A Primer on Coordinate Descent Algorithms

Hao-Jun Michael Shi
Department of Industrial Engineering and Management Sciences
Northwestern University
hjmshi@u.northwestern.edu

Shenyinying Tu
Department of Industrial Engineering and Management Sciences
Northwestern University
ShenyinyingTu2021@u.northwestern.edu

Yangyang Xu
Department of Mathematics
University of Alabama
yangyang.xu@ua.edu

Wotao Yin
Department of Mathematics
University of California, Los Angeles
wotaoyin@math.ucla.edu

January 11, 2017

Contents

Abstract

This monograph presents a class of algorithms called coordinate descent algorithms for mathematicians, statisticians, and engineers outside the field of optimization. This particular class of algorithms has recently gained popularity due to their effectiveness in solving large-scale optimization problems in machine learning, compressed sensing, image processing, and computational statistics. Coordinate descent algorithms solve optimization problems by successively minimizing along each coordinate or coordinate hyperplane, which is ideal for parallelized and distributed computing. Avoiding detailed technicalities and proofs, this monograph gives relevant theory and examples for practitioners to effectively apply coordinate descent to modern problems in data science and engineering.

To keep the primer up-to-date, we intend to publish this monograph only after no additional topics need to be added and we foresee no further major advances in the area.

1 Introduction

1.1 Overview

This monograph discusses a class of algorithms, called *coordinate descent* (CD) algorithms, which is useful in solving large-scale optimization problems with smooth or non-smooth and convex or non-convex objective functions. Although these methods have existed since the early development of the discipline and the optimization community did not emphasize them until recently, various modern applications in machine learning, compressed sensing, and large-scale computational statistics have yielded new problems well suited for CD algorithms. These methods are generally applicable to a variety of problems involving large or high-dimensional data sets since they naturally break down complicated optimization problems into simpler subproblems, which are easily parallelized or distributed. For some structured problems, CD has been shown to perform faster than traditional algorithms, such as gradient descent. In addition, CD is generally applicable to non-convex problems and are easier to understand than splitting methods such as the Alternating Direction Method of Multipliers in this aspect. Also, few assumptions are needed to prove convergence to minima for convex problems and stationary points for non-convex problems. In fact, certain CD variants have also been shown to converge for non-convex functions with fairly loose properties.

CD algorithms follow the universal approach to algorithmic, numerical optimization: solving an optimization problem by solving a sequence of simpler subproblems. Each iterate is found by fixing most components of the variable vector $\mathbf{x}$ at their current values and approximately minimizing the objective function with the remaining chosen components. In this monograph, we will explore a variety of interesting variants, extensions, and applications of CD connected to many different topics in optimization, statistics, and applied mathematics.

1.2 Formulations

We will consider the following general unconstrained minimization problem:

$$\underset{\mathbf{x}}{\text{minimize}}\, f(\mathbf{x}) = f(x_1, \ldots, x_n) \tag{1}$$

where $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ and the function $f : \mathbb{R}^n \to \mathbb{R}$ is continuous. Further assumptions may be made on the structure of f, such as convexity, Lipschitz continuity, differentiability, etc., while discussing theoretical guarantees for specific algorithms.

In addition, we will consider the following structured problem:

$$\underset{\mathbf{x}}{\text{minimize}}\, F(\mathbf{x}) = f(\mathbf{x}) + \sum_{i=1}^{n} r_i(x_i) \tag{2}$$

where f is differentiable, and r_i's are extended-valued and possibly nondifferentiable functions. Problems appearing in many recent applications such as compressed sensing, statistical variable selection, and model selection can be formulated in the form of (2). Since we allow each r_i to be extended-valued, it can model constraints on x_i by including an indicator function, as discussed in Appendix A. The function r_i can also include certain regularization terms to promote the structure of solutions, such as sparsity and low-rankness. We will further generalize the coordinate separable function r_i's to block separable ones in Section 2.1.

1.3 Framework of Coordinate Descent

The basic coordinate descent (CD) framework for (1) and (2) is shown in Algorithm 1. At each iteration, we choose one component x_{i_k} and adjust it by a certain update scheme while holding all other components fixed.

Algorithm 1 Coordinate Descent

1: Set $k = 0$ and initialize $\mathbf{x}^0 \in \mathbb{R}^n$;
2: **repeat**
3: Choose index $i_k \in \{1, 2, ..., n\}$;
4: Update x_{i_k} to $x_{i_k}^k$ by a certain scheme depending on x^{k-1} and f or F;
5: Keep x_j unchanged, i.e., $x_j^k = x_j^{k-1}$, for all $j \neq i_k$;
6: Let $k = k + 1$;
7: **until** termination condition is satisfied;

Intuitively, CD methods are easily visualized, particularly in the 2-dimensional case. Rather than moving all coordinates along a descent direction, CD changes a chosen coordinate at each iterate, moving as if it were on a grid, with each axis corresponding to each component. Figure 1 illustrates this process, where the coordinate minimization scheme is applied.

Within this framework, there are many different approaches for choosing an index and updating the selected coordinate. These various rules and updates affect the convergence properties of CD on different types of problems. They may exploit problem-specific structures, such as sparsity. They may also perform differently depending on the conditioning of the problem or how much each coordinate subproblem depends on one another.

The most natural approach to choosing an index is to select components cyclically, i.e. $i_0 = 1$, $i_1 = 2$, $i_2 = 3$, and so on. Alternatively, we can select a component at random at each iteration (not necessarily with equal probability). Lastly, we can choose components greedily, choosing the component corresponding to the greatest descent, strongest descent potential, or other scores, at the current iteration. The index rule may also satisfy an *essentially cyclic* condition, in which every component is guaranteed to be updated at least once within every N iterations. For example, we can perform a cyclic choice of components that are randomly shuffled after each cycle.

Regarding the update schemes, we can simply renew the selected component x_{i_k} by minimizing the objective with respect to x_{i_k} while fixing the remaining ones. Specifically, for problem (1), we can perform the update:

$$x_{i_k}^k = \arg\min_{x_{i_k}} f(x_1^{k-1}, \ldots, x_{i_k-1}^{k-1}, x_{i_k}, x_{i_k+1}^{k-1}, \ldots, x_n^{k-1}), \tag{3}$$

and for problem (2), one can have a similar update by replacing f with F. Other update schemes can be performed if the problem has more structure. Suppose f is differentiable in (1), then one can apply coordinate-gradient descent along each chosen component, i.e.

$$x_{i_k}^k = x_{i_k}^{k-1} - \alpha_{i_k} \nabla_{i_k} f(\mathbf{x}^{k-1}) \tag{4}$$

where α_{i_k} is a step size that can be set by line search or according to the property of f.

The scheme in (4) can be easily extended to solving (2) as

$$x_{i_k}^k = x_{i_k}^{k-1} - \alpha_{i_k} (\nabla_{i_k} f(\mathbf{x}^{k-1}) + \tilde{\nabla} r_{i_k}(x_{i_k}^{k-1})) \tag{5}$$

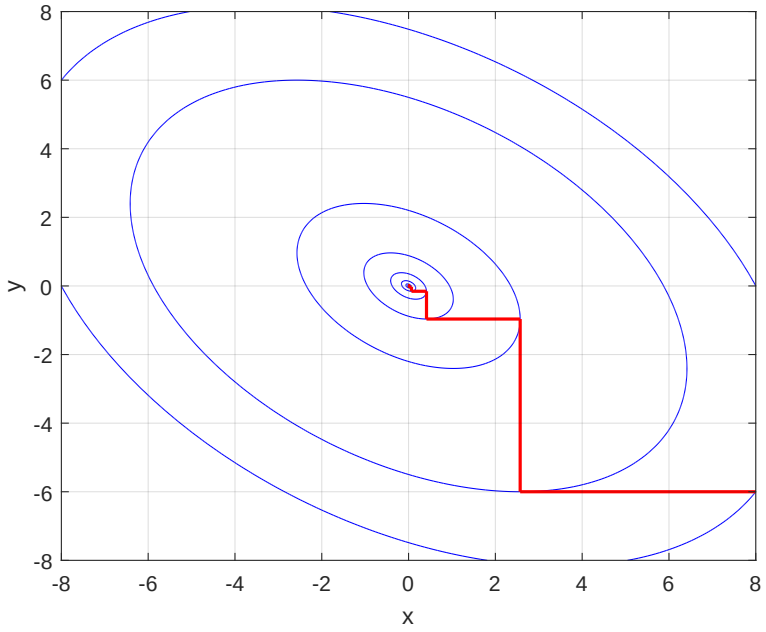

Figure 1: CD applied on the quadratic function $f(x, y) = 7x^2 + 6xy + 8y^2$ with initial point $(8, -6)$. It minimizes f alternatingly with respect to one of x and y while fixing the other. The blue curves correspond to different level curves of the function.

where $\tilde{\nabla} r_{i_k}(x_{i_k}^{k-1})$ is a subgradient of r_{i_k} at $x_{i_k}^{k-1}$. In addition, *proximal* or *prox-linear updates* can handle (2) when r_i's are not differentiable. These updates minimize a surrogate function that dominates the original objective around the current iterate. The proximal update uses as the surrogate function the sum of the original function and a proximal term, and the prox-linear update employs a surrogate function to be the linearization of the differentiable part plus a proximal term and the nondifferentiable function. They both involve the *proximal operator*, which for the function αf is defined as

$$\text{prox}_{\alpha f}(\mathbf{y}) = \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{2\alpha}\|\mathbf{x} - \mathbf{y}\|_2^2.$$

For many functions with favorable structures, the proximal operator is cheap to compute. We call these functions *proximable functions*. Please refer to Appendix C for more detail.

If we are minimizing the sum, or the average, of a vast number of functions, we can also apply *stochastic updates*, which use sample gradients computed by selecting one or a few of these functions at each update instead of the exact gradient. We will discuss these alternative update schemes further in Section 2.

The step size α_{i_k} in (4) and (5) can also be chosen in many fashions. The choice of stepsize is important because a descent direction is not sufficient to guarantee descent. If α_{i_k} is too large, the function value may increase; if α_{i_k} is too small, the algorithm will converge at a relatively slow rate. To select the step size α_{i_k}, we may perform an exact line search along the i_kth component, use traditional, backtracking line search methods to obtain sufficient descent, which may be more economical for certain separable objectives, or make predefined choices of α_{i_k} based on known properties of f.

Useful examples that shed light on the performance of CD on differently structured problems are given in Section 4. We will also discuss related *coordinate friendly* analysis and theory, as well as useful heuristics, applied to various problems in Section 3.

1.4 Other Surveys

Coordinate descent algorithms have existed since the formation of the discipline. In response to the rising interest in large-scale optimization, a few articles have recently surveyed this class of algorithms. Wright [83] gives an in-depth review of coordinate descent algorithms, including convergence theory, which we highly recommend for optimizers and practitioners with a stronger background in optimization. Lange [34] provides a survey of optimization algorithms for statistics, including block coordinate descent algorithms.

We emphasize that this paper is specifically targeted towards engineers, scientists, and mathematicians outside of the optimization field, who may not have the requisite knowledge in optimization to understand research articles in this area. Because of this, we do not present the theory of coordinate descent algorithms in a formal manner but emphasize performance on real-world applications. In addition, we avoid discussing specific parallelized and distributed coordinate method implementations in detail since this remains an active area of research and conclusions cannot be drawn without discussing many implementation aspects. We instead give a list of possible approaches toward developing parallelized and distributed algorithms. We believe that the contents of this paper may serve as a guide and toolbox for practitioners to apply coordinate descent to more problems and applications in the future.

1.5 Outline

In Section 2, we give different classes of variants, including different update schemes, indexing schemes, as well as introduce the more generalized block coordinate formulation. In Section 3, we give the relevant theory to analyze problems in practice for CD, and discuss useful heuristics to exploit *coordinate friendly* structures. In Section 4, we describe modern applications of CD in engineering and data science. In Section 5, we give resources for parallelizing CD for solving large-scale systems. Lastly, in Section 6, we summarize our results from this monograph.

1.6 Notation

We introduce some notation before proceeding. Let L be the gradient Lipschitz constant of the differentiable part f, i.e. for any $\mathbf{x}, \mathbf{y}$, it holds that

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\|_2 \leq L\|\mathbf{x} - \mathbf{y}\|_2.$$

Let L_i denote the block-wise gradient Lipschitz constant, i.e. for any $\mathbf{x}, \mathbf{y}$,

$$\|\nabla_i f(\mathbf{x}_1, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_s) - \nabla_i f(\mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{y}_i, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_s)\|_2 \leq L_i\|\mathbf{x}_i - \mathbf{y}_i\|_2,$$

where note that L_i may depend on the value of $\mathbf{x}_j$ for all $j \neq i$. In addition we introduce the notation

$$f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}) = f(\mathbf{x}_1, ..., \mathbf{x}_{i_k}, ..., \mathbf{x}_s)$$

when the i_kth block is chosen.

2 Algorithm Variants and Implementations

A wide range of implementation variants of CD have been developed for a large variety of applications. We discuss variants on the classic CD method introduced in Section 1 and describe their strengths and weaknesses below.

2.1 Block Coordinate Descent

Up until this point, we have only considered the method that updates one component of the variable $\mathbf{x}$ at each iterate. We may generalize these coordinate updates to block coordinate updates. This method is particularly useful for applications with variables partitioned into blocks, such as non-negative matrix/tensor factorization (e.g., see [10]), group LASSO [89], and many distributed computing problems, where blocks of variables naturally appear.

Consider the following optimization problem:

$$\underset{\mathbf{x}}{\text{minimize}}\, F(\mathbf{x}) = f(\mathbf{x}_1, ..., \mathbf{x}_s) + \sum_{i=1}^{s} r_i(\mathbf{x}_i) \tag{6}$$

where $\mathbf{x} \in \mathbb{R}^n$ is decomposed into s block variables $\mathbf{x}_1, ..., \mathbf{x}_s$, f is differentiable, and r_i for $i = 1, ..., s$ are extended-valued and possibly nondifferentiable functions. Note that if we consider the block formed by each component, we obtain formulation (2). We may similarly adjust formulation (1) for block variables.

From now on, we consider formulation (6) since it is a generalization of formulation (2). We can simply modify Algorithm 1 to fit this block structured problem, and the modified method is dubbed as Block Coordinate Descent (BCD), given in Algorithm 2.

Algorithm 2 Block Coordinate Descent

1: Set $k = 0$ and choose $\mathbf{x}^0 \in \mathbb{R}^n$;
2: **repeat**
3: Choose index $i_k \in \{1, 2, ..., s\}$;
4: Update $\mathbf{x}_{i_k}$ to $\mathbf{x}_{i_k}^k$ by a certain scheme depending on $\mathbf{x}^{k-1}$ and F;
5: Keep $\mathbf{x}_j^k = \mathbf{x}_j^{k-1}$ for $j \neq i_k$;
6: Let $k = k + 1$
7: **until** termination condition is satisfied;

Rather than updating a chosen coordinate at each iterate, block CD seeks to renew a chosen block of coordinates while other blocks are fixed. This method lends itself well for distributed or parallel computing since the update of a block coordinate is typically cheaper than that of all block variables. This will be discussed further in Section 3.

Block CD is also a generalization of the alternating minimization method that has been applied to a variety of problems, and also the expectation-maximization (EM) algorithm [15], that performs essentially a 2-block CD.

2.2 Update Schemes

As that done in (3), one can simply update $\mathbf{x}_{i_k}$ by minimizing F with respect to $\mathbf{x}_{i_k}$ while fixing the remaining block variables. However, this update scheme can be hard since its corresponding subproblem may be difficult to solve exactly. In addition, BCD with this update scheme may not converge for some non-smooth and/or non-convex problems. This deficiency motivates the introduction of alternative update schemes that may give easier subproblems and ensure the convergence of the algorithm.

All of the update schemes are summarized below:

1. *(Block) Coordinate Minimization*:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}^{k-1}) + r_{i_k}(\mathbf{x}_{i_k});$$

2. *(Block) Proximal Point Update*:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}^{k-1}) + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2 + r_{i_k}(\mathbf{x}_{i_k});$$

3. *(Block) Proximal Linear Update (Prox-Linear)*:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}^{k-1}) + \langle \nabla_{i_k} f(\mathbf{x}_{i_k}^{k-1}, \mathbf{x}_{\neq i_k}^{k-1}), \mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1} \rangle + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2 + r_{i_k}(\mathbf{x}_{i_k});$$

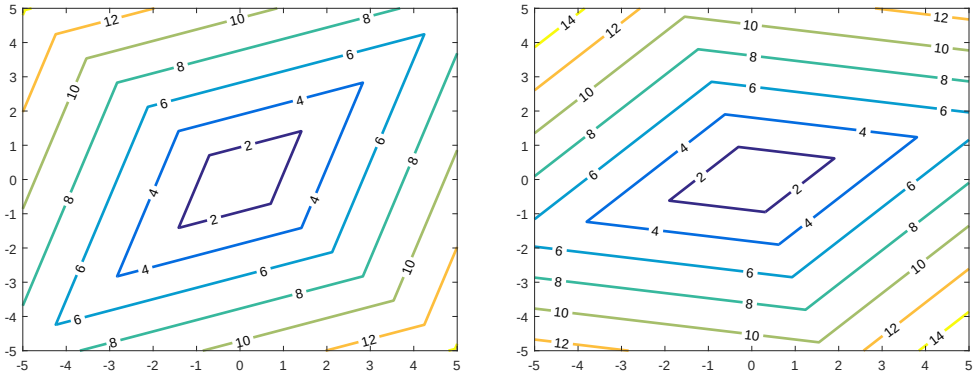

Figure 2: The function $\ell(x, y) = |x| + 2|y|$ is a separable, non-smooth, convex function. The left side is a $\pi/4$-radian rotation of ℓ for which CD gets stuck at a non-stationary point using any of the presented updates. The right side is a $\pi/10$-radian rotation of ℓ for which CD can correctly minimize.

where in the proximal point update, the step size $\alpha_{i_k}^{k-1}$ can be any bounded positive number, and in the prox-linear update, the step size $\alpha_{i_k}^{k-1}$ can be set to $1/L_{i_k}^{k-1}$.

Since each update scheme solves a different subproblem, the updates may generate different sequences that converge to different solutions. The coordinate minimization, proximal point, and prox-linear updates may be interpreted as minimizing a surrogate function that upper bounds the original objective function when α_{i_k} is chosen appropriately, as noted in the BSUM algorithm [29]. It is important to understand the nuances of each scheme to apply the correct variant for a given application. We describe each update scheme in more detail below.

2.2.1 Block Coordinate Minimization

A natural way to update the selected block $\mathbf{x}_{i_k}$ is to minimize the objective with respect to $\mathbf{x}_{i_k}$ with all other blocks fixed, i.e., by the block coordinate minimization scheme:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}^{k-1}) + r_{i_k}(\mathbf{x}_{i_k}). \tag{7}$$

This classic scheme is most intuitive and was first introduced in [27] and further analyzed in [4, 16, 23, 42, 78, 81]. BCD with this scheme is guaranteed to converge to a stationary point when the objective is convex, continuously differentiable, and strictly convex on each coordinate. However, BCD may not converge for some nonconvex problems. Powell [56] gives an example for which cyclic BCD with the update scheme in (7) fails to converge to a stationary point. Although the block minimization scheme is most easily accessible and intuitive, alternative update schemes can have greater stability, better convergence properties, or easier subproblems. For Powell's example, BCD with the proximal point or prox-linear update schemes does converge.

Warga [81] provides a convex but non-smooth example, $f(x, y) = |x - y| - \min(x, y)$, for which BCD will get stuck at a non-stationary point. We illustrate this issue by rotating the simple function

$$\ell(x, y) = |x| + 2|y|.$$

Figure 2 depicts the $\pi/4$ and $\pi/10$ radian rotations of ℓ. If we consider the $\pi/4$ radian rotation of ℓ and start at a point of the form (β, β) where $\beta \neq 0$, then coordinate minimization along x or y will not decrease the objective value since the point is already optimal along each coordinate direction. Since the optimal solution is at $(0, 0)$, the algorithm will get stuck at a non-optimal point. However, this problem does not occur for the $\pi/10$ radian rotation of ℓ, since it can decrease along the x direction.

