, \quad \gamma = 0, \qquad \alpha^0_{\text{init}} = 1, \quad \alpha^k_{\text{init}} = \min\left\{ \frac{\alpha^{k-1}}{\beta^5}, 1 \right\} \ \forall k \geq 1.$$

We experimented with other values of the above parameters, but the above choice works well.

Each CGD iteration requires 1 gradient evaluation and 1 Hessian diagonal evaluation to find the direction d^k (finding the direction d^k costs $O(mn)$ operations), and at least 1 function evaluation to find the stepsize α^k. These are the dominant computations.

We terminate the CGD method when

$$\|H^k d^{H^k}(x^k)\|_\infty \leq 10^{-6}. \tag{38}$$

5.2 Numerical Results

In this subsection, we report the performance of the CGD method using either the Gauss-Southwell-r (CGD-GS-r) rule or the Gauss-Southwell-q (CGD-GS-q) rule and compare it with the performance of l1-logreg and SpaRSA on three two-class data classification benchmark problems, namely, the leukemia cancer gene expression data [17], the rcv1 data [22], and the real-sim data from the LIBSVM data webpage [6]. Also we compare the CGD method with l1-logreg on randomly generated problems described in [20] to see the effect of problem size. Each randomly generated problem has an equal number of positive and negative examples. Features of positive (negative) examples are independent and identically distributed, drawn from a normal distribution $\mathcal{N}(\nu, 1)$, where ν is in turn drawn from a uniform distribution on $[0, 1]([-1, 0])$.

Table 9 reports the number of iterations, the final objective value, and the CPU time (in seconds) of three benchmark problems. For each instance, we chose $\mu = 0.01\mu_{\max}$ and $\mu = 0.001\mu_{\max}$ where $\mu_{\max} = \frac{1}{m}\|\frac{m_-}{m}\sum_{b_i=1} a_i + \frac{m_+}{m}\sum_{b_i=-1} a_i\|_\infty$, m_- is the number of negative examples, and m_+ is the number of positive examples. If $\mu \geq \mu_{\max}$, we get a maximally sparse weight vector, i.e. $w = 0$; see [20] for details. To perform this comparison, we first ran the l1-logreg algorithm with a stopping tolerance 10^{-4} and then each of the other algorithms until each reached the same value of the objective function reached by l1-logreg. From Table 9, we see that CGD is faster than l1-logreg when μ is large ($\mu = 0.01\mu_{\max}$). SpaRSA is as fast as CGD, when it reaches the final objective value that is founded by l1-logreg, but is not able to reach the final objective of l1-logreg on the leukemia problem.

To examine the effect of problem size on the number of iterations and the CPU time, we generated 10 random problems for each case ($n = 1001,\ 10001,\ m = 0.1(n-1)$ or $n = 101,\ 1001,\ m = 10(n-1)$). Note that the data of each randomly generated problem is dense. Tables 10 and 11 report the number of iterations, the final objective value, and the CPU time (in seconds) of the first 2 ones out of 10 random instances. Also Tables 10 and 11 report the mean values of the number of iterations and the CPU time of 10 random instances. For each instance, we chose $\mu = 0.1\mu_{\max}$ and $\mu = 0.01\mu_{\max}$. We terminated the CGD method when (38) was satisfied and terminated the l1-logreg algorithm with a stopping tolerance 10^{-4}. As described in [20], the computational cost for the search direction of l1-logreg is $O(\min(n-1, m)^2 \max(n-1, m))$ operations per iteration. In contrast, the computational cost of CGD is $O(mn)$ operations per iteration. Hence, based on the average iteration and CPU time, when we increase the number of observed examples and the dimension of the

variable w by 10 times, we see that the CPU time of l1-logreg is increased by more than 1000 times but the CPU time of CGD is increased by less than 100 times. And so CGD is typically much faster than l1-logreg when either $m = 10000$ or $n = 10001$.

For the benchmark problems in Table 9, the data sets are sparse, and l1-logreg can fully exploit the sparsity of the data. Hence l1-logreg is slower but not significantly slower than CGD even when the data is large. Based on the numerical results presented in Tables 9–11, the CGD method can be expected to be more efficient than the l1-logreg algorithm for a large-scale problem for which the data is not highly sparse.

6 Conclusions and Extensions

In this paper we have proposed a block coordinate gradient descent method for solving ℓ_1-regularized convex nonsmooth optimization problems arising in compressed sensing, image deconvolution, and classification. We have established the Q-linear convergence rate for our method when the coordinate block is chosen by either a *Gauss-Southwell-r* rule or a *Gauss-Southwell-q* rule. We evaluated the numerical performance of the CGD method when it is applied to solve ℓ_1-regularized linear least squares (the smooth convex function f of (1) is quadratic) problems and ℓ_1-regularized logistic regression (the smooth convex function f of (1) is nonquadratic) problems. We also reported numerical results of comparisons to state-of-the-art algorithms. The numerical results show that, for the large-scale problems arising in compressed sensing and classification, the CGD method, overall, outperforms state-of-the-art algorithms. Hence the CGD method is efficient not only in minimizing a quadratic convex smooth function with ℓ_1-regularization but also in minimizing a nonquadratic convex smooth function with ℓ_1-regularization.

