Greed is Good

Greedy Optimization Methods for Large-Scale Structured Problems

by

Julie Nutini

B.Sc., The University of British Columbia (Okanagan), 2010
M.Sc., The University of British Columbia (Okanagan), 2012

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies
(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA
(Vancouver)
May 2018

The following individuals certify that they have read, and recommend to the
Faculty of Graduate and Postdoctoral Studies for acceptance, the dissertation entitled:

Greed is Good: Greedy Optimization Methods for Large-Scale Structured Problems

submitted by Julie Nutini in partial fulfillment of the requirements for

the degree of Doctor of Philosophy

in Computer Science

Examining Committee:

Mark Schmidt, Computer Science
Supervisor

Chen Greif, Computer Science
Supervisory Committee Member

Will Evans, Computer Science
Supervisory Committee Member

Bruce Shepherd, Computer Science
University Examiner

Ozgur Yilmaz, Mathematics
University Examiner

Abstract

This work looks at large-scale machine learning, with a particular focus on greedy methods. A recent trend caused by big datasets is to use optimization methods that have a cheap iteration cost. In this category are (block) coordinate descent and Kaczmarz methods, as the updates of these methods only rely on a reduced subspace of the problem at each iteration. Prior to our work, the literature cast greedy variations of these methods as computationally expensive with comparable convergence rates to randomized versions. In this dissertation, we show that *greed is good*. Specifically, we show that greedy coordinate descent and Kaczmarz methods have efficient implementations and can be faster than their randomized counterparts for certain common problem structures in machine learning. We show linear convergence for greedy (block) coordinate descent methods under a revived relaxation of strong convexity from 1963, which we call the Polyak-Łojasiewicz (PL) inequality. Of the proposed relaxations of strong convexity in the recent literature, we show that the PL inequality is the weakest condition that still ensures a global minimum. Further, we highlight the exploitable flexibility in block coordinate descent methods, not only in the different types of selection rules possible, but also in the types of updates we can use. We show that using second-order or exact updates with greedy block coordinate descent methods can lead to superlinear or finite convergence (respectively) for popular machine learning problems. Finally, we introduce the notion of "active-set complexity", which we define as the number of iterations required before an algorithm is guaranteed to reach the optimal active manifold, and show explicit bounds for two common problem instances when using the proximal gradient or the proximal coordinate descent method.

Lay Summary

A recent trend caused by big datasets is to use methods that are computationally inexpensive to solve large-scale problems in machine learning. This work looks at several of these methods, with a particular focus on greedy variations, that is, methods that try to make the most possible progress at each step. Prior to our work, these greedy methods were regarded as computationally expensive with similar performance to cheaper versions that make a random amount of progress at each step. In this dissertation, we show that *greed is good*. Specifically, we show that these greedy methods can be very efficient and can be faster relative to their randomized counterparts for solving machine learning problems with certain structure. We exploit the flexibility of these methods and show various ways (both theoretically and empirically) to speed them up.

Preface

The body of research in this dissertation (Chapters 2 - 6) is based off of several collaborative papers that have either been previously published or are currently under review.

- The work in Chapter 2 was published in the Proceedings of the 32nd International Conference on Machine Learning (ICML) [Nutini et al., 2015]:

  > J. Nutini, Mark Schmidt, Issam H. Laradji, Michael Friedlander and Hoyt Koepke. Coordinate Descent Converges Faster with the Gauss-Southwell Rule Than Random Selection, *ICML 2015* [arXiv].

  The majority of this chapter's manuscript was written by J. Nutini and Mark Schmidt. The new convergence rate analysis for the greedy coordinate descent method (Section 2.2.3) was done by Michael Friedlander and Mark Schmidt. The analysis of strong-convexity constants (Section 2.3) was contributed to by Michael Friedlander. The Gauss-Southwell-Lipschitz rule and nearest neighbour analysis in Sections 2.5.2 and 2.5.3 were a joint effort by J. Nutini, Issam Laradji and Mark Schmidt. The majority of the work in Section 2.8 (numerical experiments) was done by Issam Laradji, with help from J. Nutini. Appendix A.1 showing how to calculate the greedy Gauss-Southwell rule efficiently for sparse problems was primary researched by Issam Laradji and Mark Schmidt. All other sections were primarily researched by J. Nutini and Mark Schmidt. The final co-author on this paper, Hoyt Koepke, was the primary researcher on extending the greedy coordinate descent analysis to the case of using *exact* coordinate optimization (excluded from this dissertation).

- A version of Chapter 3 was published in the Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence (UAI) [Nutini et al., 2016]:

  > J. Nutini, Behrooz Sepehry, Issam H. Laradji, Mark Schmidt, Hoyt Koepke and Alim Virani. Convergence Rates for Greedy Kaczmarz Algorithms, and Faster Randomized Kaczmarz Rules Using the Orthogonality Graph, *UAI 2016* [arXiv].

  The majority of this chapter's manuscript was researched and written by J. Nutini and Mark Schmidt. The majority of the work in Section 3.9 and Appendix B.12 (numerical experiments) was done by Issam Laradji, with help from J. Nutini and Alim Virani. A section of the corresponding paper on extending the convergence rate analysis of Kaczmarz

methods to consider more than one step (i.e., a sequence of steps) was written by my co-authors Behrooz Sepehry, Hoyt Koepke and Mark Schmidt, and therefore excluded from this dissertation.

- The material in Chapter 4 was published in the Proceedings of the 27th European Conference on Machine Learning (ECML) [Karimi et al., 2016]:

  > Hamed Karimi, J. Nutini and Mark Schmidt. Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak-Łojasiewicz Condition, *ECML 2016* [arXiv].

  Hamed Karimi, Mark Schmidt and I all made contributions to Theorem 2 and Section C.5, while the work in Section C.4 was done by Hamed Karimi. Several results presented in the corresponding paper are excluded from this dissertation as they were written by my co-authors. These include using the Polyak-Łojasiewicz (PL) inequality to give new convergence rate analyses for stochastic gradient descent methods and stochastic variance reduced gradient methods, as well as a result proving the equivalence of our proposed proximal-PL condition to two other previously proposed conditions.

- A version of Chapter 5 has been submitted for publication [Nutini et al., 2017a]:

  > J. Nutini, Issam H. Laradji and Mark Schmidt. Let's Make Block Coordinate Descent Go Fast: Faster Greedy Rules, Message-Passing, Active-Set Complexity, and Superlinear Convergence (2017) [arXiv].

  This chapter's manuscript was written by J. Nutini and Mark Schmidt. The majority of the research was joint work between J. Nutini, Issam Laradji and Mark Schmidt. Section 5.5 and Appendix D.5 (numerical experiments) were primarily done by Issam Laradji, with help from J. Nutini and Mark Schmidt.

- The material in Chapter 6 is a compilation of material from the reference for Chapter 5 [Nutini et al., 2017a] and a manuscript that has been submitted for publication [Nutini et al., 2017b]:

  > J. Nutini, Mark Schmidt and Warren Hare. "Active-set complexity" of proximal gradient: How long does it take to find the sparsity pattern? (2017) [arXiv].

  The majority of this chapter's manuscript was written by J. Nutini and Mark Schmidt. The proof of Lemmas 2 and 4 was joint work by Warren Hare, Mark Schmidt and J. Nutini. The majority of the work in Section 6.5 (numerical experiments) was done by Issam Laradji, with help from J. Nutini and Mark Schmidt.

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would first and foremost like to thank my supervisor, Mark Schmidt – thank you for helping me battle the imposter within. Your support, encouragement and mentorship are at the root of this work, and it has been an absolute pleasure learning from you. To my supervisory committee, Chen Greif and Will Evans, my external examiner, Stephen Wright, my university examiners, Bruce Shepherd and Ozgur Yilmaz, and my defence chair, Maurice Queyranne – thank you for your time, support and helpful feedback. Thank you to Michael Friedlander for supervising me in the early years of this degree. To my Masters supervisor, Warren Hare – thank you for encouraging me to pursue my studies further and for sticking by my side long enough to get another publication (no fairy dust required). Thank you to all of my co-authors and lab mates, especially Issam Laradji – I have thoroughly enjoyed learning alongside you. To all of my colleagues here at UBC and at UrtheCast – thank you for crossing your fingers with me every time my answer was, "I should be done by the end of [*insert month here*]". To all of the support staff at UBC, especially Joyce Poon and Kath Imhiran – thank you for always making the paper work an easy process. To my family and friends – thank you for your unwavering support and for providing me with a distraction when needed. And finally, to my parents, for whom there are no words – this accomplishment is as much yours as it is mine.

This work was partially funded by the Natural Sciences and Engineering Research Council of Canada, the UBC Faculty of Science, and a UBC Four Year Fellowship.

To my grandpa – an educator, a family man and a lover of life...

Chapter 1

Introduction

Machine learning is remodelling the world we live in. Coined by computer gaming and artificial intelligence pioneer Arthur Samuel in 1959, the term *machine learning* (ML) has been popularly defined as the "field of study that gives computers the ability to learn without being explicitly programmed", (credited to Samuel [1959]). Elaborating on this elegant definition, ML is the study of using computers to automatically detect patterns in data and make predictions or decisions, [Murphy, 2012]. This automation process is useful in the absence of a human expert skilled at analyzing the data, when a human cannot detect or explain patterns in the data, or when the complexity of a classification or prediction problem is beyond the capability of a human. By designing computer systems and optimization algorithms capable of parsing this data, ML has been integral in the success of several big impact applications recently, such as epileptic seizure prediction [Mirowski et al., 2008], speech recognition [Mohamed et al., 2009], music generation [Boulanger-Lewandowski et al., 2012], large-scale video classification [Karpathy et al., 2014] and autonomous vehicles [Teichmann et al., 2016]. It is evident that ML is at the forefront of technological change in our world.

1.1 Big-Data: A Barrier to Learning?

There is no shortage of data in today's data driven world. In nearly every daily task we generate and record data on the order of terabytes to exabytes. Online news articles, blog posts, Facebook likes, credit card transactions, online purchases, gene expression data, maps, satellite imagery and user interactions are a fractional sample of the day to day activities for which we record data. Fitting most ML models involves solving an optimization problem, and without methods capable of parsing and analyzing these huge datasets, the learning process becomes wildly intractable.

This presents us with the crucial task of developing ways to efficiently deal with these massive data sets. One approach is to create hardware that can handle the computational requirements of existing algorithms. For example, parallel computing using multi-core machines, outsourcing computational work to cloud computing platforms and faster GPUs have had a huge impact on the ability of the ML field to keep up with the ever-growing amount of data. However, the development of hardware is restricted by what is known as *Moore's Law*, the observation that the processing speed of computers only doubles every two years. With the rate at which we are collecting data, we cannot rely on hardware advances to keep up.

A different approach to dealing with these large datasets is to design methods that are *scalable* with problem size. That is, for large-scale data sets we want methods whose cost scales at most linearly (or almost linearly) with the data size. The best method for a given problem is often dictated by the size of the problem; the larger the problem, the greater the restrictions on what mathematical operations are feasible.

In general, optimization algorithms can be classified according to two properties:

1. **Convergence rate**: the number of iterations required to reach a solution of accuracy ϵ.

2. **Iteration cost**: the amount of computation each iteration requires.

Often times these two properties work in opposition to each other. For example, a second order method like Newton's method achieves a superlinear convergence rate but requires a quadratic-or-worse iteration cost. Alternatively, a first order method like gradient descent has a cheaper iteration cost but only achieves a linear convergence rate. If the data size is large enough, it is possible that gradient descent will find an epsilon-optimal solution in a shorter amount of total computation time compared to Newton's method (even though the number of iterations may be much higher).

For some big datasets the cost of first-order gradient methods is still infeasible. As a result, there has been a recent trend of proposing/reviving methods with even cheaper iteration costs. Generally speaking, this indicates a shift in focus from developing a robust algorithm that is scalable for all types of problems to developing algorithms that exploit problem structure to deal with scalability. By exploiting problem structure we are able to develop faster optimization methods that still maintain cheap iteration costs.

Before we discuss the details of these methods, we first need to describe the problem structure that these large-scale methods are designed to exploit. In the next section, we define the supervised learning problem that is solved in many machine learning tasks. Formally this problem is known as expected risk minimization.

1.2 The Learning Problem/Algorithm

Optimization is used to formalize the learning process, where the goal is to determine a mapping such that for any observed input feature vector $a_i \in \mathbb{R}^n$ and corresponding output classification or prediction vector $b_i \in \mathbb{R}$, the mapping yields the true output b_i. We use optimization to learn the parameterization of this mapping such that some difference measure between the output of the learned mapping and the true output is minimized.[1]

The mapping is known formally as a prediction function. There are several families of prediction functions and an optimization process can be done over entire families to select the optimal form the prediction function should take. However for the purpose of this work we assume that we have a fixed prediction function with an unknown parameterization. That is

[1]The description of the ERM problem in this section largely follows the presentation of Bottou et al. [2016].

we are given a prediction function $h(\cdot; x) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ parameterized by an unknown vector $x \in \mathbb{R}^n$. For example, a general linear prediction function is define by

$$h(a_i; x) = x^T a_i, \tag{1.1}$$

where i is the index of a data sample and the a_i could potentially be non-linear transformations of some original measurements. The goal is to learn a parameter vector x such that for any feature vector $a_i \in \mathbb{R}^n$, the output of $h(a_i; x)$, say $\hat{b}_i$, matches the known output b_i.

To carry out this learning process, we require a measure of difference between $\hat{b}_i$ and b_i. We define a loss function as some function $f_i : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ that computes a measure of difference between these two values. This loss function is usually a continuous (and often convex) approximation to the "true" loss function (see Section 1.2.1). Indeed, this loss function is often *strongly-convex*, an important property that will be discussed in this work and that we define in Section 1.3.

Given a prediction function h and a loss function f_i, we define the objective function of the ML optimization known as the *expected risk* function, which is defined by

$$\hat{f}(x) = \int_{\mathbb{R}^n \times \mathbb{R}} f_i(b_i, h(a_i; x))\ dP(a_i, b_i) = \mathbb{E}\left[f_i(b_i, h(a_i; x))\right], \tag{1.2}$$

where $P(a_i, b_i)$ is a joint probability distribution from which the observation pair (a_i, b_i) was sampled. Clearly, the function in (1.2) is impractical to evaluate as it is an expectation over the entire (infinite) distribution of possible data examples. Thus, in practice we approximate it using an *empirical risk* function. Given m independently and identically distributed random observations $(a_i, b_i) \in \mathbb{R}^n \times \mathbb{R}$ for $i = 1, 2, \ldots, m$, we evaluate the *empirical risk* function, which is defined by

$$f(x) = \frac{1}{m} \sum_{i=1}^{m} f_i(b_i, h(a_i; x)),$$

where $f_i(x)$ is commonly used as short form notation for $f_i(b_i, h(a_i; x))$. Thus, the ML optimization problem is to find x such that empirical risk of misclassification is minimized,

$$\min_{x \in \mathbb{R}^n}\ f(x) \equiv \min_{x \in \mathbb{R}^n} \frac{1}{m} \sum_{i=1}^{m} f_i(x).$$

In general, this problem is known as *empirical risk minimization* (ERM). Often we use regularization to decrease the variance in the learned estimator so that it is less sensitive to the data it is trained on. This typically improves test error but assumes some prior belief over the smoothness of the desired model. We define the regularized empirical risk minimization by

$$\min_{x \in \mathbb{R}^n}\ \frac{1}{m} \sum_{i=1}^{m} f_i(x) + \lambda g(x),$$

for some regularization function $g : \mathbb{R}^n \to \mathbb{R}$ and regularization parameter $\lambda > 0$. The most common regularizers used in ML are:

- $\|\cdot\|_2$: The ℓ_2-norm is differentiable and acts as a smoothing regularizer that when added to a convex loss function results in a strongly convex loss function. This type of regularization is a special case of Tikhonov regularization, where the Tikhonov matrix is the identity matrix. The influence of the ℓ_2-regularization encourages the entries of the parameter vector w to be small in magnitude.

- $\|\cdot\|_1$: The ℓ_1-norm promotes sparsity in the parameter vector w. The resulting non-zero entires in w correspond to the most important features needed for classification and it has been shown that this type of regularization can help with the *curse of dimensionality* [Ng, 2004]. Otherwise, when there is enough data the ℓ_1-norm has a denoising effect on the solution (e.g., the basis pursuit denoising problem in signal reconstruction applications [Chen and Donoho, 1994, Chen et al., 2001]).

Using either the regularized or non-regularized ERM objective, the learning process then uses data to train, validate and test the model. Usually the dataset is separated into a training set and a testing set. Although not the topic of this work, the variability of a training dataset has a direct influence on the accuracy of the model learned. Validation includes deciding/choosing between several prediction/loss functions to determine the one that yields the lowest percentage of error in predictions. This generated model is then used to test on a given dataset. In summary, the choice of loss function is made via empirical observation and experimentation, using training data, validation and testing datasets. The selected loss function usually has the best performance on the validation set and we explore several common loss functions used in ML in the next section. (For more details on the use of different regularization/penalty functions used in statistical learning, see [Hastie et al., 2001, Wainwright, 2014].)

1.2.1 Loss Functions

In this work we focus on methods that can be used to solve ERM and regularized ERM with convex loss functions. Convex loss functions are commonly used in ML because they are well-behaved and have a well-defined unique solution. We define the convex loss functions that are most regularly used in ML problems next (see [Rosasco et al., 2004] for a thorough comparison of the most common convex loss functions used in the machine learning).

Least-Squares Loss

The *least-squares loss* is defined as taking the squared error between the linear prediction function defined in (1.1) evaluated at a sample $a_i \in \mathbb{R}^d$ and the corresponding true output $b_i \in \mathbb{R}$,

$$f_i(x) = \frac{1}{2}(b_i - x^T a_i)^2.$$

As an optimization problem this is known as least-squares linear regression and is the minimization of the squared error over a set of samples $a_i \in \mathbb{R}^n,\ i = 1, \ldots, m$,

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \sum_{i=1}^{m} (b_i - x^T a_i)^2.$$

Least-squares loss is twice-differentiable and convex, but not strongly convex. In order to make the objective strongly convex to ensure a unique solution, we can add a strongly convex regularizer. For example, using a set of samples $A = [a_1, a_2, \ldots, a_m]^T \in \mathbb{R}^{m \times n}$ and a set of outputs $b \in \mathbb{R}^m$, the *ridge regression* problem uses ℓ_2-regularization,

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_2^2.$$

Alternatively, the *LASSO* problem uses ℓ_1-regularization,

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} \|Ax - b\|_2^2 + \lambda \|x\|_1.$$

We note, however, that this choice of regularization does not ensure a unique solution.

Logistic Loss

For problems of binary classification with $b_i \in \{-1, 1\}$, we consider the function

$$\max(0, -b_i \operatorname{sign}(x^T a_i)), \tag{1.3}$$

where $\operatorname{sign}(\cdot)$ is equal to -1 if the argument is negative and $+1$ if the argument is positive. Using this as our loss function, if $b_i \operatorname{sign}(x^T a_i) > 1$, then our model predicted the correct classification for example i and (1.3) would be equal to 0. That is, in the standard binary case $\hat{b}_i = \operatorname{sign}(x^T a_i)$ and (1.3) is a "measure of difference" between $\hat{b}_i$ and b_i. However, (1.3) is a nonsmooth problem and suffers from a degenerate solution when $x = 0$. Thus, we use the *logistic loss* function, which is a smooth approximation of (1.3),

$$\max(0, -b_i \operatorname{sign}(x^T a_i)) \approx \log(\exp(0) + \exp(-b_i x^T a_i)).$$

As an optimization problem, this translates to minimizing the penalization of the predictions made by the model over some training sample set,

$$\min_{x \in \mathbb{R}^n} \sum_{i=1}^{m} \log(1 + \exp(-b_i x^T a_i)).$$

The logistic loss function is smooth and convex, and we can add an ℓ_2-regularizer to make the objective strongly convex. For a regularized logistic loss function, Ng [2004] shows that when learning in the presence of many irrelevant features, while the worst case sample complexity for

an ℓ_2-regularized logistic loss problem grows linearly in the number of irrelevant features, the number of training examples required for learning with ℓ_1-regularization grows only logarithmically in the number of irrelevant features. Thus, ℓ_1-regularization can be used to counteract the curse of dimensionality.

Hinge Loss

For problems of classification, where b_i takes on a fixed integer value from a finite set, e.g., $b_i \in \{1, -1\}$, we have the *hinge loss* function,

$$f_i(x) := \max\{1 - b_i x^T a_i, 0\}.$$

This loss function is the tightest convex upper bound (on $[-1, 1]$) of the 0-1 indicator function, or the *true loss* function

$$\mathbf{1}_{\mathbf{b_i x^T a_i < 0}}.$$

The regularized empirical risk minimization problem using this loss is better known as the classic linear Support Vector Machine (SVM) primal problem [Cortes and Vapnik, 1995], and can equivalently be written as the hinge-loss function plus ℓ_2-regularization,

$$\min_{x \in \mathbb{R}^n} \frac{1}{m} \sum_{i=1}^{m} \max\{1 - b_i x^T a_i, 0\} + \lambda \|x\|_2^2.$$

Points that are correctly classified are not penalized, while points that are misclassified (on the wrong side of the separating hyperplane) are penalized linearly with respect to the distance to the correct boundary. This problem is differentiable but non-smooth, and unlike ℓ_1-regularization, this non-smoothness is not separable. As a result, we often consider solving the Fenchel dual of the primal problem, which reduces to a linearly constrained quadratic problem,

$$\min_{z \in [0, U]} \frac{1}{2} z^T M z - \sum_i z_i,$$

where $U \in \mathbb{R}^+$ is a constant and M is a particular positive semi-definite matrix. In [Rosasco et al., 2004], the authors use a probabilistic bound on the estimation error for the classification problem and show the convergence rate of the hinge-loss is almost identical to the logistic loss rate, and far superior to the quadratic loss rate.

1.3 First-Order Methods

The popularity of convex optimization in ML has grown significantly in recent years. There are numerous efficient convex optimization algorithms that have been proven to find globally optimal solutions and use convex geometry to prove rates of convergence for large-scale problems (see [Bertsekas, 2015, Cevher et al., 2014, Nesterov, 2004]).

In this section we consider several commonly used first-order methods for solving the unconstrained convex optimization problem,

$$\min_{x \in \mathbb{R}^n} f(x). \tag{1.4}$$

We assume that f is a convex and differentiable function, and that it satisfies a smoothness assumption. That is, we assume the gradient of f is L-Lipschitz continuous such that for all $x, y \in R^n$ we have

$$\|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\|. \tag{1.5}$$

This smoothness assumption is standard in convex optimization [Nesterov, 2004, § 2.1.1]. For twice differentiable functions, this condition implies that the maximum eigenvalue of the Hessian of f, $\nabla^2 f(x)$, is bounded above by L, [Nesterov, 2004, Lem. 1.2.2].

In some cases we also assume that f is μ-strongly convex [Nesterov, 2004, § 2.1.3], that is for all $x, y \in \mathbb{R}^n$ we have

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2,$$

where $\mu > 0$ is the strong convexity constant. If $\mu = 0$, then this is equivalent to assuming f is simply convex. For twice differentiable functions, strong convexity implies that the minimum eigenvalue of the Hessian is bounded below by μ, [Nesterov, 2004, Thm. 2.1.11].

Under these assumptions (Lipschitz continuity and strong convexity), we can obtain a q-linear convergence rate for first-order methods,

$$\lim_{k \to \infty} \frac{|f(x^{k+1}) - f(x^*)|}{|f(x^k) - f(x^*)|} = \rho, \tag{1.6}$$

where $1 > \rho > 0$ is known as the *rate of convergence*. This means that the function evaluated at the iterates x^k will converge asymptotically to the solution $f(x^*)$ at a constant rate of ρ. This is the classic version of linear convergence used in optimization. However, in this work we say a method achieves a linear convergence rate if

$$f(x^k) - f^* \leq \gamma \rho^k, \tag{1.7}$$

where we explicitly know the constants ρ and $\gamma \in \mathbb{R}$. We can express both these conditions as $O(\rho^k)$, but the condition (1.7) is a stronger condition than (1.6) and more relevant for machine learning, since it focuses on performance in the finite time (non-asymptotic) case. In Section 1.8 we discuss how several conditions have been proposed in the literature to relax the strong convexity assumption while still maintaining a linear convergence rate for first-order methods.

In this work we focus on upper bounding the convergence rates of certain first-order methods. We note that lower bounds on the convergence rate of any first-order method under the

assumptions of Lipschitz continuity and strong convexity have been analyzed, showing that the best lower bound we can obtain is linear [Nesterov, 2004, §2.1.4]. This proves that all first-order methods will converge with at least a linear rate in this setting.

1.4 Gradient Descent

The most classic first-order optimization algorithm is the gradient descent (GD) method. At each iteration a step is taken in the direction of the negative gradient evaluated at the current iteration, yielding the update

$$x^{k+1} = x^k - \alpha_k \nabla f(x^k),$$

where k is the iteration counter, x^k is the current iterate, ∇f is the gradient of f and $\alpha_k > 0$ is a per iteration step-size. The step-size α_k can either be a fixed value defined by some continuity constant of the function f or can be determined using a line search technique at each iteration.

If we assume the function f is convex and has Lipschitz continuous gradient, then using a fixed step-size of $\alpha_k = 1/L$ we can achieve a $O(1/k)$ sublinear convergence rate [Nesterov, 2004, Cor. 2.1.2],

$$f(x^k) - f^* \leq \frac{2L}{k+4}\|x^0 - x^*\|^2.$$

Thus, it requires $O(1/\epsilon)$ iterations to achieve an ϵ-accurate solution. However, under these assumptions there is no guaranteed convergence of the iterates.