More mathematically, the rotated functions correspond to setting ε to $\pi/4$ and $\pi/10$, respectively, in the following function:

$$f_\varepsilon(x, y) = \ell\big(\cos(\varepsilon)x + \sin(\varepsilon)y, \cos(\varepsilon)y - \sin(\varepsilon)x\big).$$

We can verify that $f_{\pi/4}(x, x) = \min_{\bar{y}} f_{\pi/4}(x, \bar{y})$ for any $x \in \mathbb{R}$, that is, given any x, the minimizer y of $f_{\pi/4}$ equals the value of x. (The same holds if, instead, we fix y and minimize $f_{\pi/4}$ over x. The minimizer x will equal the value of y.) Therefore, starting from any point (x, y), a coordinate minimization over x or y will move to the point (y, y) or (x, x), respectively. If the point is not 0 (the origin), then any further coordinate update cannot reduce the value of $f_{\pi/4}$. Therefore, CD converges to a non-stationary point.

Though our example $f_{\pi/4}$ gives a case where CD may converge to a non-stationary point if f is convex, non-separable, and non-smooth, this may not always be the case. CD will converge for the example $f_{\pi/10}$ since it will not get stuck at any of the contour corners.

A mathematical explanation of this phenomenon is that the componentwise subgradients $p_x \in \partial_x f_{\pi/4}(x, y)$ and $p_y \in \partial_y f_{\pi/4}(x, y)$ do *not* necessarily mean that the full vector formed by the concatenation of the component subgradients is a subgradient, i.e. $[p_x; p_y] \in \partial f_{\pi/4}(x, y)$. Thus, in the case for $f_{\pi/4}$, a point in the form of $(\beta, \beta) \neq 0$ is not a stationary point because $0 \notin \partial f_{\pi/4}(\beta, \beta)$. A further explanation on this and subdifferential calculus is detailed in Appendix B.

In general, BCD can fail to converge for an objective function that contains just one non-separable and non-smooth term, even if all the other terms are differentiable or separable. In fact, this failure can occur with any BCD update presented in this paper. However, in this case, though there are no theoretical guarantees, practitioners may still try applying BCD and check optimality conditions for convergence. When the non-separable and non-smooth term has the form $f(Ax)$ and is convex, primal-dual coordinate update algorithms may also be applied since they will decouple f from A, though this is not a focus in this monograph. Please refer to [55, 53, 21] for more information on primal-dual coordinate update algorithms.

2.2.2 Proximal Point Update

The *proximal point* update, or *proximal* update, is defined by the following:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}^{k-1}) + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2 + r_{i_k}(\mathbf{x}_{i_k}), \tag{8}$$

where $\alpha_{i_k}^{k-1}$ serves as a step size and can be any bounded positive number.

The proximal update with BCD was introduced in [2] for convex problems and further analyzed in [23, 59, 86] for possibly nonconvex problems. This update adds a quadratic proximal term to the classic block minimization update described in (7), and thus the function of each subproblem dominates the original objective around the current iterate. This modification gives the proximal update scheme better convergence properties and increased stability, particularly for non-smooth problems.

2.2.3 Prox-Linear Update

The *proximal linear* update, or *prox-linear* update, is defined by:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}^{k-1}) + \langle \nabla_{i_k} f(\mathbf{x}_{i_k}^{k-1}, \mathbf{x}_{\neq i_k}^{k-1}), \mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1} \rangle + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2 + r_{i_k}(\mathbf{x}_{i_k}), \quad (9)$$

where the step size $\alpha_{i_k}^{k-1}$ can be set as the reciprocal of the Lipschitz constant of $\nabla_{i_k} f(\mathbf{x}_{i_k}, \mathbf{x}_{\neq i_k}^{k-1})$ with respect to $\mathbf{x}_{i_k}$. The difference between the prox-linear update from the proximal update is that the former further linearizes the smooth part f to make the subproblem easier.

The prox-linear update scheme was introduced in [80], which proposed a more general framework of block coordinate gradient descent (BCGD) methods. It was later adopted and also popularized by Nesterov [47] in the randomized coordinate descent method. BCD with this update scheme has been analyzed and also applied to both convex and nonconvex problems such as in [4, 28, 47, 80, 87, 91, 90, 95]. In essence, this scheme minimizes a surrogate function that dominates the original objective around the current iterate $\mathbf{x}_{i_k}^{k-1}$. Note that when the regularization term $r_{i_k}(\mathbf{x}_{i_k})$ vanishes, this update reduces to:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}^{k-1}) + \langle \nabla_{i_k} f(\mathbf{x}_{i_k}^{k-1}, \mathbf{x}_{\neq i_k}^{k-1}), \mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1} \rangle + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2, \quad (10)$$

or equivalently the block gradient descent algorithm:

$$\mathbf{x}_{i_k}^k = \mathbf{x}_{i_k}^{k-1} - \alpha_{i_k}^{k-1} \nabla_{i_k} f(\mathbf{x}_{i_k}^{k-1}, \mathbf{x}_{\neq i_k}^{k-1}).$$

When BCD with the proximal or prox-linear update scheme is applied to non-convex objectives, it often gives solutions of lower objective values compared with the block coordinate minimization, since small regions containing certain local minima may be avoided by its local proximal or prox-linear approximation. Note that the prox-linear update may take more iterations to reach the same accuracy than the other two schemes. However, it is easier to perform the update and thus may take less total time as demonstrated in [71, 85, 86].

2.2.4 Extrapolation

Though the prox-linear update is easily computed and gives a better solution overall, coordinate minimization and proximal updates tend to make larger objective decreases per iteration. This observation motivates the use of extrapolation to accelerate the convergence of the prox-linear update scheme.

Extrapolation uses the information at an extrapolated point in place of the current point to update the next iterate [3, 35, 48, 68, 86]. In particular, rather than using the partial gradient at $\mathbf{x}_{i_k}^{k-1}$ for the next update, we instead consider an extrapolated point

$$\hat{\mathbf{x}}_{i_k}^{k-1} = \mathbf{x}_{i_k}^{k-1} + \omega_{i_k}^{k-1}(\mathbf{x}_{i_k}^{k-1} - \mathbf{x}_{i_k}^{k-2}), \quad (11)$$

where $\omega_{i_k}^{k-1} \geq 0$ is an extrapolation weight. This extrapolated point is then used to compute our next update, and it gives the update:

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}^{k-1}) + \langle \nabla_{i_k} f(\hat{\mathbf{x}}_{i_k}^{k-1}, \mathbf{x}_{\neq i_k}^{k-1}), \mathbf{x}_{i_k} - \hat{\mathbf{x}}_{i_k}^{k-1} \rangle + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \hat{\mathbf{x}}_{i_k}^{k-1}\|_2^2 + r_{i_k}(\mathbf{x}_{i_k}).$$

Note that if $\omega_{i_k}^{k-1} = 0$, the above update reduces to that in (9). Appropriate positive weight can significantly accelerate the convergence of BCD with prox-linear update as demonstrated in [47, 85] while the per-iteration complexity remains almost the same.

2.2.5 Stochastic Gradients

Often in machine learning and other large data applications, we encounter problems with datasets consisting of tens of millions of datapoints and millions of features. Due to the size and scope of these problems, sometimes computing a coordinate gradient may still be extremely expensive. To remedy this, we can introduce *stochastic gradients*.

Consider the following stochastic program, called the regularized *expected risk minimization* problem,

$$\underset{\mathbf{x}}{\text{minimize}} \ \mathbb{E}_\xi f_\xi(\mathbf{x}) + \sum_{i=1}^{s} r_i(\mathbf{x}_i), \tag{12}$$

where ξ is a random variable, $f(\mathbf{x}) = \mathbb{E}_\xi f_\xi(\mathbf{x})$ is differentiable, and r_i's are certain regularization terms.

The function f may represent some loss due to inaccurate predictions from some classifier or prediction function. To minimize loss for any set of predicted and true parameters, we take the expectation of the loss with respect to some probability distribution modeled by the random variable ξ. This expectation takes the form of an integral or summation that weights losses for all possible predictions and true parameters.

An interesting case of (12) is when ξ follows a uniform distribution over $1, \ldots, m$, representing a noninformative prior distribution. In this case, if the potential outcomes are discrete, then the stochastic program in (12) reduces to the *empirical risk minimization* problem

$$\underset{\mathbf{x}}{\text{minimize}} \ \frac{1}{m} \sum_{i=1}^{m} f_i(\mathbf{x}) + \sum_{i=1}^{s} r_i(\mathbf{x}_i) \tag{13}$$

where each f_i represents the loss incurred with respect to one sample from the dataset. Therefore, since the problems we are considering often rely on millions of training points, m is very large. The empirical risk minimization problem is also often used in place of the expected risk minimization problem when the probability distribution of ξ is unknown.

The stochastic gradient method, also called the stochastic approximation method, (e.g., see [46]) is useful for minimizing objectives in the form of (12) or (13) with large m, in which computing the exact gradient or objective becomes overly expensive. To compute the full gradient for (13), one would have to compute

$$\nabla f(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\mathbf{x})$$

by processing every training example in the dataset. Since this is often infeasible, we can instead sample either one or a small batch of loss functions f_i, called a *mini-batch*, to compute a subsampled gradient to use in place of the full gradient, i.e. we use

$$\tilde{\mathbf{g}}_{i_k} = \frac{1}{|S_k|} \sum_{l \in S_k} \nabla_{\mathbf{x}_{i_k}} f_{k_l}(\mathbf{x})$$

where $S_k \subset \{1, ..., m\}$ is a mini-batch and $|S_k|$ is the number of sample functions selected from the loss functions f_i's.

More generally, for solving (12) or (13) by prox-linear BCD, we may replace the true coordinate gradient with a stochastic approximation, i.e. if the i_kth block is selected,

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}^{k-1}) + \langle \tilde{\mathbf{g}}_{i_k}^{k-1}, \mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1} \rangle + \frac{1}{2\alpha_{i_k}^{k-1}} \|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|^2 + r_{i_k}(\mathbf{x}_{i_k}), \tag{14}$$

where $\tilde{\mathbf{g}}_{i_k}^{k-1}$ is a stochastic approximation of $\nabla_{i_k} f(\mathbf{x}^{k-1})$, $\tilde{\mathbf{g}}_{i_k}^{k-1}$ is a subsampled gradient, etc.

Though the stochastic prox-linear update may be less accurate, it works well when there is a limited amount of memory available, or when a solution is needed quickly, as discussed in [13, 88].

2.2.6 Variance Reduction Techniques

Alternatively, we can also consider *stochastic variance-reduced gradients*, which use a combination of stale gradients with new gradients to reduce the variance in the chosen stochastic gradients. These variance-reduced stochastic gradient algorithms gain a significant speedup in convergence; whereas stochastic gradients only have sublinear convergence rate guarantees, variance-reduced stochastic gradients can have linear convergence rates similar to traditional gradient descent methods on problems with strongly convex objectives.

Consider the problem given above in (13). Let ϕ_i^k denote the past stored point used at the prior gradient evaluation for function f_i and $\nabla f_i(\phi_i^k)$ denote the stored gradient. We list some common stochastic variance-reduced gradients below:

- SAG [65]: If the jth indexed function is chosen at iterate k,

$$\tilde{\mathbf{g}}^{k-1} = \frac{\nabla f_j(\mathbf{x}^{k-1}) - \nabla f_j(\phi_j^{k-1})}{m} + \frac{1}{m} \sum_{l=1}^{m} \nabla f_l(\phi_l^{k-1}).$$

  The current iterate $\mathbf{x}^{k-1}$ is then taken as ϕ_j^k and $\nabla f_j(\phi_j^k)$ is explicitly stored in a table of gradients.

- SAGA [14]: If the jth indexed function is chosen at iterate k,

$$\tilde{\mathbf{g}}^{k-1} = \nabla f_j(\mathbf{x}^{k-1}) - \nabla f_j(\phi_j^{k-1}) + \frac{1}{m} \sum_{l=1}^{m} \nabla f_l(\phi_l^{k-1}).$$

  The current iterate $\mathbf{x}^{k-1}$ is then taken as ϕ_j^k and $\nabla f_j(\phi_j^k)$ is explicitly stored in a table of gradients.

- SVRG [33]: If the jth indexed function is chosen at iterate k,

$$\tilde{\mathbf{g}}^{k-1} = \nabla f_j(\mathbf{x}^{k-1}) - \nabla f_j(\tilde{\mathbf{x}}) + \frac{1}{m} \sum_{l=1}^{m} \nabla f_l(\tilde{\mathbf{x}}),$$

  where $\tilde{\mathbf{x}}$ is not updated every step but is updated after a fixed number of iterations. If enough memory is available, individual gradients $\nabla f_l(\tilde{\mathbf{x}})$'s as well as their average are all stored; otherwise, one can store only the average and evaluate $\nabla f_j(\tilde{\mathbf{x}})$ at each iteration (in addition to the usual work to evaluate $\nabla f_j(\mathbf{x}^{k-1})$).

Another approach, the stochastic dual coordinate ascent algorithm (SDCA) [67], applies randomized dual coordinate ascent to the dual formulation of the problem and gives similar variance reduction properties.

In general, though these methods give better convergence properties, they require more memory to store stale gradients or more computation to evaluate exact gradient. However, they perform better than traditional stochastic gradient methods, and work well when calculating the exact gradient is expensive.

Note that the primary difference between SVRG and SAG is that SVRG makes 2-3x more gradient evaluations if it does not store a table of gradients, whereas SAG uses less gradient evaluations but requires more memory overhead to store gradients. SAGA may be interpreted as the midpoint between SVRG and SAG. The usage of SAG, SAGA, and SVRG is, therefore, problem-dependent.

The variance reduction technique can also be incorporated into the prox-linear BCD. For example, one can use any $\tilde{\mathbf{g}}^{k-1}$ of the above three ones in the update (14) to accelerate its convergence.

2.2.7 Summative Proximable Functions

We apply the prox-linear update (9) when the function r_{i_k} is proximable, that is, when its proximal operator can be evaluated at a low cost. Functions such as ℓ_1-norm, ℓ_2-norm, and ℓ_∞-norm, as well as the indicator functions of box constraints, one or two linear constraints, and the standard simplex, are proximable functions. The list can be quite long. Nonetheless, it is not difficult to see that, even if two functions f and g are both proximable, $f+g$ may not be proximable. Therefore, the update (9) can still be expensive to compute if r_{i_k} is the sum of two or more proximable functions.

The *summative proximable function* is the sum of proximable functions f and g that satisfy

$$\operatorname{prox}_{f+g} = \operatorname{prox}_g \circ \operatorname{prox}_f .$$

Because their proximal operator can be obtained by applying the proximal operator of f and then that of g in a sequential fashion, it is also proximable. Some common examples of summative proximable functions are:

- $f(\mathbf{x}) + g(\mathbf{x}) := f(\mathbf{x}) + \beta\|\mathbf{x}\|_2$, where $\beta \geq 0$ and $f(\mathbf{x})$ is a homogeneous function of order 1 (i.e., $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$ for $\alpha \geq 0$). Examples of $f(\mathbf{x})$ include $\alpha\|\mathbf{x}\|_1$, $\alpha\|\mathbf{x}\|_\infty$, $\iota_{\geq 0}(\mathbf{x})$, or $\iota_{\leq 0}(\mathbf{x})$ and $\alpha, \beta > 0$.

- $f(\mathbf{x}) + g(\mathbf{x}) := \beta \mathrm{TV}(\mathbf{x}) + g(\mathbf{x})$, where $\mathrm{TV}(\mathbf{x}) := \sum_{i=1}^{n-1} |x_{i+1} - x_i|$ is (discrete) total variation, and $g(\mathbf{x})$ is a function with the following property: for any $\mathbf{x} \in \mathbb{R}^n$ and coordinates $i \in [n]$ and $j = i + 1$,

$$\begin{aligned} x_i > x_j \quad &\Rightarrow \quad \big(\operatorname{prox}_g(\mathbf{x})\big)_i \geq \big(\operatorname{prox}_g(\mathbf{x})\big)_j \\ x_i < x_j \quad &\Rightarrow \quad \big(\operatorname{prox}_g(\mathbf{x})\big)_i \leq \big(\operatorname{prox}_g(\mathbf{x})\big)_j \\ x_i = x_j \quad &\Rightarrow \quad \big(\operatorname{prox}_g(\mathbf{x})\big)_i = \big(\operatorname{prox}_g(\mathbf{x})\big)_j. \end{aligned}$$

  Examples of such $g(\mathbf{x})$ include $\alpha\|\mathbf{x}\|_1$, $\alpha\|\mathbf{x}\|_2$, $\alpha\|\mathbf{x}\|_\infty$, $\iota_{\geq 0}(\mathbf{x})$, or, more generally, $\iota_{[\ell,u]}(\mathbf{x})$ for any $\ell, u \in \mathbb{R}$.

- [11, Prop. 3.6] scalar function $f_i(\rho) + g_i(\rho) := \alpha|\rho| + g_i(\rho)$, where $\rho \in \mathbb{R}$ and g_i is convex and $g_i'(0) = 0$. An example is the *elastic net* regularizer [96]: $f(\mathbf{x}) + g(\mathbf{x}) := \alpha\|\mathbf{x}\|_1 + \frac{1}{2}\|\mathbf{x}\|_2^2$.

The key to these results is an inclusion property: For any $\mathbf{x} \in \mathbb{R}^n$, let $\mathbf{y} := \text{prox}_f(\mathbf{x})$ and $\mathbf{z} := \text{prox}_g(\mathbf{y})$, whose minimization conditions are

$$0 \in \partial f(\mathbf{y}) + (\mathbf{y} - \mathbf{x}),$$
$$0 \in \partial g(\mathbf{z}) + (\mathbf{z} - \mathbf{y}),$$

respectively, and adding them yields

$$0 \in \partial f(\mathbf{y}) + \partial g(\mathbf{z}) + (\mathbf{z} - \mathbf{x}).$$

If the property of f and g gives the inclusion property $\partial f(\mathbf{y}) \subseteq \partial f(\mathbf{z})$, then we arrive at the minimization condition of $\mathbf{z} = \text{prox}_{f+g}(\mathbf{x})$:

$$0 \in \partial f(\mathbf{z}) + \partial g(\mathbf{z}) + (\mathbf{z} - \mathbf{x}).$$

Because the first two classes of summative proximable functions are not seen elsewhere to the best of our knowledge, a proof is included in Appendix D.

2.3 Choosing Update Index i_k

In this section, we elaborate on various implementation approaches in choosing the coordinate or block $i_k \in \{1, ..., s\}$. Since different paths taken in coordinate descent may lead to different minima and different schemes perform differently for both convex and non-convex problems, the choice of the update index i_k for each iterate is crucial for good performance for BCD. Often, it is easy to switch index orders. However, the choice of index affects convergence, possibly resulting in faster convergence or divergence. We describe the index rules more in detail below.

2.3.1 Cyclic Variants

The most natural, deterministic approach for choosing an index is to choose indices in a cyclic fashion, i.e. $i_0 = 1$ and

$$i_{k+1} = (k \mod s) + 1, \ k \in \mathbb{N}.$$

We may also adapt this method and instead cycle through a permutation of $\{1, ..., s\}$, called a *shuffled cyclic* method. In practice, one may reshuffle the order of the indices after each cycle, or cycle through all coordinates, which may have stronger convergence properties for some applications.

Another approach is to satisfy an *essentially cyclic* condition, in which for every consecutive $N \geq s$ iterations, each component is modified at least once. More rigorously, we require

$$\bigcup_{j=0}^{N} \{i_{k-j}\} = \{1, 2, ..., s\}$$

for all $k \geq N$.

Cyclic variants are most intuitive and easily implemented. BCD with the deterministic cyclic rule may give poorer performance than that with shuffled cyclic one, as demonstrated in [87] for solving non-negative matrix factorization. Convergence results of cyclic BCD are given in [4, 6, 16, 23, 28, 42, 59, 64, 80, 87, 92].

2.3.2 Randomized Variants

In randomized BCD algorithms, the update component or block i_k is chosen randomly at each iteration. The simplest, most commonly used randomized variant is to simply select i_k with equal probability, or *sample uniformly*, independent of all choices made in prior iterations.

Other typical randomized variants include sampling without replacement, and considering different, non-uniform probability distributions. We list common sampling methods below:

1. *Uniform sampling* [20, 47, 60, 66, 67]: Each block coordinate $j \in \{1, ...s\}$ is chosen with equal probability as we described above, i.e.

$$P(i_k = j) = \frac{1}{s}, \; j = 1, \ldots, s.$$

2. *Importance sampling* [36, 47, 60, 93, 94]: We proportionally weight each block according to its block-wise Lipschitz gradient constant $L_j > 0$ for all $j \in \{1, ..., s\}$. More rigorously, given some $\alpha \geq 0$, Nesterov [47] proposes the following distribution:

$$P(i_k = j) = p_\alpha(j) = \frac{L_j^\alpha}{\sum_{i=1}^s L_i^\alpha}, \; j = 1, \ldots, s.$$

   This scheme generalizes uniform sampling – when $\alpha = 0$, we obtain uniform sampling. Importance sampling has been further studied in [1, 12].