The Barzilai-Borwein steps used in both GPSR [14] and FPC [18] generally accelerates the convergence. But the performance of the CGD method using the Barzilai-Borwein steps was generally worse than that of the CGD method using the minimization rule when it was applied to solve ℓ_1-regularized linear least squares problems. Can the efficiency be further improved by using nonmonotone descent? When the CGD method with the continuation strategy was applied to solve ℓ_1-regularized logistic regression problems, its overall performance was worse than that of the CGD method without the continuation strategy. Can other acceleration techniques be developed for solving ℓ_1-regularized logistic regression problems? The above are some issues that need further investigation.

Recently Shi et al. [35] considered a large-scale ℓ_1-penalized log likelihood problem of the form:

$$\min_{w\in\Re^n} \ \frac{1}{m}\sum_{i=1}^{m}\Big(-b_i f(z^i) + \log(1+\exp(f(z^i)))\Big) + \mu\|w\|_1. \tag{39}$$

where $z^i = (z^i_1, ..., z^i_p)$ with $z^i_j \in \{0,1\}$, $b_i \in \{0,1\}$, and $f(z) = c + \sum_{l=1}^{n} w_l B_l(z)$ with $B_l(z) = 0$ or 1 (see [35] for details). They proposed a specialized algorithm that solve (3) to

estimate the active set (the set of components w_l that are zero at the minimizer of (39)) and use a Newton-like enhancement to the search direction using the projection of the Hessian of the log likelihood function onto the set of nonzero components w_l. Lin et al. [23] proposed a trust region Newton method to solve a large-scale ℓ_2-regularized logistic regression problem of the form:

$$\min_{w \in \Re^n} \sum_{i=1}^{m} \log(1 + \exp(-w^T a_i)) + \frac{\mu}{2}\|w\|_2^2. \tag{40}$$

Our CGD method can also be applied to solve both (39) and (40). How the CGD method performs on these problems is a topic for future study.

Acknowledgement. We thank an anonymous referee for helpful comments and suggestions.

References

[1] Bertsekas, D. P., Constrained Optimization and Lagrange Multiplier Methods, Academic Press, New York, 1982.

[2] Bertsekas, D. P., Nonlinear Programming, 2nd edition, Athena Scientific, Belmont, 1999.

[3] Bradley, P. S., Fayyad, U. M., and Mangasarian, O. L., Mathematical programming for data mining: formulations and challenges, INFORMS J. Comput. 11 (1999), 217–238.

[4] Candès, E. J., Romberg, J., and Tao, T., Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information, IEEE Trans. Info. Theory 52 (2006), 489–509.

[5] Candès, E. J. and Tao, T., Nearly optimal signal recovery from random projections: Universal encoding strategies, IEEE Trans. Info. Theory 52 (2006), 5406–5425.

[6] Chang, C.-C. and Lin, C.-J., *LIBSVM – A Library for Support Vector Machines*, `http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

[7] Chen, S., Donoho, D., and Saunders, M., Atomic decomposition by basis pursuit, SIAM J. Sci. Comput. 20 (1999), 33–61.

[8] Daubechies, I., De Friese, M., and De Mol, C., An iterative thresholding algorithm for linear inverse problems with a sparsity constraint, Comm. on Pure and Applied Math. 57 (2004), 1413–1457.

[9] Donoho, D., Compressed sensing, IEEE Trans. Info. Theory 52 (2006), 1289–1306.

[10] Donoho, D. and Tsaig, Y., Fast solution of ℓ_1-norm minimization problems when the solution may be sparse, IEEE Trans. Inform. Theory 54 (2008), 4789-4812.

[11] Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R., Least angle regression, Ann. Stat. 32 (2004), 407–499.

[12] Figueiredo, M. and Nowak, R., An EM algorithm for wavelet-bassed image restoration, IEEE Trans. Image Proc. 12 (2003), 906–916.

[13] Figueiredo, M. and Nowak, R., A bound optimization approach to wavelet-based image deconvolution, IEEE Intern. Conf. on Image Processing-ICIP'05, 2005.

[14] Figueiredo, M., Nowak, R., and Wright, S. J., Gradient projection for sparse reconstruction: Application to compressed sensing and other inverse problems, IEEE J. of Selected Topics in Signal Proc. 1 (2007), 586–598.

[15] Fuchs, J.-J., On sparse representations in arbitrary redundant bases, IEEE Trans. Inform. Theory 50 (2004), 1341-1344.

[16] Genkin, A., Lewis, D., and Madigan, D., Large-scale Bayesian logistic regression for text categorization, Technometrics 49 (2007), 291–304.

[17] Golub, T. R., Slonim, D. K., Tamayo, P., Huard, C., Gaasenbeek, M., Mesirov, J. P., Coller, H., Loh, M. L., Downing, J. R., Caligiuri, M. A., Bloomfield, C. D., and Lander, E. S., Molecular classification of cancer: class discovery and class prediction by gene expression monitoring, Science 286 (1999), 531–537.