If we also assume μ-strong convexity of f, then the derived linear rate for GD is given by [Nesterov, 2004, Thm. 2.2.8],

$$f(x^k) - f(x^*) \leq \left(1 - \frac{\mu}{L}\right)^k \left[f(x^0) - f(x^*)\right],$$

so that the rate of convergence is $O(\left(1 - \frac{\mu}{L}\right)^k)$, where $\mu \leq L$ follows directly from Lipschitz continuity and strong convexity using the results in Nesterov [2004, Thm. 2.1.5 and Thm. 2.1.10]). Under these assumptions, we guarantee convergence of both the iterates and the function values.

1.5 Stochastic Gradient Descent

As mentioned at the end of Section 1.1, there has been a recent trend towards using methods with very cheap iteration costs. One method that reduces the iteration cost of classic GD methods for large-scale problems with specific structures and that has become an important tool for modern high-dimensional ML problems is the stochastic gradient descent (SGD) method [Robbins and Monro, 1951].

Consider problems that have the following finite sum form,

$$f(x) = \frac{1}{m}\sum_{i=1}^{m} f_i(x),$$

where m is very large. At each iteration of the SGD method we randomly select an index $i \in \{1, 2, \ldots, m\}$, and update according to

$$x^{k+1} = x^k - \alpha_k f_i'(x^k).$$

In this update, if f_i is differentiable (smooth) then f_i' is the gradient, ∇f_i, and if f_i is non-differentiable (non-smooth), then f_i' is a subgradient, i.e., an element of the subdifferential of f_i, ∂f_i, where

$$\partial f_i(x) = \{v : f_i(y) \geq f_i(x) + \langle v, y - x\rangle \text{ for all } x, y \in \mathbb{R}\}.$$

This update gives an unbiased estimate of the true gradient,

$$\mathbb{E}[f_i'(x)] = \frac{1}{m}\sum_{i=1}^{m} \nabla f_i(x) = \nabla f(x).$$

In SGD we require that the step-size α_k converges asymptotically to 0 due to the variance of the gradients,

$$\frac{1}{n}\sum_{i=1}^{n} \|\nabla f_i(x^k) - \nabla f(x^k)\|^2.$$

If the variance is zero, then we have an exact gradient and every step is a descent step. If the variance is large, then many of the steps will be in the wrong direction. Therefore, the effects of variance in SGD methods can be reduced by using an appropriate step-size that "scales" this variance. The classic choice of step-size is $\alpha_k = O(1/k)$ and Bach et al. showed using $\alpha_k = O(1/k^\alpha)$ for $\alpha \in (0, 1/2)$ is more robust [Bach and Moulines, 2011] (encourages larger step-sizes).

Deterministic gradient methods [Cauchy, 1847] for this problem have an update cost linear in m, where as the cost of stochastic iterations are independent of m, i.e., m times faster than deterministic. Furthermore, they have the same rates for non-smooth problems, meaning we can solve non-smooth problems m times faster using stochastic methods. The achievable rates are sublinear for convex $O(1/\sqrt{k})$ and strongly convex $O(1/k)$ [Nemirovski et al., 2009]. This is an example of sacrificing convergence rate for cheaper iteration costs, as these rates are clearly slower than the rates obtained by the GD method for smooth problems.

A popular method that improves convergence and also battles the variance introduced by randomness in SGD methods is the stochastic average gradient (SAG) method proposed by Le Roux et al. [2012]. The SAG method still only requires one gradient evaluation at each iteration,

but unlike SGD it achieves a linear convergence rate. The iteration update is given by

$$x^{k+1} = x^k - \frac{\alpha_k}{m} \sum_{i=1}^{m} y_i^k$$

where a memory of $y_i^k = \nabla f_i(x^k)$ from the last k where i was selected. For L-smooth, convex functions f_i, Schmidt et al. [2017] showed that with a constant step-size of $\alpha_k = 1/16L$, the SAG iterations achieve a rate of

$$\mathbb{E}\left[f(\bar{x}^k) - f(x^*)\right] \leq \frac{32n}{k} C_0,$$

where $\bar{x}^k = \frac{1}{k}\sum_{i=0}^{k-1} x^i$ is the average iterate and C_0 is a constant dependent on the initialization of the method. Schmidt et al. [2017] also show a linear rate of convergence when f is μ-strongly convex,

$$\mathbb{E}\left[f(x^k) - f(x^*)\right] \leq \left(1 - \min\left\{\frac{\mu}{16L}, \frac{1}{8n}\right\}\right)^k C_0.$$

These are similar rates compared to GD, but each iteration is m times cheaper.

Alternative methods, such as the Stochastic Variance Reduced Gradient (SVRG) method [Johnson and Zhang, 2013], have also been proposed. Although SVRG is not faster, it does not have the memory requirements of SAG. However, the major challenge of the classic and variance reduced stochastic gradient methods is still choosing the step-size, as a line search is not practical given that there is no guarantee for function value decrease.

1.6 Coordinate Descent Methods

An alternative way to deal with the size of large-scale optimization problems is instead of updating all n variables at each iteration, we can select a single variable (or "block" of variables) to update. We call these methods (block) coordinate descent methods. Each iteration of a coordinate descent (CD) method carries out an approximate update along a single coordinate direction or coordinate hyperplane. In the single coordinate case, the update is given by

$$x^{k+1} = x^k - \alpha_k \nabla_{i_k} f(x^k) e_{i_k},$$

where e_{i_k} is a vector with a one in position i_k and zeros in all other positions.

Since CD methods only update one coordinate at each iteration, or in the case of block CD methods, a subset of coordinates at each iteration, this yields a low iteration cost for problems with certain structure. Nesterov [2010, 2012] brought clarity to the true power of CD methods in his seminal research on CD methods. He showed that coordinate descent can be faster than gradient descent in cases where, if we are optimizing n variables, the cost of performing one full gradient iteration is similar to the cost of performing n single coordinate updates. Essentially, this says that CD methods are highly efficient for numerous popular ML problems

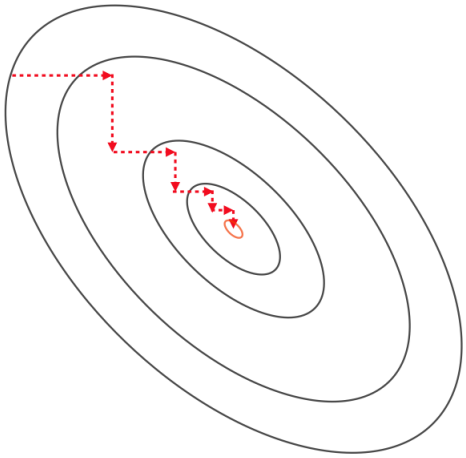


Figure 1.1: Visualization of several iterations of cyclic coordinate descent on the level curves of a quadratic function. The steps alternate between updating x_1 and x_2, and converge towards the minimum value.

like least-squares, logistic regression, LASSO, SVMs, quadratics, graph-based label propagation algorithms for semi-supervised learning and other sparse graph problems.

Like SGD, these methods have become an important tool for modern high-dimensional ML problems. However, CD methods are appealing over SGD methods because each iteration updates a single variable for all f_i whereas SGD updates all variables but only observes one f_i. Thus, we can do a line search to determine α_k at each iteration with CD methods. Further, with CD methods we can improve performance by adapting the algorithmic building blocks to exploit problem structure, which is the main focus of this work.

At each iteration a coordinate i_k (or block of coordinates b_k) is chosen to be updated using a certain selection rule. Most commonly this selection is done in a cyclic (see Figure 1.1) or random fashion. Numerous works have recently been publish on randomized coordinate descent methods and for a comprehensive overview of CD methods and their variations/extensions, see these summary papers [Shi et al., 2016, Wright, 2015].

Assuming smoothness of f and constant step-size $1/L$, randomized CD methods can be shown to achieve a convergence rate of $O(1/k)$. If we further assume strong convexity of f, it has been shown that randomized CD methods achieve a linear convergence rate in expectation,

$$\mathbb{E}[f(x^{k+1})] - f(x^*) \leq \left(1 - \frac{\mu}{Ln}\right)[f(x^k) - f(x^*)].$$

This is a special of case of Nesterov [2012, Theorem 2] with $\alpha = 0$ in his notation. Nesterov [2012] also showed that greedy CD methods (selection of i so as to maximize progress at each iteration) achieves this same bound (not in expectation).

We note that in the above rate, L is the coordinate-wise Lipschitz constant, that is, for each

$i = 1, \ldots, n$,

$$|\nabla_i f(x + \alpha e_i) - \nabla_i f(x)| \leq L|\alpha|, \quad \forall x \in \mathbb{R}^n \text{ and } \alpha \in \mathbb{R},$$

where e_i is a vector with a one in position i and zero in all other positions. For gradient methods, the Lipschitz condition in (1.5) is usually assumed to hold for some Lipschitz constant L_f, where the relationship between these constants is $L/n \leq L_f$. Thus, the above rate is faster.

1.7 Linear Systems and Kaczmarz Methods

Closely-related to CD and SGD methods is the Kaczmarz method [Kaczmarz, 1937], which is designed to solve large-scale consistent (a solution exists) systems of linear equations,

$$Ax = b,$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. This is a fundamental problem in machine learning. At each iteration of the Kaczmarz algorithm, a row i_k is selected and the current iterate x^k is projected onto the hyperplane defined by $a_{i_k}^T x^k = b_{i_k}$. This gives the iteration

$$x^{k+1} = x^k + \frac{b_{i_k} - a_{i_k}^T x^k}{\|a_{i_k}\|^2} a_{i_k},$$

and it has been proven that this algorithm converges linearly to a solution x^* under weak conditions (e.g., each i is visited infinitely often) using cyclic [Deutsch, 1985, Deutsch and Hundal, 1997, Galántai, 2005] or random Strohmer and Vershynin [2009] selection.

The Kaczmarz method projects onto a single hyperplane at each iteration, and thus, has a low-iteration cost like SGD and CD. In fact, the Kaczmarz method can be expressed as an instance of weighted SGD [Needell et al., 2013] when solving the least-squares problem. However a benefit of using Kaczmarz methods over SGD methods for these types of problems is that Kaczmarz methods use a step-size of $\alpha_k = 1$ for all iterations, and thus, avoiding the step-size selection issue of SGD methods.

Further, as discussed by Wright [2015], Kaczmarz methods applied to a linear system can also be interpreted as CD methods on the dual problem,

$$\min_y \frac{1}{2}\|A^T y\|^2 - b^T y,$$

where $x = A^T y^*$, so that $Ax = AA^T y^* = b$. As discussed by Ma et al. [2015a] there are several connections/differences between cyclic CD (also known as the Gauss-Seidel method [Seidel, 1874]) and Kaczmarz methods.

1.8 Relaxing Strong Convexity

To prove linear convergence rates for GD and CD, we assume strong convexity of f. This is a reasonable assumption as an ℓ_2-norm regularizer can be added to make any convex objective strongly convex. Nevertheless, there have been various conditions proposed over the years, all with the goal of replacing or weakening the assumption of strong convexity while still guaranteeing linear convergence for problems like least-squares and logistic regression [Anitescu, 2000, Liu and Wright, 2015, Liu et al., 2014, Łojasiewicz, 1963, Luo and Tseng, 1993, Ma et al., 2015b, Necoara et al., 2015, Polyak, 1963, Zhang and Yin, 2013].

The oldest one of these conditions is the Polyak-Łojasiewicz (PL) inequality [Łojasiewicz, 1963, Polyak, 1963], which requires that for all x, we have

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu\left(f(x) - f^*\right).$$

The PL inequality was proposed in 1963 by Polyak [1963] and is a special case of the Łojasiewicz [1963] inequality proposed in the same year. This condition implies that the gradient grows faster than a quadratic function as we move away from the optimal function value. The PL inequality is sufficient to show a global linear convergence rate for gradient descent without requiring strong convexity (or even convexity). Further, this inequality implies that every stationary point is a global minimum. However, unlike the guarantees of strong convexity, the global minimum need not be unique.

Despite the PL inequality being the oldest of the existing conditions to relaxing strong convexity, it remained relatively unknown in the literature until our recent work [Karimi et al., 2016].

1.9 Proximal First-Order Methods

Several of the first-order methods mentioned in the previous sections have variants that can be used to solve the nonsmooth problem,

$$\min_{x\in\mathbb{R}^n} f(x) + g(x), \tag{1.8}$$

where f is smooth and convex, and g is convex but not necessarily smooth. A classic example of this problem is optimization subject to non-negative constraints,

$$\operatorname*{argmin}_{x\geq 0} f(x),$$

where in this case g_i is the indicator function on the non-negative orthant,

$$g_i(x_i) = \begin{cases} 0 & \text{if } x_i \geq 0, \\ \infty & \text{if } x_i < 0. \end{cases}$$

Another example that has received significant recent attention is the case of an ℓ_1-regularizer,

$$\underset{x \in \mathbb{R}^n}{\operatorname{argmin}} f(x) + \lambda \|x\|_1,$$

where in this case $g_i = \lambda |x_i|$. Here, the ℓ_1-norm regularizer is used to encourage sparsity in the solution. It is possible to generalize GD methods to the nonsmooth problem (1.8) by simply considering subgradients instead of gradients. However, these methods only achieve sublinear rates.

One of most widely-used methods for minimizing functions of the form (1.8) is the proximal gradient (PG) method [Beck and Teboulle, 2009, Bertsekas, 2015, Levitin and Polyak, 1966, Nesterov, 2013], which uses an iteration update given by applying the proximal operator to a standard GD update,

$$x^{k+1} = \operatorname{prox}_{\alpha_k g}\left[x^k - \alpha_k \nabla f(x^k)\right],$$

where the proximal operator is given by

$$\operatorname{prox}_{\alpha g}[y] = \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2}\|x - y\|^2 + \alpha g(x).$$

The proximal versions of the gradient based methods presented in the previous section all achieve the same worst-case convergence rate bounds as regular versions when f is assumed to be smooth. In this work we consider both the smooth problem (1.4) and the nonsmooth problem (1.8).

We also note here that accelerated variants of the methods presented in this chapter exist. For example, it is well-known that the accelerated gradient method achieves the optimal rate of convergence when f is only smooth and convex (not strongly convex) [Nesterov, 1983] (as does proximal gradient descent when f is non-smooth and convex [Nesterov, 2013]). Also, recently an accelerated SGD method was proposed that is robust to noise and variance [Jain et al., 2017]. These methods are not the focus of this work but we expect that all contributions presented in this work should carry over to the accelerated setting.

1.10 Summary of Contributions

In this work we focus on greedy (block) coordinate descent methods and greedy Kaczmarz methods, where for greedy (block) coordinate descent methods, we consider relaxing the strong convexity assumption normally used to show linear convergence. Specifically, we focus on

exploiting the cheap iteration costs of these methods, showing faster rates are achievable when we exploit problem structure. Our list of contributions is as follows:

- **Chapter 2**: **We show that greedy CD methods are faster than random CD methods for problems with certain structure.** Previous bounds show that random and greedy obtain the same convergence rate bounds, but our analysis gives tighter bounds on greedy methods showing that they can be faster. Our work includes a summary of two general problem classes for which one coordinate descent iteration is n times cheaper than a full-gradient update, conditions that ensure efficient implementation of greedy coordinate selection and an exploration of when greedy selection beats random selection using a simple separable quadratic problem. We present a new greedy selection rule that uses Lipschitz gradient information and has a relationship to the nearest neighbour problem. We also present results for approximate and proximal-variants of greedy selection rules. Finally, we present numerical results to emphasize the efficacy of greedy selection rules for coordinate descent methods.

- **Chapter 3**: **We show that greedy Kaczmarz methods are faster than random Kaczmarz methods for problems where A is sparse.** Previous bounds show that random and greedy obtain the same convergence rate bounds, but our analysis gives tighter bounds on greedy methods showing that they can be faster. Our work includes efficient ways to calculate the greedy selection rules when the matrix A is sparse, simpler/tighter convergence rate analysis for randomized selection rules and analysis for greedy selection rules. We present a comparison of general convergence rates for randomized and greedy selection rules, and a comparison of rates for the specific example of a diagonal A. We also present analysis for approximate greedy selection rules and a faster randomized method using adaptive selection rules. Finally, we present numerical results to emphasize the efficiency of greedy selection rules for Kaczmarz methods.

- **Chapter 4**: **We show that of the conditions proposed to relax strong convexity, the PL inequality is the weakest condition that still ensures a global minimum despite it being much older and less popular than other existing conditions.** Our work includes presenting a formal relationship between several of these existing bounds, showing that the PL inequality can be used to establish the first linear convergence rate analysis for sign-based gradient descent methods and establishing different problem classes for which a proximal extension to the PL inequality holds.

- **Chapter 5**: **We show that by adjusting the algorithmic components of block coordinate descent methods such that they exploit problem structure, we are able to obtain significantly faster methods.** Our work includes proposing new greedy block-selection strategies that guarantee more progress per iteration than the classic greedy rule, exploring previously proposed block update strategies that exploit higher-order information and proving faster local convergence rates, and exploring the use of

message-passing to efficiently compute optimal block updates for problems with a sparse dependency between variables. We present numerical results to support all of our findings and establish the efficiency of our greedy block coordinate descent approaches.

- **Chapter 6**: **We show that greedy BCD methods have a finite-time manifold identification property for problems with separable non-smooth structures. Our analysis notably leads to bounds on the number of iterations required to reach the optimal manifold ("active-set complexity").** We show this leads to superlinear convergence when using greedy rules with variable blocks and updates with second-order information for problems with sufficiently-sparse solutions. In the special case of LASSO and SVM problems, we further show that optimal updates are possible. This leads to finite convergence for SVM and LASSO problems with sufficiently-sparse solutions when using greedy selection and sufficiently-large variable blocks. We also use this analysis to show active-set identification and active-set complexity results for the full proximal gradient method.

We note that Chapters 2- 5 include appendices, which contain extra theoretical and experimental results. The details in these appendices are for the interested reader and are not required to understand the main ideas in this dissertation.

In **Chapter 7** we discuss the impact some of these contributions have had since publication, as well as future extensions.

Chapter 2

Greedy Coordinate Descent

There has been substantial recent interest in applying coordinate descent methods to solve large-scale optimization problems because of their cheap iteration costs, low memory requirements and amenability to parallelization. The seminal work of Nesterov [2012] gave the first global rate of convergence analysis for coordinate-descent methods for minimizing convex functions. The analysis in Nesterov's work suggests that choosing a random coordinate to update gives the same performance as choosing the "best" coordinate to update via the more expensive Gauss-Southwell (GS) rule. This result gives a compelling argument to use randomized coordinate descent in contexts where the GS rule is too expensive. It also suggests that there is no benefit to using the GS rule in contexts where it is relatively cheap. However, in these contexts, the GS rule often substantially outperforms randomized coordinate selection in practice. This suggests that either the analysis of GS in [Nesterov, 2012] is not tight, or that there exists a class of functions for which the GS rule is as slow as randomized coordinate descent.

In this chapter, we present our work on greedy coordinate descent methods. We first discuss contexts in which it makes sense to use coordinate descent and the GS rule (Section 2.1). In Section 2.2 we give the existing analysis for random and greedy coordinate descent methods presented by Nesterov [2012], and then we give a tighter convergence rate analysis of the GS rule (under strong convexity and standard smoothness assumptions) that yields the same rate as the randomized method for a restricted class of functions, but is otherwise faster (and in some cases substantially faster). We further show that, compared to the usual *constant step-size* update of the coordinate, the GS method with varying step-sizes has a provably faster rate (Section 2.4). Furthermore, in Section 2.5, we propose a variant of the GS rule that, similar to Nesterov's more clever randomized sampling scheme proposed in [Nesterov, 2012], uses knowledge of the Lipschitz constants of the coordinate-wise gradients to obtain a faster rate. We also analyze approximate GS rules (Section 2.6), which provide an intermediate strategy between randomized methods and the exact GS rule. Finally, we analyze proximal gradient variants of the GS rule (Section 2.7) for optimizing problems that include a separable non-smooth term. All our findings are supported by empirical results on some classic machine learning problems (Section 2.8).

2.1 Problems of Interest

The rates of Nesterov show that coordinate descent can be faster than gradient descent in cases where, if we are optimizing n variables, the cost of performing n coordinate updates is similar to the cost of performing one full gradient iteration. Two common problem structures that satisfy this characterization and therefore are amenable to coordinate descent are:

$$h_1(x) := \sum_{i=1}^{n} g_i(x_i) + f(Ax), \qquad h_2(x) := \sum_{i \in V} g_i(x_i) + \sum_{(i,j) \in E} f_{ij}(x_i, x_j),$$

where x_i is element i of x, f is smooth and cheap, the f_{ij} are smooth, $G = \{V, E\}$ is a graph, and A is a matrix. (It is assumed that all functions are convex.)[2] The family of functions h_1 includes core machine-learning problems such as least squares, logistic regression, LASSO, and SVMs (when solved in dual form) [Hsieh et al., 2008]. Family h_2 includes quadratic functions, graph-based label propagation algorithms for semi-supervised learning [Bengio et al., 2006], and finding the most likely assignments in continuous pairwise graphical models [Rue and Held, 2005].

In general, the GS rule for problem h_2 is as expensive as a full gradient evaluation. However, the structure of G often allows efficient implementation of the GS rule. For example, if each node has at most d neighbours, we can track the gradients of all the variables and use a max-heap structure to implement the GS rule in $O(d \log n)$ time [Meshi et al., 2012]. This is similar to the cost of the randomized algorithm if $d \approx |E|/n$ (since the average cost of the randomized method depends on the average degree). This condition is true in a variety of applications. For example, in spatial statistics we often use two-dimensional grid-structured graphs, where the maximum degree is four and the average degree is slightly less than 4. As another example, for applying graph-based label propagation on the Facebook graph (to detect the spread of diseases, for example), the average number of friends is around 200 but no user has more than seven thousand friends.[3] The maximum number of friends would be even smaller if we removed edges based on proximity. A non-sparse example where GS is efficient is complete graphs, since here the average degree and maximum degree are both $(n-1)$. Thus, the GS rule is efficient for optimizing dense quadratic functions. On the other hand, GS could be very inefficient for star graphs.

If each column of A has at most c non-zeroes and each row has at most r non-zeroes, then for many notable instances of problem h_1 we can implement the GS rule in $O(cr \log n)$ time by maintaining Ax as well as the gradient and again using a max-heap (see Appendix A.1). Thus, GS will be efficient if cr is similar to the number of non-zeroes in A divided by n. Otherwise, Dhillon et al. [2011] show that we can approximate the GS rule for problem h_1 with

[2] We could also consider slightly more general cases like functions that are defined on hyper-edges [Richtárik and Takáč, 2016], provided that we can still perform n coordinate updates for a similar cost to one gradient evaluation.

[3] `https://recordsetter.com/world-record/facebook-friends`

no g_i functions by solving a nearest-neighbour problem. Their analysis of the GS rule in the convex case, however, gives the same convergence rate that is obtained by random selection (although the constant factor can be smaller by a factor of up to n). More recently, Shrivastava and Li [2014] give a general method for approximating the GS rule for problem h_1 with no g_i functions by writing it as a maximum inner-product search problem.

2.2 Analysis of Convergence Rates

We are interested in solving the convex optimization problem

$$\min_{x \in \mathbb{R}^n} \; f(x), \tag{2.1}$$

where ∇f is coordinate-wise L-Lipschitz continuous, i.e., for $i = 1, \ldots, n$,

$$|\nabla_i f(x + \alpha e_i) - \nabla_i f(x)| \leq L|\alpha|, \quad \forall x \in \mathbb{R}^n \text{ and } \alpha \in \mathbb{R},$$

where e_i is a vector with a one in position i and zero in all other positions. For twice-differentiable functions, this is equivalent to the assumption that the diagonal elements of the Hessian are bounded in magnitude by L. In contrast, the typical assumption used for gradient methods is that ∇f is L^f-Lipschitz continuous (note that $L \leq L^f \leq Ln$). The coordinate-descent method with constant step-size is based on the iteration

$$x^{k+1} = x^k - \frac{1}{L}\nabla_{i_k} f(x^k) e_{i_k}.$$

The randomized coordinate-selection rule chooses i_k uniformly from the set $\{1, 2, \ldots, n\}$. Alternatively, the GS rule

$$i_k \in \operatorname*{argmax}_i \; |\nabla_i f(x^k)|,$$

chooses the coordinate with the largest directional derivative (see Figure 2.1). Under either rule, because f is coordinate-wise Lipschitz continuous, we obtain the following bound on the progress made by each iteration:

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) + \nabla_{i_k} f(x^k)(x^{k+1} - x^k)_{i_k} + \frac{L}{2}(x^{k+1} - x^k)_{i_k}^2 \\ &= f(x^k) - \frac{1}{L}(\nabla_{i_k} f(x^k))^2 + \frac{L}{2}\left[\frac{1}{L}\nabla_{i_k} f(x^k)\right]^2 \\ &= f(x^k) - \frac{1}{2L}[\nabla_{i_k} f(x^k)]^2. \end{aligned} \tag{2.2}$$

We focus on the case where f is μ-strongly convex, meaning that, for some positive μ,

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2, \quad \forall x, y \in \mathbb{R}^n, \tag{2.3}$$

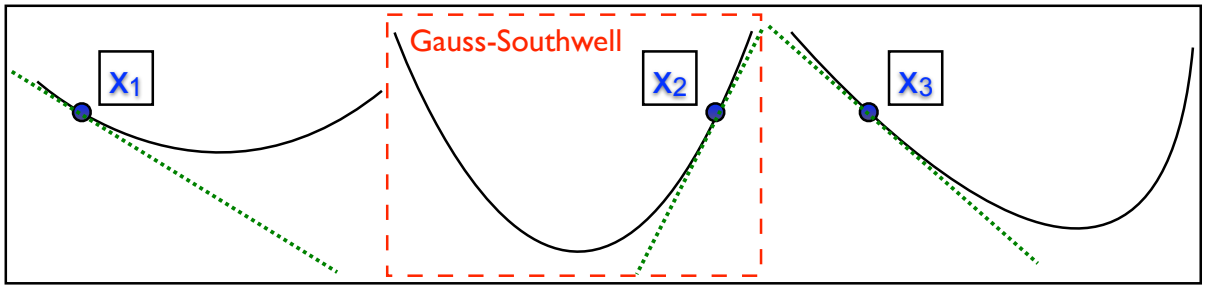


Figure 2.1: Visualization of the Gauss-Southwell selection rule. Shown here are three different projections of a function onto individual coordinates given the corresponding values of $x = [x_1, x_2, x_3]$. The dotted green lines are the individual gradient values (tangent lines) at x. We see that the Gauss-Southwell rule selects the coordinate corresponding to the largest (steepest) individual gradient value (in magnitude).

which implies that

$$f(x^*) \geq f(x^k) - \frac{1}{2\mu}\|\nabla f(x^k)\|^2, \tag{2.4}$$

where x^* is the optimal solution of (2.1). This bound is obtained by minimizing both sides of (2.3) with respect to y.