3. *Arbitrary sampling* [51, 57, 58, 60, 61]: We pick and update a block $j \in \{1, 2, ..., s\}$ arbitrarily, following some assigned probability distribution $(p_1, ..., p_s)$, i.e.

$$P(i_k = j) = p_j, \; j = 1, \ldots, s,$$

   where $0 \leq p_i \leq 1$ for all i and $\sum_{i=1}^s p_i = 1$. This sampling scheme generalizes both uniform and importance sampling.

Randomized BCD is well suited for cases in which memory or available data is limited since the computation of a partial derivative is often much cheaper and less memory demanding than computing an entire gradient [47]. Recent work also suggests that randomization improves the convergence rate of BCD in expectation, such as for minimizing generic smooth and simple nonsmooth block-separable convex functions [47, 57, 60, 74]. However, randomized BCD variants have greater per-iteration complexities than cyclic BCD variants since these algorithms have to sample from probability distributions each iteration. Since randomized variants are non-deterministic, results in practice may vary. During the running of a randomized BCD, cache misses are more likely, requiring extra time to move data from slower to faster memory in the memory hierarchy.

2.3.3 Greedy Variants

The last widely used approach for selecting indices are greedy methods. These methods choose i_k "greedily", or choose the index such that the objective function is minimized most, or the gradient or subgradient has the largest size, in that direction. The simplest variant for smooth functions,

called the *Gauss-Southwell*[1] *selection rule (GS)*, involves choosing i_k such that the gradient of the chosen block is greatest, or mathematically,

$$i_k = \arg\max_{1 \le j \le s} \|\nabla_j f(\mathbf{x}^{k-1})\| \tag{15}$$

for formulation (1). If each block consists of an individual component, then the norm reduces to the absolute value function. GS can be analyzed in the general CD framework [77, 42] for convex optimization. In the refined analysis [49], it is shown that, except in extreme cases, GS converges faster than choosing random coordinates in terms of the number of iterations, though its per-iteration complexity is higher.

Alternatively, we could choose the component or block that gives maximum improvement, called the *Maximum Block Improvement (MBI) rule* [9, 38]:

$$i_k = \arg\min_j f(\mathbf{x}_j, \mathbf{x}_{\neq j}^{k-1}) \tag{16}$$

Motivated by the performance of Lipschitz sampling and Gauss-Southwell's rule, Nutini, et. al. [49] proposed the *Gauss-Southwell-Lipschitz (GSL)* rule:

$$i_k = \arg\max_j \frac{\|\nabla_j f(\mathbf{x}^{k-1})\|}{\sqrt{L_j}}. \tag{17}$$

The GSL rule accounts for varied coordinatewise Lipschitz constants. When the gradients of two coordinates have similar sizes, updating the coordinate that has the smaller L_i will likely lead to greater reduction in the objective value and is thus preferred in the selection.

For non-smooth problems, we list some commonly used GS rules for formulation (6). We let $L \in \mathbb{R}$ denote the gradient Lipschitz constant and $L_j \in \mathbb{R}$ denote the jth coordinate gradient Lipschitz constant.

1. *Gauss-Southwell-s rule (GS-s)*: At each iteration, the coordinate $i_k \in \{1, ..., s\}$ is chosen by

$$i_k = \arg\max_j \left\{ \min_{\tilde{\nabla} r_j \in \partial r_j} \|\nabla_j f(\mathbf{x}^{k-1}) + \tilde{\nabla} r_j(\mathbf{x}_j^{k-1})\| \right\}. \tag{18}$$

   This rule chooses the coordinate with the greatest negative partial derivative, similar to (15). It is popular in ℓ_1 minimization [37, 70, 84].

2. *Gauss-Southwell-r rule (GS-r)*: At each iteration, the coordinate $i_k \in \{1, ..., s\}$ is chosen by

$$i_k = \arg\max_j \left\| \mathbf{x}_j^{k-1} - \text{prox}_{\frac{1}{L} r_j}[\mathbf{x}_j^{k-1} - \frac{1}{L}\nabla_j f(\mathbf{x}^{k-1})] \right\| \tag{19}$$

   which is equivalent to

$$i_k = \arg\max_j \left\| \arg\min_{\mathbf{d}} \left(f(\mathbf{x}^{k-1}) + \nabla_j f(\mathbf{x}^{k-1})^T \mathbf{d} + \frac{L}{2}\|\mathbf{d}\|_2^2 + r_j(\mathbf{x}_j^{k-1} + \mathbf{d}) - r_j(\mathbf{x}_j^{k-1}) \right) \right\|,$$

[1] The greedy selection rule dates back to Gauss and was popularized by Southwell [72] for linear systems.

where $\mathbf{d}$ has the same size as the block gradient direction. This rule chooses the block that maximizes the distance from the current iterate to the block's following iterate from a proximal gradient update. It has been used in [80, 17, 52]. If the coordinate-wise gradient Lipschitz constant L_j is known, then we can replace L by L_j in the GS-r update to obtain the *Gauss-Southwell-Lipschitz-r rule (GSL-r)*.

3. *Gauss-Southwell-q rule (GS-q)*: At each iteration, the coordinate $i_k \in \{1, ..., s\}$ is chosen by

$$i_k = \arg\min_j \big(\min_{\mathbf{d}} f(\mathbf{x}^{k-1}) + \nabla_j f(\mathbf{x}^{k-1})^T \mathbf{d} + \frac{L}{2}\|\mathbf{d}\|_2^2 + r_j(\mathbf{x}_j^{k-1} + \mathbf{d}) - r_j(\mathbf{x}_j^{k-1}) \big). \quad (20)$$

This rule can be interpreted as the maximum coordinate-wise descent and has been used in [80]. If the coordinate-wise gradient Lipschitz constant L_j is known, then we can replace L by L_j in the GS-q update to obtain the *Gauss-Southwell-Lipschitz-q rule (GSL-q)*.

Note that these greedy methods usually require evaluating the whole gradient vector or some other greedy score, and searching for the best index. These greedy scores may be stored and updated using a max-heap, which is further detailed in [44]. To practically implement greedy methods, some terms of these greedy scores may be cached and maintained at each iteration to make these approaches computationally worthwhile. Section 3 provides detailed further explanations.

Greedy coordinate selections are very efficient for sparse optimization since most zero components in the solution are never selected and thus remain zero throughout the iterations [37, 52]. The problem dimension effectively reduces the number of variables that are ever updated, which is relatively small. Consequently, the greedy CD iteration converges in very few iterations. The saved iterations over-weigh the extra cost of ranking the coordinates.

Other simple greedy methods involve perturbing coordinates or blocks $\mathbf{x}_j$ for all $j \in \{1, ..., s\}$ by some small step size $\beta > 0$, then evaluating the difference of objective values at those perturbed points with the current point to determine the coordinate or block of steepest descent, i.e.,

$$i_k = \arg\max_j |f(\mathbf{x}_j + \beta \mathbf{e}_j) - f(\mathbf{x}_j)|,$$

where e_j is the standard basis vector consisting of 1 at the jth entry and 0's elsewhere.

2.3.4 Comparison of Index Rules

We summarize the strengths and weaknesses of these common index rules below in Table 1. We also point readers to Section 4 for a comparison of various index rules applied to several examples. Note that no matter which index rule is chosen, CD can stagnate at non-critical points for objectives with terms that are both non-separable and non-smooth.

Method	Description	Strengths	Weaknesses
Cyclic	i_k is chosen in a cyclically: $i_k = (k \mod n) + 1$ or by a randomly shuffled cycle	• Fast, easy implementation • Lowest per-iteration complexity • Performs well for problems with low coupling between blocks	• When coordinates are highly coupled, performance may be worse than randomized and greedy methods and theoretical worst-case complexity is worse • When exact coordinate minimization is used on non-convex problems, the points may cycle and fail to converge
Randomized	i_k is chosen randomly following some probability distribution given by the vector $(p_1, ..., p_s)$: $P(i_k = j) = p_j$	• Easier to analyze; often reduces to standard case after taking expectations • Empirically avoids local solutions more for some non-convex problems • Well suited for parallel computing	• Non-deterministic • Random data moves are slower and result in cache misses • Slightly higher per-iteration complexity than cyclic CD due to pseudo-random number generation • 2-3x slower if the coordinates are weakly coupled
Greedy	i_k is chosen by a greedy rule, such as greatest descent: $i_k = \underset{1 \le i \le s}{\arg\max} \|[\nabla f(\mathbf{x}^{k-1})]_i\|$	• Convergence take the least number of iterations, both theoretically and empirically • Well suited for problems with sparse solutions; can be parallelized	• Need computation of greedy scores or rankings, such as gradients or difference vectors • Highest per-iteration complexity except when greedy scores can be updated at a low cost

Table 1: Summary and comparison of different index choosing schemes for CD.

3 Coordinate Friendly Structures

As discussed earlier in the monograph, the strong performance and parallelizability of CD rely on solving subproblems that consist of fewer variables and have low complexities and low memory requirements. Therefore, not all problems are amenable for CD, particularly if little computation is saved from using CD relative to the full update for all coordinates. Intuitively, given s blocks, a block coordinate update should cost about $1/s$ of the computation for a full update. Thus, identifying *coordinate friendly updates* for a given problem is crucial to implementing CD effectively.

In this section, we elaborate on different types of structures in optimization problems that make CD computationally worthy. We define the notion of a *coordinate friendly (CF) update* and *coordinate friendly (CF) structure* and introduce heuristics to exploit problems with CF structures. This basic theory may help practitioners identify when CD is applicable and computationally worthwhile for their problem, as well as determine which quantities to cache and maintain to improve the performance of CD.

The CF structure was originally presented in [53] for monotone set-valued operators, which apply in more general settings. We refrain from discussing this here and instead replace the notion of an operator with an *update mapping*.

3.1 Coordinate Friendly Update Mappings

Before we define coordinate friendly update and structure, we introduce terminology and notation on updates. Suppose we are working with s equally-sized blocks in $\mathbf{x} \in \mathbb{R}^n$, i.e. $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_s)$. Let $T : \mathbb{R}^n \to \mathbb{R}^n$ represent an *update mapping* or simply *update*. Applying it to $\mathbf{x}^{k-1}$, we obtain the next iterate $\mathbf{x}^k$, i.e.,

$$\mathbf{x}^k = T(\mathbf{x}^{k-1}).$$

In addition, let T_i denote the *coordinate update mapping* of T for block $\mathbf{x}_i$, i.e.,

$$T_i(\mathbf{x}) = (T(\mathbf{x}))_i,\ i = 1, \ldots, s.$$

As an example, consider the least squares problem:

$$\underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}}\ \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \tag{21}$$

where $\mathbf{A} \in \mathbb{R}^{p\times n}$ and $\mathbf{b} \in \mathbb{R}^p$. The update mapping for gradient descent on (21) is given as:

$$\mathbf{x}^k = T_{\text{GD}}(\mathbf{x}^{k-1}) = \mathbf{x}^{k-1} - \alpha(\mathbf{A}^T\mathbf{A}\mathbf{x}^{k-1} - \mathbf{A}^T\mathbf{b})$$

where $\alpha > 0$ is the step size, and the coordinate update mapping is then given as

$$T_{\text{GD},i}(\mathbf{x}^{k-1}) = \mathbf{x}_i^{k-1} - \alpha(\mathbf{A}^T\mathbf{A}\mathbf{x}^{k-1} - \mathbf{A}^T\mathbf{b})_i,\ i = 1, \ldots, s.$$

We let $\mathcal{N}[a \mapsto b]$ denote the number of basic operations necessary to compute quantity b from the input a. Then the update mapping T is called *coordinate friendly (CF)* if

$$\mathcal{N}\big[\mathbf{x} \mapsto T_i(\mathbf{x})\big] = O\left(\frac{1}{s}\mathcal{N}[\mathbf{x} \mapsto T(\mathbf{x})]\right), \forall i. \tag{22}$$

In other words, the number of basic operations necessary to compute the coordinate update is about $1/s$ times of the number of basic operations to compute the full update.

Returning to our least squares example, T_{GD} is CF since the coordinate gradient update can be computed by

$$T_{\text{GD},i}(\mathbf{x}^{k-1}) = \mathbf{x}_i^{k-1} - \alpha\left[(\mathbf{A}^T\mathbf{A})_{i,:}\mathbf{x}^{k-1} - (\mathbf{A}^T\mathbf{b})_i\right] \tag{23}$$

which takes $O(\frac{n^2}{s})$ operations after precomputing $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$. In contrast, the full gradient update T_{GD} takes $O(n^2)$ operations with precomputed $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$.

Intuitively, the definition of coordinate friendly update mappings encapsulates the notion of gaining s times speed-up per coordinate update T_i relative to the full update T. Equivalently, if we break up our variables into s equally-sized blocks, updating each block should require about $1/s$ times operations necessary to compute the full update.

This formulation also matches our intuition for extreme cases; in the coordinate case of updating individual components x_i, this will give an n times speed up for computing each coordinate update relative to computing the full update. In the case of only one block containing the entire vector $\mathbf{x}$, we gain no speed-up.

We say that the problem is *coordinate friendly* or has *coordinate friendly (CF) structure* if there exists a coordinate friendly update for the problem. To readily apply CD, we must recognize and exploit CF structures in optimization problems.

Identifying CF structure in optimization problems is not always trivial. To help with the analysis of optimization problems to find CF structure, we will give some useful heuristics applied to CD implementations in Section 3.2.

The definitions for CF update and structure may be further generalized for blocks of arbitrary lengths, but we refrain from doing so to maintain simplicity.

3.2 Common Heuristics for Exploiting CF Structures

In this section, we introduce some common heuristics that exploit structure in a problem to improve the performance of CD and gain coordinate friendliness.

3.2.1 Precomputation of Non-Variable Quantities

As noted already in the least squares example, one common approach for increasing the computational worthiness of CD is to precompute certain quantities in the update. If certain quantities that do not consist of any of the variables appear in the update, we can precompute them before applying CD to save computation.

For example, consider again the least squares problem given above. Since the full gradient is given by $\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}$, we can precompute $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ to avoid recomputing $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ at each iteration.

Note, however, that this is only efficient when $n \sim p$ or $p \gg n$ (recall that $\mathbf{A}$ has p rows and n columns). If $p \ll n$, then multiplying by $\mathbf{A}$ then $\mathbf{A}^T$ is computationally cheaper, so we avoid precomputation in this case.

3.2.2 Caching and Maintaining Variable-Dependent Quantities

Another approach to save computation is to cache and maintain variable-dependent quantities. In the CF notation, this would refer to storing some quantity $\mathcal{M}(\mathbf{x}^k)$ in the memory and updating it at each iteration.

For example, for the least squares problem, instead of performing the coordinate update as in (23), we can save the quantity

$$\mathcal{M}(\mathbf{x}^{k-1}) = \mathbf{A}^T\mathbf{A}\mathbf{x}^{k-1}.$$

Then for any i, $T_{\mathrm{GD},i}$ can be evaluated by

$$T_{\mathrm{GD},i}(\mathbf{x}^{k-1}) = \mathbf{x}_i^{k-1} - \alpha\left((\mathcal{M}(\mathbf{x}^{k-1}))_i - (\mathbf{A}^\top\mathbf{b})_i\right),$$

which takes $O(\frac{n}{s})$ operations. Let $\mathbf{x}^k$ be the vector obtained by block coordinate descent update from $\mathbf{x}^{k-1}$, i.e.,

$$\mathbf{x}_i^k = \begin{cases} T_{\mathrm{GD},i}(\mathbf{x}^{k-1}), & \text{if } i = i_k, \\ \mathbf{x}_i^{k-1}, & \text{if } i \neq i_k. \end{cases} \tag{24}$$

Then $\mathcal{M}(\mathbf{x}^k)$ can be obtained by

$$\mathcal{M}(\mathbf{x}^k) = \mathcal{M}(\mathbf{x}^{k-1}) + (\mathbf{A}^T\mathbf{A})_{:,i_k}(\mathbf{x}_{i_k}^k - \mathbf{x}_{i_k}^{k-1}),$$

which takes $O(\frac{n}{s}) + O(\frac{n^2}{s}) + O(\frac{n}{s}) = O(\frac{n^2}{s})$ operations, since block coordinate addition/subtraction takes $O(\frac{n}{s})$ operations and block matrix-vector multiplication takes $O(\frac{n^2}{s})$ operations. Hence, if $\mathbf{A}^T\mathbf{A}$ is cached or $n = O(p)$, we have that

$$\mathcal{N}\big[\{\mathbf{x}^{k-1}, \mathcal{M}(\mathbf{x}^{k-1})\} \to \{\mathbf{x}^k, \mathcal{M}(\mathbf{x}^k)\}\big] = O\left(\frac{1}{s}\mathcal{N}\big[\mathbf{x}^{k-1} \to T_{\mathrm{GD}}(\mathbf{x}^{k-1})\big]\right). \tag{25}$$

Consider the logistic regression as another example:

$$\underset{\mathbf{w}}{\text{minimize}}\, F(\mathbf{w}) = \sum_{j=1}^m \log\left(1 + \exp[-y_j\mathbf{w}^T\mathbf{x}_j]\right),$$

where $\{\mathbf{x}_j \in \mathbb{R}^n : j = 1, \ldots, m\}$ are training data points, and $y_j \in \{-1, 1\}$ is the label of $\mathbf{x}_j$ for $j = 1, 2, \ldots, m$. The variable $\mathbf{w} \in \mathbb{R}^n$ and its block components $\mathbf{w}_i \in \mathbb{R}^{n/s}$. Its gradient is

$$\nabla F(\mathbf{w}) = \sum_{j=1}^m \frac{-y_j \exp[-y_j\mathbf{w}^T\mathbf{x}_j]}{1 + \exp[-y_j\mathbf{w}^T\mathbf{x}_j]}\mathbf{x}_j,$$

and the gradient descent update with step size α is given by

$$T_{\mathrm{GD}}(\mathbf{w}) = \mathbf{w} - \alpha\nabla F(\mathbf{w}). \tag{26}$$

To achieve an efficient coordinate update from $\mathbf{x}^{k-1}$, we maintain the quantity

$$\mathcal{M}(\mathbf{w}^{k-1}) = \left\{\exp\left[-y_j(\mathbf{w}^{k-1})^T\mathbf{x}_j\right],\, j = 1, \ldots, m\right\}.$$

Then for any i, the block gradient descent update from $\mathbf{w}^{k-1}$ can be computed by

$$T_{\mathrm{GD},i}(\mathbf{w}^{k-1}) = \mathbf{w}_i^{k-1} - \alpha\sum_{j=1}^m \frac{-y_j\mathcal{M}(\mathbf{w}^{k-1})}{1 + \mathcal{M}(\mathbf{w}^{k-1})}(\mathbf{x}_j)_i,$$

which only takes $O(\frac{mn}{s})$ because computing $\exp[-y_j(\mathbf{w}^{k-1})^T\mathbf{x}_j]$ for $j = 1, ..., m$ has been avoided. Let $\mathbf{w}^k$ be obtained by applying the block gradient descent update from $\mathbf{w}^{k-1}$, as given in (24). Note that

$$\exp\big[-y_j(\mathbf{w}^k)^\top\mathbf{x}_j\big] = \exp\big[-y_j(\mathbf{w}^{k-1})^\top\mathbf{x}_j\big] \cdot \exp\big[-y_j(\mathbf{w}_{i_k}^k - \mathbf{w}_{i_k}^{k-1})^\top(\mathbf{x}_j)_{i_k}\big], \forall j.$$

Thus obtaining $\mathcal{M}(\mathbf{w}^k)$ from $\mathcal{M}(\mathbf{w}^{k-1})$ takes $O(\frac{mn}{s})$ operations. Since evaluating $\nabla F(\mathbf{w})$ takes $O(mn)$ operations because only the block $\mathbf{w}_{i_k}$ is needed, we have (25), and thus T_{GD} defined in (26) is CF.

Note that CF structure may also be found in more complicated update mappings, such as the proximal-point update, prox-linear update, and updates derived from primal-dual methods. More details may be found in [53].

4 Applications

Since we have presented and compared various update schemes and index rules for CD and investigated the coordinate friendliness of problems, we now present canonical examples that apply CD effectively from the literature. These examples demonstrate the efficiency and capability for block CD to handle larger data sets. Each example is initially expressed in its natural form, then re-formulated into a canonical form amenable to block CD methods. A CF analysis is presented for each problem by analyzing the computational cost or number of flops for each coordinate update relative to the full update. Experimental results by the block CD methods are also reported.