[18] Hale, E. T., Yin, W., and Zhang, Y., A fixed-point continuation method for ℓ_1-regularized minimization with applications to compressed sensing, CAAM Technical Report TR07-07, Department of Computational and Applied Mathematics, Rice University, July 2007.

[19] Kim, S.-J., Koh, K., Lustig, M., Boyd, S., and Gorinevsky, D., An interior-point method for large-scale ℓ_1-regularized least squares, IEEE J. Selected Topics in Signal Proc. 1 (2007), 606–617.

[20] Koh, K., Kim, S.-J., and Boyd, S., An interior-point method for large-scale ℓ_1-regularized logistic regression, J. Mach. Learn. Res. 8 (2007), 1519–1555.

[21] Lee, S., Lee, H., Abeel, P., and Ng, A., Efficient ℓ_1-regularized logistic regression, In Proceedings of the 21st National Conference on Artificial Intelligence, 2006.

[22] Lewis, D. D., Yang, Y., Rose, T. G., and Li, F., RCV1: A new benchmark collection for text categorization research, J. Mach. Learn. Res. 5 (2004), 361–397.

[23] Lin, C.-J., Weng, R. C., and Keerthi, S. S., Trust region Newton method for large-scale logistic regression. J. Mach. Learn. Res. 9 (2008), 627–650.

[24] Luo, Z.-Q. and Tseng, P., Error bounds and convergence analysis of feasible descent methods: a general approach, Ann. Oper. Res. 46 (1993), 157–178.

[25] Mangasarian, O. L, Sparsity-preserving SOR algorithms for separable quadratic and linear programming, Comput. Oper. Res. 11 (1984), 105–112.

[26] Mangasarian, O. L. and Musicant, D. R., Large scale kernel regression via linear programming, Machine Learning 46 (2002), 255–269.

[27] Ng, A. Y., Feature selection, ℓ_1 vs. ℓ_2 regularization, and rotational invariance, In Proceedings of the 21st International Conference on Machine Learning, 2004.

[28] Nocedal, J. and Wright S. J., Numerical Optimization, Springer-Verlag, New York, 1999.

[29] Osborne, M., Presnell, B., and Turlach, B., A new approach to variable selection in least squares problems, IMA J. Numer. Anal. 20 (2000), 389–403.

[30] Park, M. and Hastie, T., An ℓ_1 regularization-path algorithm for generalized linear models, J. R. Statist. Soc. B 69 (2007), 659-677.

[31] Rockafellar, R. T., Convex Analysis, Princeton University Press, Princeton, 1970.

[32] Rockafellar, R. T. and Wets R. J.-B., Variational Analysis, Springer-Verlag, New York, 1998.

[33] Sardy, S., Bruce, A., and Tseng, P., Block coordinate relaxation methods for nonparametric wavelet denoising, J. Comput. Graph. Stat. 9 (2000), 361–379.

[34] Sardy, S. and Tseng, P., AMlet, RAMlet, and GAMlet: automatic nonlinear fitting of additive models, robust and generalized, with wavelets, J. Comput. Graph. Statist. 13 (2004), 283–309.

[35] Shi, W., Wahba, G., Wright, S. J., Lee, K., Klein, R., and Klein, B., Lasso-patternsearch algorithm with application to ophthalmology and genomic data, Stat. Interface 1 (2008), 137-153.

[36] Starck, J.-L. Nguyen, M., and Murtagh, F., Wavelets and curvelets for image deconvolution: a combined approach, Signal Processing 83 (2003), 2279-2283.

[37] Tropp, J. A., Just relax: Convex programming methods for identifying sparse signals, IEEE Trans. Info. Theory 51 (2006), 1030–1051.

[38] Tsaig, Y. and Donoho, D., Extensions of compressed sesning, Signal Processing 86 (2005), 533–548.

[39] Tseng, P. and Yun S., A coordinate gradient descent method for nonsmooth separable minimization, Math. Prog. 117 (2009), 387–423.

[40] Wang, L., Efficient Regularized Solution Path Algorithms with Applications in Machine Learning and Data Mining, PhD thesis, University of Michigan, 2008.

[41] Wright, S. J., Nowak, R., and Figueiredo, M., Sparse reconstruction by separable approximation, October 2007; to appear in IEEE Trans. Signal Proc.