2.2.1 Randomized Coordinate Descent

Conditioning on the σ-field $\mathcal{F}_{k-1}$ generated by the sequence $\{x^0, x^1, \ldots, x^{k-1}\}$, and taking expectations of both sides of (2.2), when i_k is chosen with uniform sampling we obtain

$$\begin{aligned}
\mathbb{E}[f(x^{k+1})] &\leq \mathbb{E}\left[f(x^k) - \frac{1}{2L}\left(\nabla_{i_k} f(x^k)\right)^2\right] \\
&= f(x^k) - \frac{1}{2L}\sum_{i=1}^{n}\frac{1}{n}\left(\nabla_i f(x^k)\right)^2 \\
&= f(x^k) - \frac{1}{2Ln}\|\nabla f(x^k)\|^2.
\end{aligned}$$

Using (2.4) and subtracting $f(x^*)$ from both sides, we get

$$\mathbb{E}[f(x^{k+1})] - f(x^*) \leq \left(1 - \frac{\mu}{Ln}\right)[f(x^k) - f(x^*)]. \tag{2.5}$$

This is a special of case of Nesterov [2012, Theorem 2] with $\alpha = 0$ in his notation.

2.2.2 Gauss-Southwell

We now consider the progress implied by the GS rule. By the definition of i_k,

$$(\nabla_{i_k} f(x^k))^2 = \|\nabla f(x^k)\|_\infty^2 \geq (1/n)\|\nabla f(x^k)\|^2. \tag{2.6}$$

Applying this inequality to (2.2), we obtain

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2Ln}\|\nabla f(x^k)\|^2,$$

which together with (2.4), implies that

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \frac{\mu}{Ln}\right)[f(x^k) - f(x^*)]. \tag{2.7}$$

This is a special case of Boyd and Vandenberghe [2004, §9.4.3], viewing the GS rule as performing steepest descent in the 1-norm. While this is faster than known rates for cyclic coordinate selection [Beck and Tetruashvili, 2013] and holds deterministically rather than in expectation, this rate is the same as the randomized rate given in (2.5).

2.2.3 Refined Gauss-Southwell Analysis

The deficiency of the existing GS analysis is that too much is lost when we use the inequality in (2.6). To avoid the need to use this inequality we propose measuring strong convexity in the 1-norm, i.e.,

$$f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\mu_1}{2}\|y - x\|_1^2,$$

which is the analogue of (2.3). Minimizing both sides with respect to y, we obtain

$$\begin{aligned} f(x^*) &\geq f(x) - \sup_y\{\langle -\nabla f(x), y - x\rangle - \frac{\mu_1}{2}\|y - x\|_1^2\} \\ &= f(x) - \left(\frac{\mu_1}{2}\|\cdot\|_1^2\right)^*(-\nabla f(x)) \\ &= f(x) - \frac{1}{2\mu_1}\|\nabla f(x)\|_\infty^2, \end{aligned} \tag{2.8}$$

which makes use of the convex conjugate $(\frac{\mu_1}{2}\|\cdot\|_1^2)^* = \frac{1}{2\mu_1}\|\cdot\|_\infty^2$ [Boyd and Vandenberghe, 2004, §3.3]. Using (2.8) in (2.2), and the fact that $(\nabla_{i_k} f(x^k))^2 = \|\nabla f(x^k)\|_\infty^2$ for the GS rule, we obtain

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \frac{\mu_1}{L}\right)[f(x^k) - f(x^*)]. \tag{2.9}$$

It is evident that if $\mu_1 = \mu/n$, then the rates implied by (2.5) and (2.9) are identical, but (2.9) is faster if $\mu_1 > \mu/n$. In Appendix A.2, we show that the relationship between μ and μ_1 can be obtained through the relationship between the squared norms $\|\cdot\|^2$ and $\|\cdot\|_1^2$. In particular, we have

$$\frac{\mu}{n} \leq \mu_1 \leq \mu.$$

Thus, at one extreme the GS rule obtains the same rate as uniform selection ($\mu_1 \approx \mu/n$). However, at the other extreme, it could be faster than uniform selection by a factor of n ($\mu_1 \approx \mu$). This analysis, that the GS rule only obtains the same bound as random selection in an extreme case, supports the better practical behaviour of GS.

2.3 Comparison for Separable Quadratic

We illustrate these two extremes with the simple example of a quadratic function with a diagonal Hessian $\nabla^2 f(x) = \text{diag}(\lambda_1, \ldots, \lambda_n)$. In this case,

$$\mu = \min_i \ \lambda_i, \quad \text{and} \quad \mu_1 = \left(\sum_{i=1}^{n} \frac{1}{\lambda_i} \right)^{-1}.$$

We prove the correctness of this formula for μ_1 in Appendix A.3. The parameter μ_1 achieves its lower bound when all λ_i are equal, $\lambda_1 = \cdots = \lambda_n = \alpha > 0$, in which case

$$\mu = \alpha \quad \text{and} \quad \mu_1 = \alpha/n.$$

Thus, uniform selection does as well as the GS rule if all elements of the gradient change at *exactly* the same rate. This is reasonable: under this condition, there is no apparent advantage in selecting the coordinate to update in a clever way. Intuitively, one might expect that the favourable case for the Gauss-Southwell rule would be where one λ_i is much larger than the others. However, in this case, μ_1 is again similar to μ/n. To achieve the other extreme, suppose that $\lambda_1 = \beta$ and $\lambda_2 = \lambda_3 = \cdots = \lambda_n = \alpha$ with $\alpha \geq \beta$. In this case, we have $\mu = \beta$ and

$$\mu_1 = \frac{\beta\alpha^{n-1}}{\alpha^{n-1} + (n-1)\beta\alpha^{n-2}} = \frac{\beta\alpha}{\alpha + (n-1)\beta}.$$

If we take $\alpha \to \infty$, then we have $\mu_1 \to \beta$, so $\mu_1 \to \mu$. This case is much less intuitive; GS is n times faster than random coordinate selection if one element of the gradient changes much more *slowly* than the others.

2.3.1 'Working Together' Interpretation

In the separable quadratic case above, μ_1 is given by the harmonic mean of the eigenvalues of the Hessian divided by n. The harmonic mean is dominated by its smallest values, and this is why having one small value is a notable case. Furthermore, the harmonic mean divided by n has an interpretation in terms of processes 'working together' [Ferger, 1931]. If each λ_i represents the time taken by each process to finish a task (e.g., large values of λ_i correspond to slow workers), then μ is the time needed by the fastest worker to complete the task, and μ_1 is the time needed to complete the task if all processes work together (and have independent effects). Using this interpretation, the GS rule provides the most benefit over random selection when *working together is not efficient*, meaning that if the n processes work together, then the task is not solved much faster than if the fastest worker performed the task alone. This gives an interpretation of the non-intuitive scenario where GS provides the most benefit: if all workers have the same efficiency, then working together solves the problem n times faster. Similarly, if there is one slow worker (large λ_i), then the problem is solved roughly n times faster by working

together. On the other hand, if most workers are slow (many large λ_i), then working together has little benefit and we should be greedy in our selection of the workers.

2.3.2 Fast Convergence with Bias Term

Consider the standard linear-prediction framework,

$$\underset{x,\beta}{\operatorname{argmin}} \sum_{i=1}^{m} f(a_i^T x + \beta) + \frac{\lambda}{2}\|x\|^2 + \frac{\sigma}{2}\beta^2,$$

where we have included a bias variable β (an example of problem h_1). Typically, the regularization parameter σ of the bias variable is set to be much smaller than the regularization parameter λ of the other covariates, to avoid biasing against a global shift in the predictor. Assuming that there is no hidden strong convexity in the sum, this problem has the structure described in the previous section ($\mu_1 \approx \mu$) where GS has the most benefit over random selection.

2.4 Rates with Different Lipschitz Constants

Consider the more general scenario where we have a Lipschitz constant L_i for the partial derivative of f with respect to each coordinate i,

$$|\nabla_i f(x + \alpha e_i) - \nabla_i f(x)| \leq L_i|\alpha|, \quad \forall x \in \mathbb{R}^n \text{ and } \alpha \in \mathbb{R},$$

and we use a coordinate-dependent step-size at each iteration:

$$x^{k+1} = x^k - \frac{1}{L_{i_k}} \nabla_{i_k} f(x^k) e_{i_k}. \tag{2.10}$$

By the logic of (2.2), in this setting we have

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L_{i_k}} [\nabla_{i_k} f(x^k)]^2, \tag{2.11}$$

and thus a convergence rate of

$$f(x^k) - f(x^*) \leq \left[\prod_{j=1}^{k} \left(1 - \frac{\mu_1}{L_{i_j}}\right)\right] [f(x^0) - f(x^*)]. \tag{2.12}$$

Noting that $L = \max_i\{L_i\}$, we have

$$\prod_{j=1}^{k} \left(1 - \frac{\mu_1}{L_{i_j}}\right) \leq \left(1 - \frac{\mu_1}{L}\right)^k. \tag{2.13}$$

Thus, the convergence rate based on the L_i will be faster, provided that at least one iteration chooses an i_k with $L_{i_k} < L$. In the worst case, however, (2.13) holds with equality even if the L_i are distinct, as we might need to update a coordinate with $L_i = L$ on every iteration. (For example, consider a separable function where all but one coordinate is initialized at its optimal value, and the remaining coordinate has $L_i = L$.) In Section 2.5, we discuss selection rules that incorporate the L_i to achieve faster rates whenever the L_i are distinct.

2.5 Rules Depending on Lipschitz Constants

If the L_i are known, Nesterov [2012] showed that we can obtain a faster convergence rate by sampling proportional to the L_i. We review this result below and compare it to the GS rule, and then propose an improved GS rule for this scenario. Although in this section we will assume that the L_i are known, this assumption can be relaxed using a backtracking procedure [Nesterov, 2012, §6.1].

2.5.1 Lipschitz Sampling

Taking the expectation of (2.11) under the distribution $p_i = L_i / \sum_{j=1}^n L_j$ and proceeding as before, we obtain

$$\mathbb{E}[f(x^{k+1})] - f(x^*) \leq \left(1 - \frac{\mu}{n\bar{L}}\right)[f(x^k) - f(x^*)],$$

where $\bar{L} = \frac{1}{n}\sum_{j=1}^n L_j$ is the average of the Lipschitz constants. This was shown by Leventhal and Lewis [2010] and is a special case of Nesterov [2012, Theorem 2] with $\alpha = 1$ in his notation. This rate is faster than (2.5) for uniform sampling if any L_i differ.

Under our analysis, this rate may or may not be faster than (2.9) for the GS rule. On the one extreme, if $\mu_1 = \mu/n$ and any L_i differ, then this Lipschitz sampling scheme is faster than our rate for GS. Indeed, in the context of the problem from Section 2.3, we can make Lipschitz sampling faster than GS by a factor of nearly n by making one λ_i much larger than all the others (recall that our analysis shows no benefit to the GS rule over randomized selection when only one λ_i is much larger than the others). At the other extreme, in our example from Section 2.3 with many large α and one small β, the GS and Lipschitz sampling rates are the same when $n = 2$, with a rate of $(1 - \beta/(\alpha + \beta))$. However, the GS rate will be faster than the Lipschitz sampling rate for any $\alpha > \beta$ when $n > 2$, as the Lipschitz sampling rate is $(1-\beta/((n-1)\alpha+\beta))$, which is slower than the GS rate of $(1 - \beta/(\alpha + (n-1)\beta))$.

2.5.2 Gauss-Southwell-Lipschitz Rule

Since neither Lipschitz sampling nor GS dominates the other in general, we are motivated to consider if faster rules are possible by combining the two approaches. Indeed, we obtain a faster

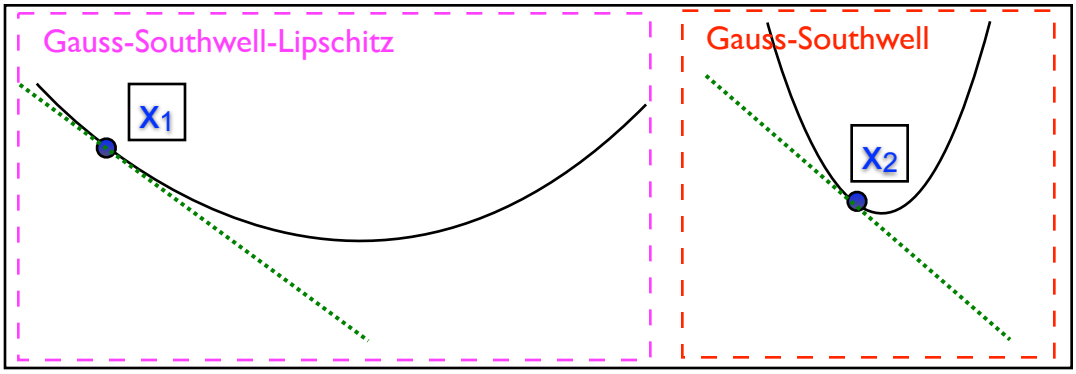


Figure 2.2: Visualization of the Gauss-Southwell-Lipschitz selection rule compared to the Gauss-Southwell selection rule. When the slopes of the tangent lines (gradient values) are similar, the GSL will make more progress by selecting the coordinate with the slower changing derivative (smaller L_i).

rate by choosing the i_k that minimizes (2.11), leading to the rule

$$i_k \in \underset{i}{\operatorname{argmax}} \frac{|\nabla_i f(x^k)|}{\sqrt{L_i}},$$

which we call the *Gauss-Southwell-Lipschitz* (GSL) rule. We see in Figure 2.2 that if the directional derivative between two coordinates is equal, then compared to the GS rule, the GSL rule will select the coordinate that leads to more function value progress (smaller L_i).

Following a similar argument to Section 2.2.3, but using (2.11) in place of (2.2), the GSL rule obtains a convergence rate of

$$f(x^{k+1}) - f(x^*) \leq (1 - \mu_L)[f(x^k) - f(x^*)],$$

where μ_L is the strong convexity constant with respect to the norm $\|x\|_L = \sum_{i=1}^n \sqrt{L_i}|x_i|$. This is shown in Appendix A.4, and in Appendix A.5 we show that

$$\max\left\{\frac{\mu}{n\bar{L}}, \frac{\mu_1}{L}\right\} \leq \mu_L \leq \frac{\mu_1}{\min_i\{L_i\}}.$$

Thus, the GSL rule is always at least as fast as the fastest of the GS rule and Lipschitz sampling. Indeed, it can be more than a factor of n faster than using Lipschitz sampling, while it can obtain a rate closer to the minimum L_i, instead of the maximum L_i that the classic GS rule depends on.

An interesting property of the GSL rule for quadratic functions is that it is the *optimal* myopic coordinate update. That is, if we have an oracle that can choose the coordinate and the step-size that decreases f by the largest amount, i.e.,

$$f(x^{k+1}) \equiv \underset{i,\alpha}{\operatorname{argmin}}\{f(x^k + \alpha e_i)\}, \tag{2.14}$$

this is equivalent to using the GSL rule and the update in (2.10). This follows because (2.11) holds with equality in the quadratic case, and the choice $\alpha_k = 1/L_{i_k}$ yields the optimal step-size. Thus, although faster schemes could be possible with non-myopic strategies that cleverly choose the sequence of coordinates or step-sizes, if we can only perform one iteration, then the GSL rule cannot be improved.

For general f, (2.14) is known as the *maximum improvement* (MI) rule. This rule has been used in the context of boosting [Rätsch et al., 2001], graphical models [Della Pietra et al., 1997, Lee et al., 2006, Scheinberg and Rish, 2009], Gaussian processes [Bo and Sminchisescu, 2012], and low-rank tensor approximations [Li et al., 2015]. By the argument

$$\begin{aligned} f(x^{k+1}) &= \min_{\alpha}\{f(x^k + \alpha e_{i_k})\} \\ &\leq f\left(x^k - \frac{1}{L_{i_k}}\nabla_{i_i} f(x^k) e_{i_k}\right) \\ &\leq f(x^k) - \frac{1}{2}\frac{[\nabla_{i_k} f(x^k)]^2}{L_{i_k}}, \end{aligned} \tag{2.15}$$

our GSL rate also applies to the MI rule, improving existing bounds on this strategy. However, the GSL rule is much cheaper and does not require any special structure (recall that we can estimate L_i as we go).

2.5.3 Connection between GSL Rule and Normalized Nearest Neighbour Search

Dhillon et al. [2011] discuss an interesting connection between the GS rule and the nearest-neighbour-search (NNS) problem for objectives of the form

$$\min_{x \in \mathbb{R}^n} F(x) = f(Ax), \tag{2.16}$$

This is a special case of h_1 with no g_i functions, and its gradient has the special form

$$\nabla F(x) = A^T r(x),$$

where $r(x) = \nabla f(Ax)$. We use the symbol r because it is the residual vector $(Ax - b)$ in the special case of least squares. For this problem structure the GS rule has the form

$$\begin{aligned} i_k &\in \operatorname*{argmax}_i |\nabla_i f(x^k)| \\ &\equiv \operatorname*{argmax}_i |r(x^k)^T a_i|, \end{aligned}$$

where a_i denotes column i of A for $i = 1, \ldots, n$. Dhillon et al. [2011] propose to approximate

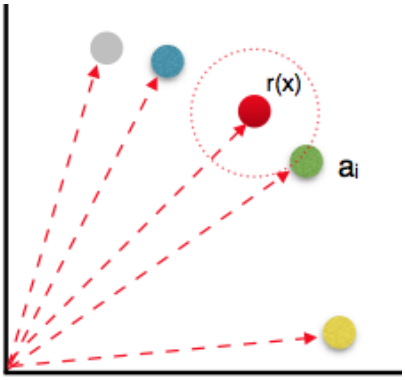


Figure 2.3: Visualization of the GS rule as a nearest neighbour problem. The "nearest neighbour" corresponds to the vector that is the closest (in distance) to $r(x)$, i.e., we want to minimize the distance between two vectors. Alternatively, the GS rule is evaluated as a maximization of an inner product. This makes sense as the smaller the angle between two vectors, the larger the cosine of that angle and in turn, the larger the inner product.

the above argmax by solving the following NNS problem (see Figure 2.3)

$$i_k \in \operatorname*{argmin}_{i \in [2n]} \|r(x^k) - a_i\|,$$

where i in the range $(n+1)$ through $2n$ refers to the negation $-(a_{i-n})$ of column $(i-n)$ and if the selected i_k is greater than n we return $(i-n)$. We can justify this approximation using the logic

$$\begin{aligned}
i_k &\in \operatorname*{argmin}_{i \in [2n]} \|r(x^k) - a_i\| \\
&\equiv \operatorname*{argmin}_{i \in [2n]} \frac{1}{2}\|r(x^k) - a_i\|^2 \\
&\equiv \operatorname*{argmin}_{i \in [2n]} \underbrace{\frac{1}{2}\|r(x^k)\|^2}_{\text{constant}} - r(x^k)^T a_i + \frac{1}{2}\|a_i\|^2 \\
&\equiv \operatorname*{argmax}_{i \in [n]} |r(x^k)^T a_i| - \frac{1}{2}\|a_i\|^2 \\
&\equiv \operatorname*{argmax}_{i \in [n]} |\nabla_i f(x^k)| - \frac{1}{2}\|a_i\|^2.
\end{aligned}$$

Thus, the NNS computes an approximation to the GS rule that is biased towards coordinates where $\|a_i\|$ is small. Note that this formulation is equivalent to the GS rule in the special case that $\|a_i\| = 1$ (or any other constant) for all i. Shrivastava and Li [2014] have more recently considered the case where $\|a_i\| \leq 1$ and incorporate powers of $\|a_i\|$ in the NNS to yield a better approximation.

A further interesting property of the GSL rule is that we can often formulate the *exact* GSL

rule as a *normalized* NNS problem. In particular, for problem (2.16) the Lipschitz constants will often have the form $L_i = \gamma\|a_i\|^2$ for a some positive scalar γ. For example, least squares has $\gamma = 1$ and logistic regression has $\gamma = 0.25$. When the Lipschitz constants have this form, we can compute the exact GSL rule by solving a normalized NNS problem,

$$i_k \in \underset{i\in[2n]}{\operatorname{argmin}} \left|\left| r(x^k) - \frac{a_i}{\|a_i\|} \right|\right|. \tag{2.17}$$

The exactness of this formula follows because

$$\begin{aligned}
i_k &\in \underset{i\in[2n]}{\operatorname{argmin}} \left|\left| r(x^k) - \frac{a_i}{\|a_i\|} \right|\right| \\
&\equiv \underset{i\in[2n]}{\operatorname{argmin}} \frac{1}{2}\|r(x^k) - a_i/\|a_i\|\|^2 \\
&\equiv \underset{i\in[2n]}{\operatorname{argmin}} \underbrace{\frac{1}{2}\|r(x^k)\|^2}_{\text{constant}} - \frac{r(x^k)^T a_i}{\|a_i\|} + \underbrace{\frac{1}{2}\frac{\|a_i\|^2}{\|a_i\|^2}}_{\text{constant}} \\
&\equiv \underset{i\in[n]}{\operatorname{argmax}} \frac{|r(x^k)^T a_i|}{\|a_i\|} \\
&\equiv \underset{i\in[n]}{\operatorname{argmax}} \frac{|r(x^k)^T a_i|}{\sqrt{\gamma}\|a_i\|} \\
&\equiv \underset{i\in[n]}{\operatorname{argmax}} \frac{|\nabla_i f(x^k)|}{\sqrt{L_i}}.
\end{aligned}$$

Thus, the form of the Lipschitz constant conveniently removes the bias towards smaller values of $\|a_i\|$ that gets introduced when we try to formulate the classic GS rule as a NNS problem. Interestingly, in this setting we *do not need to know* γ to implement the GSL rule as a NNS problem.

2.6 Approximate Gauss-Southwell

In many applications, computing the exact GS rule is too inefficient to be of any practical use. However, a computationally cheaper *approximate* GS rule might be available. Approximate GS rules under multiplicative and additive errors were considered by Dhillon et al. [2011] in the convex case, but in this setting the convergence rate is similar to the rate achieved by random selection. In this section, we give rates depending on μ_1 for approximate GS rules.

2.6.1 Multiplicative Errors

In the multiplicative error regime, the approximate GS rule chooses an i_k satisfying

$$|\nabla_{i_k} f(x^k)| \geq \|\nabla f(x^k)\|_\infty(1 - \epsilon_k),$$

for some $\epsilon_k \in [0, 1)$. In this regime, our basic bound on the progress (2.2) still holds, as it was defined for any i_k. We can incorporate this type of error into our lower bound (2.8) to obtain

$$\begin{aligned} f(x^*) &\geq f(x^k) - \frac{1}{2\mu_1}\|\nabla f(x^k)\|_\infty^2 \\ &\geq f(x^k) - \frac{1}{2\mu_1(1-\epsilon_k)^2}|\nabla_{i_k} f(x^k)|^2. \end{aligned}$$

This implies a convergence rate of

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \frac{\mu_1(1-\epsilon_k)^2}{L}\right)[f(x^k) - f(x^*)].$$

Thus, the convergence rate of the method is nearly identical to using the exact GS rule for small ϵ_k (and it degrades gracefully with ϵ_k). This is in contrast to having an error in the gradient [Friedlander and Schmidt, 2012], where the error ϵ must decrease to zero over time.

2.6.2 Additive Errors

In the additive error regime, the approximate GS rule chooses an i_k satisfying

$$|\nabla_{i_k} f(x^k)| \geq \|\nabla f(x^k)\|_\infty - \epsilon_k,$$

for some $\epsilon_k \geq 0$. In Appendix A.6, we show that under this rule, we have

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \frac{\mu_1}{L}\right)^k \left[f(x^0) - f(x^*) + A_k\right],$$

where

$$\begin{aligned} A_k \leq \min\Bigg\{ &\sum_{i=1}^{k} \left(1 - \frac{\mu_1}{L}\right)^{-i} \epsilon_i \frac{\sqrt{2L_1}}{L}\sqrt{f(x^0) - f(x^*)}, \\ &\sum_{i=1}^{k} \left(1 - \frac{\mu_1}{L}\right)^{-i} \left(\epsilon_i \sqrt{\frac{2}{L}}\sqrt{f(x^0) - f(x^*)} + \frac{\epsilon_i^2}{2L}\right)\Bigg\}, \end{aligned}$$

where L_1 is the Lipschitz constant of ∇f with respect to the 1-norm. Note that L_1 could be substantially larger than L, so the second part of the minimum in A_k is likely to be the smaller part unless the ϵ_i are large. This regime is closer to the case of having an error in the gradient, as to obtain convergence the ϵ_k must decrease to zero. This result implies that a sufficient condition for the algorithm to obtain a linear convergence rate is that the errors ϵ_k converge to zero at a linear rate. Further, if the errors satisfy $\epsilon_k = O(\rho^k)$ for some $\rho < (1 - \mu_1/L)$, then the convergence rate of the method is the same as if we used an exact GS rule. On the other hand, if ϵ_k does not decrease to zero, we may end up repeatedly updating the same wrong coordinate and the algorithm will not converge (though we could switch to the randomized method if this

is detected).