4.1 LASSO

In compressed sensing and machine learning, the *Least Absolute Shrinkage and Selection Operator (LASSO)* problem [76] seeks to recover a *sparse signal*, or a vector with small support, $\mathbf{x}^* \in \mathbb{R}^n$ satisfying the underdetermined linear system of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{b} \in \mathbb{R}^m$. It is closely related to the basis pursuit problem [69] in signal processing. Under certain conditions, ℓ_1 minimization returns a sparse solution. This gives the following minimization problem:

$$\underset{\mathbf{x}}{\text{minimize}}\ \|\mathbf{x}\|_1 + \frac{\lambda}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

where $\mathbf{b} \in \mathbb{R}^m$ is our compressed signal, $\lambda \in \mathbb{R}^+$ is a parameter that controls the tradeoff between sparsity and reconstruction, and $\mathbf{A} \in \mathbb{R}^{m\times n}$ is our measurement matrix with $m \ll n$. Our approach is influenced by the work of [25, 37, 79].

4.1.1 Update Derivation

Since the problem has a mixed differentiable-nondifferentiable objective, we apply the prox-linear update. Let

$$F(\mathbf{x}) = \|\mathbf{x}\|_1 + \frac{\lambda}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

Recall that the prox-linear update minimizes the component surrogate function

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} f(\mathbf{x}^{k-1}) + \langle \nabla_{i_k} f(\mathbf{x}^k), \mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\rangle + \frac{1}{2\alpha_{i_k}^{k-1}}\|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2 + r_{i_k}(\mathbf{x}_{i_k}).$$

Note that for the LASSO problem, we let $f(\mathbf{x}) = \frac{\lambda}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, and thus

$$\nabla f(\mathbf{x}) = \lambda \mathbf{A}^T(\mathbf{A}\mathbf{x} - \mathbf{b}).$$

It is also natural to choose our step size as $\alpha_{i_k}^{k-1} = 1/L_{i_k}$ where $L_{i_k} = \lambda(\mathbf{A}^T\mathbf{A})_{i_k,i_k} = \lambda\|\mathbf{A}_{:,i_k}\|^2$ is the Lipschitz constant of the gradient of the i_kth component of $f(\mathbf{x})$. Combined with $r_i(\mathbf{x}_i) = |\mathbf{x}_i|$ in the LASSO problem, the prox-linear update becomes

$$\mathbf{x}_{i_k}^k = \arg\min_{\mathbf{x}_{i_k}} \|\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}\|_2^2 + \langle \lambda \mathbf{A}_{:,i_k}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}), \mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\rangle + \frac{L_{i_k}}{2}\|\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}\|_2^2 + |\mathbf{x}_{i_k}|.$$

Using the first-order optimality conditions of this minimization, we get that

$$\begin{aligned} 0 &\in \lambda \mathbf{A}_{:,i_k}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}) + L_{i_k}(\mathbf{x}_{i_k} - \mathbf{x}_{i_k}^{k-1}) + \partial|\mathbf{x}_{i_k}|, \\ \text{or equivalently,} \quad \mathbf{x}_{i_k}^{k-1} - \frac{\lambda}{L_{i_k}}\mathbf{A}_{:,i_k}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}) &\in (I + \frac{1}{L_{i_k}}\partial|\cdot|)(\mathbf{x}_{i_k}), \end{aligned}$$

and thus

$$\mathbf{x}_{i_k}^k = \text{prox}_{\frac{1}{L_{i_k}}|\cdot|}(\mathbf{x}_{i_k}^{k-1} - \frac{\lambda}{L_{i_k}}\mathbf{A}_{:,i_k}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}))$$

as desired.

Note that $\text{prox}_{\mu|\cdot|}$, which is the proximal operator of the scaled absolute value function $\mu|\cdot|$, is the well-known shrink operator defined by

$$\text{shrink}(x, \mu) = \begin{cases} x - \mu, & \text{if } x > \mu \\ 0, & \text{if } -\mu \le x \le \mu \\ x + \mu, & \text{if } x < -\mu \end{cases}$$

which gives the coordinate update

$$\mathbf{x}_{i_k}^k = \text{shrink}\left(\mathbf{x}_{i_k}^{k-1} - \frac{\lambda}{L_{i_k}}\mathbf{A}_{:,i_k}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}), \frac{1}{L_{i_k}}\right).$$

Combining this with $L_{i_k} = \lambda\|\mathbf{A}_{:,i_k}\|^2$ gives the final coordinate update

$$\mathbf{x}_{i_k}^k = \text{shrink}\left(\mathbf{x}_{i_k}^{k-1} - \frac{1}{\|\mathbf{A}_{:,i_k}\|^2}\mathbf{A}_{:,i_k}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}), \frac{1}{\lambda\|\mathbf{A}_{:,i_k}\|^2}\right).$$

4.1.2 Continuation

Since the step size $1/L$ may be small, we may improve the speed of convergence of the algorithm by introducing *continuation*. Note that smaller values of λ dictate sparser solutions while larger values of λ admit less sparse solutions. This remark motivates defining a series of problems with increasing λ_k that reaches the final λ. Each problem is solved consecutively, with the solution to the current problem acting as the starting point for the next problem. A simple algorithm for λ_{k+1} is given by

$$\lambda_{k+1} = \eta\lambda_k$$

where $\eta > 1$. For more details, please refer to [25].

4.1.3 Derivations for Gauss-Southwell Rules

We also show the derivation for the Gauss-Southwell rules:

1. *GS-s index rule*: Recall that the GS-s rule chooses an index i_k at each iteration k by (18). Note that for the LASSO problem,

$$\nabla_j f(\mathbf{x}^{k-1}) = \lambda \mathbf{A}_{:,j}^T(\mathbf{A}\mathbf{x} - \mathbf{b})$$

$$\partial r_j(\mathbf{x}_j^{k-1}) = \begin{cases} 1 & \text{if } \mathbf{x}_j^{k-1} > 0 \\ [-1, 1] & \text{if } \mathbf{x}_j^{k-1} = 0 \\ -1 & \text{if } \mathbf{x}_j^{k-1} < 0. \end{cases}$$

   Therefore, following $g_j(\mathbf{x}^{k-1}) = \min_{\tilde{\nabla} r_j \in \partial r_j} \|\nabla_j f(\mathbf{x}^{k-1}) + \tilde{\nabla} r_j(\mathbf{x}_j^{k-1})\|$, we get

$$g_j(\mathbf{x}^{k-1}) = \begin{cases} \|\lambda \mathbf{A}_{:,j}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}) + \text{sign}(\mathbf{x}_j^{k-1})\| & \text{if } \mathbf{x}_j^{k-1} \neq 0 \\ \|\text{ shrink}(\lambda \mathbf{A}_{:,j}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}), 1)\| & \text{otherwise.} \end{cases}$$

   Choosing the largest score $g_j(\mathbf{x}^{k-1})$ gives the index i_k.

2. *GS-r index rule*: Recall that the GS-r rule chooses an index i_k at each iteration k by (19). Therefore,

$$i_k = \arg\max_j \left\| \mathbf{x}_j^{k-1} - \text{prox}_{\frac{1}{L}|\cdot|}(\mathbf{x}_j^{k-1} - \frac{\lambda}{L}\mathbf{A}_{:,j}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b})) \right\|$$

   where $L = \lambda\|\mathbf{A}^T\mathbf{A}\|$ is the Lipschitz constant of the smooth quadratic function.

3. *GS-q index rule*: Recall that the GS-q rule chooses an index i_k at each iteration k by (20). Since

$$d_j = \text{prox}_{\frac{1}{L}|\cdot|}(\mathbf{x}_j^{k-1} - \frac{\lambda}{L}\mathbf{A}_{:,j}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b})) - \mathbf{x}_j^{k-1}$$

   for the LASSO problem, we can plug in $\mathbf{d}$ into our equation

$$\frac{\lambda}{2}\|\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}\|_2^2 + \lambda d_j(\mathbf{A}_{:,j}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b})) + \frac{L}{2}|d_j|^2 + |\mathbf{x}_j^{k-1} + d_j| - |\mathbf{x}_j^{k-1}|.$$

   Finding the index for the smallest score gives the index i_k. Note that the first term is constant in j, so it can be dropped from the score computation.

4.1.4 CF Analysis

Following our analysis of least squares in Section 3.1, we give a CF analysis of the LASSO problem. Note that though the LASSO and least squares problems are similar, $\mathbf{A} \in \mathbb{R}^{m \times n}$ is a short-and-fat matrix, i.e. $m \ll n$. Therefore, the precomputation of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}^T\mathbf{b}$ is not incentivized.

Instead, we cache and maintain the quantity $\mathbf{A}\mathbf{x}^k$. Recall that for the full update, we can simply multiply first by $\mathbf{A}$ then $\mathbf{A}^T$ to take advantage of our short-and-fat matrix $\mathbf{A}$:

$$\mathbf{x}^k = \text{shrink}(\mathbf{x}^{k-1} - \frac{1}{L}\mathbf{A}^T(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b}), \frac{1}{\lambda L}).$$

This gives $\mathcal{N}[\mathbf{x}^k \mapsto T(\mathbf{x}^k)] = O(mn)$ since matrix-vector multiplication and vector addition takes $O(mn)$ operations and $O(m)$ or $O(n)$ operations, respectively.

For the coordinate update, caching $\mathbf{A}\mathbf{x}^{k-1}$ gives the cheap coordinate update

$$\mathbf{x}^k_{i_k} = \text{shrink}(\mathbf{x}^{k-1}_{i_k} - \frac{1}{L_{i_k}}(\mathbf{A}^T_{:,i_k}(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b})), \frac{1}{\lambda L_{i_k}}),$$

where computing $\mathbf{A}^T_{:,i_k}(\mathbf{A}\mathbf{x}^{k-1} - \mathbf{b})$ involves only two $O(m)$ vector-vector operations. We maintain $\mathbf{A}\mathbf{x}^k$ by adding $(\mathbf{x}^k_{i_k} - \mathbf{x}^{k-1}_{i_k})\mathbf{A}_{:,i_k}$ to $\mathbf{A}\mathbf{x}^{k-1}$, i.e.

$$\mathbf{A}\mathbf{x}^k = \mathbf{A}\mathbf{x}^{k-1} + (\mathbf{x}^k_{i_k} - \mathbf{x}^{k-1}_{i_k})\mathbf{A}_{:,i_k}.$$

Since both the coordinate update and maintenance of $\mathbf{A}\mathbf{x}^k$ take $O(m)$ operations,

$$\mathcal{N}[\{\mathbf{x}^k, \mathbf{A}\mathbf{x}^k\} \mapsto \{T_{i_k}(\mathbf{x}^k), \mathbf{A}T_{i_k}(\mathbf{x}^k)\}] = O(m) = O(\frac{1}{n}\mathcal{N}[\mathbf{x}^k \mapsto T(\mathbf{x}^k)])$$

which shows that this approach is CF.

4.1.5 Numerical Examples

For the following numerical examples, we apply our approach to a compressed, randomly generated, sparse signal, and compare the performance of the algorithm for multiple index rules. We first specify the size and type of matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$, $k = |\text{supp}(\mathbf{x}_s)|$, and standard deviation of noise σ. The entries of the matrix $\mathbf{A}$ are taken from a standard normal distribution. We choose our support $\text{supp}(\mathbf{x}_s)$ by using a random permutation, and we set non-zero entries following a normal distribution $\mathcal{N}(0, 2)$.

Next, we introduce Gaussian noise to our compressed signal $\mathbf{b} = \mathbf{A}\mathbf{x}_s + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma)$ is a Gaussian noise vector.

For our experiments, we set $m = 50$ and $n = 100$, and use $\sigma = 10^{-4}$ and $\lambda = 10^3$. We plot both the objective, norm of the gradient map, and distance to the solution in Figure 3. Note that the gradient map here is defined as

$$\mathbf{G} = \mathbf{x} - \text{prox}_{\frac{1}{L}\|\cdot\|_1}(\mathbf{x} - \frac{1}{L}\nabla f(\mathbf{x})).$$

We average the results of randomized and shuffled cyclic CD over 100 trials. We use CVX with high precision to find $\mathbf{x}^*$ for comparison [22].

Due to the sparsity of the solutions, the Gauss-Southwell rules perform significantly better than their randomized, shuffled cyclic, and cyclic counterparts. Cyclic and shuffled cyclic variants also perform better than randomized coordinate descent for this particular application, since randomized coordinate descent does not discover the support of the solution. The drop in objective also suggests that greedy and cyclic CD uses the first number of iterations to discover the support, then solves for the solution over the support, whereas randomized CD fails to exploit the sparsity of the solution.

4.2 Non-Negative Matrix Factorization for Data Mining and Dimensionality Reduction

Non-negative Matrix Factorization (NMF) [50, 10] aims at factorizing a non-negative matrix $\mathbf{M} \in \mathbb{R}^{m\times n}_+$ into two non-negative matrices of column-size r, i.e. finding non-negative $\mathbf{X} \in \mathbb{R}^{m\times r}_+$ and

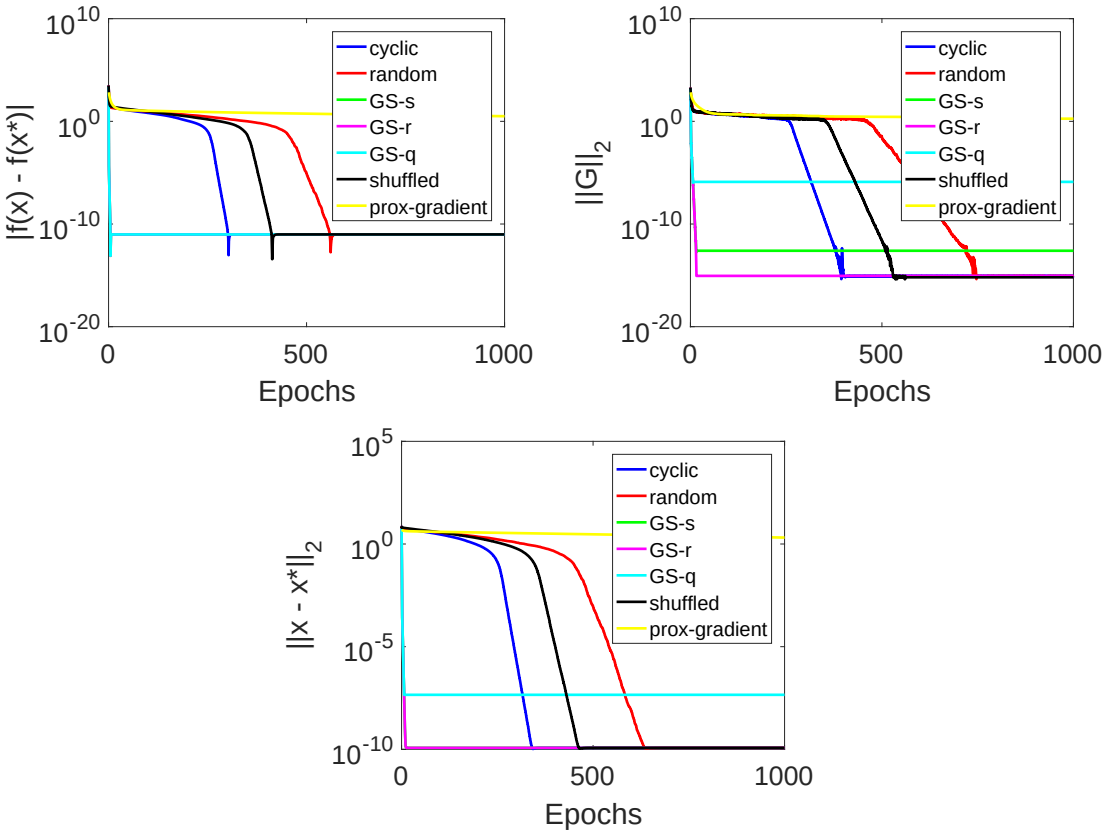

Figure 3: Convergence results for LASSO application with $m = 50$, $n = 100$, and $\sigma = 10^{-4}$. Randomized and shuffled cyclic CD results are averaged over 100 trials. The top left, top right, and bottom graphs compare the objective value $\|\mathbf{x}\|_1 + \frac{\lambda}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, norm of the gradient map, and distance to the minimizer against the number of epochs, or n coordinate updates.

$\mathbf{Y} \in \mathbb{R}_+^{n\times r}$ such that

$$\mathbf{M} = \mathbf{X}\mathbf{Y}^T, \text{ or } \mathbf{M} \approx \mathbf{X}\mathbf{Y}^T. \tag{27}$$

When r is significantly smaller than $\min\{m, n\}$, the matrix $\mathbf{M}$ is (approximately) low rank. Since the resulting matrices $\mathbf{X}$ and $\mathbf{Y}$ are both non-negative, they interpret meaningfully and are easy to inspect. This particular advantage motivates the use of NMF in a variety of fields, including text mining, computer vision, recommender systems, and signal processing.

The goal in (27) can be achieved by solving the following minimization problem:

$$\begin{aligned} \underset{\mathbf{X},\mathbf{Y}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{M} - \mathbf{X}\mathbf{Y}^T\|_F^2 \\ \text{subject to} \quad & \mathbf{X}, \mathbf{Y} \geq 0 \end{aligned} \tag{28}$$

Note that (28) is a non-convex problem due to the bilinear term $\mathbf{X}\mathbf{Y}^T$. Toward finding a solution to (28), we follow [86, 53] and apply block coordinate descent with prox-linear update. By incorporating the constraints into the objective function as indicator variables, we obtain the problem

$$\underset{\mathbf{X},\mathbf{Y}}{\text{minimize}} \, \frac{1}{2}\|\mathbf{M} - \mathbf{X}\mathbf{Y}^T\|_F^2 + \iota_{\geq 0}(\mathbf{X}) + \iota_{\geq 0}(\mathbf{Y}), \tag{29}$$

where

$$\iota_{\geq 0}(\mathbf{X}) = \begin{cases} 0 & \text{if } \mathbf{X}_{ij} \geq 0 \text{ for all } i, j \\ \infty & \text{otherwise.} \end{cases}$$

4.2.1 Update Derivation

Let

$$F(\mathbf{X}, \mathbf{Y}) = \frac{1}{2}\|\mathbf{M} - \mathbf{X}\mathbf{Y}^T\|_F^2.$$

The gradient can be easily obtained as

$$\nabla F(\mathbf{X}, \mathbf{Y}) = [\nabla_{\mathbf{X}} F(\mathbf{X}, \mathbf{Y}), \nabla_{\mathbf{Y}} F(\mathbf{X}, \mathbf{Y})] = \left[(\mathbf{X}\mathbf{Y}^T - \mathbf{M})\mathbf{Y}, \, (\mathbf{X}\mathbf{Y}^T - \mathbf{M})^T\mathbf{X}\right]. \tag{30}$$

To enforce nonnegativity, the projected gradient method goes a step along the gradient descent direction and then projects the iterate to the nonnegative orthant, i.e.,

$$\mathbf{X}^k = \max\left(0, \mathbf{X}^{k-1} - \eta_{k-1}\nabla_{\mathbf{X}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right), \tag{31a}$$

$$\mathbf{Y}^k = \max\left(0, \mathbf{Y}^{k-1} - \eta_{k-1}\nabla_{\mathbf{Y}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right), \tag{31b}$$

where η_{k-1} is the step size that can be determined by line search.