	l1-ls	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
Gaussian	$n = 4096$, $m = 1024$, $\mu = 0.05\|A^T b\|_\infty$				
iterations	13	19	22	40	40
nnz(x)	376	229	198	310	342
obj value	3.288e+00	3.285e+00	3.285e+00	3.285e+00	3.283e+00
CPU time	6.8e+00	3.0e-01	3.4e-01	1.1e+00	1.1e+00
error	1.2e-01	1.4e-01	1.3e-01	1.4e-01	1.3e-01
iterations	13	16	17	31	36
nnz(x)	628	195	213	324	339
obj value	3.595e+00	3.593e+00	3.586e+00	3.590e+00	3.578e+00
CPU time	6.7e+00	2.6e-01	2.6e-01	8.3e-01	9.8e-01
error	1.3e-01	1.6e-01	1.5e-01	1.5e-01	1.4e-01
mean iterations	12	17	19	33	39
mean nnz(x)	474	214	215	302	323
mean CPU time	6.2e+00	2.6e-01	3.0e-01	8.8e-01	1.0e+00
mean error	1.3e-01	1.5e-01	1.5e-01	1.5e-01	1.4e-01
Gaussian	$n = 4096$, $m = 1024$, $\mu = 0.01\|A^T b\|_\infty$				
iterations	13	62	41	126	57
nnz(x)	491	573	527	640	690
obj value	6.879e-01	6.876e-01	6.876e-01	6.875e-01	6.867e-01
CPU time	8.5e+00	9.4e-01	6.1e-01	3.3e+00	1.6e+00
error	3.1e-02	4.2e-02	4.0e-02	4.0e-02	3.3e-02
iterations	13	55	39	156	57
nnz(x)	782	665	647	896	624
obj value	7.541e-01	7.537e-01	7.537e-01	7.528e-01	7.505e-01
CPU time	8.0e+00	8.4e-01	5.8e-01	4.1e+00	1.5e+00
error	3.7e-02	5.6e-02	5.4e-02	4.8e-02	3.5e-02
mean iterations	13	48	44	125	59
mean nnz(x)	559	480	602	600	616
mean CPU time	8.4e+00	7.3e-01	6.6e-01	3.3e+00	1.6e+00
mean error	3.3e-02	4.7e-02	4.5e-02	4.4e-02	3.5e-02
Gaussian	$n = 4096$, $m = 1024$, $\mu = 0.005\|A^T b\|_\infty$				
iterations	13	70	50	1001	58
nnz(x)	657	735	813	4090	1087
obj value	3.464e-01	3.460e-01	3.460e-01	6.316e-01	3.464e-01
CPU time	9.3e+00	1.0e+00	7.4e-01	2.6e+01	1.5e+00
error	2.2e-02	2.8e-02	2.7e-02	7.3e-01	3.1e-02
iterations	14	67	50	1001	59
nnz(x)	580	750	874	4089	770
obj value	3.785e-01	3.783e-01	3.784e-01	7.012e-01	3.783e-01
CPU time	9.4e+00	1.0e+00	7.5e-01	2.6e+01	1.6e+00
error	2.1e-02	2.9e-02	2.9e-02	7.1e-01	3.0e-02
mean iterations	13	57	53	1001	60
mean nnz(x)	569	670	859	4090	788
mean CPU time	9.8e+00	8.8e-01	8.0e-01	2.6e+01	1.6e+00
mean error	2.1e-02	2.8e-02	2.8e-02	7.3e-01	3.0e-02

Table 1: Comparing the CGD method using the Gauss-Southwell rules with l1-ls, GPSR-BB and FPC-BB on the problem (2) with A being a Gaussian matrix.

	l1-ls	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
Gaussian	$n = 8192$, $m = 2048$, $\mu = 0.05\|A^T b\|_\infty$				
iterations	12	17	21	26	33
nnz(x)	1007	394	432	615	639
obj value	9.310e+00	9.301e+00	9.293e+00	9.291e+00	9.276e+00
CPU time	1.9e+01	1.0e+00	1.2e+00	2.7e+00	3.5e+00
error	1.7e-01	2.0e-01	1.9e-01	1.9e-01	1.9e-01
iterations	13	18	23	28	41
nnz(x)	717	431	453	488	687
obj value	7.659e+00	7.653e+00	7.653e+00	7.643e+00	7.647e+00
CPU time	2.3e+01	1.0e+00	1.3e+00	2.9e+00	4.2e+00
error	1.4e-01	1.6e-01	1.6e-01	1.5e-01	1.6e-01
mean iterations	12	17	21	28	39
mean nnz(x)	842	399	437	574	629
mean CPU time	2.0e+01	9.2e-01	1.1e+00	2.8e+00	3.8e+00
mean error	1.4e-01	1.5e-01	1.5e-01	1.6e-01	1.5e-01
Gaussian	$n = 8192$, $m = 2048$, $\mu = 0.01\|A^T b\|_\infty$				
iterations	12	46	37	84	53
nnz(x)	1228	919	1009	1010	1138
obj value	1.979e+00	1.979e+00	1.977e+00	1.976e+00	1.972e+00
CPU time	2.5e+01	2.4e+00	1.9e+00	8.0e+00	5.1e+00
error	4.2e-02	6.2e-02	5.6e-02	5.4e-02	4.5e-02
iterations	14	57	54	104	67
nnz(x)	645	824	963	800	1276
obj value	1.610e+00	1.610e+00	1.610e+00	1.610e+00	1.610e+00
CPU time	3.3e+01	2.9e+00	2.8e+00	9.8e+00	6.5e+00
error	3.2e-02	4.1e-02	3.9e-02	3.6e-02	3.9e-02
mean iterations	13	39	45	111	57
mean nnz(x)	1026	800	1122	928	1204
mean CPU time	2.8e+01	2.0e+00	2.3e+00	1.1e+01	5.5e+00
mean error	3.4e-02	4.6e-02	4.5e-02	4.0e-02	3.7e-02
Gaussian	$n = 8192$, $m = 2048$, $\mu = 0.005\|A^T b\|_\infty$				
iterations	13	58	47	1001	57
nnz(x)	1020	1088	1467	8155	878
obj value	9.962e-01	9.956e-01	9.957e-01	1.854e+00	9.957e-01
CPU time	3.1e+01	2.9e+00	2.4e+00	9.3e+01	5.5e+00
error	2.4e-02	3.4e-02	3.3e-02	7.0e-01	3.3e-02
iterations	13	64	60	1001	64
nnz(x)	1339	1235	1754	8173	1574
obj value	8.134e-01	8.121e-01	8.121e-01	1.488e+00	8.124e-01
CPU time	3.2e+01	3.3e+00	3.1e+00	9.5e+01	6.2e+00
error	2.4e-02	3.1e-02	3.0e-02	7.4e-01	3.4e-02
mean iterations	13	48	53	1001	58
mean nnz(x)	1245	1206	1704	8175	1588
mean CPU time	3.2e+01	2.5e+00	2.8e+00	9.4e+01	5.6e+00
mean error	2.3e-02	2.8e-02	2.8e-02	7.1e-01	3.1e-02