2.7 Proximal Gradient Gauss-Southwell

One of the key motivations for the resurgence of interest in coordinate descent methods is their performance on problems of the form

$$\min_{x\in\mathbb{R}^n} F(x) \equiv f(x) + \sum_{i=1}^{n} g_i(x_i),$$

where f is smooth and convex and the g_i are convex, but possibly non-smooth. This includes problems with ℓ_1-regularization, and optimization with lower and/or upper bounds on the variables. Similar to proximal gradient methods, we can apply the proximal operator to the coordinate update,

$$x^{k+1} = \text{prox}_{\frac{1}{L}g_{i_k}}\left[x^k - \frac{1}{L}\nabla_{i_k} f(x^k)e_{i_k}\right],$$

where

$$\text{prox}_{\alpha g_i}[y] = \underset{x\in\mathbb{R}^n}{\text{argmin}} \frac{1}{2}\|x-y\|^2 + \alpha g_i(x_i).$$

We note that all variables other than the selected x_i stay at their existing values in the optimal solution to this problem. With random coordinate selection, Richtárik and Takáč [2014] show that this method has a convergence rate of

$$\mathbb{E}[F(x^{k+1}) - F(x^*)] \leq \left(1 - \frac{\mu}{nL}\right)[F(x^k) - F(x^*)],$$

similar to the unconstrained/smooth case.

There are several generalizations of the GS rule to this scenario. Here we consider three possibilities, all of which are equivalent to the GS rule if the g_i are not present. First, the GS-s rule chooses the coordinate with the most negative directional derivative. This strategy is popular for ℓ_1-regularization [Li and Osher, 2009, Shevade and Keerthi, 2003, Wu and Lange, 2008] and in general is given by [see Bertsekas, 2016, §8.4]

$$i_k \in \underset{i}{\text{argmax}} \left\{ \min_{s\in\partial g_i} |\nabla_i f(x^k) + s| \right\}.$$

However, the length of the step ($\|x^{k+1} - x^k\|$) could be arbitrarily small under this choice. In contrast, the GS-r rule chooses the coordinate that maximizes the length of the step [Dhillon et al., 2011, Tseng and Yun, 2009b],

$$i_k \in \underset{i}{\text{argmax}} \left\{ \left| x_i^k - \text{prox}_{\frac{1}{L}g_i}\left[x_i^k - \frac{1}{L}\nabla_i f(x^k)\right]\right| \right\}.$$

This rule is effective for bound-constrained problems, but it ignores the change in the non-

smooth term $(g_i(x_i^{k+1}) - g_i(x_k^k))$. Finally, the GS-$q$ rule maximizes progress assuming a quadratic upper bound on f [Tseng and Yun, 2009b],

$$i_k \in \operatorname*{argmin}_i \left\{ \min_d \left\{ f(x^k) + \nabla_i f(x^k) d + \frac{L}{2} d^2 + g_i(x_i^k + d) - g_i(x_i^k) \right\} \right\}.$$

While the least intuitive rule, the GS-q rule seems to have the best theoretical properties. Further, if we use L_i in place of L in the GS-q rule (which we call the GSL-q strategy), then we obtain the GSL rule if the g_i are not present. In contrast, using L_i in place of L in the GS-r rule (which we call the GSL-r strategy) does not yield the GSL rule as a special case.

In Appendix A.7, we show that using the GS-q rule yields a convergence rate of

$$F(x^{k+1}) - F(x^*) \leq \min \left\{ \left(1 - \frac{\mu}{Ln}\right) [f(x^k) - f(x^*)], \left(1 - \frac{\mu_1}{L}\right) [f(x^k) - f(x^*)] + \epsilon_k \right\}, \tag{2.18}$$

where ϵ_k is bounded above by a measure of the non-linearity of the g_i along the possible coordinate updates times the inverse condition number μ_1/L. Note that ϵ_k goes to zero as k increases. In contrast, in Appendix A.7 we also give counter-examples showing that the rate in (2.18) does not hold with $\epsilon_k = 0$ for the GS-s or GS-r rule, even if the minimum is replaced by a maximum. Thus, any bound for the GS-s or GS-r rule would be slower than the expected rate under random selection, while the GS-q rule leads to a better bound.

Recently, Song et al. [2017] proposed an alternative GS-q rule for ℓ_1-regularized problems ($g(x) := \|x\|_1$) that uses an ℓ_1-norm square approximation. A generalized version of this update rule is given by

$$\begin{aligned} d^k &\in \min_{d \in \mathbb{R}^n} \left\{ \langle \nabla f(x^k), d \rangle + \frac{L}{2} \|d\|_1^2 + g(x^k + d) \right\} \\ x^{k+1} &= x^k + d^k, \end{aligned}$$

which is equivalent to the GS rule in the smooth case [Boyd and Vandenberghe, 2004, §9.4.2]. Song et al. [2017] show that this version of the GS-q rule improves the convergence rate by a constant factor over random in the convex, ℓ_1-regularized ERM setting. In Appendix A.8 we show that the ϵ_k term in (2.18) is zero when using this update and assuming L_1-Lipschitz continuity. However, unlike the other non-smooth generalizations of the GS rule, this generalization may select more than one variable to update at each iteration (update all coordinates corresponding to non-zero entries in d^k) making this method more like block coordinate descent.

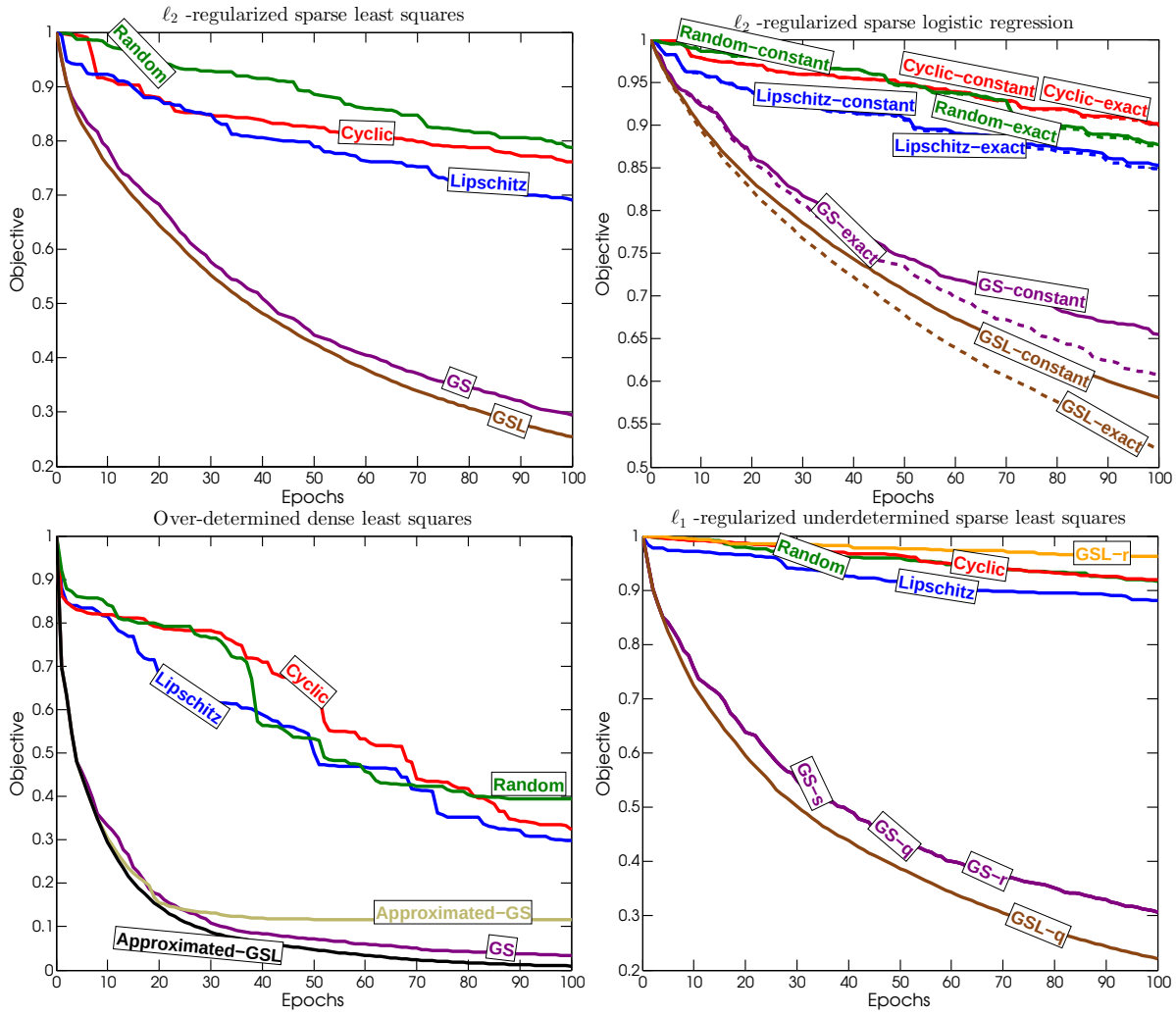


Figure 2.4: Comparison of coordinate selection rules for 4 instances of problem h_1.

2.8 Experiments

We first compare the efficacy of different coordinate selection rules on the following simple instances of h_1.

ℓ_2-regularized sparse least squares: Here we consider the problem

$$\min_x \frac{1}{2m}\|Ax - b\|^2 + \frac{\lambda}{2}\|x\|^2,$$

an instance of problem h_1. We set A to be an m by n matrix with entries sampled from a $\mathcal{N}(0,1)$ distribution (with $m = 1000$ and $n = 1000$). We then added 1 to each entry (to induce a dependency between columns), multiplied each column by a sample from $\mathcal{N}(0,1)$ multiplied by ten (to induce different Lipschitz constants across the coordinates), and only kept each entry of A non-zero with probability $10\log(n)/n$ (a sparsity level that allows the Gauss-Southwell rule to be applied with cost $O(\log^3(n))$. We set $\lambda = 1$ and $b = Ax + e$, where the entries of x and e were drawn from a $\mathcal{N}(0,1)$ distribution. In this setting, we used a step-size of $1/L_i$ for each coordinate i, which corresponds to exact coordinate optimization.

ℓ_2-regularized sparse logistic regression: Here we consider the problem

$$\min_x \frac{1}{m}\sum_{i=1}^{m} \log(1 + \exp(-b_i a_i^T x)) + \frac{\lambda}{2}\|x\|^2.$$

We set the a_i^T to be the rows of A from the previous problem, and set $b = \text{sign}(Ax)$, but randomly flipping each b_i with probability 0.1. In this setting, we compared using a step-size of $1/L_i$ to using exact coordinate optimization.

Over-determined dense least squares: Here we consider the problem

$$\min_x \frac{1}{2m}\|Ax - b\|^2,$$

but, unlike the previous case, we do not set elements of A to zero and we make A have dimension 1000 by 100. Because the system is over-determined, it does not need an explicit strongly convex regularizer to induce global strong convexity. In this case, the density level means that the exact GS rule is not efficient. Hence, we use a balltree structure [Omohundro, 1989] to implement an efficient approximate GS rule based on the connection to the NNS problem discovered by Dhillon et al. [2011]. On the other hand, we can compute the exact GSL rule for this problem as a NNS problem as discussed in Section 2.5.3.

ℓ_1-regularized underdetermined sparse least squares: Here we consider the non-smooth

problem

$$\min_x \frac{1}{2m}\|Ax - b\|^2 + \lambda\|x\|_1.$$

We generate A as we did for the ℓ_2-regularized sparse least squares problem, except with the dimension 1000 by 10000. This problem is not globally strongly convex, but will be strongly convex along the dimensions that are non-zero in the optimal solution.

We plot the objective function (divided by its initial value) of coordinate descent under different selection rules in Figure 2.4. Even on these simple datasets, we see dramatic differences in performance between the different strategies. In particular, the GS rule outperforms random coordinate selection (as well as cyclic selection) by a substantial margin in all cases. The Lipschitz sampling strategy can narrow this gap, but it remains large (even when an approximate GS rule is used). The difference between GS and randomized selection seems to be most dramatic for the ℓ_1-regularized problem; the GS rules tend to focus on the non-zero variables while most randomized/cyclic updates focus on the zero variables, which tend not to move away from zero.[4] Exact coordinate optimization and using the GSL rule seem to give modest but consistent improvements. The three non-smooth GS-$*$ rules had nearly identical performance despite their different theoretical properties. The GSL-q rule gave better performance than the GS-$*$ rules, while the GSL-r variant performed worse than even cyclic and random strategies. We found it was also possible to make the GS-s rule perform poorly by perturbing the initialization away from zero.

While the results in this section are in terms of epochs, we direct the reader to Nutini et al. [2015, Appendix I] for runtime experiments for the ℓ_2-regularized sparse least squares problem defined above. The authors use the efficient max-heap implementation in `scikit-learn` [Pedregosa et al., 2011] and show that the GS and GSL rules also offer benefits in terms of runtime over cyclic, random and Lipschitz sampling.

We next consider an instance of problem h_2, that is, performing label propagation for semi-supervised learning in the 'two moons' dataset [Zhou et al., 2003]. We generate 500 samples from this dataset, randomly label five points in the data, and connect each node to its five nearest neighbours. This high level of sparsity is typical of graph-based methods for semi-supervised learning, and allows the exact Gauss-Southwell rule to be implemented efficiently. We use the quadratic labeling criterion of Bengio et al. [2006], which allows exact coordinate optimization and is normally optimized with cyclic coordinate descent. We plot the performance under different selection rules in Figure 2.5. Here, we see that even cyclic coordinate descent outperforms randomized coordinate descent, but that the GS and GSL rules give even better performance. We note that the GS and GSL rules perform similarly on this problem since the Lipschitz constants do not vary much.

[4] To reduce the cost of the GS-s method in this context, Shevade and Keerthi [2003] consider a variant where we first compute the GS-s rule for the non-zero variables and if an element is sufficiently large then they do not consider the zero variables.

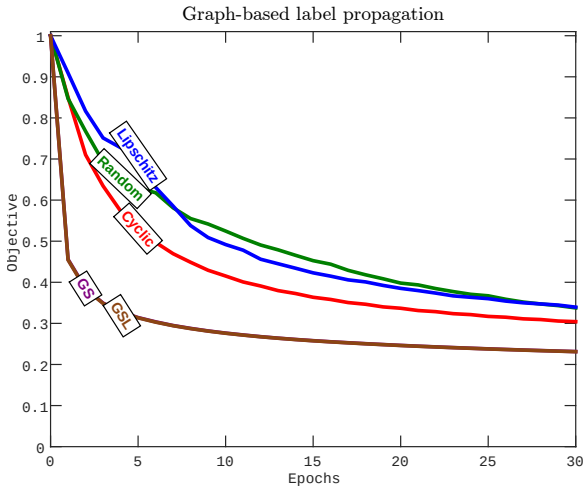


Figure 2.5: Comparison of coordinate selection rules for graph-based semi-supervised learning.

2.9 Discussion

It is clear that the GS rule is not practical for every problem where randomized methods are applicable. Nevertheless, we have shown that even approximate GS rules can obtain better convergence rate bounds than fully-randomized methods. We have given a similar justification for the use of exact coordinate optimization, and we note that our argument could also be used to justify the use of exact coordinate optimization within randomized coordinate descent methods (as used in our experiments). We have also proposed the improved GSL rule, and considered approximate/proximal variants. In Chapter 4 we show that our analysis can be used for scenarios without strong convexity and in Chapter 5 we show it also applies to block updates. We expect it could also be used for accelerated/parallel methods [Fercoq and Richtárik, 2015], for primal-dual rates of dual coordinate ascent [Shalev-Shwartz and Zhang, 2013], for successive projection methods [Leventhal and Lewis, 2010] and for boosting algorithms [Rätsch et al., 2001].

Chapter 3

Greedy Kaczmarz

Solving large linear systems is a fundamental problem in machine learning. Applications range from least-squares problems to Gaussian processes to graph-based semi-supervised learning. All of these applications (and many others) benefit from advances in solving large-scale linear systems. The Kaczmarz method is a particular iterative algorithm suited for solving consistent linear systems of the form $Ax = b$. This method as we know it today was originally proposed by Polish mathematician Stefan Kaczmarz [1937] and later re-invented by Gordon et al. [1970] under the name *algebraic reconstruction technique* (ART). However, this method is closely related to the *method of alternating projections*, which was proposed by von Neumann in 1933 for the case of 2 subspaces (published in 1950, von Neumann [1950]). It has been used in numerous applications including image reconstruction and digital signal processing, and belongs to several general categories of methods including *row-action*, *component-solution*, *cyclic projection*, *successive projection* methods [Censor, 1981] and *stochastic gradient descent* (when applied to a least-squares problem [Needell et al., 2013]).

At each iteration k, the Kaczmarz method uses a *selection rule* to choose some row i_k of A and then projects the current iterate x^k onto the corresponding hyperplane $a_{i_k}^T x^k = b_{i_k}$. Classically, the two categories of selection rules are *cyclic* and *random*. Cyclic selection repeatedly cycles through the coordinates in sequential order, making it simple to implement and computationally inexpensive. There are various linear convergence rates for cyclic selection [see Deutsch, 1985, Deutsch and Hundal, 1997, Galántai, 2005], but these rates are in terms of cycles through the entire dataset and involve constants that are not easily interpreted. Further, the performance of cyclic selection worsens if we have an undesirable ordering of the rows of A.

Randomized selection has recently become the default selection rule in the literature on Kaczmarz-type methods. Empirically, selecting i_k randomly often performs substantially better in practice than cyclic selection [Feichtinger et al., 1992, Herman and Meyer, 1993]. Although a number of asymptotic convergence rates for randomized selection have been presented [Censor et al., 1983, Hanke and Niethammer, 1990, Tanabe, 1971, Whitney and Meany, 1967], the pivotal theoretical result supporting the use of randomized selection for the Kaczmarz method was given by Strohmer and Vershynin [2009]. They proved a simple non-asymptotic linear convergence rate (in expectation) in terms of the number of iterations, when rows are selected proportional to their squared norms. This work spurred numerous extensions and generalizations of the randomized Kaczmarz method [Lee and Sidford, 2013, Leventhal and Lewis, 2010, Liu and Wright, 2014, Ma et al., 2015a, Needell, 2010, Zouzias and Freris, 2013], including similar rates

when we replace the equality constraints with inequality constraints.

Rather than cyclic or randomized, in this chapter we consider *greedy* selection rules. There are very few results in the literature that explore the use of greedy selection rules for Kaczmarz-type methods. Griebel and Oswald [2012] present the *maximum residual rule* for multiplicative Schwarz methods, for which the randomized Kaczmarz iteration is a special case. Their theoretical results show similar convergence rate estimates for both greedy and random methods, suggesting there is no advantage of greedy selection over randomized selection (since greedy selection has additional computational costs). Eldar and Needell [2011] propose a greedy *maximum distance rule*, which they approximate using the Johnson-Lindenstrauss [1984] transform to reduce the computation cost. They show that this leads to a faster algorithm in practice, and show that this rule may achieve more progress than random selection on certain iterations.

In the next section, we define several relevant problems of interest in machine learning that can be solved via Kaczmarz methods. Subsequently, we define the greedy selection rules and discuss cases where they can be computed efficiently. In Section 3.3 we give faster convergence rate analyses for both the maximum residual rule and the maximum distance rule, which clarify the relationship of these rules to random selection and show that greedy methods will typically have better convergence rates than randomized selection. Section 3.4 contrasts Kaczmarz methods with coordinate descent methods, Section 3.5 considers a simplified setting where we explicitly compute the constants in the convergence rates, Section 3.6 considers how these convergence rates are changed under *approximations* to the greedy rules, and Section 3.7 discusses the case of inequality constraints. We also propose provably-faster randomized selection rules for matrices A with pairwise-orthogonal rows by using the so-called "orthogonality graph" (Section 3.8). Finally, in Section 3.9 we present numerical experiments evaluating greedy Kaczmarz methods.

3.1 Problems of Interest

We first consider systems of linear equations,

$$Ax = b, \tag{3.1}$$

where A is an $m \times n$ matrix and $b \in \mathbb{R}^m$. We assume the system is *consistent*, meaning a solution x^* exists. We denote the rows of A by $a_1^\top, \ldots, a_m^\top$, where each $a_i \in R^n$ and all rows have at least one non-zero entry, and use $b = (b_1, \ldots, b_m)^\top$, where each $b_i \in \mathbb{R}$. One of the most important examples of a consistent linear system, and a fundamental model in machine learning, is the least squares problem,

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2.$$

An appealing way to write a least squares problem as a linear system is to solve the $(n+m)$-variable consistent system [see also Zouzias and Freris, 2013]

$$\begin{pmatrix} A & -I \\ \mathbf{0} & A^T \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

Other applications in machine learning that involve solving consistent linear systems include: least-squares support vector machines, Gaussian processes, fitting the final layer of a neural network (using squared-error), graph-based semi-supervised learning or other graph-Laplacian problems [Bengio et al., 2006], and finding the optimal configuration in Gaussian Markov random fields [Rue and Held, 2005].

Kaczmarz methods can also be applied to solve consistent systems of linear *inequalities*,

$$Ax \leq b,$$

or combinations of linear equalities and inequalities. We believe there is a lot potential to use this application of Kaczmarz methods in machine learning. Indeed, a classic example of solving linear inequalities is finding a linear separator for a binary classification problem. The classic perceptron algorithm is a generalization of the Kaczmarz method, but unlike the classic sublinear rates of perceptron methods [Novikoff, 1962] we can show a linear rate for the Kaczmarz method.

Kaczmarz methods could also be used to solve the ℓ_1-regularized robust regression problem,

$$\min_x f(x) := \|Ax - b\|_1 + \lambda\|x\|_1,$$

for $\lambda \geq 0$. We can formulate finding an x with $f(x) \leq \tau$ for some constant τ as a set of linear inequalities. By doing a binary search for τ and using *warm-starting*, this can be substantially faster than existing approaches like stochastic subgradient methods (which have a sublinear convergence rate) or formulating as a linear program (which is not scaleable due to the super-linear cost). The above logic applies to many piecewise-linear problems in machine learning like variants of support vector machines/regression with the ℓ_1-norm, regression under the ℓ_∞-norm, and linear programming relaxations for decoding in graphical models.

3.2 Kaczmarz Algorithm and Greedy Selection Rules

The Kaczmarz algorithm for solving linear systems begins from an initial guess x^0, and each iteration k chooses a row i_k and projects the current iterate x^k onto the hyperplane defined by $a_{i_k}^T x^k = b_{i_k}$. This gives the iteration

$$x^{k+1} = x^k + \frac{b_{i_k} - a_{i_k}^T x^k}{\|a_{i_k}\|^2} a_{i_k}, \tag{3.2}$$

and the algorithm converges to a solution x^* under weak conditions (e.g., each i is visited infinitely often). We consider two greedy selection rules: the *maximum residual* rule and the *maximum distance* rule. The maximum residual (MR) rule selects i_k according to

$$i_k \in \underset{i}{\operatorname{argmax}} \ |a_i^T x^k - b_i|, \tag{3.3}$$

which is the equation i_k that is 'furthest' from being satisfied. The maximum distance (MD) rule selects i_k according to

$$i_k \in \underset{i}{\operatorname{argmax}} \ \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|, \tag{3.4}$$

which is the rule that maximizes the distance between iterations, $\|x^{k+1} - x^k\|$.

3.2.1 Efficient Calculations for Sparse A

In general, computing these greedy selection rules exactly is too computationally expensive, but in some applications we can compute them efficiently. For example, consider a *sparse* A with at most c non-zeros per column and at most r non-zeros per row. In this setting, we show in Appendix B.1 that using a *max-heap structure* both rules can be computed exactly in $O(cr \log m)$ time. We show a simple example of this process in Figure 3.1 for a matrix with the following sparsity pattern,

$$A = \begin{pmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & 0 & * & 0 \\ 0 & 0 & 0 & 0 & * \end{pmatrix}.$$

We exploit the fact that projecting onto row i does not change the residual of row j if a_i and a_j do not share a non-zero index.

The above sparsity condition guarantees that row i is orthogonal to row j, and indeed projecting onto row i will not change the residual of row j under the more general condition that a_i and a_j are orthogonal. Consider what we call the *orthogonality graph*: an undirected graph on m nodes where we place on edge between nodes i and j if a_i is not orthogonal to a_j. Given this graph, to update all residuals after we update a row i we only need to update the neighbours of node i in this graph. Even if A is dense ($r = n$ and $c = m$), if the maximum number of neighbours is g, then tracking the maximum residual costs $O(gr + g\log(m))$. If g is small, this could still be comparable to the $O(r + \log(m))$ cost of using existing randomized selection strategies.