To derive the projected block coordinate gradient update, we partition the variables into disjoint blocks by columns, i.e. $\mathbf{X} = (\mathbf{X}_{:,1}, \ldots, \mathbf{X}_{:,r})$ and $\mathbf{Y} = (\mathbf{Y}_{:,1}, \ldots, \mathbf{Y}_{:,r})$. (Partition by rows or by block is also possible.) At each iteration k, we select one block variable and perform one of the following updates:

$$\mathbf{X}_{:,i_k}^k = \max\left(0, \mathbf{X}_{:,i_k}^{k-1} - \eta_{k-1}\nabla_{\mathbf{X}_{:,i_k}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right), \text{ if } i_k\text{th column of } \mathbf{X} \text{ selected} \tag{32a}$$

$$\mathbf{Y}_{:,i_k}^k = \max\left(0, \mathbf{Y}_{:,i_k}^{k-1} - \eta_{k-1}\nabla_{\mathbf{Y}_{:,i_k}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right), \text{ if } i_k\text{th column of } \mathbf{Y} \text{ selected,} \tag{32b}$$

where the partial gradients can be explicitly evaluated via

$$\nabla_{\mathbf{X}_{:,i_k}} F(\mathbf{X},\mathbf{Y}) = \mathbf{X}(\mathbf{Y}^T\mathbf{Y}_{:,i_k}) - \mathbf{M}\mathbf{Y}_{:,i_k}, \tag{33a}$$

$$\nabla_{\mathbf{Y}_{:,i_k}} F(\mathbf{X},\mathbf{Y}) = \mathbf{Y}(\mathbf{X}^T\mathbf{X}_{:,i_k}) - \mathbf{M}^T\mathbf{X}_{:,i_k}, \tag{33b}$$

and the step size η_{k-1} can be set to the reciprocal of the Lipschitz constant of the corresponding partial gradient, i.e.,

$$\eta_{k-1} = \begin{cases} \frac{1}{\|\mathbf{Y}^{k-1}_{:,i_k}\|_2^2}, & \text{for } \mathbf{X}\text{-update,} \\ \frac{1}{\|\mathbf{X}^{k-1}_{:,i_k}\|_2^2}, & \text{for } \mathbf{Y}\text{-update.} \end{cases} \tag{34}$$

To avoid *zero*-columns during each iteration that would give an infinite step sizes in (34), we can simply restrict $\mathbf{X}$ to have unit-norm columns, since $\mathbf{X}\mathbf{Y}^T = (\mathbf{X}\mathbf{D})(\mathbf{Y}\mathbf{D}^{-1})^T$ for any invertible diagonal matrix $\mathbf{D}$; see [87] for more details.

Note that if the subproblems in (32) are still expensive for extremely large problems, CD may further break $\mathbf{X}$ and $\mathbf{Y}$ into more blocks that can be updated in a sequential fashion. One can also apply direct minimization of the objective rather than the prox-linear approach described above. That causes each subproblem to be a non-negative least squares problem, which has many off-the-shelf solvers.

4.2.2 Derivations for Gauss-Southwell Rules

We give the derivations for the Gauss-Southwell rules:

1. *GS-s index rule*: Recall that the GS-s rule chooses an index i_k at each iteration k by (18). Since the subdifferential of the indicator function of a closed convex set C is the normal cone, i.e.

   $$\partial\iota_C(\mathbf{x}) = N_C(\mathbf{x}) = \{\mathbf{g} \in \mathbb{R}^n : \mathbf{g}^T\mathbf{x} \geq \mathbf{g}^T\mathbf{y} \text{ for all } \mathbf{y} \in C\},$$

   the subdifferential of $\iota_{\geq 0}$ at any $\mathbf{x} \geq 0$ is given as

   $$\partial\iota_{\geq 0}(\mathbf{x}) = \{\mathbf{g} \leq 0 : g_i x_i = 0, \forall i\}.$$

   Since $\mathbf{X}^k, \mathbf{Y}^k \geq 0, \forall k$, the indices by GS-s rule for $\mathbf{X}$ and $\mathbf{Y}$ are given respectively by

   $$i_k = \arg\max_j \|(\mathbf{G}_\mathbf{X})_{:,j}\|$$

   $$i_k = \arg\max_j \|(\mathbf{G}_\mathbf{Y})_{:,j}\|$$

   where

   $$(\mathbf{G}_\mathbf{X})_{i,j} = \begin{cases} \min(\nabla_{\mathbf{X}_{i,j}} F(\mathbf{X}^{k-1},\mathbf{Y}^{k-1}), 0) & \text{if } \mathbf{X}^{k-1}_{i,j} = 0 \\ \nabla_{\mathbf{X}_{i,j}} F(\mathbf{X}^{k-1},\mathbf{Y}^{k-1}) & \text{otherwise} \end{cases}$$

   $$(\mathbf{G}_\mathbf{Y})_{i,j} = \begin{cases} \min(\nabla_{\mathbf{Y}_{i,j}} F(\mathbf{X}^{k-1},\mathbf{Y}^{k-1}), 0) & \text{if } \mathbf{Y}^{k-1}_{i,j} = 0 \\ \nabla_{\mathbf{Y}_{i,j}} F(\mathbf{X}^{k-1},\mathbf{Y}^{k-1}) & \text{otherwise.} \end{cases}$$

2. *GS-r index rule*: Recall that the GS-r rule chooses an index i_k at each iteration k by (19). Thus,

$$i_k = \arg\max_j \|\mathbf{X}_{:,j}^{k-1} - \max\left(0, \mathbf{X}_{:,j}^{k-1} - \eta_{k-1}\nabla_{\mathbf{X}_{:,j}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right)\|$$

$$i_k = \arg\max_j \|\mathbf{Y}_{:,j}^{k-1} - \max\left(0, \mathbf{Y}_{:,j}^{k-1} - \eta_{k-1}\nabla_{\mathbf{Y}_{:,j}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right)\|$$

are the selected indices for $\mathbf{X}$ and $\mathbf{Y}$, respectively.

3. *GS-q index rule*: Recall that the GS-q rule chooses an index i_k at each iteration k by (20). Let

$$\mathbf{d}_j^x = \max\left(0, \mathbf{X}_{:,j}^{k-1} - \eta_{k-1}\nabla_{\mathbf{X}_{:,j}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right) - \mathbf{X}_{:,j}^{k-1},\ j = 1, \ldots, r,$$
$$\mathbf{d}_j^y = \max\left(0, \mathbf{Y}_{:,j}^{k-1} - \eta_{k-1}\nabla_{\mathbf{Y}_{:,j}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})\right) - \mathbf{Y}_{:,j}^{k-1},\ j = 1, \ldots, r.$$

Then the indices for $\mathbf{X}$ and $\mathbf{Y}$ are given respectively by

$$i_k = \arg\min_j \nabla_{\mathbf{X}_{:,j}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})^T \mathbf{d}_j^x + \frac{\eta_{k-1}}{2}\|\mathbf{d}_j^x\|^2,$$

$$i_k = \arg\min_j \nabla_{\mathbf{Y}_{:,j}} F(\mathbf{X}^{k-1}, \mathbf{Y}^{k-1})^T \mathbf{d}_j^y + \frac{\eta_{k-1}}{2}\|\mathbf{d}_j^y\|^2,$$

where we have used the fact that $\mathbf{X}_j^{k-1} + \mathbf{d}_j^x$ and $\mathbf{Y}_j^{k-1} + \mathbf{d}_j^y$ are both componentwise nonnegative.

4.2.3 CF Analysis

To show the CF property of the projected gradient method, we compare its per-update cost to that of the projected coordinate gradient method. Note that the full gradient update in (30) costs $O(mnr)$ operations, and with the full gradient computed, the nonnegativity projection in (31) takes another $O(mn)$ operations. Hence, one update of the projected gradient costs $O(mnr)$.

For the projected coordinate gradient update, the evaluation of partial gradient in (33) costs $O(mn)$, and with the partial gradient computed, the projection in (32) takes another $O(m)$ for the $\mathbf{X}$-update and $O(n)$ for the $\mathbf{Y}$-update. Hence, one update of the projected coordinate gradient costs $O(mn)$, and according to our definition of CF property in section 3, the projected gradient mapping is CF.

4.2.4 Numerical Example

We apply the projected coordinate gradient update in (32) to the NMF problem (28) on the ORL database from AT&T Laboratories Cambridge. This data set consists of 40 different faces taken from 10 different directions and with different expressions. We vectorize each image and obtain a matrix $\mathbf{M}$ of size 10304×400. We test its performance using cyclic, shuffled cyclic, random, and all greedy index rules. For the performance of randomized projected CD, we average the results from 100 trials. The objective and relative error decrease are plotted in Figure 4.

The GS-q outperforms all other rules while the shuffled cyclic and cyclic rules edge out the other rules. GS-s and GS-r perform surprisingly poor here despite their complexity and are comparable to randomized CD. This observation may suggest that shuffled cyclic or cyclic rules may be better than more expensive Gauss-Southwell rules for NMF.

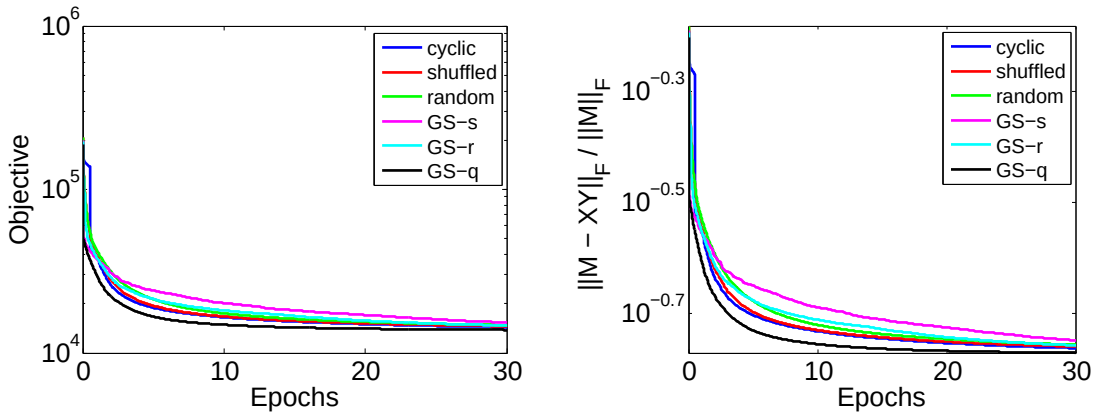

Figure 4: Convergence results comparing cyclic, shuffled cyclic, random, and greedy projected coordinate descent for the NMF problem (28) on the ORL data set. The graphs compare the objective value $\frac{1}{2}\|\mathbf{M} - \mathbf{X}\mathbf{Y}^T\|_F^2$ and relative error $\|\mathbf{M} - \mathbf{X}\mathbf{Y}^T\|_F^2/\|\mathbf{M}\|_F$ against the number of epochs.

4.3 Sparse Logistic Regression for Classification

In machine learning and statistics, we are often concerned with predicting the categories of new observations. One common approach to do this is to learn a model from a training data set with correctly classified observations, and then use the learned model to predict category membership for new observations.

Logistic regression [30] is one popularly used model. It estimates the probability of a binary response based on given multivariable data points. To fit the model, one can solve an optimization problem to search its parameters. We describe our approach below.

Given a set of training data

$$\mathcal{D} = \{(\mathbf{x}_i, y_i) |\ \mathbf{x}_i \in \mathbb{R}^n,\ y_i \in \{-1, +1\}\}, \tag{35}$$

consisting of features $\mathbf{x}_i$ and labels y_i, ℓ_1-regularized logistic regression [76] can be formulated as the following optimization problem:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \|\mathbf{w}\|_1 + C\sum_{i=1}^{l} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i})$$

where $C > 0$ is a parameter that controls the balance between regularization and loss function. Note that the ℓ_1-regularization term can be replaced by some other regularizer depending on the application.

4.3.1 Update Derivation

We follow [84, 24] and apply the *one-dimensional Newton direction* coordinate update to solve the sparse logistic regression problem (4.3). Let

$$F(\mathbf{w}) = \|\mathbf{w}\|_1 + C\sum_{i=1}^{l}\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})$$

Instead of updating $\mathbf{w}$, we update one coordinate $\mathbf{w}_{i_k}$. Therefore, we solve the one-variable sub-problem:

$$\begin{aligned}\underset{d}{\text{minimize}}\ g_{i_k}(d) &= F(\mathbf{w}^k + d\mathbf{e}_{i_k}) - F(\mathbf{w}^k)\\ &= |\mathbf{w}^k_{i_k}+d| - |\mathbf{w}^k_{i_k}| + f(\mathbf{w}^k + d\mathbf{e}_{i_k}) - f(\mathbf{w}^k)\end{aligned}$$

where

$$f(\mathbf{w}) = C\sum_{i=1}^{l}\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}).$$

At the kth iteration, given index i_k, we first find a coordinate descent direction by minimizing the second-order approximation of the smooth function,

$$\underset{d}{\text{minimize}}\,|\mathbf{w}^k_{i_k}+d| - |\mathbf{w}_{i_k}| + f'_{i_k}(\mathbf{w}^k)d + \frac{1}{2}f''_{i_k}(\mathbf{w}^k)d^2 \tag{36}$$

where $f'_{i_k}(\mathbf{w})$ and $f''_{i_k}(\mathbf{w})$ denote the first and second derivative with respect to the i_kth component, respectively. It is easy to see that (36) has a closed-form solution by using the shrinkage operator:

$$d = \begin{cases} -\frac{f'_{i_k}(\mathbf{w}^k)+1}{f''_{i_k}(\mathbf{w}^k)} & \text{if } f'_{i_k}(\mathbf{w}^k)+1 \le f''_{i_k}(\mathbf{w}^k)\mathbf{w}^k_{i_k}\\ -\frac{f'_{i_k}(\mathbf{w}^k)-1}{f''_{i_k}(\mathbf{w}^k)} & \text{if } f'_{i_k}(\mathbf{w}^k)-1 \ge f''_{i_k}(\mathbf{w}^k)\mathbf{w}^k_{i_k}\\ -\mathbf{w}^k_{i_k} & \text{otherwise.}\end{cases}$$

To guarantee a decreasing in the function value, armijo rule [80] is applied to find a stepsize $\alpha^k \in (0,1)$ such that:

$$g_{i_k}(\alpha^k d) \le \sigma\alpha^k(f'_{i_k}(\mathbf{w}^k)d + |\mathbf{w}^k_{i_k}+d| - |\mathbf{w}^k_{i_k}|)$$

where σ is an arbitrary constant in (0,1). Thus we get the update

$$\mathbf{w}^{k+1}_{i_k} = \mathbf{w}^k_{i_k} + \alpha^k d. \tag{37}$$

Note that $\mathbf{w}^k_{i_k}$ reaches the optimal value when $\frac{\partial g_{i_k}}{\partial d}(0) = 0$.

4.3.2 Derivations for Gauss-Southwell Rules

We derive the update schemes for the Gauss-Southwell rules:

1. *GS-s index rule*: Recall that the GS-s rule chooses an index i_k at each iteration k by (18). Note for the Logistic Regression problem, we have $f(\mathbf{w}) = C\sum_{i=1}^{l}\log(1+e^{-y_i\mathbf{w}^T\mathbf{x}_i})$ and $r_i(\mathbf{w}) = |\mathbf{w}_i|$, and thus

$$\nabla_j f(\mathbf{w}^{k-1}) = C\sum_{i=1}^{l}\frac{-y_i x_{ij} e^{-y_i\mathbf{w}^T\mathbf{x}_i}}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}},$$

$$\partial r_j(\mathbf{w}_j^{k-1}) = \begin{cases} 1 & \text{if } \mathbf{w}_j > 0 \\ [-1,1] & \text{if } \mathbf{w}_j = 0 \\ -1 & \text{if } \mathbf{w}_j < 0, \end{cases}$$

where x_{ij} is the jth entry of $\mathbf{x}_i$. Thus, if $g_j(\mathbf{w}^{k-1}) = \min_{\tilde{\nabla} r_j \in \partial r_j} \|\nabla_j f(\mathbf{w}^{k-1}) + \tilde{\nabla} r_j(\mathbf{w}_j^{k-1})\|$,

$$g_j(\mathbf{w}^{k-1}) = \begin{cases} \|C\sum_{i=1}^{l}\frac{-y_i x_{ij} e^{-y_i\mathbf{w}^T\mathbf{x}_i}}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}} + \text{ sign}(\mathbf{w}_j^{k-1})\| & \text{if } \mathbf{w}_j \neq 0 \\ \|\text{ shrink }(C\sum_{i=1}^{l}\frac{-y_i x_{ij} e^{-y_i\mathbf{w}^T\mathbf{x}_i}}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}, 1)\| & \text{otherwise.} \end{cases}$$

Choosing the largest score $|g_j(\mathbf{w}^{k-1})|$ gives the index i_k.

2. *GS-r index rule*: Recall that the GS-r rule chooses an index i_k at each iteration k by (19). Thus,

$$i_k = \arg\max_j \|\mathbf{w}_j^{k-1} - \text{prox}_{\frac{1}{L}|\cdot|}(\mathbf{w}_j^{k-1} - \frac{C}{L}\sum_{i=1}^{l}\frac{-y_i x_{ij} e^{-y_i\mathbf{w}^T\mathbf{x}_i}}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}})\|,$$

where L is the Lipschitz constant of ∇f. Choosing the largest score $|d|$ gives the index i_k.

3. *GS-q index rule*: Recall that the GS-q rule chooses an index i_k at each iteration k by (20). Since

$$d = \text{prox}_{\frac{1}{L}|\cdot|}(\mathbf{w}_j^{k-1} - \frac{C}{L}\sum_{i=1}^{l}\frac{-y_i x_{ij} e^{-y_i\mathbf{w}^T\mathbf{x}_i}}{1+e^{-y_i\mathbf{w}^T\mathbf{x}_i}}) - \mathbf{w}_j^{k-1},$$

we can plug d into equation

$$f(\mathbf{w}^{k-1}) + d\nabla_j f(\mathbf{w}^{k-1}) + \frac{L}{2}|d|^2 + |\mathbf{w}_j^{k-1} + d| - |\mathbf{w}_j^{k-1}|$$

Finding the index for the smallest score gives the index i_k.

4.3.3 CF analysis

Note that

$$f_j'(\mathbf{w}^{k-1}) = C\sum_{i=1}^{l} y_i x_{ij}(\frac{1}{1+e^{-y_i\mathbf{w}^{k-1}\mathbf{x}_i}} - 1),$$

$$f_j''(\mathbf{w}^{k-1}) = C\sum_{i=1}^{l} x_{ij}^2(\frac{1}{1+e^{-y_i\mathbf{w}^{k-1}\mathbf{x}_i}})(1 - \frac{1}{1+e^{-y_i\mathbf{w}^{k-1}\mathbf{x}_i}}),$$

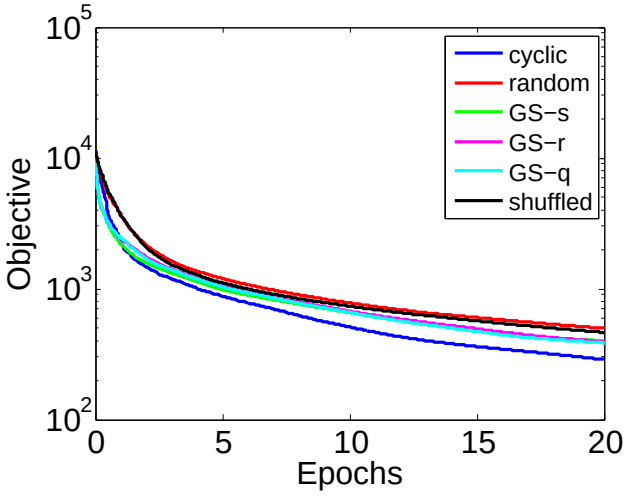

Figure 5: The graph shows the objective decrease for each iteration of CD with various schemes of coordinate selection on solving the sparse logistic regression.

where we have eliminated $y_i^2 \equiv 1$. We cache $e^{-y_i \mathbf{w}^{k-1} \mathbf{x}_i}$ and, following (37), update $e^{-y_i \mathbf{w}^k \mathbf{x}_i}$ by

$$e^{-y_i \mathbf{w}^k \mathbf{x}_i} = e^{-y_i \mathbf{w}^{k-1} \mathbf{x}_i} \cdot e^{-y_i \lambda d x_{ij}}$$

Then the computations take $O(l)$ operations, so

$$\mathcal{N}[\{\mathbf{w}_{i_k}^k \mapsto \{T_{i_k}(\mathbf{w}_{i_k}^k)\}] = O(l) = O(\frac{1}{l}\mathcal{N}[\mathbf{w}^k \mapsto T(\mathbf{w}^k)]),$$

which shows that this problem has CF structure.

4.3.4 Numerical Example

We generate $m = 100$ data points from two different bivariate normal distributions and apply CD to (4.3). The objective decrease is plotted in Figure 5. The randomized and shuffled cyclic variants were averaged over 100 trials.

Note that cyclic CD outperforms all other variants, including all of the Gauss-Southwell rules. Peculiarly, shuffled cyclic performs significantly worse than cyclic CD, which is worthy of investigation.

4.4 Support Vector Machines for Classification

We consider another popular classification model, support vector machines (SVM) [73], to predict categories of new observations. Different from the logistic regression that estimates the probability of a binary response, the SVM is a non-probabilistic classifier, and it can be formulated as the

following optimization problem:

$$\begin{aligned} \underset{\mathbf{w},b,\xi}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{m}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) \geq 1-\xi_i,\ \xi_i \geq 0, \quad \forall i. \end{aligned} \tag{38}$$

where ξ_i's are slack variables, and $C > 0$ is a penalty parameter. For an optimal solution $(\mathbf{w}, b, \xi)$ to (38), it is easy to see that $\xi_i = \max(1 - y_i(\mathbf{w}^T\mathbf{x}_i - b), 0)$, $\forall i$.