Table 2: Comparing the CGD method using the Gauss-Southwell rules with l1-ls, GPSR-BB and FPC-BB on the problem (2) with A being a Gaussian matrix.

	l1-ls	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
partial DCT	$n = 4096$, $m = 1024$, $\mu = 0.01\|A^T b\|_\infty$				
iterations	13	20	36	121	56
nnz(x)	423	306	503	458	578
obj value	7.056e-01	7.054e-01	7.052e-01	7.054e-01	7.043e-01
CPU time	1.2e+00	9.0e-02	1.4e-01	4.0e-01	1.9e-01
error	2.8e-02	3.6e-02	3.6e-02	3.6e-02	3.0e-02
iterations	13	26	39	141	63
nnz(x)	417	392	576	576	586
obj value	6.878e-01	6.876e-01	6.877e-01	6.873e-01	6.864e-01
CPU time	1.2e+00	1.2e-01	1.6e-01	4.6e-01	2.0e-01
error	2.7e-02	3.7e-02	3.8e-02	3.4e-02	2.9e-02
mean iterations	12	41	39	109	58
mean nnz(x)	585	454	530	539	603
mean CPU time	1.1e+00	1.7e-01	1.6e-01	3.6e-01	1.9e-01
mean error	3.3e-02	4.6e-02	4.5e-02	4.5e-02	3.4e-02
partial DCT	$n = 4096$, $m = 1024$, $\mu = 0.005\|A^T b\|_\infty$				
iterations	13	28	41	1001	56
nnz(x)	711	537	800	4086	1086
obj value	3.556e-01	3.547e-01	3.550e-01	6.779e-01	3.554e-01
CPU time	1.2e+00	1.2e-01	1.7e-01	3.3e+00	1.8e-01
error	2.1e-02	2.3e-02	2.6e-02	7.0e-01	2.9e-02
iterations	14	35	47	1001	65
nnz(x)	489	675	776	4089	750
obj value	3.458e-01	3.457e-01	3.456e-01	6.541e-01	3.458e-01
CPU time	1.4e+00	1.5e-01	2.0e-01	3.3e+00	2.1e-01
error	1.7e-02	2.3e-02	2.3e-02	7.3e-01	2.4e-02
mean iterations	13	50	48	1001	59
mean nnz(x)	616	641	824	4090	899
mean CPU time	1.4e+00	2.3e-01	2.1e-01	3.5e+00	2.0e-01
mean error	2.1e-02	2.7e-02	2.7e-02	7.1e-01	3.0e-02

Table 3: Comparing the CGD method using the Gauss-Southwell rules with l1-ls, GPSR-BB and FPC-BB on the problem (2) with A being a partial DCT matrix.

	l1-ls	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
partial DCT	$n = 8192$, $m = 2048$, $\mu = 0.01\|A^T b\|_\infty$				
iterations	13	38	40	154	58
nnz(x)	847	785	945	763	1174
obj value	1.454e+00	1.454e+00	1.453e+00	1.451e+00	1.451e+00
CPU time	2.5e+00	3.4e-01	3.6e-01	1.2e+00	4.0e-01
error	2.9e-02	4.0e-02	3.8e-02	3.1e-02	3.2e-02
iterations	13	31	43	168	60
nnz(x)	817	593	900	1029	1151
obj value	1.567e+00	1.566e+00	1.566e+00	1.566e+00	1.564e+00
CPU time	2.5e+00	2.7e-01	3.5e-01	1.3e+00	3.9e-01
error	3.1e-02	3.9e-02	4.1e-02	3.7e-02	3.5e-02
mean iterations	13	46	46	119	58
mean nnz(x)	912	830	1090	917	1215
mean CPU time	2.6e+00	4.2e-01	4.1e-01	9.4e-01	4.3e-01
mean error	3.3e-02	4.4e-02	4.3e-02	3.8e-02	3.6e-02
partial DCT	$n = 8192$, $m = 2048$, $\mu = 0.005\|A^T b\|_\infty$				
iterations	13	46	47	1001	57
nnz(x)	1361	1142	1464	8182	2178
obj value	7.328e-01	7.312e-01	7.312e-01	1.355e+00	7.326e-01
CPU time	2.6e+00	4.2e-01	4.2e-01	7.9e+00	4.3e-01
error	2.1e-02	2.6e-02	2.5e-02	7.0e-01	3.0e-02
iterations	13	39	51	1001	61
nnz(x)	1239	901	1381	8178	1972
obj value	7.899e-01	7.882e-01	7.884e-01	1.486e+00	7.896e-01
CPU time	2.8e+00	3.5e-01	4.6e-01	7.9e+00	4.7e-01
error	2.2e-02	2.6e-02	2.7e-02	7.0e-01	3.2e-02
mean iterations	13	55	54	1001	60
mean nnz(x)	1204	1199	1584	8179	1593
mean CPU time	2.9e+00	5.2e-01	4.9e-01	7.9e+00	4.6e-01
mean error	2.2e-02	2.8e-02	2.8e-02	7.2e-01	3.1e-02