3.2.2 Approximate Calculation

Many applications, particularly those arising from graphical models with a simple structure, will allow efficient calculation of the greedy rules using the method of the previous section. However,

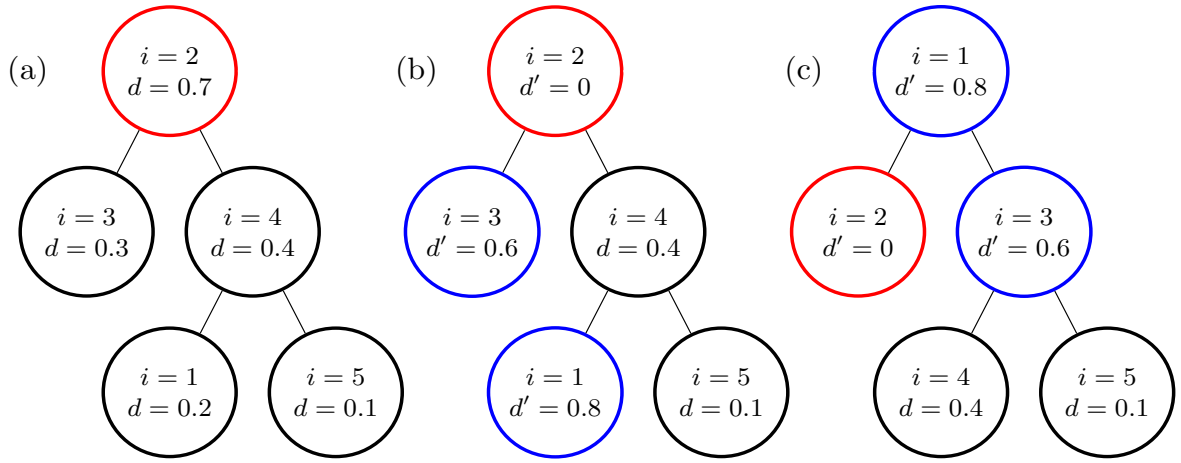


Figure 3.1: Example of the updating procedure for a max-heap structure on a 5×5 sparse matrix: (a) select the node with highest d value; (b) update selected sample and neighbours; (c) reorder max-heap structure.

in other applications it will be too inefficient to calculate the greedy rules. Nevertheless, Eldar and Needell [2011] show that it is possible to efficiently select an i_k that *approximates* the greedy rules by making use of the dimensionality reduction technique of Johnson and Lindenstrauss [1984]. Their experiments show that approximate greedy rules can be sufficiently accurate and that they still outperform random selection. After first analyzing exact greedy rules in the next section, we analyze the effect of using approximate rules in Section 3.6.

3.3 Analyzing Selection Rules

All the convergence rates we discuss use a relationship between the terms $\|x^{k+1} - x^*\|$ and $\|x^k - x^*\|$. To derive this relationship, we consider the following expansion of $\|x^k - x^*\|^2$:

$$\begin{aligned}\|x^k - x^*\|^2 &= \|x^k - x^{k+1} + x^{k+1} - x^*\|^2 \\ &= \|x^k - x^{k+1}\|^2 + \|x^{k+1} - x^*\|^2 - 2\langle x^{k+1} - x^k, x^{k+1} - x^*\rangle.\end{aligned}$$

Rearranging, we obtain

$$\|x^{k+1} - x^*\|^2 = \|x^k - x^*\|^2 - \|x^{k+1} - x^k\|^2 + 2\langle x^{k+1} - x^k, x^{k+1} - x^*\rangle. \tag{3.5}$$

Given the Kaczmarz update (3.2), we can say that $x^{k+1} - x^k = \gamma^k a_{i_k}$ for some scalar γ^k and selected i_k. Then the last term in (3.5) equals 0, as shown by the following argument,

$$\begin{aligned}(x^{k+1} - x^k)^T(x^{k+1} - x^*) &= \gamma a_{i_k}^T(x^{k+1} - x^*) \\ &= \gamma(a_{i_k}^T x^{k+1} - a_{i_k}^T x^*) \\ &= \gamma(b_{i_k} - b_{i_k}) \\ &= 0,\end{aligned}$$

where the last equality follows from the fact that both x^{k+1} and x^* solve the equality $a_{i_k}^T \cdot = b_{i_k}$. This proves that $(x^{k+1} - x^k)$ is orthogonal to $(x^{k+1} - x^*)$ (see Figure 3.2). Then using the

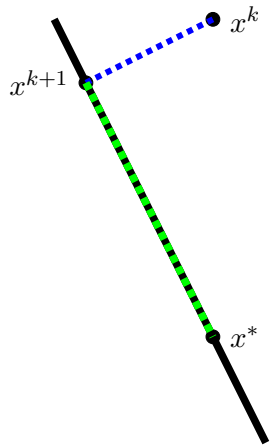


Figure 3.2: Visualizing the orthogonality of vectors $x^{k+1} - x^k$ and $x^{k+1} - x^*$.

definition of x^{k+1} from (3.2) and simplifying (3.5), we obtain

$$\|x^{k+1} - x^*\|^2 = \|x^k - x^*\|^2 - \frac{\left(a_{i_k}^T x^k - b_{i_k}\right)^2}{\|a_{i_k}\|^2}. \tag{3.6}$$

3.3.1 Randomized and Maximum Residual

We first give an analysis of the Kaczmarz method with *uniform* random selection of the row to update i (which we abbreviate as 'U'). Conditioning on the σ-field $\mathcal{F}_k$ generated by the sequence $\{x^0, x^1, \ldots, x^k\}$, and taking expectations of both sides of (3.6), when i_k is selected using U we

obtain

$$\begin{aligned}
\mathbb{E}[\|x^{k+1} - x^*\|^2] &= \|x^k - x^*\|^2 - \mathbb{E}\left[\frac{\left(a_i^T x^k - b_i\right)^2}{\|a_i\|^2}\right] \\
&= \|x^k - x^*\|^2 - \sum_{i=1}^{m} \frac{1}{m} \frac{(a_i^\top (x^k - x^*))^2}{\|a_i\|^2} \\
&\leq \|x^k - x^*\|^2 - \frac{1}{m\|A\|_{\infty,2}^2} \sum_{i=1}^{m} (a_i^\top (x^k - x^*))^2 \\
&= \|x^k - x^*\|^2 - \frac{1}{m\|A\|_{\infty,2}^2} \|A(x^k - x^*)\|^2 \\
&\leq \left(1 - \frac{\sigma(A,2)^2}{m\|A\|_{\infty,2}^2}\right) \|x^k - x^*\|^2, \qquad (3.7)
\end{aligned}$$

where $\|A\|_{\infty,2}^2 := \max_i\{\|a_i\|^2\}$ and $\sigma(A,2)$ is the Hoffman [1952] constant.[5] We have assumed that x^k is not a solution, allowing us to use Hoffman's bound (the inequality is trivially satisfied if x^k is a solution to $Ax = b$). When A has independent columns, $\sigma(A,2)$ is the nth singular value of A and in general it is the smallest non-zero singular value.

The argument above is related to the analysis of Vishnoi [2013] but is simpler due to the use of the Hoffman bound. Further, this simple argument makes it straightforward to derive bounds on other rules. For example, we can derive the convergence rate bound of Strohmer and Vershynin [2009] by following the above steps but selecting i non-uniformly with probability $\|a_i\|^2/\|A\|_F^2$ (where $\|A\|_F$ is the Frobenius norm of A). We review these steps in Appendix B.2, showing that this non-uniform (NU) selection strategy has

$$\mathbb{E}[\|x^{k+1} - x^*\|^2] \leq \left(1 - \frac{\sigma(A,2)^2}{\|A\|_F^2}\right) \|x^k - x^*\|^2. \qquad (3.8)$$

This strategy requires prior knowledge of the row norms of A, but this is a one-time computation and can be reused for any linear system involving A. Because $\|A\|_F^2 \leq m\|A\|_{\infty,2}^2$, the NU rate (3.8) is at least as fast as the uniform rate (3.7).

While a trivial analysis shows that the MR rule also satisfies (3.7) in a deterministic sense, in Appendix B.2 we give a tighter analysis of the MR rule showing it has the convergence rate

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - \frac{\sigma(A,\infty)^2}{\|A\|_{\infty,2}^2}\right) \|x^k - x^*\|^2, \qquad (3.9)$$

where the Hoffman-like constant $\sigma(A,\infty)$ satisfies the relationship

$$\frac{\sigma(A,2)}{\sqrt{m}} \leq \sigma(A,\infty) \leq \sigma(A,2).$$

[5] In this work, any reference to the Hoffman constant is in fact the *inverse* of the Hoffman constant defined by Hoffman [1952].

Thus, at one extreme the maximum residual rule obtains the same rate as (3.7) for uniform selection when $\sigma(A,2)^2/m \approx \sigma(A,\infty)^2$. However, at the other extreme the maximum residual rule could be faster than uniform selection by a factor of m ($\sigma(A,\infty)^2 \approx \sigma(A,2)^2$). Thus, although the uniform and MR bounds are the same in the worst case, the MR rule can be superior by a large margin.

In contrast to comparing U and MR, the MR rate may be faster or slower than the NU rate. This is because

$$\|A\|_{\infty,2} \leq \|A\|_F \leq \sqrt{m}\|A\|_{\infty,2},$$

so these quantities and the relationship between $\sigma(A,2)$ and $\sigma(A,\infty)$ influence which bound is tighter.

3.3.2 Tighter Uniform and MR Analysis

In our derivations of rates (3.7) and (3.9), we use the inequality

$$\|a_i\|^2 \leq \|A\|_{\infty,2}^2 \quad \forall\, i, \tag{3.10}$$

which leads to a simple result but could be very loose if the range of the row norms is large. In this section, we give tighter analyses of the U and MR rules that are less interpretable but are tighter because they avoid this inequality.

In order to avoid using this inequality for our analysis of U, we can absorb the row norms of A into a row weighting matrix D, where $D = \text{diag}(\|a_1\|, \|a_2\|, \ldots, \|a_m\|)$. Defining $\bar{A} := D^{-1}A$, we show in Appendix B.3 that this results in the following upper bound on the convergence rate for uniform random selection,

$$\mathbb{E}[\|x^{k+1} - x^*\|^2] \leq \left(1 - \frac{\sigma(\bar{A},2)^2}{m}\right)\|x^k - x^*\|^2. \tag{3.11}$$

A similar result is given by Needell et al. [2013] under the stronger assumption that A has independent columns. The rate in (3.11) is tighter than (3.7), since $\sigma(A,2)/\|A\|_{\infty,2} \leq \sigma(\bar{A},2)$ [van der Sluis, 1969]. Further, *this rate can be faster than the non-uniform sampling method* of Strohmer and Vershynin [2009]. For example, suppose row i is orthogonal to all other rows but has a significantly larger row norm than all other row norms. In other words, $\|a_i\| >> \|a_j\|$ for all $j \neq i$. In this case, NU selection will repeatedly select row i (even though it only needs to be selected once), whereas U will only select it on each iteration with probability $1/m$. It has been previously pointed out that Strohmer and Vershynin's method can perform poorly if the problem has one row norm that is significantly larger than the other row norms [Censor et al., 2009]. This result theoretically shows that U can have a tighter bound than the NU method of Strohmer and Vershynin.

In Appendix B.3, we also give a simple modification of our analysis of the MR rule, which

leads to the rate

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - \frac{\sigma(A,\infty)^2}{\|a_{i_k}\|^2}\right)\|x^k - x^*\|^2. \tag{3.12}$$

This bound depends on the *specific* $\|a_{i_k}\|$ corresponding to the i_k selected at each iteration k. This convergence rate will be faster whenever we select an i_k with $\|a_{i_k}\| < \|A\|_{\infty,2}$. However, in the worst case we repeatedly select i_k values with $\|a_{i_k}\| = \|A\|_{\infty,2}$ so there is no improvement. This issue is considered in [Sepehry, 2016], where the authors give tighter bounds on the *sequence* of $\|a_{i_k}\|$ values for problems with sparse orthogonality graphs.

3.3.3 Maximum Distance Rule

If we can only perform one iteration of the Kaczmarz method, the *optimal* rule (with respect to iteration progress) is in fact the MD rule. In Appendix B.4, we show that this strategy achieves a rate of

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - \sigma(\bar{A},\infty)^2\right)\|x^k - x^*\|^2, \tag{3.13}$$

where $\sigma(\bar{A},\infty)$ satisfies

$$\max\left\{\frac{\sigma(\bar{A},2)}{\sqrt{m}}, \frac{\sigma(A,2)}{\|A\|_F}, \frac{\sigma(A,\infty)}{\|A\|_{\infty,2}}\right\} \leq \sigma(\bar{A},\infty) \leq \sigma(\bar{A},2).$$

Thus, the maximum distance rule is at least as fast as the fastest among the U/NU/MR_∞ rules, where MR_∞ refers to rate (3.9). Further, in Appendix B.9 we show that this new rate is not only simpler but is strictly tighter than the rate reported by Eldar and Needell [2011] for the exact MD rule. In Table 3.1, we summarize the relationships we have discussed in this section

Table 3.1: Comparison of Convergence Rates

	U_∞	U	NU	MR_∞	MR	MD
U_∞	$=$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
U		$=$	P	P	P	$\leq$
NU			$=$	P	P	$\leq$
MR_∞				$=$	$\leq$	$\leq$
MR					$=$	$\leq$
MD						$=$

among the different selection rules. We use the following abbreviations: U_∞ - uniform (3.7), U - tight uniform (3.11), NU - non-uniform (3.8), MR_∞ - maximum residual (3.9), MR - tight maximum residual (3.12) and MD - maximum distance (3.13). The inequality sign ($\leq$) indicates that the bound for the selection rule listed in the row is slower or equal to the rule listed in the column, while we have written 'P' to indicate that the faster method is problem-dependent.

3.4 Kaczmarz and Coordinate Descent

With the exception of the tighter U and MR rate, the results of the previous section are analogous to the results in Chapter 2 for coordinate descent methods. Indeed, if we apply coordinate descent methods to minimize the squared error between Ax and b then we obtain similar-looking rates and analogous conclusions. With cyclic selection this is called the Gauss-Seidel method [Seidel, 1874], and as discussed by Ma et al. [2015a] there are several connections/differences between this method and Kaczmarz methods. In this section we highlight some key differences.

First, the previous work required strong convexity which would require that A has independent columns. This is often unrealistic, and our results from the previous section hold for any A.[6] Second, here our results are in terms of the iterates $\|x^k - x^*\|$, which is the natural measure for linear systems. The coordinate descent results are in terms of $f(x^k) - f(x^*)$ and although it is possible to use strong convexity to turn this into a rate on $\|x^k - x^*\|$, this would result in a looser bound and would again require strong convexity to hold (see Ma et al. [2015a]). On the other hand, coordinate descent gives the least squares solution for inconsistent systems. However, this is also true of the Kaczmarz method using the formulation in Section 3.1. Another subtle issue is that the Kaczmarz rates depend on the row norms of A while the coordinate descent rates depend on the column norms. Thus, there are scenarios where we expect Kaczmarz methods to be much faster and vice versa. Finally, we note that Kaczmarz methods can be extended to allow inequality constraints (see Section 3.7).

As discussed by Wright [2015], Kaczmarz methods can also be interpreted as coordinate descent methods on the dual problem

$$\min_y \frac{1}{2}\|A^T y\|^2 - b^T y, \tag{3.14}$$

where $x = A^T y^*$ so that $Ax = AA^T y^* = b$. Applying the Gauss-Southwell rule in this setting yields the MR rule while applying the Gauss-Southwell-Lipschitz rule yields the MD rule (see Appendix B.5 for details and numerical comparisons, indicating that in some cases Kaczmarz substantially outperforms CD). However, applying the analysis of Chapter 2 to this dual problem would require that A has independent rows and would only yield a rate on the dual objective, unlike the convergence rates in terms of $\|x^k - x^*\|$ that hold for general A from the previous section. In Chapter 4 we revisit the strong-convexity assumption on CD methods and show that, in fact, it is not a necessary assumption to guarantee a linear convergence rate.

3.5 Example: Diagonal A

To give a concrete example of these rates, we consider the simple case of a diagonal A. While such problems are not particularly interesting, this case provides a simple setting to understand

[6] In Chapter 4 we show that the results of Chapter 2 apply for general least squares problems.

these different rates without referring to Hoffman bounds.

Consider a square diagonal matrix A with $a_{ii} > 0$ for all i. In this case, the diagonal entries are the eigenvalues λ_i of the linear system. The convergence rate constants for this scenario are given in Table 3.2. We provide the details in Appendix B.6 of the derivations for $\sigma(A, \infty)$

Table 3.2: Convergence Rate Constants for Diagonal A

U_∞	$\left(1 - \frac{\lambda_m^2}{m\lambda_1^2}\right)$
U	$\left(1 - \frac{1}{m}\right)$
NU	$\left(1 - \frac{\lambda_m^2}{\sum_i \lambda_i^2}\right)$
MR_∞	$\left(1 - \frac{1}{\lambda_1^2}\left[\sum_i \frac{1}{\lambda_i^2}\right]^{-1}\right)$
MR	$\left(1 - \frac{1}{\lambda_{i_k}^2}\left[\sum_i \frac{1}{\lambda_i^2}\right]^{-1}\right)$
MD	$\left(1 - \frac{1}{m}\right)$

and $\sigma(\bar{A}, \infty)$, as well as substitutions for the uniform, non-uniform, and uniform tight rates to yield the above table. We note that the uniform tight rate follows from $\lambda_m^2(\bar{A})$ being equivalent to the minimum eigenvalue of the identity matrix.

If we consider the most basic case when all the eigenvalues of A are equal, then all the selection rules yield the same rate of $(1 - 1/m)$ and the method converges in at most m steps for greedy selection rules and in at most $O(m \log m)$ steps (in expectation) for the random rules (due to the 'coupon collector' problem). Further, this is the worst situation for the greedy MR and MD rules since they satisfy their lower bounds on $\sigma(A, \infty)$ and $\sigma(\bar{A}, \infty)$.

Now consider the extreme case when all the eigenvalues are equal except for one. For example, consider when $\lambda_1 = \lambda_2 = \cdots = \lambda_{m-1} > \lambda_m$ with $m > 2$. Letting $\alpha = \lambda_i^2(A)$ for any $i = 1, \ldots, m-1$ and $\beta = \lambda_m^2(A)$, we have

$$\underbrace{\frac{\beta}{m\alpha}}_{\text{U}_\infty} < \underbrace{\frac{\beta}{\alpha(m-1)+\beta}}_{\text{NU}} < \underbrace{\frac{\beta}{\alpha+\beta(m-1)}}_{\text{MR}_\infty} \leq \underbrace{\frac{1}{\lambda_{i_k}^2}\frac{\alpha\beta}{\alpha+\beta(m-1)}}_{\text{MR}} < \underbrace{\frac{1}{m}}_{\text{U, MD}}.$$

Thus, Strohmer and Vershynin's NU rule would actually be the worst rule to use, whereas U and MD are the best. In this case $\sigma(A, \infty)^2$ is closer to its upper bound ($\approx \beta$) so we would expect greedy rules to perform well.

3.6 Approximate Greedy Rules

In many applications computing the exact MR or MD rule will be too inefficient, but we can always approximate it using a cheaper *approximate* greedy rule, as in the method of Eldar and Needell [2011]. In this section we consider methods that compute the greedy rules up to multiplicative or additive errors.

3.6.1 Multiplicative Error

Suppose we have approximated the MR rule such that there is a multiplicative error in our selection of i_k,

$$|a_{i_k}^T x^k - b_{i_k}| \geq \max_i |a_i^T x^k - b_i|(1-\epsilon_k),$$

for some $\epsilon_k \in [0,1)$. In this scenario, using the tight analysis for the MR rule, we show in Appendix B.7 that

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - \frac{(1-\epsilon_k)^2\sigma(A,\infty)^2}{\|a_{i_k}\|^2}\right)\|x^k - x^*\|^2.$$

Similarly, if we approximate the MD rule up to a multiplicative error,

$$\left|\frac{a_{i_k}^T x^k - b_{i_k}}{\|a_{i_k}\|}\right| \geq \max_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|(1-\bar{\epsilon}_k),$$

for some $\bar{\epsilon}_k \in [0,1)$, then we show in Appendix B.7 that the following rate holds,

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - (1-\bar{\epsilon}_k)^2\sigma(\bar{A},\infty)^2\right)\|x^k - x^*\|^2.$$

These scenarios do not require the error to converge to 0. However, if ϵ_k or $\bar{\epsilon}_k$ is large, then the convergence rate will be slow.

3.6.2 Additive Error

Suppose we select i_k using the MR rule up to additive error,

$$|a_{i_k}^T x^k - b_{i_k}|^2 \geq \max_i |a_i^T x^k - b_i|^2 - \epsilon_k,$$

or similarly for the MD rule,

$$\left|\frac{a_{i_k}^T x^k - b_{i_k}}{\|a_{i_k}\|}\right|^2 \geq \max_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|^2 - \bar{\epsilon}_k,$$

for some $\epsilon_k \geq 0$ or $\bar{\epsilon}_k \geq 0$, respectively. We show in Appendix B.8 that this results in the following convergence rates for the MR and MD rules with additive error (respectively),

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - \frac{\sigma(A,\infty)^2}{\|a_{i_k}\|^2}\right)\|x^k - x^*\|^2 + \frac{\epsilon_k}{\|a_{i_k}\|^2},$$

and

$$\|x^{k+1} - x^*\|^2 \leq \left(1 - \sigma(\bar{A},\infty)^2\right)\|x^k - x^*\|^2 + \bar{\epsilon_k}.$$

With an additive error, we need the errors to go to 0 in order for the algorithm to converge; if it does go to 0 fast enough, we obtain the same rate as if we were calculating the exact greedy rule. In the approximate greedy rule used by Eldar and Needell [2011], there is unfortunately a constant additive error. To address this, they compare the approximate greedy selection to a randomly selected i_k and take the one with the largest distance. This approach can be substantially faster when far from the solution, but may eventually revert to random selection. We give details comparing Eldar and Needell's rate to our above rate in Appendix B.9, but here we note that the above bounds will typically be much stronger.

3.7 Systems of Linear Inequalities

Kaczmarz methods have been extended to systems of linear inequalities,

$$\begin{cases} a_i^T x \leq b_i & (i \in I_{\leq}) \\ a_i^T x = b_i & (i \in I_{=}). \end{cases} \tag{3.15}$$

where the disjoint index sets $I_{\leq}$ and $I_{=}$ partition the set $\{1, 2, \ldots, m\}$ [Leventhal and Lewis, 2010]. In this setting the method takes the form

$$x^{k+1} = x^k - \frac{\beta^k}{\|a_i\|^2}a_i, \quad \text{with} \quad \beta^k = \begin{cases} (a_i^T x^k - b_i)^+ & (i \in I_{\leq}) \\ a_i^T x^k - b_i & (i \in I_{=}), \end{cases}$$

where $(\gamma)^+ = \max\{\gamma, 0\}$. In Appendix B.10 we derive analogous greedy rules and convergence results for this case. The main difference in this setting is that the rates are in terms of the distance of x^k to the feasible set S of (3.15),

$$d(x^k, S) = \min_{z \in S} \|x^k - z\|_2 = \|x^k - P_S(x^k)\|_2,$$

where $P_S(x)$ is the projection of x onto S. This generalization is needed because with inequality constraints the different iterates x^k may have different projections onto S.

3.8 Faster Randomized Kaczmarz Methods

In this section we use the orthogonality graph presented in Section 3.2.1 to design new selection rules and derive faster convergence rates for randomized Kaczmarz methods. Similar to the classic convergence rate analyses of cyclic Kaczmarz algorithms, these new rates/rules depend in some sense on the 'angle' between rows, which is a property that is not captured by existing randomized/greedy analyses (only depend on the row norms).

If two rows a_i and a_j are orthogonal, then if the equality $a_i^T x^k = b_i$ holds at iteration x^k and we select $i_k = j$, then we know that $a_i^T x^{k+1} = b_i$. More generally, updating i_k makes equality i_k satisfied but could make any equality j unsatisfied where a_j is not orthogonal to a_{i_k}. Thus, after we have selected row i_k, equation i_k will remain satisfied for all subsequent iterations until one of its neighbours is selected in the orthogonality graph.[7]

Based on this, we call a row i 'selectable' if i has never been selected or if a neighbour of i in the orthogonality graph has been selected since the last time i was selected.[8] We use the notation $s_i^k = 1$ to denote that row i is 'selectable' on iteration k, and otherwise we use $s_i^k = 0$ and say that i is 'not selectable' at iteration k. There is no reason to ever update a 'not selectable' row, because by definition the equality is already satisfied. Based on this, we propose two simple randomized schemes:

1. **Adaptive Uniform**: select i_k uniformly from the selectable rows.

2. **Adaptive Non-Uniform**: select i_k proportional to $\|a_i\|^2$ among the selectable rows.

Let $A_k/\bar{A}_k$ denote the sub-matrix of $A/\bar{A}$ formed by concatenating the selectable rows on iteration k, and let m_k denote the number of selectable rows. If we are given the set of selectable nodes at iteration k, then for adaptive uniform we obtain the bound

$$\mathbb{E}[\|x^{k+1} - x^*\|^2] \leq \left(1 - \frac{\sigma(\bar{A}_k, 2)^2}{m_k}\right)\|x^k - x^*\|^2,$$

while for adaptive non-uniform we obtain the bound

$$\mathbb{E}[\|x^{k+1} - x^*\|^2] \leq \left(1 - \frac{\sigma(A_k, 2)^2}{\|A_k\|_F^2}\right)\|x^k - x^*\|^2.$$

If we are not on the first iteration, then at least one node is not selectable and these are strictly faster than the previous bounds. The gain will be small if most nodes are selectable (which would be typical of dense orthogonality graphs), but the gain can be very large if only a few nodes are selectable (which would be typical of sparse orthogonality graphs).

Practical Issues: In order for the adaptive methods to be efficient, we must be able to efficiently form the orthogonality graph and update the set of selectable nodes. If each node

[7] Although we only consider randomized Kaczmarz methods in this section, [Sepehry, 2016] uses the orthogonality graph to derive a tighter *multi-step* analysis for the MR rule.

[8] If we initialize with $x^0 = 0$, then instead of considering all nodes as initially selectable we can restrict to the nodes i with $b_i \neq 0$ since otherwise we have $a_i^T x^0 = b_i$ already.

has at most g neighbours in the orthogonality graph, then the cost of updating the set of selectable nodes and then sampling from the set of selectable nodes is $O(g\log(m))$ (we give details in Appendix B.11). In order for this to not increase the iteration cost compared to the NU method, we only require the very-reasonable assumption that $g\log(m) = O(n + \log(m))$. In many applications where orthogonality is present, g will be far smaller than this.