This problem can be interpreted as finding the best separating hyperplane that classifies the data by maximizing the margin width from the hyperplane to the nearest data point of either class. To simplify the problem, we enforce $b = 0$ that corresponds to the unbiased case, and we reformulate and solve its dual problem by substituting $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{minimize}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \leq \boldsymbol{\alpha}_i \leq C, \quad \forall i, \end{aligned}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^m$ is the dual variable, and $Q_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$.

4.4.1 Update Derivation

Let

$$F(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - \mathbf{1}^T\boldsymbol{\alpha}.$$

Then our goal is to solve

$$\underset{\boldsymbol{\alpha}}{\text{minimize}}\, F(\boldsymbol{\alpha}) + \iota_{[0,C]}(\boldsymbol{\alpha}). \tag{39}$$

where $\iota_{[0,C]}(\boldsymbol{\alpha})$ denotes the indicator function over the feasible set $\{\boldsymbol{\alpha} : 0 \leq \boldsymbol{\alpha}_i \leq C, \forall i\}$. We apply the prox-linear update (9) to solve this problem.

The prox-linear update solves the subproblem

$$\boldsymbol{\alpha}_{i_k}^k = \underset{\boldsymbol{\alpha}_{i_k}}{\arg\min}\langle \mathbf{Q}_{i_k,:}\boldsymbol{\alpha}^{k-1} - 1, \boldsymbol{\alpha}_{i_k} - \boldsymbol{\alpha}_{i_k}^{k-1}\rangle + \frac{L_{i_k}}{2}\|\boldsymbol{\alpha}_{i_k} - \boldsymbol{\alpha}_{i_k}^{k-1}\|^2 + \iota_{[0,C]}(\boldsymbol{\alpha}_{i_k}).$$

The first-order optimality condition of this minimization is

$$\begin{aligned} & 0 \in \mathbf{Q}_{i_k,:}\boldsymbol{\alpha}^{k-1} - 1 + L_{i_k}(\boldsymbol{\alpha}_{i_k} - \boldsymbol{\alpha}_{i_k}^{k-1}) + \partial\iota_{[0,C]}(\boldsymbol{\alpha}_{i_k}), \\ \text{equivalently,} \quad & \boldsymbol{\alpha}_{i_k}^{k-1} - \frac{1}{L_{i_k}}(\mathbf{Q}_{i_k,:}\boldsymbol{\alpha}^{k-1} - 1) \in (I + \frac{1}{L_{i_k}}\partial\iota_{[0,C]})(\boldsymbol{\alpha}_{i_k}), \end{aligned}$$

and thus

$$\mathbf{x}_{i_k}^k = \text{prox}_{\iota_{[0,C]}}(\boldsymbol{\alpha}_{i_k}^{k-1} - \frac{1}{L_{i_k}}(\mathbf{Q}_{i_k,:}\boldsymbol{\alpha}^{k-1} - 1))$$

as desired. Since the proximal operator of the indicator function is the projection operator, this yields the projected coordinate update

$$\boldsymbol{\alpha}_{i_k}^{k+1} = \min(\max(\boldsymbol{\alpha}_{i_k}^k - \frac{\nabla_{i_k} f(\boldsymbol{\alpha}^k)}{Q_{i_k,i_k}}, 0), C). \tag{40}$$

4.4.2 Derivations for Gauss-Southwell Rules

We derive the update schemes for the Gauss-Southwell rules:

1. *GS-s index rule*: Recall that the GS-s rule chooses an index i_k at each iteration k by (18). Note for the SVM problem, $f(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\mathbf{Q}\boldsymbol{\alpha} - 1^T\boldsymbol{\alpha}$, and thus

$$\nabla f(\boldsymbol{\alpha}^{k-1}) = \mathbf{Q}\boldsymbol{\alpha}^{k-1} - 1.$$

   In addition, for any $\boldsymbol{\alpha}$ with $0 \leq \alpha_i \leq C$, $\forall i$, we have that any $\mathbf{g} \in \partial\iota_{[0,C]}(\boldsymbol{\alpha})$ satisfies

$$g_i \begin{cases} \leq 0, & \text{if } \alpha_i = 0, \\ = 0, & \text{if } 0 < \alpha_i < C, \\ \geq 0, & \text{if } \alpha_i = C, \end{cases}$$

   for all i. Therefore, by (18), the scores are computed by

$$S_j(\boldsymbol{\alpha}^{k-1}) = \begin{cases} \min(\mathbf{Q}_{j,:}\boldsymbol{\alpha}^{k-1} - 1, 0), & \text{if } \boldsymbol{\alpha}_j^{k-1} = 0, \\ \max(\mathbf{Q}_{j,:}\boldsymbol{\alpha}^{k-1} - 1, 0), & \text{if } \boldsymbol{\alpha}_j^{k-1} = C, \\ \mathbf{Q}_{j,:}\boldsymbol{\alpha}^{k-1} - 1, & \text{otherwise.} \end{cases}$$

   We choose the index corresponding to the largest score, i.e., $i_k = \arg\max_j |S_j(\boldsymbol{\alpha}^{k-1})|$.

2. *GS-r index rule*: Recall that the GS-r rule chooses an index i_k at each iteration k by (19). Since the proximal operator of the indicator $\iota_{[0,C]}$ is the projection operator, the index is given by

$$i_k = \arg\max_j \left|\boldsymbol{\alpha}_j^{k-1} - \min(\max(\boldsymbol{\alpha}_j^{k-1} - \frac{1}{\mathbf{Q}_{j,j}}\nabla_j f(\boldsymbol{\alpha}^{k-1}), 0), C)\right|.$$

3. *GS-q index rule*: Recall that the GS-q rule chooses an index i_k at each iteration k by (20). Let

$$d = \min(\max(\boldsymbol{\alpha}_j^{k-1} - \frac{1}{\mathbf{Q}_{j,j}}\nabla_j f(\boldsymbol{\alpha}^{k-1}), 0), C) - \boldsymbol{\alpha}_j^{k-1}.$$

   Plugging d into the quadratic approximation

$$f(\boldsymbol{\alpha}^{k-1}) + \nabla_j f(\boldsymbol{\alpha}^{k-1})^T d + \frac{L_j}{2}\|d\|^2$$

   yields the greedy scores. Choosing the index corresponding to the smallest score yields the index i_k.

4.4.3 CF Analysis

It is easy to see that one coordinate update (40) for SVM costs $O(m)$, mainly in evaluating $\mathbf{Q}_{i_k,:}\boldsymbol{\alpha}^k$, while a full projected gradient update needs to first compute the gradient $\mathbf{Q}\boldsymbol{\alpha}^k$ that costs $O(m^2)$ and then project at an additional cost of $O(m)$. Hence, by the definition of CF property discussed in section 3, the update (40) is CF.

For the coordinate update, we cache $\mathbf{Q}\boldsymbol{\alpha}^{k-1}$ and update $\boldsymbol{\alpha}^k$ as

$$\boldsymbol{\alpha}_{i_k}^k = \min(\max(\boldsymbol{\alpha}_{i_k}^{k-1} - \frac{(\mathbf{Q}\boldsymbol{\alpha}^{k-1})_{i_k} - 1}{Q_{i_k,i_k}}, 0), C).$$

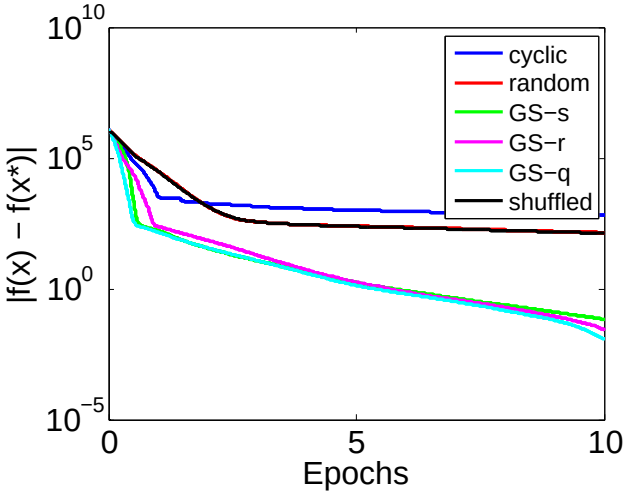

Figure 6: The graph shows the objective decrease with respect to the number of iterations by applying CD to SVM on the a2a dataset.

We then maintain $\mathbf{Q}\boldsymbol{\alpha}^k$ by

$$\mathbf{Q}\boldsymbol{\alpha}^k = \mathbf{Q}\boldsymbol{\alpha}^{k-1} + \mathbf{Q}_{i_k,:}(\boldsymbol{\alpha}^k_{i_k} - \boldsymbol{\alpha}^{k-1}_{i_k}).$$

Both steps take $O(m)$ operations, so this way also shows the update (40) is CF.

4.4.4 Numerical Example

We train a classifier by applying our approach to the a2a training set, consisting of 20,242 training examples with 123 features [8, 39]. We average our results over 10 trials. We use the update given in (40) and compare cyclic, randomized, shuffled cyclic, GS-s, GS-r, and GS-q index rules. The convergence results for the objective decrease over 10 epochs are plotted in Figure 6.

We note, in particular, that the GS-q and GS-s rule perform significantly better than the other rules for this application. Randomized, GS-r, and shuffled cyclic CD perform similarly.

4.5 Semidefinite Programming

A standard semidefinite program (SDP) is of the following form:

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \langle \mathbf{C}, \mathbf{X} \rangle \\ \text{subject to} \quad & \mathcal{A}(\mathbf{X}) = \mathbf{b}, \\ & \mathbf{X} \succeq 0, \end{aligned} \tag{41}$$

where $\mathcal{A}$ is a linear operator. Typically, SDPs can be solved in polynomial time using interior-point methods, but in practice, large-scale SDPs require an enormous amount of work at each iteration if an interior-point method is used. Because of the increasing size of the SDPs encountered in

modern applications, [82] develops a block coordinate descent approach that can solve large-scale SDPs much more cheaply per iteration than interior-point methods.

The procedure in [82] may be interpreted as a row-by-row block minimization method. It can be summarized as cyclically updating the rows and columns of $\mathbf{X}$, one pair at a time, by minimizing the objective of (41) and keeping the constraints satisfied. Namely, let $\mathbf{X}^{k,i}$ denote the current value of $\mathbf{X}$ before performing the ith inner update in the kth outer iteration. At a given outer iteration k and inner iteration i, the algorithm seeks to update $\mathbf{X}$ to $\mathbf{X}^{k,i+1}$ by solving the following optimization problem:

$$\begin{aligned} \underset{\mathbf{X}}{\text{minimize}} \quad & \langle \mathbf{C}, \mathbf{X} \rangle \\ \text{subject to} \quad & \mathcal{A}(\mathbf{X}) = \mathbf{b}, \\ & \mathbf{X} \succeq 0, \\ & \mathbf{X}_{\neq i, \neq i} = \mathbf{X}^{k,i}_{\neq i, \neq i}, \end{aligned}$$

where $\mathbf{X}_{\neq i, \neq i}$ denotes the submatrix of $\mathbf{X}$ excluding its ith row and column. This method, called the RBR method, has theoretical guarantees for SDPs with simple bound constraints, but may also be applied to more general bound constraints. We refer interested readers to [82] for more information about the method.

5 Implementations for Large-Scale Systems

Although serial CD performs well by relying on cheap updates, serial CD may not be capable of solving some large-scale systems due to memory and speed constraints in the problem. In this case, additional speedup must be gained by implementing portions of the solver in a parallel or distributed fashion. Many large-scale applications already utilize parallelisms, such as video processing, 4D-CT processing, large-scale dynamical systems, systems with streaming data, and tensor factorizations.

Parallel computing breaks a problem into simpler parts that are executed simultaneously by multiple agents while being coordinated by a controller. By using multiple cores, CPUs, or networked computers in parallel, we can overcome potential memory and speed constraints in solving our problem.

In this section, we motivate the use of parallel and distributed computing for scaling coordinate descent algorithms for larger systems. We introduce some solutions for scaling CD for larger problems in both multicore and multi-machine architectures. We also discuss parallelized numerical linear algebra methods and parallelized CD methods and give relevant problem structures and resources for implementation.

5.1 Parallelization of Coordinate Updates

By leveraging the CF properties of a problem as we have discussed in Section 3, coordinate descent algorithms are made computationally worthwhile by relying on cheap updates to iteratively solve optimization problems. However, as our problem scales in size, the amount of work for each coordinate block update increases, stalling the computation time of each update.

For example, consider the computation of the gradient for the least-squares problem, $\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}$, which we have discussed earlier. If our data matrix $\mathbf{A}$ has dimensions $1,000,000 \times 1,000,000$, then the cost of computing a coordinate update using 400 coordinate blocks is equivalent to the cost

of calculating a full gradient update for a problem with dimensions $50,000 \times 50,000$. Therefore, to solve large-scale problems quickly, we must leverage multicore systems to gain additional speedup.

5.1.1 Parallelized Numerical Linear Algebra

Consider the least squares problem as discussed in Section 3. Recall that the block coordinate update for least squares is given by:

$$\mathbf{x}_{i_k}^k = \mathbf{x}_{i_k}^{k-1} - \alpha((\mathbf{A}^T\mathbf{A})_{i_k}\mathbf{x}_{i_k}^{k-1} - (\mathbf{A}^T\mathbf{b})_{i_k})$$

In order to compute a coordinate update for least squares, assuming $\mathbf{A}^T\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{A}^T\mathbf{b} \in \mathbb{R}^m$ are precomputed and we have s blocks, the cost of each operation is as follows:

1. Matrix-Vector Multiplication $(\mathbf{A}^T\mathbf{A})_{i_k}\mathbf{x}_{i_k}^{k-1}$: $O(\frac{m^2}{s})$
2. Vector Difference $(\mathbf{A}^T\mathbf{A})_{i_k}\mathbf{x}_{i_k}^{k-1} - (\mathbf{A}^T\mathbf{b})_{i_k}$: $O(\frac{m}{s})$
3. Scalar-Vector Multiplication $\alpha((\mathbf{A}^T\mathbf{A})_{i_k}\mathbf{x}_{i_k}^{k-1} - (\mathbf{A}^T\mathbf{b})_{i_k})$: $O(\frac{m}{s})$
4. Vector Difference $\mathbf{x}_{i_k}^{k-1} - \alpha((\mathbf{A}^T\mathbf{A})_{i_k}\mathbf{x}_{i_k}^{k-1} - (\mathbf{A}^T\mathbf{b})_{i_k})$: $O(\frac{m}{s})$

Assuming no communication cost, we see that the major bottleneck in computing each coordinate update consists of numerical linear algebra operations. Therefore, we can improve the efficiency of our coordinate updates by parallelizing our numerical linear algebra operations.

Since writing stable, efficient parallel numerical linear algebra solvers may be difficult, we list some common libraries and packages that are useful for implementing parallelized numerical linear algebra and point the reader to additional reports and references readily available in the public domain:

- **BLAS**: Basic Linear Algebra Subprograms (BLAS) is the standard low-level routines for performing linear algebra operations. BLAS has been implemented in both sequential and parallel fashions and has been designed to be highly optimized for high-performance computing. BLAS is categorized into three levels: Level 1 consists of vector operations, Level 2 consists of matrix-vector operations, and Level 3 consists of matrix-matrix operations. Please refer to the BLAS website at `http://www.netlib.org/blas/` for more information.
- **LAPACK**: Linear Algebra PACKage (LAPACK) provides routines for solving more complex linear algebra operations, such as solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and beyond. These libraries are designed to run on shared-memory parallel processors. Please refer to the LAPACK website at `http://www.netlib.org/lapack/` for more information.
- **PLASMA**: Parallel Linear Algebra Software for Multicore Architectures (PLASMA) is a dense linear algebra package designed for multicore computing. PLASMA offers routines for solving linear systems of equations, least squares problems, eigenvalue problems, etc. Please refer to the PLASMA website at `http://icl.cs.utk.edu/plasma/` for more details.

5.1.2 Parallelized Coordinate Descent

Alternatively, we can also improve the timeliness of CD for large-scale problems by parallelizing the coordinate updates, and gain an approximate speedup (ideally) proportional to the number of processors. In particular, rather than performing each operation faster, each core performs a partial coordinate update, which together form a full coordinate or gradient update. In order for parallel CD to be effective, it again depends on the CF analysis of our problem. We list some fairly common problem structures which lend themselves well to parallelizing CD:

- *Separability*: A function F is separable if it can be written as

$$F(\mathbf{x}) = \sum_{i=1}^{s} f_i(\mathbf{x}_i)$$

  where each f_i only depends on a non-overlapping coordinate block $\mathbf{x}_i$ of $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_s)$.

- *Partial separability*: A function F is partially separable if it can be written as

$$F(\mathbf{x}) = \sum_{J \in \mathcal{J}} f_J(\mathbf{x})$$

  where $\mathcal{J}$ is a finite collection of nonempty subsets of $\{1, ..., s\}$ and f_J depends on blocks $\mathbf{x}_i$ for $i \in J$ only. If

$$|J| \leq \omega \text{ for all } J \in \mathcal{J}$$

  then we say that f is partially separable with degree ω.

Clearly, minimizing a separable function $F(\mathbf{x})$ is equivalent to the independent minimization of each f_i over $\mathbf{x}_i$, which is obviously parallelizable since the objective is minimized if each core minimizes over each or partition of functions f_i.

Partially separable functions are also useful since they only couple some components of $\mathbf{x}$ together. Some common examples of partially separable functions include:

1. Square Loss: $f_j(\mathbf{x}, \mathbf{A}_j, \mathbf{y}_j) = \frac{1}{2}(\mathbf{A}_j^T\mathbf{x} - \mathbf{y}_j)^2$
2. Logistic Loss: $f_j(\mathbf{x}, \mathbf{A}_j, \mathbf{y}_j) = \log(1 + e^{-\mathbf{y}_j\mathbf{A}_j^T\mathbf{x}})$
3. Hinge Square Loss: $f_j(\mathbf{x}, \mathbf{A}_j, \mathbf{y}_j) = \frac{1}{2}\max\{0, 1 - \mathbf{y}_j\mathbf{A}_j^T\mathbf{x}\}^2$

where $\mathbf{A}_j \in \mathbb{R}^n$ is a training example with label $\mathbf{y}_j \in \mathbb{R}$ and $F(\mathbf{x}) = \sum_{j=1}^{m} f_j(\mathbf{x}, \mathbf{A}_j, \mathbf{y}_j)$. As $\mathbf{A}$ is a sparse matrix, each example may depend only on a few features, and thus the objective is partially separable. The maximum number of dependencies over all examples is then the degree of partial separability ω [62].

When parallelizing CD, we typically solve problems of the form

$$\underset{\mathbf{x}}{\text{minimize}} \;\; F(\mathbf{x}) = f(\mathbf{x}) + r(\mathbf{x})$$

where f is a partially separable smooth convex function and r is a simple separable convex function, also usually proximable. This is an example of a problem that parallelizes well with CD.

Many parallel CD implementations using partial coordinate updates have already been investigated with various update schemes, sampling schemes, or step sizes, exploiting certain coordinate-friendly structures, or reducing operations by maintaining additional quantities involved in the algorithm. We refer the reader to additional work done in [7, 19, 20, 32, 43, 45, 52, 61, 62, 62, 75] for more detail on specific implementations.

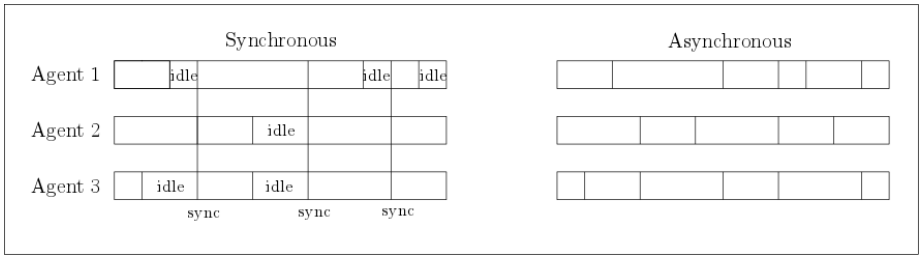

Figure 7: Comparison of synchronous parallel and asynchronous parallel implementations.