Table 4: Comparing the CGD method using the Gauss-Southwell rules with l1-ls, GPSR-BB and FPC-BB on the problem (2) with A being a partial DCT matrix.

	l1-ls	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
Gaussian	$n = 4096$, $m = 1024$, $\mu = 0.05\|A^T b\|_\infty$				
iterations	17	10	12	15	20
nnz(x)	102	68	69	90	108
obj value	1.019e+02	1.019e+02	1.019e+02	1.018e+02	1.018e+02
CPU time	3.9e+00	3.1e-01	2.1e-01	4.8e-01	6.0e-01
error	1.3e-01	1.4e-01	1.4e-01	1.4e-01	1.4e-01
iterations	16	8	10	24	22
nnz(x)	141	80	77	152	133
obj value	8.878e+01	8.849e+01	8.853e+01	8.857e+01	8.838e+01
CPU time	3.6e+00	1.3e-01	1.5e-01	6.3e-01	5.8e-01
error	1.1e-01	1.2e-01	1.2e-01	1.3e-01	1.3e-01
mean iterations	16	9	11	22	21
mean nnz(x)	125	77	75	121	124
mean CPU time	3.8e+00	1.6e-01	1.8e-01	6.0e-01	5.8e-01
mean error	1.2e-01	1.3e-01	1.3e-01	1.2e-01	1.3e-01
Gaussian	$n = 4096$, $m = 1024$, $\mu = 0.01\|A^T b\|_\infty$				
iterations	14	12	14	50	31
nnz(x)	182	112	111	166	345
obj value	2.209e+01	2.205e+01	2.205e+01	2.204e+01	2.202e+01
CPU time	4.8e+00	2.0e-01	2.2e-01	1.3e+00	8.5e-01
error	3.4e-02	3.7e-02	3.8e-02	3.5e-02	4.0e-02
iterations	14	10	12	53	32
nnz(x)	217	110	111	328	355
obj value	1.896e+01	1.895e+01	1.891e+01	1.895e+01	1.888e+01
CPU time	4.9e+00	1.7e-01	1.9e-01	1.4e+00	8.6e-01
error	2.9e-02	3.4e-02	3.0e-02	4.2e-02	3.3e-02
mean iterations	14	11	13	56	32
mean nnz(x)	208	113	114	306	347
mean CPU time	5.2e+00	1.9e-01	2.2e-01	1.5e+00	8.9e-01
mean error	3.1e-02	3.3e-02	3.3e-02	3.8e-02	3.5e-02
Gaussian	$n = 4096$, $m = 1024$, $\mu = 0.005\|A^T b\|_\infty$				
iterations	13	13	15	92	33
nnz(x)	323	114	113	616	497
obj value	1.121e+01	1.119e+01	1.119e+01	1.115e+01	1.117e+01
CPU time	5.8e+00	2.1e-01	2.4e-01	2.4e+00	8.9e-01
error	2.3e-02	2.6e-02	2.5e-02	2.4e-02	2.9e-02
iterations	14	13	14	117	35
nnz(x)	264	138	138	515	508
obj value	9.568e+00	9.553e+00	9.561e+00	9.558e+00	9.558e+00
CPU time	6.2e+00	2.2e-01	2.2e-01	3.1e+00	9.5e-01
error	1.9e-02	1.9e-02	2.0e-02	2.4e-02	2.5e-02
mean iterations	13	13	15	103	34
mean nnz(x)	297	128	145	516	648
mean CPU time	6.1e+00	2.1e-01	2.4e-01	2.7e+00	9.2e-01
mean error	2.0e-02	2.1e-02	2.2e-02	2.7e-02	2.9e-02

Table 5: Comparing the CGD method using the Gauss-Southwell rules with l1-ls, GPSR-BB and FPC-BB on the problem (2) with A being a Gaussian matrix, the original solution has nonzero components with various magnitudes.