However, forming the orthogonality graph at the start may be prohibitive: it would cost $O(m^2n)$ in the worst case to test pairwise orthogonality of all nodes. In the sparse case where each column has at most c non-zeros, we can find an approximation to the orthogonality graph in $O(c^2n)$ by assuming that all rows which share a non-zero are non-orthogonal. Alternately, in many applications the orthogonality graph is easily derived from the structure of the problem. For example, in graph-based semi-supervised learning where the graph is constructed based on the k-nearest neighbours, the orthogonality graph will simply be the given k-nearest neighbour graph as these correspond to the columns that will be mutually non-zero in A.

3.9 Experiments

Eldar and Needell [2011] have already shown that approximate greedy rules can outperform randomized rules for dense problems. Thus, in our experiments we focus on comparing the effectiveness of different rules on very sparse problems where our max-heap strategy allows us to efficiently compute the exact greedy rules. The first problem we consider is generating a dataset A with a 50 by 50 lattice-structured dependency (giving $n = 2500$). The corresponding A has the following non-zero elements: the diagonal elements $A_{i,i}$, the upper/lower diagonal elements $A_{i,i+1}$ and $A_{i+1,i}$ when i is not a multiple of 50 (horizontal edges), and the diagonal bands $A_{i,i+50}$ and $A_{i+50,i}$ (vertical edges). We generate these non-zero elements from a $\mathcal{N}(0,1)$ distribution and generate the target vector $b = Az$ using $z \sim \mathcal{N}(0, I)$. Each row in this problem has at most four neighbours, and this type of sparsity structure is typical of spatial Gaussian graphical models and linear systems that arise from discretizing two-dimensional partial differential equations.

The second problem we consider is solving an overdetermined consistent linear system with a very sparse A of size 2500×1000. We generate each row of A independently such that there are $\log(m)/2m$ non-zero entries per row drawn from a uniform distribution between 0 and 1. To explore how having different row norms affects the performance of the selection rules, we randomly multiply one out of every 11 rows by a factor of 10,000.

For the third problem, we solve a label propagation problem for semi-supervised learning in the 'two moons' dataset [Zhou et al., 2003]. From this dataset, we generate 2000 samples and randomly label 100 points in the data. We then connect each node to its 5 nearest neighbours. Constructing a data set with such a high sparsity level is typical of graph-based methods for semi-supervised learning. We use a variant of the quadratic labelling criterion of Bengio et al.

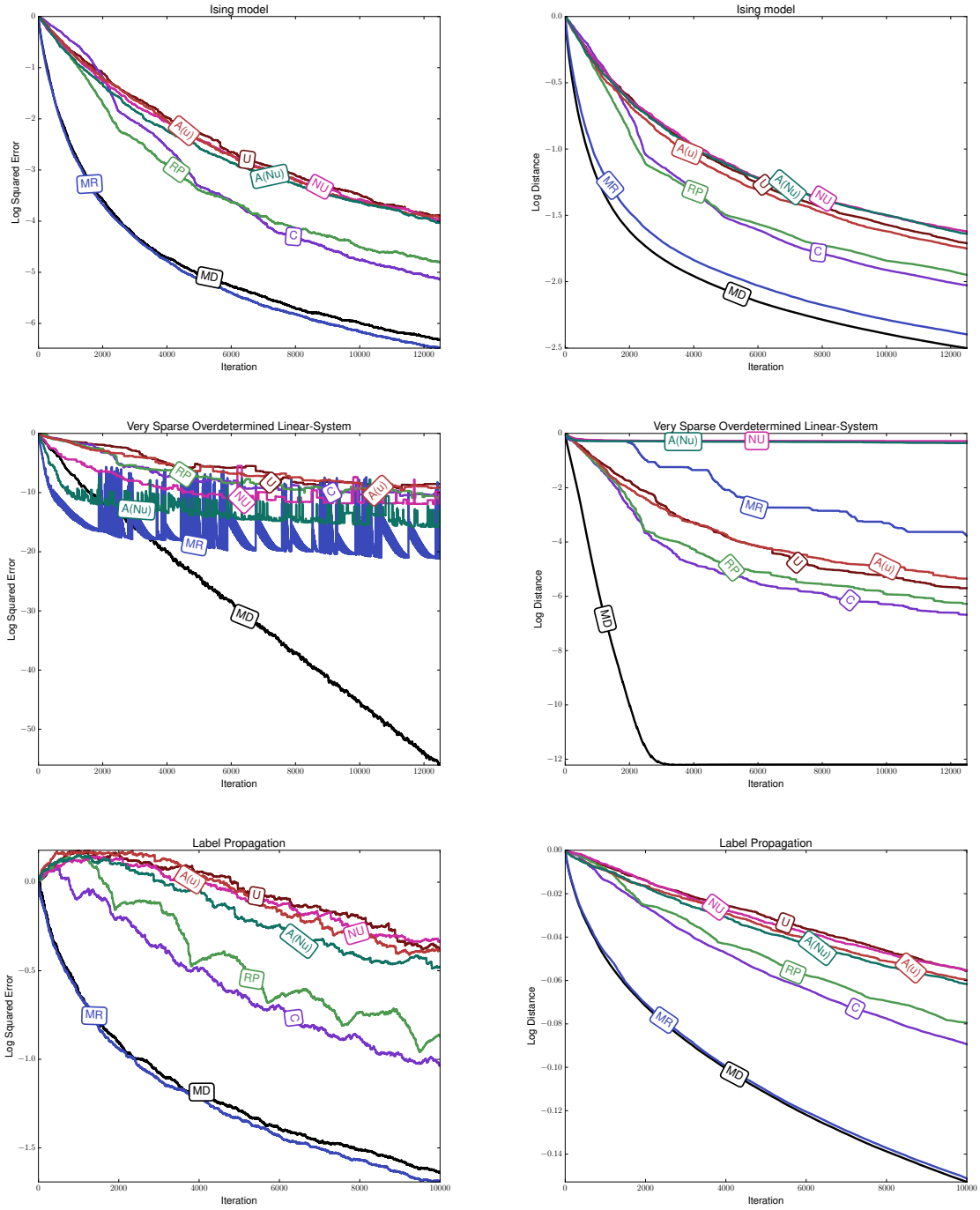


Figure 3.3: Comparison of Kaczmarz selection rules for squared error (left) and distance to solution (right).

[2006],

$$\min_{y_i | i \notin S} \quad \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} (y_i - y_j)^2,$$

where y is our label vector (each y_i can take one of 2 values), S is the set of labels that we do know and $w_{ij} \geq 0$ are the weights assigned to each y_i describing how strongly we want the label y_i and y_j to be similar. We can express this quadratic problem as a linear system that is consistent by construction (see Appendix B.12), and hence apply Kaczmarz methods. As we labelled 100 points in our data, the resulting linear system has a matrix of size 1900×1900 while the number of neighbours g in the orthogonality graph is at most 5.

In Figure 3.3 we compare the normalized squared error and distance against the iteration number for 8 different selection rules: cyclic (C), random permutation (RP - where we change the cycle order after each pass through the rows), uniform random (U), adaptive uniform random (A(u)), non-uniform random NU, adaptive non-uniform random (A(Nu)), maximum residual (MR), and maximum distance (MD).

In experiments 1 and 3, MR performs similarly to MD and both outperform all other selection rules. For experiment 2, the MD rule outperforms all other selection rules in terms of distance to the solution although MR performs better on the early iterations in terms of squared error. In Appendix B.12 we explore a 'hybrid' method on the overdetermined linear system problem that does well on both measures. For runtime results on these experiments, see Nutini et al. [2016, Appendix M].

The new randomized A(u) method did not give significantly better performance than the existing U method on any dataset. This agrees with our bounds which show that the gain of this strategy is modest. In contrast, the new randomized A(Nu) method performed much better than the existing NU method on the over-determined linear system in terms of squared error. This again agrees with our bounds which suggest that the A(Nu) method has the most to gain when the row norms are very different. Interestingly, in most experiments we found that *cyclic* selection worked better than any of the randomized algorithms. However, cyclic methods were clearly beaten by greedy methods.

3.10 Discussion

In this chapter, we proved faster convergence rate bounds for a variety of row-selection rules in the context of Kaczmarz methods for solving linear systems. We have also provided new randomized selection rules that make use of orthogonality in the data in order to achieve better theoretical and practical performance. While we have focused on the case of non-accelerated and single-variable variants of the Kaczmarz algorithm, we expect that all of our conclusions also hold for accelerated Kaczmarz and block Kaczmarz methods [Gower and Richtárik, 2015, Lee and Sidford, 2013, Liu and Wright, 2014, Needell and Tropp, 2014, Oswald and Zhou, 2015].

Chapter 4

Relaxing Strong Convexity

Fitting most machine learning models involves solving some sort of optimization problem. Gradient descent, and variants of it like coordinate descent and stochastic gradient, are the workhorse tools used by the field to solve very large instances of these problems. In this chapter we consider the basic problem of minimizing a smooth function and the convergence rate of gradient descent methods. It is well-known that if f is strongly convex, then gradient descent achieves a global linear convergence rate for this problem [Nesterov, 2004]. However, many of the fundamental models in machine learning like least squares and logistic regression yield objective functions that are convex but not strongly convex. Further, if f is only convex, then gradient descent only achieves a sub-linear rate.

This situation has motivated a variety of alternatives to strong convexity (SC) in the literature, in order to show that we can obtain linear convergence rates for problems like least squares and logistic regression. One of the oldest of these conditions is the *error bounds* (EB) of Luo and Tseng [1993], but four other recently-considered conditions are *essential strong convexity* (ESC) [Liu et al., 2014], *weak strong convexity* (WSC) [Necoara et al., 2015], the *restricted secant inequality* (RSI) [Zhang and Yin, 2013], and the *quadratic growth* (QG) condition [Anitescu, 2000]. Some of these conditions have different names in the special case of convex functions. For example, a convex function satisfying RSI is said to satisfy *restricted strong convexity* (RSC) [Zhang and Yin, 2013]. Names describing convex functions satisfying QG include *optimal strong convexity* (OSC) [Liu and Wright, 2015], *semi-strong convexity* (SSC) [Gong and Ye, 2014], and (confusingly) WSC [Ma et al., 2015b]. The proofs of linear convergence under all of these relaxations are typically not straightforward, and it is rarely discussed how these conditions relate to each other.

In this work, we consider a much older condition that we refer to as the Polyak-Łojasiewicz (PL) inequality. This inequality was originally introduced by Polyak [1963], who showed that it is a sufficient condition for gradient descent to achieve a linear convergence rate. We describe it as the PL inequality because it is also a special case of the inequality introduced in the same year by Łojasiewicz [1963]. We review the PL inequality in the next section and how it leads to a trivial proof of the linear convergence rate of gradient descent. Next, in terms of showing a global linear convergence rate to the optimal solution, we show that the PL inequality is *weaker* than all of the more recent conditions discussed in the previous paragraph. This suggests that we can replace the long and complicated proofs under any of the conditions above with simpler proofs based on the PL inequality. Subsequently, we show how this result

implies gradient descent achieves linear rates for standard problems in machine learning like least squares and logistic regression that are not necessarily strongly convex, and even for some non-convex problems (Section 4.1.3). In Section 4.2 we use the PL inequality to give new convergence rates for randomized and greedy coordinate descent (implying a new convergence rate for certain variants of boosting) and sign-based gradient descent methods. Next we turn to the problem of minimizing the sum of a smooth function and a simple non-smooth function. We propose a generalization of the PL inequality that allows us to show linear convergence rates for proximal gradient methods without strong convexity. This leads to a simple analysis showing linear convergence of methods for training support vector machines. It also implies that we obtain a linear convergence rate for ℓ_1-regularized least squares problems, showing that the extra conditions previously assumed to derive linear converge rates in this setting are in fact not needed.

4.1 Polyak-Łojasiewicz Inequality

We first focus on the basic unconstrained optimization problem

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} f(x), \tag{4.1}$$

and we assume that the first derivative of f is L-Lipschitz continuous. This means that

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}||y - x||^2, \tag{4.2}$$

for all $x,\ y \in \mathbb{R}^n$. For twice-differentiable objectives this assumption means that the eigenvalues of $\nabla^2 f(x)$ are bounded above by some L, which is typically a reasonable assumption. We also assume that the optimization problem has a non-empty solution set $\mathcal{X}^*$, and we use f^* to denote the corresponding optimal function value. We will say that a function satisfies the PL inequality if the following holds for some $\mu > 0$,

$$\frac{1}{2}||\nabla f(x)||^2 \geq \mu(f(x) - f^*), \quad \forall\ x. \tag{4.3}$$

This inequality requires that the gradient grows faster than a quadratic function as we move away from the optimal function value. Note that this inequality implies that every stationary point is a global minimum. But unlike strong convexity, it does not imply that there is a unique solution. Linear convergence of gradient descent under these assumptions was first proved by Polyak [1963]. Below we give a simple proof of this result when using a step-size of $1/L$.

Theorem 1. *Consider problem* (4.1)*, where f has an L-Lipschitz continuous gradient (4.2), a non-empty solution set $\mathcal{X}^*$, and satisfies the PL inequality (4.3). Then the gradient method*

with a step-size of $1/L$,

$$x^{k+1} = x^k - \frac{1}{L}\nabla f(x^k), \tag{4.4}$$

has a global linear convergence rate,

$$f(x^k) - f^* \leq \left(1 - \frac{\mu}{L}\right)^k (f(x^0) - f^*).$$

Proof. By using update rule (4.4) in the Lipschitz inequality condition (4.2) we have

$$f(x^{k+1}) - f(x^k) \leq -\frac{1}{2L}||\nabla f(x^k)||^2.$$

Now by using the PL inequality (4.3) we get

$$f(x^{k+1}) - f(x^k) \quad \leq \quad -\frac{\mu}{L}(f(x^k) - f^*).$$

Re-arranging and subtracting f^* from both sides gives us $f(x^{k+1}) - f^* \leq \left(1 - \frac{\mu}{L}\right)(f(x^k) - f^*)$, where using the same argument as in [Csiba and Richtárik, 2017, Lem 1] we can see that $(1 - \frac{\mu}{L}) < 1$ since $f(x^k) - f^* \geq 0$ for any $k = 0, 1, \ldots$, and thus it must hold that $\mu \leq L$. Applying the inequality recursively gives the result. □

Note that the above result also holds if we use the optimal step-size at each iteration, because under this choice we have

$$f(x^{k+1}) = \min_{\alpha}\{f(x^k - \alpha\nabla f(x^k))\} \leq f\left(x^k - \frac{1}{L}\nabla f(x^k)\right).$$

A beautiful aspect of this proof is its simplicity; in fact it is *simpler* than the proof of the same fact under the usual strong convexity assumption. It is certainly simpler than typical proofs which rely on the other conditions mentioned at the beginning of this chapter. Further, it is worth noting that the proof does *not* assume convexity of f. Thus, this is one of the few general results we have for global linear convergence on non-convex problems.

4.1.1 Relationships Between Conditions

As mentioned at the beginning of this chapter, several other assumptions have been explored over the last 25 years in order to show that gradient descent achieves a linear convergence rate. We state these conditions next, noting that all of these definitions involve some constant $\mu > 0$ (which may not be the same across conditions). We use the convention that x_p is the projection of x onto the solution set $\mathcal{X}^*$.

1. **Strong Convexity** (SC): For all x and y we have

$$f(y) \geq f(x) + \langle\nabla f(x), y - x\rangle + \frac{\mu}{2}\|y - x\|^2.$$

2. **Essential Strong Convexity** (ESC): For all x and y such that $x_p = y_p$ we have

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2.$$

3. **Weak Strong Convexity** (WSC): For all x we have

$$f^* \geq f(x) + \langle \nabla f(x), x_p - x \rangle + \frac{\mu}{2}\|x_p - x\|^2.$$

4. **Restricted Secant Inequality** (RSI): For all x we have

$$\langle \nabla f(x), x - x_p \rangle \geq \mu\|x_p - x\|^2.$$

   If the function f is also convex it is called **restricted strong convexity** (RSC).

5. **Error Bound** (EB): For all x we have

$$\|\nabla f(x)\| \geq \mu\|x_p - x\|.$$

6. **Polyak-Łojasiewicz** (PL): For all x we have

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu(f(x) - f^*).$$

7. **Quadratic Growth** (QG): For all x we have

$$f(x) - f^* \geq \frac{\mu}{2}\|x_p - x\|^2.$$

   If the function f is also convex it is called **optimal strong convexity** (OSC) or **semi-strong convexity** or sometimes WSC (but we will reserve the expression WSC for the definition above).

These conditions typically assume that f is convex, and lead to more complicated proofs than the one of Theorem 1. However, it is rarely discussed how the conditions relate to each other. Indeed, all of the relationships that have been explored have only been in the context of convex functions [Liu and Wright, 2015, Necoara et al., 2015, Zhang, 2015]. In Figure 4.1 we show how these conditions relate to each other with and without the assumption of convexity, and we formalize these relationships in the following theorem.

Theorem 2. *For a function f with a Lipschitz-continuous gradient, the following implications hold:*

$$(SC) \subset (ESC) \subseteq (WSC) \subseteq (RSI) \subseteq (EB) \equiv (PL) \subseteq (QG).$$

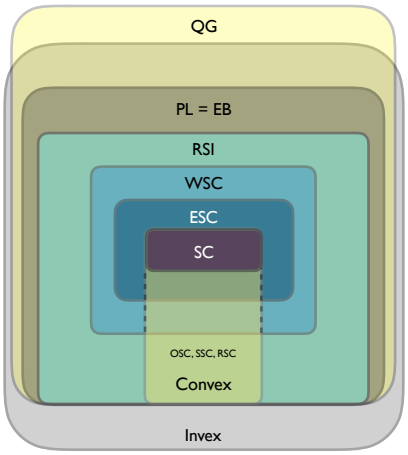


Figure 4.1: Visual of the implications shown in Theorem 2 between the various relaxations of strong convexity.

If we further assume that f is convex then we have

$$(RSI) \equiv (EB) \equiv (PL) \equiv (QG).$$

Proof. See Appendix C.1. □

Note that Zhang [2016] independently also recently gave the relationships between RSI, EB, PL, and QG.[9] This result shows that QG is the weakest assumption among those considered. However, QG allows non-global local minima so it is not enough to guarantee that gradient descent finds a global minimizer. This means that, among those considered above, *PL and the equivalent EB are the most general conditions* that allow linear convergence to a global minimizer. Note that in the convex case QG is called OSC or SSC, but the result above shows that in the convex case it is also equivalent to EB and PL (as well as RSI which is known as RSC in this case).

4.1.2 Invex and Non-Convex Functions

While the PL inequality does not imply convexity of f, it does imply the weaker condition of *invexity*. A function is invex if it is differentiable and there exists a vector valued function η such that for any $x, y \in \mathbb{R}^n$, the following inequality holds

$$f(x) \geq f(y) + \eta(x, y)^T \nabla f(y).$$

It is clear that if $\eta(x, y) = x - y$, then f is convex.

Invexity was first introduced by Hanson [1981], and has been used in the context of learning

[9] Drusvyatskiy and Lewis [2016] is a recent work discussing the relationships among many of these conditions for non-smooth functions.

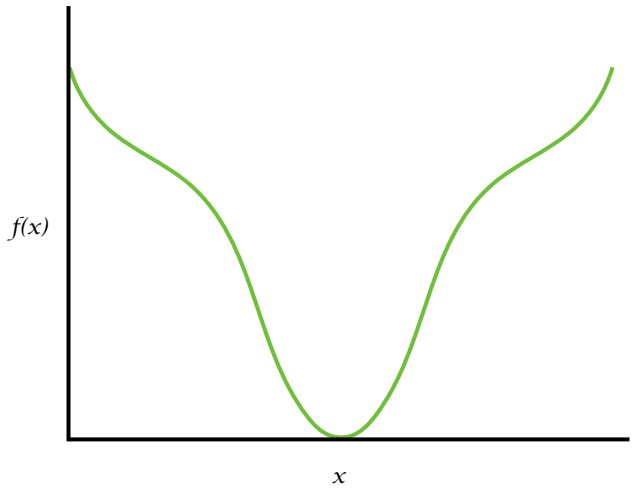


Figure 4.2: Example: $f(x) = x^2 + 3\sin^2(x)$ is an invex but non-convex function that satisfies the PL inequality.

output kernels [Dinuzzo et al., 2011]. Craven and Glover [1985] show that a smooth f is invex if and only if every stationary point of f is a global minimum. Since the PL inequality implies that all stationary points are global minimizers, functions satisfying the PL inequality must be invex. It is easy to see this by noting that at any stationary point $\bar{x}$, we have $\nabla f(\bar{x}) = 0$, so for $\mu > 0$ by the PL inequality, we have

$$0 = \frac{1}{2}\|\nabla f(\bar{x})\|^2 \geq \mu(f(x) - f^*) \geq 0,$$

which means $f(x) = f^*$ and we must be at the global minimum.

Indeed, Theorem 2 shows that all of the previous conditions (except QG) imply invexity. The function $f(x) = x^2 + 3\sin^2(x)$ shown in Figure 4.2 is an example of an invex but non-convex function satisfying the PL inequality (with $\mu = 1/32$). Thus, Theorem 1 implies gradient descent obtains a global linear convergence rate on this function.

Unfortunately, many complicated models have non-optimal stationary points. For example, typical deep feed-forward neural networks have sub-optimal stationary points and are thus not invex. A classic way to analyze functions like this is to consider a *global convergence phase* and a *local convergence phase*. The global convergence phase is the time spent to get "close" to a local minimum, and then once we are "close" to a local minimum the local convergence phase characterizes the convergence rate of the method. Usually, the local convergence phase starts to apply once we are locally strongly convex around the minimizer. But this means that the local convergence phase may be arbitrarily small: for example, for $f(x) = x^2 + 3\sin^2(x)$ the local convergence rate would not even apply over the interval $x \in [-1, 1]$. If we instead defined the local convergence phase in terms of locally satisfying the PL inequality, then we see that it can be *much* larger ($x \in \mathbb{R}$ for this example).

4.1.3 Relevant Problems

If f is μ-strongly convex, then it also satisfies the PL inequality with the same μ (see Appendix C.2). Further, by Theorem 2, f satisfies the PL inequality if it satisfies any of ESC, WSC, RSI, or EB (while for convex f, QG is also sufficient). Although it is hard to precisely characterize the general class of functions for which the PL inequality is satisfied, we note one important special case below.

Strongly convex composed with linear: This is the case where f has the form $f(x) = g(Ax)$ for some σ-strongly convex function g and some matrix A. In Appendix C.2, we use the Hoffman bound (previously used in Chapter 3) to show that this class of functions satisfies the PL inequality, and we note that this form frequently arises in machine learning. For example, least squares problems have the form

$$f(x) = \|Ax - b\|^2,$$

and by noting that $g(z) \triangleq \|z - b\|^2$ is strongly convex we see that least squares falls into this category. Indeed, this class includes all convex quadratic functions.

In the case of logistic regression we have

$$f(x) = \sum_{i=1}^{m} \log(1 + \exp(b_i a_i^T x)).$$

This can be written in the form $g(Ax)$, where g is strictly convex but not strongly convex. In cases like this where g is only strictly convex, the PL inequality will still be satisfied over any compact set. Thus, if the iterations of gradient descent remain bounded, the linear convergence result still applies. It is reasonable to assume that the iterates remain bounded when the set of solutions is finite, since each step must decrease the objective function. Thus, for practical purposes, we can relax the above condition to "strictly-convex composed with linear" and the PL inequality implies a linear convergence rate for logistic regression.

4.2 Convergence of Huge-Scale Methods

In this section, we use the PL inequality to analyze variants of one of the most widely-used techniques for handling large-scale machine learning problems: coordinate descent methods. In particular, the PL inequality yields very simple analyses of this method that apply to more general classes of functions than previously analyzed. We also note that the PL inequality has recently been used by Garber and Hazan [2015] to analyze the Frank-Wolfe algorithm and by Karimi et al. [2016] to analyze stochastic gradient and stochastic variance reduced gradient methods. Further, inspired by the resilient backpropagation (RPROP) algorithm of Riedmiller and Braun [1992], we give a convergence rate analysis for a sign-based gradient descent method.

4.2.1 Randomized Coordinate Descent

Nesterov [2012] shows that randomized coordinate descent achieves a faster convergence rate than gradient descent for problems where we have n variables and it is n times cheaper to update one coordinate than it is to compute the entire gradient. The expected linear convergence rates in this previous work rely on strong convexity, but in this section we show that randomized coordinate descent achieves an expected linear convergence rate if we only assume that the PL inequality holds.

To analyze coordinate descent methods, we assume that the gradient is coordinate-wise Lipschitz continuous, meaning that for any x and y we have

$$f(x + \alpha e_i) \leq f(x) + \alpha \nabla_i f(x) + \frac{L}{2}\alpha^2, \quad \forall \alpha \in \mathbb{R}, \quad \forall x \in \mathbb{R}^n, \tag{4.5}$$

for any coordinate i, and where e_i is the ith unit vector.