Synchrony and Asynchrony Performing partial coordinate updates with each core may also be interpreted as performing coordinate updates on a finer partition of coordinate blocks, with a synchronization step after each set of updates. The relaxation of this synchronization gives another set of methods for parallelizing CD: asynchronous parallel CD algorithms.

Synchronous and asynchronous parallelism may be summarized as:

1. *Synchronous Parallelism*: Synchronous algorithms distribute the coordinate computation across multiple agents and regularly synchronize across all agents to ensure consistency. The synchronization step consists of sharing the results of all coordinate updates across all agents before further computation.

2. *Asynchronous Parallelism*: Asynchronous algorithms weaken or eliminate consistent synchronization across agents while still partitioning computation for execution in parallel on multiple agents. Each agent may compute with the possibly stale information it has, even if the results from other agents have not been received.

One can easily describe this difference through an intuitive example. Suppose the lead of a project would like to divide up operations between multiple employees. One approach would be to delegate a batch of operations for a set of employees, one operation for each employee, and wait until all employees' work has been completed before moving onto the next phase of the project. This setting would correspond to synchronous parallelization. Alternatively, the project lead may delegate a new operation to each employee as each finishes their work regardless of the other employees' progress. This setting would correspond to asynchronous parallelization.

We highlight some major differences between synchronous and asynchronous CD methods. In particular, synchronous CD methods have been studied for much longer and are much more reliable. They have already been implemented for a variety of systems, and software is publicly available. However, synchronization requires every core, no matter how efficient, to wait for the slowest core to be communicated. Concurrent data exchanges during synchronization lead to slowdown due to lock contention and bus contention. Consequently, the speedup factor of synchronous parallel algorithms seldom reaches the number of cores or processors.

On the other hand, asynchronous methods relax the synchronization step and therefore reduce idle time and contentions that synchronization creates. In asynchronous methods, each core or processor instead performs computations using whatever information it has, regardless of the current state of the data. As a result, asynchronous methods have the potential to perform much faster - in one experiment on solving sparse logistic regression, demonstrating a 25x speedup on 32 cores rather than 4x speedup with synchronization. However, asynchrony introduces input delay or age

to the components of $\mathbf{x}$ used. This change makes it challenging to determine rigorous analysis. Since components are constantly updated in asynchronous computing, line search cannot be used to select a step size. Also, very few open-source software packages are available for asynchronous coordinate methods at this time. A recent open-source package is TMAC [18].

Figure 7 summarizes the main difference between synchronous and asynchronous approaches [83]. Asynchronous CD variants differ in the assumptions they make on the choice of update components i_k, on the step size, and on the "age" of the components of $\mathbf{x}^k$ used. We refer the reader to additional work in [5, 41, 40, 31, 54, 26] for discussion on specific asynchronous variants.

5.1.3 Resources

We briefly list some popular programming resources for implementing parallelized coordinate methods on a multicore machine or cluster.

- **Multithreading**: Multiple threads can also be executed concurrently by one or more cores while sharing memory resources. This introduces thread-level parallelism and increases utilization of the cores. The usage and syntax for threads differ for various programming languages but are supported in common languages such as C/C++, Java, and Python.
- **OpenMP**: Open Multi-Processing (OpenMP) is an API for multi-platform shared-memory parallel programming in C/C++ and Fortran. It has its set of compiler directives, library routines, and environment variables that influence run-time behavior, and provides a simple interface for developing parallel applications. More information may be found at `http://openmp.org`.
- **MPI**: Message Passing Interface (MPI) is a standardized message-passing system for parallel computing. There are several efficient open-source implementations of MPI available for parallel software development, supported in common languages like C/C++, Java, Python, Matlab, and R.

6 Conclusion

Coordinate descent has become an important optimization tool used to solve many problems arising from machine learning and large data analysis. We introduced and surveyed modern coordinate descent methods, including both elementary and block settings, for engineers and practitioners. We gave relevant theory and examples to help practitioners implement CD for various applications. Interested readers may refer to the bibliography for further elaborations and extensions on the topics explored in this monograph. We expect new adaptations and variants on coordinate descent methods as well as the new theory for understanding the nuances of CD to be developed as this class of algorithms become more understood and utilized in the major application areas.

Acknowledgements

We would like to thank Charlotte Abrahamson, Yan Dong, Brent Edmunds, Xiaoyi Gu, Robert Hannah, Zhimin Peng, Tianyu Wu, and Wenli Yan for their helpful and insightful comments.

A Modeling Using Extended Valued Functions

It is often useful in optimization to reformulate problems using *extended valued functions*. They may be used to incorporate domain of functions or feasible set constraints in the objective function. We give a precise treatment below.

Definition 1. *An extended valued function is a function that maps to elements in the extended real line* $f : \mathbf{X} \mapsto \bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, \infty\}$.

To motivate the use of extended valued functions in optimization, we present some examples.

Example 1. *Let* $\mathcal{X} \subset \mathbb{R}^n$. *Then the indicator function of* $\mathcal{X}$, *defined as*

$$\iota_{\mathcal{X}}(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in \mathcal{X} \\ \infty & \text{otherwise,} \end{cases}$$

is an extended valued function.

Indicator functions allow us to write constraints into the objective function and treat our problem as an unconstrained minimization problem. In particular, consider the problem

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} \;\; f(\mathbf{x}) \\ &\text{subject to} \;\; \mathbf{x} \in \mathcal{X} \end{aligned}$$

where f is a convex function and $\mathcal{X} \subset \mathbb{R}^n$ is a convex set. Then our constrained problem may be rewritten as

$$\underset{\mathbf{x}}{\text{minimize}} \;\; f(\mathbf{x}) + \iota_{\mathcal{X}}(\mathbf{x}).$$

In addition, we can use extended valued functions to ignore the domain of the function.

Example 2. *Consider the problem*

$$\underset{x>0}{\text{minimize}} \;\; f(x) = \frac{1}{\sqrt{x}} + x.$$

We can define a new function

$$\tilde{f}(x) = \begin{cases} \frac{1}{\sqrt{x}} + x & \text{if } x > 0 \\ \infty & \text{otherwise} \end{cases}$$

to remove implicit domain constraints from the function. Since f *and* $\tilde{f}$ *share the same set of minimizers, it is sufficient to consider the minimization of* $\tilde{f}$.

By similarly rewriting optimization problems using extended valued functions, we can ignore constraints and instead optimize an unconstrained extended-valued problem, and therefore apply unconstrained minimization techniques, such as coordinate descent methods.

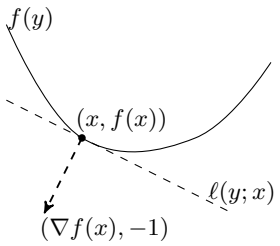

Figure 8: Geometric interpretation of the definition of convexity for differentiable functions.

B Subdifferential Calculus

Recall that for differentiable convex functions, we have that

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$$

for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$. In words, the line $\ell(\mathbf{y}; \mathbf{x}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$ is a linear underestimator of the function f at $\mathbf{x}$. This interpretation of convexity is shown in Figure 8. In particular, this inequality is equivalent to

$$\langle (\nabla f(\mathbf{x}), -1), (\mathbf{y} - \mathbf{x}, f(\mathbf{y}) - f(\mathbf{x})) \rangle = \langle \nabla f(\mathbf{x}), \mathbf{y} - \mathbf{x} \rangle + f(\mathbf{x}) - f(\mathbf{y}) \leq 0.$$

In words, the line connecting $(\mathbf{x}, f(\mathbf{x}))$ and $(\mathbf{y}, f(\mathbf{y}))$ makes an obtuse angle with the vector $(\nabla f(\mathbf{x}), -1)$, as shown in the figure.

This definition of convexity motivates a more general notion of the gradient that applies to non-differentiable convex functions, called a *subgradient*.

Definition 2. *A subgradient at* $\mathbf{x} \in \mathrm{dom} f$ *is any element* $\mathbf{g} \in \mathbb{R}^n$ *such that*

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}), \ \forall \mathbf{y} \in \mathrm{dom} f.$$

The subdifferential $\partial f(\mathbf{x})$ *is the set of all subgradients at* $\mathbf{x}$*, i.e.*

$$\partial f(\mathbf{x}) := \{\mathbf{g} \in \mathbb{R}^n : f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}), \ \forall \mathbf{y} \in \mathrm{dom} f\}.$$

Note that subdifferentials are nonempty for proper convex functions in the interiors of their domains. Subdifferentials may be empty on the domain boundaries for convex functions and anywhere for non-convex functions.

Subgradients and subdifferentials help guide the development and analysis of optimization algorithms for non-differentiable functions. The most common example of a non-differentiable convex function is the absolute value function $|x|$. The subdifferential of $|x|$ is

$$\partial |x| = \begin{cases} \{1\} & \text{if } x > 0 \\ [-1, 1] & \text{if } x = 0 \\ \{-1\} & \text{if } x < 0 \end{cases}.$$

A subgradient of $|x|$ at $x = 0$ is shown in Figure 9.

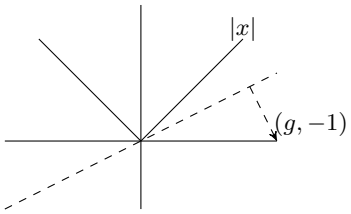

Figure 9: A subgradient of $f(x) = |x|$ at $x = 0$.

Another common example is the indicator function of a closed nonempty convex set C. Then by definition, the subdifferential of the indicator function at $\mathbf{x} \in C$ is

$$\begin{aligned}\partial \iota_C(\mathbf{x}) &= \{\mathbf{g} \in \mathbb{R}^n : \iota_C(\mathbf{y}) \geq \iota_C(\mathbf{x}) + \mathbf{g}^T(\mathbf{y} - \mathbf{x}),\ \forall \mathbf{y} \in \mathbb{R}^n\} \\ &= \{\mathbf{g} \in \mathbb{R}^n : \mathbf{g}^T(\mathbf{y} - \mathbf{x}) \leq 0,\ \forall \mathbf{y} \in C\} \\ &:= N_C(\mathbf{x}),\end{aligned}$$

also called the *normal cone*. Note if $\mathbf{x} \notin C$, then the subdifferential $\partial \iota_C(\mathbf{x}) = \emptyset$.

We state some well-known properties of subgradients and subdifferentials.

Properties. *Assume all functions are proper convex, and ϕ_i is differentiable for all i.*

1. *If f is differentiable, then $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$.*
2. *If $f(\mathbf{x}) = c_1 f_1(\mathbf{x}) + c_2 f_2(\mathbf{x})$ with $c_1, c_2 \geq 0$, then*
$$\partial f(\mathbf{x}) \supseteq c_1 \partial f_1(\mathbf{x}) + c_2 \partial f_2(\mathbf{x}).$$
3. *If $f(\mathbf{x}) = h(\mathbf{A}\mathbf{x} + \mathbf{b})$, then*
$$\partial f(\mathbf{x}) \supseteq \mathbf{A}^T \partial h(\mathbf{A}\mathbf{x} + \mathbf{b}).$$
4. *If $\lambda \geq 0$ and $f(\mathbf{x}) = h(\lambda \mathbf{x})$, then*
$$\partial f(\mathbf{x}) = \lambda \partial h(\lambda \mathbf{x}).$$
5. *If $f(\mathbf{x}) = \max\{\phi_1(\mathbf{x}), ..., \phi_m(\mathbf{x})\}$, then for $I(\mathbf{x}) = \{i : \phi_i(\mathbf{x}) = f(\mathbf{x})\}$,*
$$\partial f(\mathbf{x}) = \text{conv}\{\nabla \phi_i(\mathbf{x}) : i \in I(\mathbf{x})\}$$
where $\text{conv}\{\cdot\}$ denotes the convex hull.

Under more technical assumptions, such as constraint qualification, Properties 2 and 3 hold with equality. For most cases in this monograph, they indeed do. We refer the reader to [63] for more detail since they lie outside of the scope of this monograph.

Using subdifferentials, we can immediately characterize optimal solutions.

Theorem 1. *If $f : \mathbb{R}^n \to (-\infty, \infty]$ is proper convex, then $\mathbf{x}^*$ is a global minimizer if and only if $0 \in \partial f(\mathbf{x}^*)$.*

Proof. By definition, $\mathbf{x}^*$ is a global minimizer of f if and only if for all $\mathbf{x} \in \mathbb{R}^n$,

$$\begin{aligned} & f(\mathbf{x}) \geq f(\mathbf{x}^*) \\ \iff & f(\mathbf{x}) \geq f(\mathbf{x}^*) + 0^T(\mathbf{x} - \mathbf{x}^*) \\ \iff & 0 \in \partial f(\mathbf{x}^*). \end{aligned}$$

□

Theorem 2. *Let $f : \mathbb{R}^n \to (-\infty, \infty]$ be a proper closed convex function and C be a nonempty closed convex set. Then $x^* = \arg\min\{f(\mathbf{x}) : \mathbf{x} \in C\}$ if there exists $\mathbf{p} \in \partial f(\mathbf{x}^*) \cap (-N_C(\mathbf{x}^*))$.*

Proof. By the first order optimality condition, $\mathbf{x}^*$ is a global minimizer of $f(\mathbf{x}) + \iota_C(\mathbf{x})$ if and only if

$$0 \in \partial f(\mathbf{x}^*) + \partial \iota_C(\mathbf{x}^*) = \partial f(\mathbf{x}^*) + N_C(\mathbf{x}^*)$$

which holds if there exists a $\mathbf{p}$ such that $\mathbf{p} \in -N_C(\mathbf{x}^*)$ and $\mathbf{p} \in \partial f(\mathbf{x}^*)$, as desired.

□

Note that unlike gradients, which can be formed by partial derivatives, partial subgradients do not necessarily form a gradient. In particular, if

$$\mathbf{p}_1 \in \partial_1 f(\mathbf{x}_1, ..., \mathbf{x}_n), ..., \mathbf{p}_n \in \partial_n f(\mathbf{x}_1, ..., \mathbf{x}_n),$$

then $(\mathbf{p}_1, ..., \mathbf{p}_n)$ may *not* be in $\partial f(\mathbf{x}_1, ..., \mathbf{x}_n)$.

C Proximal Operators

The *proximal mapping* or *proximal operator* appears in many algorithms for minimizing convex, non-smooth functions. It involves a smaller minimization problem that may be solved cheaply in certain cases.

Definition 3. *Given a closed, proper, and convex function f, the proximal operator for αf is defined as*

$$\text{prox}_{\alpha f}(\mathbf{y}) = \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{2\alpha}\|\mathbf{x} - \mathbf{y}\|_2^2.$$

We first show that the proximal operator is a generalization of projections.

Example 3. *Given a convex set $\mathcal{X}$, we will show that the proximal operator for the indicator variable $\iota_{\mathcal{X}}(\mathbf{x})$ is the projection onto $\mathcal{X}$.*

Note that by definition, $\text{prox}_{\iota_{\mathcal{X}}}(\mathbf{x}) = \arg\min_{\mathbf{u}} \iota_{\mathcal{X}}(\mathbf{u}) + \frac{1}{2}\|\mathbf{u} - \mathbf{x}\|_2^2$, which is equivalent to the problem

$$\begin{aligned} \underset{\mathbf{u}}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{u} - \mathbf{x}\|_2^2 \\ \text{subject to} \quad & \mathbf{u} \in \mathcal{X} \end{aligned}$$

This problem, by definition, is the projection operator, $\text{proj}_{\mathcal{X}}(\mathbf{x})$.

The second prime example of proximal operators is for minimizing the ℓ_1-norm. In particular, we can recover the shrinkage operator, as in [3].

Example 4. *We will show that the proximal operator for the ℓ_1 norm is the shrinkage operator, i.e.*

$$\text{prox}_{\mu\|\cdot\|_1}(\mathbf{x})_i = \begin{cases} x_i - \mu & \text{if } x_i > \mu \\ 0 & \text{if } x_i \in [-\mu, \mu] \\ x_i + \mu & \text{if } x_i < -\mu. \end{cases}$$

First note that since the minimization is separable:

$$\|\mathbf{u}\|_1 + \frac{1}{2\mu}\|\mathbf{x} - \mathbf{u}\|_2^2 = \sum_{i=1}^{n} |u_i| + \frac{1}{2\mu}(x_i - u_i)^2,$$

it is sufficient to consider

$$\text{prox}_{\mu|\cdot|}(x) = \arg\min_u |u| + \frac{1}{2\mu}(u - x)^2.$$

By the first order optimality condition of the minimization problem,

$$0 \in \partial|u| + \frac{1}{\mu}(u - x).$$

This gives three cases:

1. $u > 0 \iff 0 = \mu + (u - x) \iff u = x - \mu \iff x > \mu$
2. $u = 0 \iff 0 \in \mu[-1, 1] + (0 - x) \iff x \in [-\mu, \mu]$
3. $u < 0 \iff 0 = -\mu + (u - x) \iff u = x + \mu \iff x < -\mu.$

Combining these three cases gives the result.

For the sake of completeness, we list some common properties and interpretations of proximal operators that appear in the monograph. Please refer to [63] for more details on proximal operators.

Theorem 3. *Let $(I + \alpha\partial f)$ denote the operator that takes $(I + \alpha\partial f)(\mathbf{x}) = \mathbf{x} + \alpha\partial f(\mathbf{x})$. Then $(I + \alpha\partial f(\mathbf{x}))^{-1}(\mathbf{x}) = \text{prox}_{\alpha f}(\mathbf{x})$.*

Proof. Suppose $\mathbf{u} = \text{prox}_{\alpha f}(\mathbf{x})$. Then

$$\begin{aligned} \mathbf{x} = \text{prox}_{\alpha f}(\mathbf{y}) &\iff 0 \in \partial f(\mathbf{x}) + \frac{1}{\alpha}(\mathbf{x} - \mathbf{y}) \\ &\iff 0 \in \alpha\partial f(\mathbf{x}) + (\mathbf{x} - \mathbf{y}) \\ &\iff \mathbf{y} \in x + \alpha\partial f(\mathbf{x}) \\ &\iff \mathbf{y} \in (I + \alpha\partial f)(\mathbf{x}) \\ &\iff \mathbf{x} \in (I + \alpha\partial f)^{-1}(\mathbf{y}). \end{aligned}$$

□

Theorem 4. *For a separable function $f(\mathbf{x}, \mathbf{y}) = g(\mathbf{x}) + h(\mathbf{y})$, then*

$$\text{prox}_f(\mathbf{x}, \mathbf{y}) = (\text{prox}_f(\mathbf{x}), \text{prox}_g(\mathbf{y})).$$

Proof. This follows from the definition of the proximal operator and that the minimization of f is equivalent to minimizing g and h independently.

□

D Proofs for Summative Proximable Functions

We give relevant propositions and proofs for our results for summative proximable functions. Recall that we are interested in evaluating the proximal operator of the form

$$\text{prox}_{\alpha(f+g)}(\mathbf{y}) = \arg\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x}) + \frac{1}{2\alpha}\|\mathbf{x}-\mathbf{y}\|_2^2$$

where both f and g have inexpensive proximal operators.

D.1 Proof for ℓ_2-Regularized Proximable Functions

Proposition 1. *Let $\mathbf{x} \in \mathbb{R}^n$. If $f(\mathbf{x})$ is a convex, homogeneous function of order 1 (i.e., $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$ for $\alpha \geq 0$) and $g(\mathbf{x}) := \beta\|\mathbf{x}\|_2$, then $\text{prox}_{f+g} = \text{prox}_g \circ \text{prox}_f$.*

Lemma 1. *If $f(\mathbf{x})$ is a convex, homogeneous function of order 1 (i.e., $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$ for $\alpha \geq 0$), then the following results hold*

1. *$\partial f(\mathbf{x}) = \partial f(\alpha\mathbf{x})$ for any $\alpha > 0$ and $\mathbf{x} \in \mathbb{R}^n$;*
2. *$\partial f(\mathbf{x}) \subset \partial f(0)$ for any $\mathbf{x} \in \mathbb{R}^n$.*

Proof. Part 1. Let $\alpha > 0$. By the chain rule, $\partial_{\mathbf{x}} f(\alpha\mathbf{x}) = \alpha\partial f(\alpha\mathbf{x})$. By $f(\alpha\mathbf{x}) = \alpha f(\mathbf{x})$, we also have $\partial_{\mathbf{x}} f(\alpha\mathbf{x}) = \partial_{\mathbf{x}}(\alpha f(\mathbf{x})) = \alpha\partial_{\mathbf{x}} f(\mathbf{x}) = \alpha\partial f(\mathbf{x})$. Hence, $\partial f(\mathbf{x}) = \partial f(\alpha\mathbf{x})$.