		l1-ls(10^{-8})	l1-ls(10^{-2})	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
Gaussian		$n = 4096$, $m = 1024$, $\mu = 0.05\|A^T b\|_\infty$					
#1	mean error	1.2e-01	1.3e-01	1.5e-01	1.5e-01	1.5e-01	1.4e-01
	mean error-L	1.2e-01	1.3e-01	1.5e-01	1.5e-01	1.4e-01	1.4e-01
#2	mean error	1.2e-01	1.2e-01	1.3e-01	1.3e-01	1.2e-01	1.3e-01
	mean error-L	8.1e-02	8.3e-02	8.6e-02	8.7e-02	8.9e-02	9.1e-02
Gaussian		$n = 4096$, $m = 1024$, $\mu = 0.01\|A^T b\|_\infty$					
#1	mean error	2.6e-02	3.3e-02	4.7e-02	4.5e-02	4.4e-02	3.5e-02
	mean error-L	2.6e-02	3.3e-02	4.4e-02	4.3e-02	4.2e-02	3.4e-02
#2	mean error	2.7e-02	3.1e-02	3.3e-02	3.3e-02	3.8e-02	3.5e-02
	mean error-L	1.7e-02	2.0e-02	2.0e-02	2.1e-02	2.4e-02	2.2e-02
Gaussian		$n = 4096$, $m = 1024$, $\mu = 0.005\|A^T b\|_\infty$					
#1	mean error	1.6e-02	2.1e-02	2.8e-02	2.8e-02	7.3e-01	3.0e-02
	mean error-L	1.6e-02	2.0e-02	2.6e-02	2.6e-02	6.4e-01	2.8e-02
#2	mean error	1.7e-02	2.0e-02	2.1e-02	2.2e-02	2.7e-02	2.9e-02
	mean error-L	1.0e-02	1.2e-02	1.2e-02	1.3e-02	1.7e-02	1.8e-02

Table 6: Comparing the errors for the problem (2) with A being a Gaussian matrix and the original solution having nonzero components with same magnitude with those for the problem (2) with A being a Gaussian matrix and the original solution having nonzero components with various magnitudes.

#1: experiments in the Table 1, #2: experiments in the Table 5

Problem	**blur kernel**	σ^2
1	9×9 uniform	0.56^2
2	$h_{ij} = 1/(i^2 + j^2)$	2
3	$h_{ij} = 1/(i^2 + j^2)$	8

Table 7: Image Deconvolution Experiments

	IST	CGD-GS-q	CGD-GS-r	GPSR-BB	FPC-BB
image	$n = 256^2$, $m = 256^2$, $\rho = 0.35$, experiment 1				
iterations	60	35	29	20	205
nnz(x)	8326	4449	6100	8944	4009
obj value	4.551e+05	4.549e+05	4.541e+05	4.544e+05	4.333e+05
CPU time	5.3e+00	2.9e+00	2.3e+00	1.5e+00	1.4e+01
error	1.2e-01	1.4e-01	1.4e-01	1.2e-01	1.2e-01
image	$n = 256^2$, $m = 256^2$, $\rho = 0.35$, experiment 2				
iterations	52	33	26	19	156
nnz(x)	12640	7823	10495	13369	8599
obj value	5.360e+05	5.351e+05	5.347e+05	5.350e+05	5.187e+05
CPU time	4.6e+00	2.6e+00	2.4e+00	1.3e+00	1.0e+01
error	8.5e-02	1.0e-01	1.0e-01	8.4e-02	8.6e-02
image	$n = 256^2$, $m = 256^2$, $\rho = 0.35$, experiment 3				
iterations	48	33	25	16	167
nnz(x)	17726	10070	13602	19449	11473
obj value	7.081e+05	7.065e+05	7.070e+05	7.076e+05	6.760e+05
CPU time	4.2e+00	2.6e+00	2.2e+00	1.1e+00	1.1e+01
error	8.8e-02	1.1e-01	1.0e-01	8.8e-02	9.7e-02

Table 8: Comparing the CGD method using the Gauss-Southwell rules with IST, GPSR-BB and FPC-BB on the three experiments from Table (7).

	l1-logreg (10^{-4})	CGD-GS-q	CGD-GS-r	SpaRSA
leu	$n = 7130$, $m = 38$, $\mu = 0.01\mu_{\max}$			
iterations	25	92	146	fail
obj value	3.14477e-02	3.14458e-02	3.14212e-02	
CPU time	1.1e+00	6.6e-01	1.0e+00	
leu	$n = 7130$, $m = 38$, $\mu = 0.001\mu_{\max}$			
iterations	37	54	207	fail
obj value	4.44821e-03	4.44633e-03	4.44361e-03	
CPU time	1.4e+00	4.1e-01	1.5e+00	
rcv1	$n = 47237$, $m = 20242$, $\mu = 0.01\mu_{\max}$			
iterations	28	118	105	191
obj value	2.12420e-01	2.12420e-01	2.12420e-01	2.12420e-01
CPU time	1.1e+01	7.5e+00	6.9e+00	7.5e+00
rcv1	$n = 47237$, $m = 20242$, $\mu = 0.001\mu_{\max}$			
iterations	43	484	312	676
obj value	6.91300e-02	6.91300e-02	6.91294e-02	6.91298e-02
CPU time	3.0e+01	3.4e+01	2.3e+01	2.7e+01
real-sim	$n = 20959$, $m = 72309$, $\mu = 0.01\mu_{\max}$			
iterations	22	71	72	88
obj value	2.14289e-01	2.14289e-01	2.14287e-01	2.14289e-01
CPU time	1.4e+01	7.5e+00	7.9e+00	7.6e+00
real-sim	$n = 20959$, $m = 72309$, $\mu = 0.001\mu_{\max}$			
iterations	21	185	214	232
obj value	8.70759e-02	8.70755e-02	8.70744e-02	8.70752e-02
CPU time	2.1e+01	1.9e+01	2.3e+01	2.0e+01