Theorem 3. *Consider problem* (4.1)*, where f has a coordinate-wise L-Lipschitz continuous gradient (4.5), a non-empty solution set $\mathcal{X}^*$, and satisfies the PL inequality (4.3). Consider the coordinate descent method with a step-size of $1/L$,*

$$x^{k+1} = x^k - \frac{1}{L}\nabla_{i_k} f(x^k) e_{i_k}. \tag{4.6}$$

If we choose the variable to update i_k uniformly at random, then the algorithm has an expected linear convergence rate of

$$\mathbb{E}[f(x^k) - f^*] \leq \left(1 - \frac{\mu}{nL}\right)^k [f(x^0) - f^*].$$

Proof. By using the update rule (4.6) in the Lipschitz condition (4.5) we have

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L}|\nabla_{i_k} f(x^k)|^2.$$

By taking the expectation of both sides with respect to i_k we have

$$\begin{aligned}
\mathbb{E}\left[f(x^{k+1})\right] &\leq f(x^k) - \frac{1}{2L}\mathbb{E}\left[|\nabla_{i_k} f(x^k)|^2\right] \\
&= f(x^k) - \frac{1}{2L}\sum_i \frac{1}{n}|\nabla_i f(x^k)|^2 \\
&= f(x^k) - \frac{1}{2nL}||\nabla f(x^k)||^2.
\end{aligned}$$

By using the PL inequality (4.3) and subtracting f^* from both sides, we get

$$\mathbb{E}[f(x^{k+1}) - f^*] \leq \left(1 - \frac{\mu}{nL}\right)[f(x^k) - f^*].$$

Applying this recursively and using iterated expectations yields the result. □

As before, instead of using $1/L$ we could perform exact coordinate optimization and the result would still hold. If we have a Lipschitz constant L_i for each coordinate and sample proportional to the L_i as suggested by Nesterov [2012], then the above argument (using a step-size of $1/L_{i_k}$) can be used to show that we obtain a faster rate of

$$\mathbb{E}[f(x^k) - f^*] \leq \left(1 - \frac{\mu}{n\bar{L}}\right)^k [f(x^0) - f^*],$$

where $\bar{L} = \frac{1}{n}\sum_{j=1}^n L_j$.

4.2.2 Greedy Coordinate Descent

In Chapter 2 we analyzed coordinate descent under the greedy Gauss-Southwell (GS) rule, and argued that this rule may be suitable for problems with a large degree of sparsity. We showed that a faster convergence rate can be obtained for the GS rule by measuring strong convexity in the ℓ_1-norm. Since the PL inequality is defined on the dual (gradient) space, in order to derive an analogous result we could measure the PL inequality in the ∞-norm,

$$\frac{1}{2}\|\nabla f(x)\|_\infty^2 \geq \mu_1(f(x) - f^*).$$

Because of the equivalence between norms, this is not introducing any additional assumptions beyond that the PL inequality is satisfied. Further, if f is μ_1-strongly convex in the ℓ_1-norm, then it satisfies the PL inequality in the ∞-norm with the same constant μ_1. By using that $|\nabla_{i_k} f(x^k)| = \|\nabla f(x^k)\|_\infty$ when the GS rule is used, the above argument can be used to show that coordinate descent with the GS rule achieves a convergence rate of

$$f(x^k) - f^* \leq \left(1 - \frac{\mu_1}{L}\right)^k [f(x^0) - f^*],$$

when the function satisfies the PL inequality in the ∞-norm with a constant of μ_1. By the equivalence between norms we have that $\mu/n \leq \mu_1$, so this is faster than the rate with random selection.

Meir and Rätsch [2003] show that we can view some variants of boosting algorithms as implementations of coordinate descent with the GS rule. They use the error bound property to argue that these methods achieve a linear convergence rate, but this property does not lead to an explicit rate. Our simple result above thus provides the first explicit convergence rate for these variants of boosting.

4.2.3 Sign-Based Gradient Methods

The learning heuristic RPROP (Resilient backPROPagation) is a classic iterative method used for supervised learning problems in feedforward neural networks [Riedmiller and Braun, 1992].

The general update for some vector of step-sizes $\alpha_k \in \mathbb{R}^n$ is given by

$$x^{k+1} = x^k - \alpha^k \circ \operatorname{sign} \nabla f(x^k),$$

where the $\circ$ operator indicates coordinate-wise multiplication. Although this method has been used for many years in the machine learning community, we are not aware of any previous convergence rate analysis of such a method. We assume the individual step-sizes α_i^k are chosen proportional to $1/\sqrt{L_i}$, where the L_i are constants such that the gradient is 1-Lipschitz continuous in the norm defined by

$$\|z\|_{L^{-1}[1]} \triangleq \sum_i \frac{1}{\sqrt{L_i}} |z_i|.$$

Formally, we assume that the L_i are set so that for all x and y we have

$$\|\nabla f(y) - \nabla f(x)\|_{L^{-1}[1]} \leq \|y - x\|_{L[\infty]}, \tag{4.7}$$

and where the dual norm of the $\|\cdot\|_{L^{-1}[1]}$ norm above is given by the $\|\cdot\|_{L[\infty]}$ norm,

$$\|z\|_{L[\infty]} \triangleq \max_i \sqrt{L_i} |z_i|.$$

We note that such L_i always exist if the gradient is Lipschitz continuous, so this is not adding any assumptions on the function f. The particular choice of the step-sizes α_i^k that we will analyze is

$$\alpha_i^k = \frac{\|\nabla f(x^k)\|_{L^{-1}[1]}}{\sqrt{L_i}},$$

which yields a linear convergence rate for problems where the PL inequality is satisfied in the $L^{-1}[1]$-norm,

$$\frac{1}{2}\|\nabla f(x^k)\|_{L^{-1}[1]}^2 \geq \mu_{L[\infty]} \left(f(x^k) - f^*\right). \tag{4.8}$$

This choice of α^k yields steepest descent in the L_∞-norm. The coordinate-wise iteration update under this choice of α_i^k is given by

$$x_i^{k+1} = x_i^k - \frac{\|\nabla f(x^k)\|_{L^{-1}[1]}}{\sqrt{L_i}} \operatorname{sign} \nabla_i f(x^k).$$

Defining a diagonal matrix Λ with $1/\sqrt{L_i}$ along the diagonal, the update can be written as

$$x^{k+1} = x^k - \|\nabla f(x^k)\|_{L^{-1}[1]} \Lambda \circ \operatorname{sign} \nabla f(x^k). \tag{4.9}$$

Theorem 4. *Consider problem* (4.1)*, where f has a Lipschitz continuous gradient* (4.7)*, a non-empty solution set $\mathcal{X}^*$, and satisfies the PL inequality* (4.8)*. Consider the sign-based gradient*

update defined in (4.9)*. Then the algorithm has a linear convergence rate of*

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \mu_{L[\infty]}\right)\left(f(x^k) - f(x^*)\right).$$

Proof. See Appendix C.3. □

4.3 Proximal Gradient Generalization

Attouch and Bolte [2009] consider a generalization of the the PL inequality due to Kurdyka to give conditions under which the classic proximal-point algorithm achieves a linear convergence rate for non-smooth problems (called the KL inequality). However, in practice proximal-*gradient* methods are more relevant to many machine learning problems. The KL inequality has been used to show local linear convergence of proximal gradient methods [Li and Pong, 2016], and it has been used to show global linear convergence of proximal gradient methods under the assumption that the objective function is convex [Bolte et al., 2015]. In this section we propose a different generalization of the PL inequality that yields a *simple* global linear convergence analysis without assuming convexity of the objective function.

Proximal gradient methods apply to problems of the form

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} F(x) = f(x) + g(x), \tag{4.10}$$

where f is a differentiable function with an L-Lipschitz continuous gradient and g is a simple but potentially non-smooth convex function. Typical examples of simple functions g include a scaled ℓ_1-norm of the parameter vectors, $g(x) = \lambda\|x\|_1$, and indicator functions that are zero if x lies in a simple convex set and are infinity otherwise.

In order to analyze proximal gradient algorithms, a natural (though not particularly intuitive) generalization of the PL inequality is that there exists a $\mu > 0$ satisfying

$$\frac{1}{2}\mathcal{D}_g(x, L) \geq \mu(F(x) - F^*), \tag{4.11}$$

where

$$\mathcal{D}_g(x, \alpha) \equiv -2\alpha \min_y \left[\langle \nabla f(x), y - x\rangle + \frac{\alpha}{2}||y - x||^2 + g(y) - g(x)\right]. \tag{4.12}$$

We call this the *proximal-PL* inequality, and we note that if g is constant (or linear) then it reduces to the standard PL inequality. Below we show that this inequality is sufficient for the proximal gradient method to achieve a global linear convergence rate.

Theorem 5. *Consider problem* (4.10)*, where f has an L-Lipschitz continuous gradient (4.2), F has a non-empty solution set $\mathcal{X}^*$, g is convex, and F satisfies the proximal-PL inequality* (4.11)*.*

Then the proximal gradient method with a step-size of $1/L$,

$$x^{k+1} = \underset{y \in \mathbb{R}^n}{argmin} \left[\langle \nabla f(x^k), y - x^k \rangle + \frac{L}{2} ||y - x^k||^2 + g(y) - g(x^k) \right] \tag{4.13}$$

converges linearly to the optimal value F^*,

$$F(x^k) - F^* \leq \left(1 - \frac{\mu}{L}\right)^k [F(x^0) - F^*].$$

Proof. By using Lipschitz continuity of the gradient of f we have

$$\begin{aligned} F(x^{k+1}) &= f(x^{k+1}) + g(x^k) + g(x^{k+1}) - g(x^k) \\ &\leq F(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k \rangle + \frac{L}{2} ||x^{k+1} - x^k||^2 + g(x^{k+1}) - g(x^k) \\ &\leq F(x^k) - \frac{1}{2L} \mathcal{D}_g(x^k, L) \\ &\leq F(x^k) - \frac{\mu}{L} [F(x^k) - F^*], \end{aligned}$$

which uses the definition of x^{k+1} and $\mathcal{D}_g$ followed by the proximal-PL inequality (4.11). This subsequently implies that

$$F(x^{k+1}) - F^* \leq \left(1 - \frac{\mu}{L}\right) [F(x^k) - F^*], \tag{4.14}$$

which applied recursively gives the result. □

While other conditions have been proposed to show linear convergence rates of proximal gradient methods without strong convexity [Kadkhodaie et al., 2014, Zhang, 2015], their analyses tend to be much more complicated than the above while, as we discuss in the next section, the proximal-PL inequality includes the standard scenarios where these apply.

4.3.1 Relevant Problems

As with the PL inequality, we now list several important function classes that satisfy the proximal-PL inequality (4.11). We give proofs that these classes satisfy the inequality in Appendix C.5.

1. The inequality is satisfied if f satisfies the PL inequality and g is constant. Thus, the above result generalizes Theorem 1.

2. The inequality is satisfied if f is strongly convex. This is the usual assumption used to show a linear convergence rate for the proximal gradient algorithm [Schmidt et al., 2011], although we note that the above analysis is much simpler than standard arguments.

3. The inequality is satisfied if f has the form $f(x) = h(Ax)$ for a strongly convex function h and a matrix A, while g is an indicator function for a polyhedral set.

4. The inequality is satisfied if F is convex and satisfies the QG property.

It has also been shown that the inequality is satisfied if F satisfies the proximal-EB condition or the KL inequality [Karimi et al., 2016]. By this equivalence, the proximal-PL inequality also holds for other problems where a linear convergence rate has been shown like group ℓ_1-regularization [Tseng, 2010], sparse group ℓ_1-regularization [Zhang et al., 2013], nuclear-norm regularization [Hou et al., 2013], and other classes of functions [Drusvyatskiy and Lewis, 2016, Zhou and So, 2015].

4.3.2 Least Squares with ℓ_1-Regularization

Perhaps the most interesting example of problem (4.10) is the ℓ_1-regularized least squares problem,

$$\operatorname*{argmin}_{x\in\mathbb{R}^n} \frac{1}{2}\|Ax-b\|^2 + \lambda\|x\|_1,$$

where $\lambda > 0$ is the regularization parameter. This problem has been studied extensively in machine learning, signal processing, and statistics. This problem structure seems well-suited to using proximal gradient methods, but the first works analyzing proximal gradient methods for this problem only showed sub-linear convergence rates [Beck and Teboulle, 2009]. Subsequent works show that linear convergence rates can be achieved under additional assumptions. For example, Gu et al. [2013] prove that their algorithm achieves a linear convergence rate if A satisfies a *restricted isometry property* (RIP) and the solution is sufficiently sparse. Xiao and Zhang [2013] also assume the RIP property and show linear convergence using a homotopy method that slowly decreases the value of λ. Agarwal et al. [2012] give a linear convergence rate under a *modified restricted strong convexity* and *modified restricted smoothness* assumption. But these problems have also been shown to satisfy proximal variants of the EB condition [Necoara and Clipici, 2016, Tseng, 2010], and thus by the equivalence result in [Karimi et al., 2016] we have that *any* ℓ_1-regularized least squares problem satisfies the proximal-PL inequality. Thus, Theorem 5 gives a simple proof of global linear convergence for these problems without making additional assumptions or making any modifications to the algorithm.

4.3.3 Proximal Coordinate Descent

It is also possible to adapt our results on coordinate descent and proximal gradient methods in order to give a linear convergence rate for coordinate-wise proximal gradient methods for problem (4.10). To do this, we require the extra assumption that g is a separable function. This means that $g(x) = \sum_i g_i(x_i)$ for a set of univariate functions g_i. The update rule for the coordinate-wise proximal gradient method is

$$x^{k+1} = \operatorname*{argmin}_{\alpha\in\mathbb{R}} \left[\alpha\nabla_{i_k} f(x^k) + \frac{L}{2}\alpha^2 + g_{i_k}(x_{i_k}+\alpha) - g_{i_k}(x_{i_k})\right], \tag{4.15}$$

We state the convergence rate result below.

Theorem 6. *Assume the setup of Theorem 5 and that g is a separable function $g(x) = \sum_i g_i(x_i)$, where each g_i is convex. Then the coordinate-wise proximal gradient update rule (4.15) achieves a convergence rate*

$$\mathbb{E}[F(x^k) - F^*] \leq \left(1 - \frac{\mu}{nL}\right)^k [F(x^0) - F^*], \tag{4.16}$$

when i_k is selected uniformly at random.

The proof is given in Appendix C.6 and although it is more complicated than the proof of Theorem 5, it is still simpler than existing proofs for proximal coordinate descent under strong convexity [Richtárik and Takáč, 2014]. It is also possible to analyze stochastic proximal gradient algorithms, and indeed Reddi et al. [2016a] use the proximal-PL inequality to analyze finite-sum methods in the proximal stochastic case. We also note that Zhang [2016] has recently analyzed cyclic coordinate descent for convex functions satisfying QG.

4.3.4 Support Vector Machines

Another important model problem that arises in machine learning is support vector machines,

$$\underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \sum_{i=1}^{m} \max(0, 1 - b_i x^T a_i) + \frac{\lambda}{2}\|x\|^2. \tag{4.17}$$

where (a_i, b_i) are the labelled training set with $a_i \in \mathbb{R}^n$ and $b_i \in \{-1, 1\}$. We often solve this problem by performing coordinate optimization on its Fenchel dual, which has the form

$$\min_{\bar{w}} f(\bar{w}) = \frac{1}{2}\bar{w}^T M \bar{w} - \sum \bar{w}_i, \quad \bar{w}_i \in [0, U], \tag{4.18}$$

for a particular positive semi-definite matrix M and constant U. This convex function satisfies the QG property and thus Theorem 6 implies that coordinate optimization achieves a linear convergence rate in terms of optimizing the dual objective. Further, note that Hush et al. [2006] show that we can obtain an ϵ-accurate solution to the primal problem with an $O(\epsilon^2)$-accurate solution to the dual problem. Thus this result also implies we can obtain a linear convergence rate on the primal problem by showing that stochastic dual coordinate ascent has a linear convergence rate on the dual problem. Global linear convergence rates for SVMs have also been shown by others [Ma et al., 2015b, Tseng and Yun, 2009a, Wang and Lin, 2014], but again we note that these works lead to much more complicated analyses. Although the constants in these convergence rates may be quite bad (depending on the smallest non-zero singular value of the Gram matrix), we note that the existing sublinear rates still apply in the early iterations while, as the algorithm begins to identify support vectors, the constants improve (depending on the smallest non-zero singular value of the block of the Gram matrix corresponding to the support vectors).

The result of the previous section is not only restricted to SVMs. Indeed, the result of

the previous section implies a linear convergence rate for many ℓ_2-regularized linear prediction problems, the framework considered in the stochastic dual coordinate ascent (SDCA) work of Shalev-Shwartz and Zhang [2013]. While Shalev-Shwartz and Zhang [2013] show that this is true when the primal is smooth, our result gives linear rates in many cases where the primal is non-smooth.

4.4 Discussion

We believe that the work in this chapter provides a unifying and simplifying view of a variety of optimization and convergence rate issues in machine learning. Indeed, we have shown that many of the assumptions used to achieve linear convergence rates can be replaced by the PL inequality and its proximal generalization. While we have focused on sufficient conditions for linear convergence, another recent work has turned to the question of necessary conditions for convergence [Zhang, 2016]. Further, while we have focused on non-accelerated methods, Zhang [2016] has recently analyzed Nesterov's accelerated gradient method without strong convexity. We also note that, while we have focused on first-order methods, Nesterov and Polyak [2006] have used the PL inequality to analyze a second-order Newton-style method with cubic regularization. They also consider a generalization of the inequality under the name *gradient-dominated* functions.

Throughout this chapter, we pointed out how our analyses imply convergence rates for a variety of machine learning models and algorithms. Some of these were previously known, typically under stronger assumptions or with more complicated proofs, but many of these are novel. We expect that going forward efficiency will no longer be decided by the issue of whether functions are strongly convex, but rather by whether they satisfy a variant of the PL inequality.

Chapter 5

Greedy Block Coordinate Descent

Block coordinate descent (BCD) methods have become one of our key tools for solving some of the most important large-scale optimization problems. This is due to their typical ease of implementation, low memory requirements, cheap iteration costs, adaptability to distributed settings, ability to use problem structure, and numerical performance. Notably, they have been used for almost two decades in the context of ℓ_1-regularized least squares (LASSO) [Fu, 1998, Sardy et al., 2000] and support vector machines (SVMs) [Chang and Lin, 2011, Joachims, 1999]. Indeed, randomized BCD methods have recently been used to solve instances of these widely-used models with a billion variables [Richtárik and Takáč, 2014], while for "kernelized" SVMs *greedy* BCD methods remain among the state of the art methods [You et al., 2016]. Due to the wide use of these models, any improvements on the convergence of BCD methods will affect a myriad of applications.

While there are a variety of ways to implement a BCD method, the three main building blocks that affect its performance are:

1. **Blocking strategy**. We need to define a set of possible "blocks" of problem variables that we might choose to update at a particular iteration. Two common strategies are to form a partition of the coordinates into disjoint sets (we call this *fixed* blocks) or to consider any possible subset of coordinates as a "block" (we call this *variable* blocks). Typical choices include using a set of fixed blocks related to the problem structure, or using variable blocks by randomly sampling a fixed number of coordinates.

2. **Block selection rule**. Given a set of possible blocks, we need a rule to select a block of corresponding variables to update. Classic choices include cyclically going through a fixed ordering of blocks, choosing random blocks, choosing the block with the largest gradient (the *Gauss-Southwell* rule), or choosing the block that leads to the largest improvement.

3. **Block update rule**. Given the block we have selected, we need to decide how to update the block of corresponding variables. Typical choices include performing a gradient descent iteration, computing the Newton direction and performing a line search, or computing the optimal update of the block by subspace minimization.

In the next section we introduce our notation, review the standard choices behind BCD algorithms, and discuss problem structures where BCD is suitable. Subsequently, the following sections explore a wide variety of ways to speed up BCD by modifying the three building blocks above.

1. In Section 5.2 we propose block selection rules that are variants of the Gauss-Southwell rule, but that incorporate knowledge of Lipschitz constants in order to give better bounds on the progress made at each iteration. We also give a general result characterizing the convergence rate obtained using the Gauss-Southwell rule as well as the new greedy rules, under both the Polyak-Łojasiewicz inequality and for general (potentially non-convex) functions.

2. In Section 5.3 we discuss practical implementation issues. This includes how to approximate the new rules in the variable-block setting, how to estimate the Lipschitz constants, how to efficiently implement line searches, how the blocking strategy interacts with greedy rules, and why we should prefer Newton updates over the "matrix updates" of recent works.

3. In Section 5.4 we show how second-order updates, or the exact update for quadratic functions, can be computed in linear-time for problems with sparse dependencies when using "forest-structured" blocks. This allows us to use huge block sizes for problems with sparse dependencies, and uses a connection between sparse quadratic functions and Gaussian Markov random fields (GMRFs) by exploiting the "message-passing" algorithms developed for GMRFs.

We note that many related ideas have been explored by others in the context of BCD methods and we will go into detail about these related works in subsequent sections. In Section 5.5 we use a variety of problems arising in machine learning to evaluate the effects of these choices on BCD implementations. These experiments indicate that in some cases the performance improvement obtained by using these enhanced methods can be dramatic. The source code and data files required to reproduce the experimental results of this paper can be downloaded from: `https://github.com/IssamLaradji/BlockCoordinateDescent`.

5.1 Block Coordinate Descent Algorithms

We first consider the problem of minimizing a differentiable multivariate function,

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} f(x). \tag{5.1}$$

At iteration k of a BCD algorithm, we first select a block $b_k \subseteq \{1, 2, \dots, n\}$ and then update the subvector $x_{b_k} \in \mathbb{R}^{|b_k|}$ corresponding to these variables,

$$x_{b_k}^{k+1} = x_{b_k}^k + d^k.$$

Coordinates of x^{k+1} that are not included in b_k are simply kept at their previous value. The vector $d^k \in \mathbb{R}^{|b_k|}$ is typically selected to provide descent in the direction of the reduced dimensional

subproblem,

$$d^k \in \underset{d \in \mathbb{R}^{|b_k|}}{\operatorname{argmin}} f(x^k + U_{b_k} d), \tag{5.2}$$

where we construct $U_{b_k} \in \{0,1\}^{n \times |b_k|}$ from the columns of the identity matrix corresponding to the coordinates in b_k. Using this notation, we have

$$x_{b_k} = U_{b_k}^T x,$$

which allows us to write the BCD update of all n variables in the form

$$x^{k+1} = x^k + U_{b_k} d^k.$$

There are many possible ways to define the block b_k as well as the direction d^k. Typically we have a maximum block size τ, which is chosen to be the largest number of variables that we can efficiently update at once. Given τ that divides evenly into n, consider a simple ordered fixed partition of the coordinates into a set $\mathcal{B}$ of n/τ blocks,

$$\mathcal{B} = \{\{1, 2, \dots, \tau\}, \{\tau + 1, \tau + 2, \dots, 2\tau\}, \dots, \{(n - \tau) + 1, (n - \tau) + 2, \dots, n\}\}.$$

To select the block in $\mathcal{B}$ to update at each iteration we could simply repeatedly cycle through this set in order. A simple choice of d^k is the negative gradient corresponding to the coordinates in b_k, multiplied by a scalar step-size α_k that is sufficiently small to ensure that we decrease the objective function. This leads to a gradient update of the form

$$x^{k+1} = x^k - \alpha_k U_{b_k} \nabla_{b_k} f(x^k), \tag{5.3}$$

where $\nabla_{b_k} f(x^k)$ are the elements of the gradient $\nabla f(x^k)$ corresponding to the coordinates in b_k. While this gradient update and cyclic selection among an ordered fixed partition is simple, we can often drastically improve the performance using more clever choices. We highlight some common alternative choices in the next three subsections.

5.1.1 Block Selection Rules

Repeatedly going through a fixed partition of the coordinates is known as *cyclic* selection [Bertsekas, 2016], and this is referred to as Gauss-Seidel when solving linear systems [Seidel, 1874]. The performance of cyclic selection may suffer if the order the blocks are cycled through is chosen poorly, but it has been shown that random permutations can fix a worst case for cyclic CD [Lee and Wright, 2016]. A variation on cyclic selection is "essentially" cyclic selection where each block must be selected at least every m iterations for some fixed m that is larger than the number of blocks [Bertsekas, 2016]. Alternately, several authors have explored the advantages of *randomized* block selection [Nesterov, 2010, Richtárik and Takáč, 2014]. The simplest randomized selection rule is to select one of the blocks uniformly at random. However,

several recent works show dramatic performance improvements over this naive random sampling by incorporating knowledge of the Lipschitz continuity properties of the gradients of the blocks [Nesterov, 2010, Qu and Richtárik, 2016, Richtárik and Takáč, 2016] or more recently by trying to estimate the optimal sampling distribution online [Namkoong et al., 2017].

An alternative to cyclic and random block selection is *greedy* selection. Greedy methods solve an optimization problem to select the "best" block at each iteration. A classic example of greedy selection is the **block Gauss-Southwell** (GS) rule, which chooses the block whose gradient has the largest Euclidean norm,

$$b_k \in \underset{b \in \mathcal{B}}{\operatorname{argmax}} \|\nabla_b f(x^k)\|, \tag{5.4}$$

where we use $\mathcal{B}$ as the set of possible blocks. As we saw in Chapter 2, this rule tends to make more progress per iteration in theory and practice than randomized selection. Unfortunately, for many problems it is more expensive than cyclic or randomized selection. However, several recent works show that certain problem structures allow efficient calculation of GS-style rules (see Chapter 2 as well as [Fountoulakis et al., 2016, Lei et al., 2016, Meshi et al., 2012]), allow efficient approximation of GS-style rules [Dhillon et al., 2011, Stich et al., 2017, Thoppe et al., 2014], or allow other rules that try to improve on the progress made at each iteration [Csiba et al., 2015, Glasmachers and Dogan, 2013].

The ideal selection rule is the maximum improvement (MI) rule, which chooses the block that decreases the function value by the largest amount. Notable recent applications of this rule include leading eigenvector computation [Li et al., 2015], polynomial optimization [Chen et al., 2012], and fitting Gaussian processes [Bo and Sminchisescu, 2012]. While recent works explore computing or approximating the MI rule for quadratic functions [Bo and Sminchisescu, 2012, Thoppe et al., 2014], in general the MI rule is much more expensive than the GS rule.