Part 2. For any $\mathbf{x}$ and $\alpha > 0$, let $\mathbf{y} = \alpha\mathbf{x}$. Let $\mathbf{p} \in \partial f(\mathbf{x})$. By part 1, $\mathbf{p} \in \partial f(\mathbf{y})$. Thus, by definition, $\mathbf{p}$ obeys $f(\mathbf{z}) \geq f(\mathbf{y}) + \langle \mathbf{p}, \mathbf{z}-\mathbf{y}\rangle$ for any $\mathbf{z} \in \mathbb{R}^n$, and this inequality holds for any $\alpha > 0$. Now let $\alpha \to 0$ and, by continuity, we have $f(\mathbf{z}) \geq f(0) + \langle \mathbf{p}, \mathbf{z}-0\rangle$. Hence, $\mathbf{p} \in \partial f(0)$, and $\partial f(\mathbf{x}) \subset \partial f(0)$. □

Proof of Prop. 1. Let $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} := \text{prox}_f(\mathbf{x})$, and $\mathbf{z} := \text{prox}_g(\mathbf{y})$. We shall show that $\mathbf{z} = \text{prox}_{f+g}(\mathbf{x})$.

From $\mathbf{y} := \text{prox}_f(\mathbf{x})$ and $\mathbf{z} := \text{prox}_g(\mathbf{y})$, we obtain the optimality conditions of their minimization problems, respectively,

$$0 \in \partial f(\mathbf{y}) + (\mathbf{y}-\mathbf{x}),$$
$$0 \in \partial g(\mathbf{z}) + (\mathbf{z}-\mathbf{y}),$$

and adding them gives us

$$0 \in \partial f(\mathbf{y}) + \partial g(\mathbf{z}) + (\mathbf{z}-\mathbf{x}).$$

Now using $g(\cdot) := \beta\|\cdot\|_2$, we have $\mathbf{z} := \lambda_{\mathbf{y}}\mathbf{y}$, where $\lambda_{\mathbf{y}} = \frac{\mathbf{y}}{\|\mathbf{y}\|_2}\max\{0, \|\mathbf{y}\|_2 - \beta\} \geq 0$. By Lemma 1, Part 1 for the case $\lambda > 0$ and Part 2 for the case $\lambda = 0$, we arrive at $\partial f(\mathbf{y}) \subseteq \partial f(\mathbf{z})$ and thus

$$0 \in \partial f(\mathbf{z}) + \partial g(\mathbf{z}) + (\mathbf{z}-\mathbf{x}),$$

which is the optimality condition for $\mathbf{z} = \text{prox}_{f+g}(\mathbf{x})$. □

In the proof, the formula of $\lambda_{\mathbf{y}}$ is not important; only $\lambda_{\mathbf{y}} \geq 0$ is. Therefore, Prop. 1 remains valid for $g(\mathbf{x}) := \|\mathbf{x}\|_2^p$ for any $p \geq 1$.

D.2 Proof for TV-Regularized Proximable Functions

Proposition 2. *Let* $\mathbf{x} \in \mathbb{R}^n$. *Define the total variation semi-norm:* $\mathrm{TV}(\mathbf{x}) := \sum_{i=1}^{n-1} |x_{i+1} - x_i|$. *If* $f(\mathbf{x}) = \beta \mathrm{TV}(\mathbf{x})$ *and* $g(\mathbf{x})$ *is a closed, proper, convex function that satisfies*

$$x_i > x_{i+1} \implies \mathrm{prox}_g(\mathbf{x})_i \geq \mathrm{prox}_g(\mathbf{x})_{i+1} \tag{42a}$$
$$x_i < x_{i+1} \implies \mathrm{prox}_g(\mathbf{x})_i \leq \mathrm{prox}_g(\mathbf{x})_{i+1} \tag{42b}$$
$$x_i = x_{i+1} \implies \mathrm{prox}_g(\mathbf{x})_i = \mathrm{prox}_g(\mathbf{x})_{i+1}, \tag{42c}$$

then $\mathrm{prox}_{f+g} = \mathrm{prox}_f \circ \mathrm{prox}_g$.

Proof. Let $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{y} := \mathrm{prox}_f(\mathbf{x})$, and $\mathbf{z} := \mathrm{prox}_g(\mathbf{y})$. We shall show that $\mathbf{z} = \mathrm{prox}_{f+g}(\mathbf{x})$.

From $\mathbf{y} := \mathrm{prox}_f(\mathbf{x})$ and $\mathbf{z} := \mathrm{prox}_g(\mathbf{y})$, we obtain the optimality conditions of their minimization problems, respectively,

$$0 \in \partial f(\mathbf{y}) + (\mathbf{y} - \mathbf{x}),$$
$$0 \in \partial g(\mathbf{z}) + (\mathbf{z} - \mathbf{y}),$$

and adding them gives us

$$0 \in \partial f(\mathbf{y}) + \partial g(\mathbf{z}) + (\mathbf{z} - \mathbf{x}). \tag{43}$$

By definition, $f(\mathbf{y}) = \sum_{i=1}^{n-1} |y_{i+1} - y_i|$, which satisfies $\partial f(\mathbf{y}) = \sum_{i=1}^{n-1} \partial_{\mathbf{y}} |y_{i+1} - y_i|$. In this Minkovski sum, each term satisfies

$$\partial_{(y_i, y_{i+1})} |y_{i+1} - y_i| = \begin{cases} (1, -1), & \text{if } y_i > y_{i+1} \\ (-1, 1), & \text{if } y_i < y_{i+1} \\ \{\alpha(1,-1) + (1-\alpha)(-1,1) : \alpha \in [0,1]\} & \text{otherwise,} \end{cases} \tag{44}$$

where the "otherwise" case is the convex hull of the first two cases. Since g satisfies (42) and $\mathbf{z} = \mathrm{prox}_g(\mathbf{y})$, from (y_i, y_{i+1}) to (z_i, z_{i+1}) it holds that

$$y_i > y_{i+1} \implies z_i \geq z_{i+1}$$
$$y_i < y_{i+1} \implies z_i \leq z_{i+1}$$
$$y_i = y_{i+1} \implies z_i = z_{i+1}.$$

Hence, either (z_i, z_{i+1}) holds for the same case of (y_i, y_{i+1}) in (44), or it belongs to the "otherwise" case, which is a superset of the first two cases. Therefore, $\partial_{(y_i, y_{i+1})} |y_{i+1} - y_i| \subseteq \partial_{(z_i, z_{i+1})} |z_{i+1} - z_i|$ and thus $\partial f(\mathbf{y}) \subseteq \partial f(\mathbf{z})$. This together with (43) yields

$$0 \in \partial f(\mathbf{z}) + \partial g(\mathbf{z}) + (\mathbf{z} - \mathbf{x}),$$

which is the optimality condition for $\mathbf{z} = \mathrm{prox}_{f+g}(\mathbf{x})$. □

References

[1] Z. Allen-Zhu, P. Richtárik, Z. Qu, and Y. Yuan. Even Faster Accelerated Coordinate Descent Using Non-Uniform Sampling. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[2] A. Auslender. Asymptotic properties of the fenchel dual functional and applications to decomposition problems. *Journal of Optimization Theory and Applications*, 73(3):427–449, 1992.

[3] A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, 2009.

[4] A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM Journal on Optimization*, 23(4):2037–2060, 2013.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and distributed computation: numerical methods*, volume 23. Prentice hall Englewood Cliffs, NJ, 1989.

[6] S. Bonettini. Inexact block coordinate descent methods with application to non-negative matrix factorization. *IMA Journal of Numerical Analysis*, 31(4):1431–1452, 2011.

[7] J. K. Bradley, A. Kyrola, D. Bickson, and C. Guestrin. Parallel coordinate descent for l1-regularized loss minimization. *ICML*, pages 321–328, 2011.

[8] C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[9] B. Chen, S. He, Z. Li, and S. Zhang. Maximum block improvement and polynomial optimization. *SIAM Journal on Optimization*, 22(1):87–107, 2012.

[10] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari. *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation*. John Wiley & Sons, 2009.

[11] P. Combettes and J. Pesquet. Proximal thresholding algorithm for minimization over orthonormal bases. *SIAM Journal on Optimization*, 18(4):1351–1376, Nov. 2007.

[12] D. Csiba, Z. Qu, and P. Richtarik. Stochastic dual coordinate ascent with adaptive probabilities. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 674–683, 2015.

[13] C. D. Dang and G. Lan. Stochastic block mirror descent methods for nonsmooth and stochastic optimization. *SIAM Journal on Optimization*, 25(2), 2015.

[14] A. Defazio, F. Bach, and S. Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2014.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[16] D. d'Esopo. A convex programming procedure. *Naval Research Logistics Quarterly*, 6(1):33–42, 1959.

[17] I. S. Dhillon, P. K. Ravikumar, and A. Tewari. Nearest neighbor based greedy coordinate descent. In *Advances in Neural Information Processing Systems*, pages 2160–2168, 2011.

[18] B. Edmunds, Z. Peng, and W. Yin. TMAC: A Toolbox of Modern Async-Parallel, Coordinate, Splitting, and Stochastic Methods. *arXiv:1606.04551 [math]*, June 2016.

[19] O. Fercoq and P. Richtárik. Smooth minimization of nonsmooth functions with parallel coordinate descent methods. *arXiv preprint arXiv:1309.5885*, 2013.

[20] O. Fercoq and P. Richtárik. Accelerated, parallel, and proximal coordinate descent. *SIAM Journal on Optimization*, 25(4):1997–2023, 2015.

[21] X. Gao, Y. Xu, and S. Zhang. Randomized primal-dual proximal block coordinate updates. *arXiv preprint arXiv:1605.05969*, 2016.

[22] M. Grant, S. Boyd, and Y. Ye. CVX: Matlab software for disciplined convex programming, 2008.

[23] L. Grippo and M. Sciandrone. On the convergence of the block nonlinear gauss–seidel method under convex constraints. *Operations Research Letters*, 26(3):127–136, 2000.

[24] C.-J. H. Guo-Xun Yuan, Kai-Wei Chang and C.-J. Lin. A comparison of optimization methods and software for large-scale l1-regularized linear classification. *Journal of Machine Learning Research*, 11:3183–3234, 2010.

[25] E. T. Hale, W. Yin, and Y. Zhang. Fixed-point continuation for ℓ_1-minimization: Methodology and convergence. *SIAM Journal on Optimization*, 19(3):1107–1130, 2008.

[26] R. Hannah and W. Yin. On unbounded delay in asynchronous parallel fixed-point algorithms. *UCLA CAM Report 16-64*, 2016.

[27] C. Hildreth. A quadratic programming procedure. *Naval research logistics quarterly*, 4(1): 79–85, 1957.

[28] M. Hong, X. Wang, M. Razaviyayn, and Z.-Q. Luo. Iteration complexity analysis of block coordinate descent methods. *arXiv preprint arXiv:1310.6957*, 2013.

[29] M. Hong, M. Razaviyayn, Z.-Q. Luo, and J.-S. Pang. A unified algorithmic framework for block-structured optimization involving big data: With applications in machine learning and signal processing. *Signal Processing Magazine, IEEE*, 33(1):57–77, 2016.

[30] D. W. Hosmer Jr., S. Lemeshow, and R. X. Sturdivant. Introduction to the Logistic Regression Model. In *Applied Logistic Regression*, pages 1–33. John Wiley & Sons, Inc., Hoboken, NJ, USA, Aug. 2013. ISBN 978-1-118-54838-7 978-0-470-58247-3.

[31] C.-J. Hsieh, H.-F. Yu, and I. S. Dhillon. PASSCoDe: Parallel asynchronous stochastic dual coordinate descent. *Proceedings of The 32nd International Conference on Machine Learning*, pages 2370–2379, 2015.

[32] M. Jaggi, V. Smith, M. Takác, J. Terhorst, S. Krishnan, T. Hofmann, and M. I. Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014.

[33] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[34] K. Lange, E. C. Chi, and H. Zhou. A brief survey of modern optimization for statisticians. *International Statistical Review*, 82(1):46–70, 2014.

[35] Y. T. Lee and A. Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 147–156. IEEE, 2013.

[36] D. Leventhal and A. S. Lewis. Randomized methods for linear constraints: convergence rates and conditioning. *Mathematics of Operations Research*, 35(3):641–654, 2010.

[37] Y. Li and S. Osher. Coordinate descent optimization for ℓ_1 minimization with application to compressed sensing; a greedy algorithm. *Inverse Problems and Imaging*, 3(3):487–503, 2009.

[38] Z. Li, A. Uschmajew, and S. Zhang. On convergence of the maximum block improvement method. *SIAM Journal on Optimization*, 25(1):210–233, 2015.

[39] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[40] J. Liu and S. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM Journal on Optimization*, 25(1):351–376, 2015.

[41] J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *The Journal of Machine Learning Research*, 16(1):285–322, 2015.

[42] Z.-Q. Luo and P. Tseng. On the convergence of the coordinate descent method for convex differentiable minimization. *Journal of Optimization Theory and Applications*, 72(1):7–35, 1992.

[43] J. Mareček, P. Richtárik, and M. Takáč. Distributed block coordinate descent for minimizing partially separable functions. In *Numerical Analysis and Optimization*, pages 261–288. Springer, 2015.

[44] O. Meshi, A. Globerson, and T. S. Jaakkola. Convergence rate analysis of map coordinate minimization algorithms. In *Advances in Neural Information Processing Systems*, pages 3014–3022, 2012.

[45] I. Necoara and D. Clipici. Efficient parallel coordinate descent algorithm for convex optimization problems with separable constraints: application to distributed MPC. *Journal of Process Control*, 23(3):243–253, 2013.

[46] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[47] Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM Journal on Optimization*, 22(2):341–362, 2012.

[48] Y. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

[49] J. Nutini, M. Schmidt, I. Laradji, M. Friedlander, and H. Koepke. Coordinate descent converges faster with the gauss-southwell rule than random selection. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1632–1641, 2015.

[50] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994.

[51] A. Patrascu and I. Necoara. Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *Journal of Global Optimization*, 61(1):19–46, 2015.

[52] Z. Peng, M. Yan, and W. Yin. Parallel and distributed sparse optimization. In *Signals, Systems and Computers, 2013 Asilomar Conference on*, pages 659–646. IEEE, 2013.

[53] Z. Peng, T. Wu, Y. Xu, M. Yan, and W. Yin. Coordinate friendly structures, algorithms and applications. *Annals of Mathematical Sciences and Applications*, 1(1):57–119, 2016.

[54] Z. Peng, Y. Xu, M. Yan, and W. Yin. ARock: an algorithmic framework for asynchronous parallel coordinate updates. *SIAM Journal on Scientific Computing*, 38(5):A2851–A2879, 2016.

[55] J.-C. Pesquet and A. Repetti. A class of randomized primal-dual algorithms for distributed optimization. *arXiv:1406.6404 [math]*, June 2014.

[56] M. J. Powell. On search directions for minimization algorithms. *Mathematical Programming*, 4(1):193–201, 1973.

[57] Z. Qu and P. Richtárik. Coordinate descent with arbitrary sampling i: algorithms and complexity. *Optimization Methods and Software*, 31(5):829–857, 2016.

[58] Z. Qu and P. Richtárik. Coordinate descent with arbitrary sampling ii: expected separable overapproximation. *Optimization Methods and Software*, 31(5):858–884, 2016.

[59] M. Razaviyayn, M. Hong, and Z.-Q. Luo. A unified convergence analysis of block successive minimization methods for nonsmooth optimization. *SIAM Journal on Optimization*, 23(2): 1126–1153, 2013.

[60] P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Mathematical Programming*, 144(1-2):1–38, 2014.

[61] P. Richtárik and M. Takáč. On optimal probabilities in stochastic coordinate descent methods. *Optimization Letters*, pages 1–11, 2015.

[62] P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Mathematical Programming*, 156(1-2):433–484, 2016.

[63] R. T. Rockafellar. *Convex analysis*. Princeton university press, 2015.

[64] A. Saha and A. Tewari. On the nonasymptotic convergence of cyclic coordinate descent methods. *SIAM Journal on Optimization*, 23(1):576–601, 2013.

[65] M. Schmidt, N. L. Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, pages 1–30, 2016. ISSN 1436-4646. doi: 10.1007/s10107-016-1030-6. URL http://dx.doi.org/10.1007/s10107-016-1030-6.

[66] S. Shalev-Shwartz and A. Tewari. Stochastic methods for ℓ_1-regularized loss minimization. *The Journal of Machine Learning Research*, 12:1865–1892, 2011.

[67] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss. *The Journal of Machine Learning Research*, 14(1):567–599, 2013.

[68] S. Shalev-Shwartz and T. Zhang. Accelerated proximal stochastic dual coordinate ascent for regularized loss minimization. *Mathematical Programming*, 155(1-2):105–145, 2016.

[69] Shaobing Chen and D. Donoho. Basis pursuit. In *The 28th Asilomar Conference on Signals, Systems and Computers*, volume 1, pages 41–44. IEEE Comput. Soc. Press, 1994. ISBN 978-0-8186-6405-2.

[70] S. K. Shevade and S. S. Keerthi. A simple and efficient algorithm for gene selection using sparse logistic regression. *Bioinformatics*, 19(17):2246–2253, 2003.

[71] J. V. Shi, Y. Xu, and R. G. Baraniuk. Sparse bilinear logistic regression. *arXiv preprint arXiv:1404.4104*, 2014.

[72] R. V. Southwell. *Relaxation Methods In Engineering Science - A Treatise On Approximate Computation*. Oxford University Press, London, New York, 1940.

[73] J. a. K. Suykens and J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*, 9(3):293–300, 1999.

[74] Q. Tao, K. Kong, D. Chu, and G. Wu. Stochastic coordinate descent methods for regularized smooth and nonsmooth losses. In *Machine Learning and Knowledge Discovery in Databases*, pages 537–552. Springer, 2012.

[75] R. Tappenden, M. Takáč, and P. Richtárik. On the complexity of parallel coordinate descent. *arXiv preprint arXiv:1503.03033*, 2015.

[76] R. Tibshirani. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[77] P. Tseng. Dual ascent methods for problems with strictly convex costs and linear constraints: A unified approach. *SIAM Journal on Control and Optimization*, 28(1):214–242, 1990.

[78] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *Journal of optimization theory and applications*, 109(3):475–494, 2001.

[79] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming. A Publication of the Mathematical Programming Society*, 117 (1-2):387–423, 2009.

[80] P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Mathematical Programming*, 117(1-2):387–423, 2009.

[81] J. Warga. Minimizing certain convex functions. *Journal of the Society for Industrial & Applied Mathematics*, 11(3):588–593, 1963.

[82] Z. Wen, D. Goldfarb, and K. Scheinberg. Block coordinate descent methods for semidefinite programming. In *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 533–564. Springer, 2012.

[83] S. J. Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

[84] T. T. Wu and K. Lange. Coordinate descent algorithms for lasso penalized regression. *The Annals of Applied Statistics*, 2(1):224–244, 2008.

[85] Y. Xu. Alternating proximal gradient method for sparse nonnegative tucker decomposition. *Mathematical Programming Computation*, 7(1):39–70, 2015.

[86] Y. Xu and W. Yin. A block coordinate descent method for regularized multiconvex optimization with applications to nonnegative tensor factorization and completion. *SIAM Journal on Imaging Sciences*, 6(3):1758–1789, 2013.

[87] Y. Xu and W. Yin. A globally convergent algorithm for nonconvex optimization based on block coordinate update. *arXiv preprint arXiv:1410.1386*, 2014.

[88] Y. Xu and W. Yin. Block stochastic gradient iteration for convex and nonconvex optimization. *SIAM Journal on Optimization*, 25(3):1686–1716, 2015.

[89] M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[90] S. Yun and K.-C. Toh. A coordinate gradient descent method for 1-regularized convex minimization. *Computational Optimization and Applications*, 48(2):273–307, 2011.

[91] S. Yun, P. Tseng, and K.-C. Toh. A block coordinate gradient descent method for regularized convex separable optimization and covariance selection. *Mathematical programming*, 129(2): 331–355, 2011.

[92] N. Zadeh. Note – A note on the cyclic coordinate ascent method. *Management Science*, 16(9): 642–644, 1970.

[93] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21st international conference on Machine learning (ICML-04)*, page 116. ACM, 2004.

[94] Y. Zhang and X. Lin. Stochastic primal-dual coordinate method for regularized empirical risk minimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 353–361, 2015.

[95] N. Zhou, Y. Xu, H. Cheng, J. Fang, and W. Pedrycz. Global and local structure preserving sparse subspace learning: an iterative approach to unsupervised feature selection. *Pattern Recognition*, 2015.

[96] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, Apr. 2005.