Table 9: Comparing the CGD method using the Gauss-Southwell rules with l1-logreg on three benchmark problems.

	l1-logreg 10^{-4}	CGD-GS-q 10^{-6}	CGD-GS-r 10^{-6}
random	$n = 10001$, $m = 1000$, $\mu = 0.1\mu_{\max}$		
iterations	23	205	218
obj value	2.10391e-01	2.10317e-01	2.10317e-01
CPU time	2.8e+02	1.1e+01	1.1e+01
iterations	23	182	231
obj value	2.07347e-01	2.07274e-01	2.07274e-01
CPU time	2.8e+02	9.5e+00	1.2e+01
mean iterations	23	191	203
mean CPU time	2.8e+02	1.0e+01	1.1e+01
random	$n = 10001$, $m = 1000$, $\mu = 0.01\mu_{\max}$		
iterations	20	239	236
obj value	3.41319e-02	3.40723e-02	3.40723e-02
CPU time	2.4e+02	1.2e+01	1.2e+01
iterations	20	242	286
obj value	3.35281e-02	3.34698e-02	3.34699e-02
CPU time	2.5e+02	1.3e+01	1.5e+01
mean iterations	20	230	260
mean CPU time	2.4e+02	1.2e+01	1.4e+01
random	$n = 1001$, $m = 100$, $\mu = 0.1\mu_{\max}$		
iterations	19	114	132
obj value	2.24251e-01	2.24178e-01	2.24178e-01
CPU time	2.1e-01	1.6e-01	1.7e-01
iterations	19	148	200
obj value	2.17554e-01	2.17489e-01	2.17489e-01
CPU time	2.4e-01	1.8e-01	2.4e-01
mean iterations	19	128	180
mean CPU time	2.3e-01	1.8e-01	2.4e-01
random	$n = 1001$, $m = 100$, $\mu = 0.01\mu_{\max}$		
iterations	16	191	192
obj value	3.73507e-02	3.72936e-02	3.72936e-02
CPU time	1.8e-01	2.2e-01	2.3e-01
iterations	16	219	253
obj value	3.56790e-02	3.56274e-02	3.56274e-02
CPU time	2.0e-01	2.7e-01	3.3e-01
mean iterations	17	187	220
mean CPU time	1.9e-01	2.4e-01	2.9e-01

Table 10: Comparing the CGD method using the Gauss-Southwell rules with l1-logreg on the randomly generated problems.

	l1-logreg 10^{-4}	CGD-GS-q 10^{-6}	CGD-GS-r 10^{-6}
random	$n = 1001$, $m = 10000$, $\mu = 0.1\mu_{\max}$		
iterations	19	74	78
obj value	2.17625e-01	2.17562e-01	2.17562e-01
CPU time	2.8e+02	4.2e+00	4.4e+00
iterations	19	60	64
obj value	2.14949e-01	2.14887e-01	2.14887e-01
CPU time	2.8e+02	3.3e+00	3.5e+00
mean iterations	19	68	75
mean CPU time	2.8e+02	3.8e+00	4.3e+00
random	$n = 1001$, $m = 10000$, $\mu = 0.01\mu_{\max}$		
iterations	16	84	99
obj value	3.61274e-02	3.60768e-02	3.60768e-02
CPU time	2.3e+02	4.6e+00	5.5e+00
iterations	16	64	65
obj value	3.56762e-02	3.56267e-02	3.56267e-02
CPU time	2.3e+02	4.5e+00	5.6e+00
mean iterations	16	82	88
mean CPU time	2.3e+02	4.8e+00	5.2e+00
random	$n = 101$, $m = 1000$, $\mu = 0.1\mu_{\max}$		
iterations	17	44	36
obj value	2.48722e-01	2.48698e-01	2.48698e-01
CPU time	2.3e-01	4.0e-02	4.0e-02
iterations	17	36	40
obj value	2.45341e-01	2.45321e-01	2.45321e-01
CPU time	2.2e-01	4.0e-02	5.0e-02
mean iterations	17	41	45
mean CPU time	2.2e-01	4.5e-02	5.9e-02
random	$n = 101$, $m = 1000$, $\mu = 0.01\mu_{\max}$		
iterations	14	54	51
obj value	4.53546e-02	4.53336e-02	4.53336e-02
CPU time	2.0e-01	5.0e-02	6.0e-02
iterations	13	63	67
obj value	4.40478e-02	4.40181e-02	4.40181e-02
CPU time	2.2e-01	7.0e-02	8.0e-02
mean iterations	14	60	66
mean CPU time	1.9e-01	6.8e-02	9.0e-02

Table 11: Comparing the CGD method using the Gauss-Southwell rules with l1-logreg on the randomly generated problems.