5.1.2 Fixed vs. Variable Blocks

While the choice of the block to update has a significant effect on performance, how we define the set of *possible* blocks also has a major impact. Although other variations are possible, we highlight below the two most common blocking strategies:

1. **Fixed blocks**. This method uses a partition of the coordinates into disjoint blocks, as in our simple example above. This partition is typically chosen prior to the first iteration of the BCD algorithm, and this set of blocks is then held fixed for all iterations of the algorithm. We often use blocks of roughly equal size, so if we use blocks of size τ this method might partition the n coordinates into n/τ blocks. Generic ways to partition the coordinates include "in order" as we did above [Bertsekas, 2016], or using a random partition [Nesterov, 2010]. Alternatively, the partition may exploit some inherent substructure of the objective function such as block separability [Meier et al., 2008], the ability to efficiently solve the corresponding sub-problem (5.2) with respect to the blocks [Sardy et al.,

2000], or based on the Lipschitz constants of the resulting blocks [Csiba and Richtárik, 2016, Thoppe et al., 2014].

2. **Variable blocks**. Instead of restricting our blocks to a pre-defined partition of the coordinates, we could instead consider choosing any of the $2^n - 1$ possible sets of coordinates as our block. In the randomized setting, this is referred to as "arbitrary" sampling [Qu and Richtárik, 2016, Richtárik and Takáč, 2016]. We say that such strategies use *variable* blocks because we are not choosing from a partition of the coordinates that is fixed across the iterations. Due to computational considerations, when using variable blocks we typically want to impose a restriction on the size of the blocks. For example, we could construct a block of size τ by randomly sampling τ coordinates without replacement, which is known as τ-nice sampling [Qu and Richtárik, 2016, Richtárik and Takáč, 2016]. Alternately, we could include each coordinate in the block b_k with some probability like τ/n (so the block size may change across iterations but we control its expected size). A version of the greedy GS rule (5.4) with variable blocks would select the τ coordinates corresponding to the elements of the gradient with largest magnitudes [Tseng and Yun, 2009b]. This can be viewed as a greedy variant of τ-nice sampling. While we can find these τ coordinates easily,[10] computing the MI rule with variable blocks is much more difficult. Indeed, while methods exist to compute the MI rule for quadratics with fixed blocks [Thoppe et al., 2014], with variable blocks it is NP-hard to compute the MI rule and existing works resort to approximations [Bo and Sminchisescu, 2012].

5.1.3 Block Update Rules

The selection of the update vector d^k can significantly affect performance of the BCD method. For example, in the gradient update (5.3) the method can be sensitive to the choice of the step-size α_k. Classic ways to set α_k include using a fixed step-size (with each block possibly having its own fixed step-size), using an approximate line search, or using the optimal step-size (which has a closed-form solution for quadratic objectives) [Bertsekas, 2016].

The most common alternative to the gradient update above is a **Newton update**,

$$d^k = -\alpha_k \left(\nabla^2_{b_k b_k} f(x^k)\right)^{-1} \nabla_{b_k} f(x^k), \tag{5.5}$$

where we might replace the instantaneous Hessian $\nabla^2_{b_k b_k} f(x^k)$ by a positive-definite approximation to it. In this context the step-size α_k is again a step-size that can be chosen using similar strategies to those mentioned above. Several recent works analyze such updates and show that they can substantially improve the convergence rate [Fountoulakis and Tappenden, 2015, Qu et al., 2016, Tappenden et al., 2016]. For special problem classes, another possible

[10] Once the max-heap (as defined in Chapter 2) has been updated with the current gradient values, we can find the maximal τ elements by repeating the process of removing the maximal element and resorting the remaining elements τ times. This yields the τ maximal elements at a cost of $\tau \log(n)$.

type of update is what we will call the **optimal update**. This update chooses d^k to solve (5.2). In other words, it updates the block b_k to maximally decrease the objective function.

5.1.4 Problems of Interest

BCD methods tend to be good choices for problems where we can update all variables for roughly the same cost as computing the gradient. As presented in Section 2.1 for the single-coordinate case this includes the following two common problem structures,

$$h_1(x) := \sum_{i=1}^{n} g_i(x_i) + f(Ax), \quad \text{or} \quad h_2(x) := \sum_{i \in V} g_i(x_i) + \sum_{(i,j) \in E} f_{ij}(x_i, x_j),$$

where f is smooth and cheap, the f_{ij} are smooth, $G = \{V, E\}$ is a graph, and A is a matrix. Examples of problems leading to functions of the form h_1 include least squares, logistic regression, LASSO, and SVMs.[11] The most important example of problem h_2 is quadratic functions, which are crucial to many aspects of scientific computing.[12]

Problems h_1 and h_2 are also suitable for BCD methods, as they tend to admit efficient block update strategies. In general, if single-coordinate descent is efficient for a problem, then BCD methods are also efficient for that problem and this applies whether we use fixed blocks or variable blocks. Other scenarios where coordinate descent and BCD methods have proven useful include matrix and tensor factorization methods [Xu and Yin, 2013, Yu et al., 2012], problems involving log-determinants [Hsieh et al., 2013, Scheinberg and Rish, 2009], and problems involving convex extensions of sub-modular functions [Ene and Nguyen, 2015, Jegelka et al., 2013].

An important point to note is that there are special problem classes where BCD with fixed blocks is reasonable even though using variable blocks (or single-coordinate updates) would not be suitable. For example, consider a variant of problem h_1 where we use *group* ℓ_1-regularization [Bakin, 1999],

$$h_3(x) := \sum_{b \in \mathcal{B}} \|x_b\| + f(Ax), \tag{5.6}$$

where $\mathcal{B}$ is a partition of the coordinates. We cannot apply single-coordinate updates to this problem due to the non-smooth norms, but we can take advantage of the group-separable structure in the sum of norms and apply BCD using the blocks in $\mathcal{B}$ [Meier et al., 2008, Qin et al., 2013]. Sardy et al. [2000] in their early work on solving LASSO problems consider problem h_1 where the columns of A are the union of a set of orthogonal matrices. By choosing the fixed blocks to correspond to the orthogonal matrices, it is very efficient to apply BCD.

[11] Coordinate descent remains suitable for multi-linear generalizations of problem h_1 like functions of the form $f(XY)$ where X and Y are both matrix variables.

[12] Problem h_2 can be generalized to allow functions between more than 2 variables, and coordinate descent remains suitable as long as the expected number of functions in which each variable appears is n-times smaller than the total number of functions (assuming each function has a constant cost).

In Appendix D.1, we outline how fixed blocks lead to an efficient greedy BCD method for the widely-used multi-class logistic regression problem when the data has a certain sparsity level.

5.2 Improved Greedy Rules

Previous works have identified that the greedy GS rule can lead to suboptimal progress, and have proposed rules that are closer to the MI rule for the special case of quadratic functions [Bo and Sminchisescu, 2012, Thoppe et al., 2014]. However, for non-quadratic functions it is not obvious how we should approximate the MI rule. As an intermediate between the GS rule and the MI rule for general functions, in Section 2.5.2 we presented the Gauss-Southwell-Lipschitz (GSL) rule in the case of single-coordinate updates. The GSL rule is equivalent to the MI rule in the special case of quadratic functions, so either rule can be used in that setting. However, the MI rule involves optimizing over a subspace which will typically be expensive for non-quadratic functions. After reviewing the classic block GS rule, in this section we consider several possible block extensions of the GSL rule that give a better approximation to the MI rule without requiring subspace optimization.

5.2.1 Block Gauss-Southwell

When analyzing BCD methods we typically assume that the gradient of each block b is L_b-Lipschitz continuous, meaning that for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^{|b|}$

$$\|\nabla_b f(x + U_b d) - \nabla_b f(x)\| \leq L_b \|d\|, \tag{5.7}$$

for some constant $L_b > 0$. This is a standard assumption, and in Appendix D.2 we give bounds on L_b for the common data-fitting models of least squares and logistic regression. If we apply the descent lemma [Bertsekas, 2016] to the reduced sub-problem (5.2) associated with some block b_k selected at iteration k, then we obtain the following upper bound on the function value progress,

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k \rangle + \frac{L_{b_k}}{2} \|x^{k+1} - x^k\|^2 \\ &= f(x^k) + \langle \nabla_{b_k} f(x^k), d^k \rangle + \frac{L_{b_k}}{2} \|d^k\|^2. \end{aligned} \tag{5.8}$$

The right side is minimized in terms of d^k under the choice

$$d^k = -\frac{1}{L_{b_k}} \nabla_{b_k} f(x^k), \tag{5.9}$$

which is simply a gradient update with a step-size of $\alpha_k = 1/L_{b_k}$. Substituting this into our upper bound, we obtain

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2L_{b_k}}\|\nabla_{b_k} f(x^k)\|^2. \tag{5.10}$$

A reasonable way to choose a block b_k at each iteration is by minimizing the upper bound in (5.10) with respect to b_k. For example, if we assume that L_b is the same for all blocks b then we derive the GS rule (5.4) of choosing the b_k that maximizes the gradient norm.

We can use the bound (5.10) to compare the progress made by different selection rules. For example, this bound indicates that the GS rule can make more progress with variable blocks than with fixed blocks (under the usual setting where the fixed blocks are a subset of the possible variable blocks). In particular, consider the case where we have partitioned the coordinates into blocks of size τ and we are comparing this to using variable blocks of size τ. The case where there is no advantage for variable blocks is when the indices corresponding to the τ-largest $|\nabla_i f(x^k)|$ values are in one of the fixed partitions; in this (unlikely) case the GS rule with fixed blocks and variable blocks will choose the same variables to update. The case where we see the largest advantage of using variable blocks is when each of the indices corresponding to the τ-largest $|\nabla_i f(x^k)|$ values are in different blocks of the fixed partition; in this case the last term in (5.10) can be improved by a factor as large as τ^2 when using variable blocks instead of fixed blocks. Thus, with larger blocks there is more of an advantage to using variable blocks over fixed blocks.

5.2.2 Block Gauss-Southwell-Lipschitz

The GS rule is not the optimal block selection rule in terms of the bound (5.10) if we know the block-Lipschitz constants L_b. Instead of choosing the block with largest norm, consider minimizing (5.10) in terms of b_k,

$$b_k \in \operatorname*{argmax}_{b \in \mathcal{B}} \left\{ \frac{\|\nabla_b f(x^k)\|^2}{L_b} \right\}. \tag{5.11}$$

We call this the **block Gauss-Southwell-Lipschitz** (GSL) rule. If all L_b are the same, then the GSL rule is equivalent to the classic GS rule. But in the typical case where the L_b differ, the GSL rule guarantees more progress than the GS rule since it incorporates the gradient information as well as the Lipschitz constants L_b. For example, it reflects that if the gradients of two blocks are similar but their Lipschitz constants are very different, then we can guarantee more progress by updating the block with the smaller Lipschitz constant. In the extreme case, for both fixed and variable blocks the GSL rule improves the bound (5.10) over the GS rule by a factor as large as $(\max_{b\in\mathcal{B}} L_b)/(\min_{b\in\mathcal{B}} L_b)$.

The block GSL rule in (5.11) is a simple generalization of the single-coordinate GSL rule to blocks of any size. However, it loses a key feature of the single-coordinate GSL rule: the block

GSL rule is *not* equivalent to the MI rule for quadratic functions. Unlike the single-coordinate case, where $\nabla^2_{ii} f(x^k) = L_i$ so that (5.8) holds with equality, for the block case we only have $\nabla^2_{bb} f(x^k) \preceq L_b$ so (5.8) may underestimate the progress that is possible in certain directions. In the next section we give a second generalization of the GSL rule that is equivalent to the MI rule for quadratics.

5.2.3 Block Gauss-Southwell-Quadratic

For single-coordinate updates, the bound in (5.10) is the tightest quadratic bound on progress we can expect given only the assumption of block Lipschitz-continuity (it holds with equality for quadratic functions). However, for block updates of more than one variable we can obtain a tighter quadratic bound using general quadratic norms of the form $\|\cdot\|_H = \sqrt{\langle H\cdot, \cdot\rangle}$ for some positive-definite matrix H. In particular, assume that each block has a Lipschitz-continuous gradient with $L_b = 1$ for a particular positive-definite matrix $H_b \in \mathbb{R}^{|b|\times|b|}$, meaning that

$$\|\nabla_b f(x + U_b d) - \nabla_b f(x)\|_{H_b^{-1}} \leq \|d\|_{H_b},$$

for all $x \in \mathbb{R}^n$ and $d \in \mathbb{R}^{|b|}$. Due to the equivalence between norms, this merely changes how we measure the continuity of the gradient and is not imposing any new assumptions. Indeed, the block Lipschitz-continuity assumption in (5.7) is just a special case of the above with $H_b = L_b I$, where I is the $|b| \times |b|$ identity matrix. Although this characterization of Lipschitz continuity appears more complex, for some functions it is actually computationally cheaper to construct matrices H_b than to find valid bounds L_b. We show this in Appendix D.2 for the cases of least squares and logistic regression.

Under this alternative characterization of the Lipschitz assumption, at each iteration k we have

$$f(x^{k+1}) \leq f(x^k) + \langle \nabla_{b_k} f(x^k), d^k\rangle + \frac{1}{2}\|d^k\|^2_{H_{b_k}}. \tag{5.12}$$

The left-hand side of (5.12) is minimized when

$$d^k = -\left(H_{b_k}\right)^{-1} \nabla_{b_k} f(x^k), \tag{5.13}$$

which we will call the **matrix update** of a block. Although this is equivalent to Newton's method for quadratic functions, we use the name "matrix update" rather than "Newton's method" here to distinguish two types of updates: Newton's method is based on the instantaneous Hessian $\nabla^2_{bb} f(x^k)$, while the matrix update is based on a matrix H_b that upper bounds the Hessian for all x.[13] We will discuss Newton updates in subsequent sections, but substituting the matrix update into the upper bound yields

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2}\|\nabla_{b_k} f(x^k)\|^2_{H^{-1}_{b_k}}. \tag{5.14}$$

[13] We say that a matrix A "upper bounds" a matrix B, written $A \succeq B$, if for all x we have $x^T A x \geq x^T B x$.

Consider a simple quadratic function $f(x) = x^T Ax$ for a positive-definite matrix A. In this case we can take H_b to be the sub-matrix A_{bb} while in our previous bound we would require L_b to be the maximum eigenvalue of this sub-matrix. Thus, in the worst case (where $\nabla_{b_k} f(x^k)$ is in the span of the principal eigenvectors of A_{bb}) the new bound is at least as good as (5.10). But if the eigenvalues of A_{bb} are spread out then this bound shows that the matrix update will typically guarantee substantially more progress; in this case the quadratic bound (5.14) can improve on the bound in (5.10) by a factor as large as the condition number of A_{bb} when updating block b. The update (5.13) was analyzed for BCD methods in several recent works [Qu et al., 2016, Tappenden et al., 2016], which considered random selection of the blocks. They show that this update provably reduces the number of iterations required, and in some cases dramatically. For the special case of quadratic functions where (5.14) holds with equality, greedy rules based on minimizing this bound have been explored for both fixed [Thoppe et al., 2014] and variable [Bo and Sminchisescu, 2012] blocks.

Rather than focusing on the special case of quadratic functions, we want to define a better greedy rule than (5.11) for functions with Lipschitz-continuous gradients. By optimizing (5.14) in terms of b_k we obtain a second generalization of the GSL rule,

$$b_k \in \underset{b \in \mathcal{B}}{\operatorname{argmax}} \left\{ \|\nabla_b f(x^k)\|_{H_b^{-1}} \right\} \equiv \underset{b \in \mathcal{B}}{\operatorname{argmax}} \left\{ \nabla_b f(x^k)^T H_b^{-1} \nabla_b f(x^k) \right\}, \tag{5.15}$$

which we call the **block Gauss-Southwell quadratic** (GSQ) rule.[14] Since (5.14) holds with equality for quadratics this new rule is equivalent to the MI rule in that case. But this rule also applies to non-quadratic functions where it guarantees a better bound on progress than the GS rule (and the GSL rule).

5.2.4 Block Gauss-Southwell-Diagonal

While the GSQ rule has appealing theoretical properties, for many problems it may be difficult to find full matrices H_b and their storage may also be an issue. Previous related works [Csiba and Richtárik, 2017, Qu et al., 2016, Tseng and Yun, 2009b] address this issue by restricting the matrices H_b to be diagonal matrices D_b. Under this choice we obtain a rule of the form

$$b_k \in \underset{b \in \mathcal{B}}{\operatorname{argmax}} \left\{ \|\nabla_b f(x^k)\|_{D_b^{-1}} \right\} \equiv \underset{b \in \mathcal{B}}{\operatorname{argmax}} \left\{ \sum_{i \in b} \frac{|\nabla_i f(x^k)|^2}{D_{b,i}} \right\}, \tag{5.16}$$

where we are using $D_{b,i}$ to refer to the diagonal element corresponding to coordinate i in block b. We call this the **block Gauss-Southwell diagonal** (GSD) rule. This bound arises if we consider a gradient update, where coordinate i has a constant step-size of $D_{b,i}^{-1}$ when updated

[14] While preparing this work for submission, we were made aware of a work that independently proposed this rule under the name "greedy mini-batch" rule [Csiba and Richtárik, 2017]. However, our focus on addressing the computational issues associated with the rule is quite different from that work, which focuses on tight convergence analyses.

as part of block b. This rule gives an intermediate approach that can guarantee more progress per iteration than the GSL rule, but that may be easier to implement than the GSQ rule.

5.2.5 Convergence Rate under Polyak-Łojasiewicz

Our discussion above focuses on the progress we can guarantee at each iteration, assuming only that the function has a Lipschitz-continuous gradient. Under additional assumptions, it is possible to use these progress bounds to derive convergence rates on the overall BCD method. For example, assume f satisfies the PL inequality presented in Chapter 4, that is, for all x we have for some $\mu > 0$ that

$$\frac{1}{2}\left(\|\nabla f(x)\|_*\right)^2 \geq \mu\left(f(x) - f^*\right), \tag{5.17}$$

where $\|\cdot\|_*$ can be any norm and f^* is the optimal function value. The function class satisfying this inequality includes all strongly convex functions but also includes a variety of other important problems like least squares (see Section 4.1.3). As seen in Section 4.2, this inequality leads to a simple proof of the linear convergence of any algorithm which has a progress bound of the form

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2}\|\nabla f(x^k)\|_*^2, \tag{5.18}$$

such as gradient descent, coordinate descent with the GS rule and sign-based updates.

Theorem 7. *Assume f satisfies the PL inequality* (5.17) *for some $\mu > 0$ and norm $\|\cdot\|_*$. Any algorithm that satisfies a progress bound of the form* (5.18) *with respect to the same norm $\|\cdot\|_*$ obtains the following linear convergence rate,*

$$f(x^{k+1}) - f^* \leq (1-\mu)^k\left[f(x^0) - f^*\right]. \tag{5.19}$$

Proof. By subtracting f^* from both sides of (5.18) and applying (5.17) directly, we obtain our result by recursion. □

Thus, if we can describe the progress obtained using a particular block selection rule and block update rule in the form of (5.18), then we have a linear rate of convergence for BCD on this class of functions. It is straightforward to do this using an appropriately defined norm, as shown in the following corollary.

Corollary 1. *Assume ∇f is Lipschitz continuous* (5.12) *and that f satisfies the PL inequality* (5.17) *in the norm defined by*

$$\|v\|_{\mathcal{B}} = \max_{b\in\mathcal{B}} \|v_b\|_{H_b^{-1}}, \tag{5.20}$$

for some $\mu > 0$ and matrix $H_b \in \mathbb{R}^{|b|\times|b|}$. Then the BCD method using either the GSQ, GSL, GSD or GS selection rule achieves a linear convergence rate of the form (5.19).

Proof. Using the definition of the GSQ rule (5.15) in the progress bound resulting from the

Lipschitz continuity of ∇f and the matrix update (5.14), we have

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2}\max_{b\in\mathcal{B}}\left\{\|\nabla_b f(x^k)\|^2_{H_b^{-1}}\right\} \\ &= f(x^k) - \frac{1}{2}\|\nabla f(x^k)\|^2_{\mathcal{B}}. \end{aligned} \tag{5.21}$$

By Theorem 7 and the observation that the GSL, GSD and GS rules are all special cases of the GSQ rule corresponding to specific choices of H_b, we have our result. □

We refer the reader to the work of Csiba and Richtárik for tools that allow alternative analyses of BCD methods [Csiba and Richtárik, 2017].

5.2.6 Convergence Rate with General Functions

The PL inequality is satisfied for many problems of practical interest, and is even satisfied for some non-convex functions. However, general non-convex functions do not satisfy the PL inequality and thus the analysis of the previous section does not apply. Without a condition like the PL inequality, it is difficult to show convergence to the optimal function value f^* (since we should not expect a local optimization method to be able to find the global solution of functions that may be NP-hard to minimize). However, the bound (5.21) still implies a weaker type of convergence rate even for general non-convex problems. The following result is a generalization of a standard argument for the convergence rate of gradient descent [Nesterov, 2004], and gives us an idea of how fast the BCD method is able to find a point resembling a stationary point even in the general non-convex setting.

Theorem 8. *Assume ∇f is Lipschitz continuous* (5.12) *and that f is bounded below by some f^*. Then the BCD method using either the GSQ, GSL, GSD or GS selection rule achieves the following convergence rate of the minimum gradient norm,*

$$\min_{t=0,1,\ldots,k-1}\|\nabla f(x^t)\|^2_{\mathcal{B}} \leq \frac{2(f(x^0)-f^*)}{k}.$$

Proof. By rearranging (5.21), we have

$$\frac{1}{2}\|\nabla f(x^k)\|^2_{\mathcal{B}} \leq f(x^k) - f(x^{k+1}).$$

Summing this inequality over iterations $t = 0$ up to $(k-1)$ yields

$$\frac{1}{2}\sum_{t=0}^{k-1}\|\nabla f(x^t)\|^2_{\mathcal{B}} \leq f(x^0) - f(x^{k+1}).$$

Using that all k elements in the sum are lower bounded by their minimum and that $f(x^{k+1}) \geq$

f^*, we get

$$\frac{k}{2}\left(\min_{t=0,1,\ldots,k-1} \|\nabla f(x^t)\|_{\mathcal{B}}^2\right) \leq f(x^0) - f^*.$$

□

Due to the potential non-convexity we cannot say anything about the gradient norm of the final iteration, but this shows that the minimum gradient norm converges to zero with an error at iteration k of $O(1/k)$. This is a global sublinear result, but note that if the algorithm eventually arrives and stays in a region satisfying the PL inequality around a set of local optima, then the local convergence rate to this set of optima will increase to be linear.

5.3 Practical Issues

The previous section defines straightforward new rules that yield a simple analysis. In practice there are several issues that remain to be addressed. For example, it seems intractable to compute any of the new rules in the case of variable blocks. Furthermore, we may not know the Lipschitz constants for our problem. For fixed blocks we also need to consider how to partition the coordinates into blocks. Another issue is that the d^k choices used above do not incorporate the local Hessian information. Although how we address these issues will depend on our particular application, in this section we discuss several issues associated with these practical considerations.

5.3.1 Tractable GSD for Variable Blocks

The problem with using any of the new selection rules above in the case of variable blocks is that they seem intractable to compute for any non-trivial block size. In particular, to compute the GSL rule using variable blocks requires the calculation of L_b for each possible block, which seems intractable for any problem of reasonable size. Since the GSL rule is a special case of the GSD and GSQ rules, these rules also seem intractable in general. In this section we show how to restrict the GSD matrices so that this rule has the same complexity as the classic GS rule.

Consider a variant of the GSD rule where each $D_{b,i}$ can depend on i but does not depend on b, so we have $D_{b,i} = d_i$ for some value $d_i \in \mathbb{R}_+$ for all blocks b. This gives a rule of the form

$$b_k \in \operatorname*{argmax}_{b\in\mathcal{B}} \left\{ \sum_{i\in b} \frac{|\nabla_i f(x^k)|^2}{d_i} \right\}. \tag{5.22}$$

Unlike the general GSD rule, this rule has essentially the same complexity as the classic GS rule since it simply involves finding the largest values of the ratio $|\nabla_i f(x^k)|^2/d_i$.

A natural choice of the d_i values would seem to be $d_i = L_i$, since in this case we recover the GSL rule if the blocks have a size of 1 (here we are using L_i to refer to the coordinate-wise Lipschitz constant of coordinate i). Unfortunately, this does not lead to a bound of the

form (5.18) as needed in Theorems 7 and 8 because coordinate-wise L_i-Lipschitz continuity with respect to the Euclidean norm does not imply 1-Lipschitz continuity with respect to the norm $\|\cdot\|_{D_b^{-1}}$ when the block size is larger than 1. Subsequently, the steps under this choice may increase the objective function. A similar restriction on the D_b matrices in (5.22) is used in the implementation of Tseng and Yun based on the Hessian diagonals [Tseng and Yun, 2009b], but their approach similarly does not give an upper bound and thus they employ a line search in their block update.

It is possible to avoid needing a line search by setting $D_{b,i} = L_i\tau$, where τ is the maximum block size in $\mathcal{B}$. This still generalizes the single-coordinate GSL rule, and in Appendix D.3 we show that this leads to a bound of the form (5.18) for twice-differentiable convex functions (thus Theorems 7 and 8 hold). If all blocks have the same size then this approach selects the same block as using $D_{b,i} = L_i$, but the matching block update uses a much-smaller step-size that guarantees descent. We do not advocate using this smaller step, but note that the bound we derive also holds for alternate updates like taking a gradient update with $\alpha_k = 1/L_{b_k}$ or using a matrix update based on H_{b_k}.

The choice of $D_{b,i} = L_i\tau$ leads to a fairly pessimistic bound, but it is not obvious even for simple problems how to choose an optimal set of D_i values. Choosing these values is related to the problem of finding an expected separable over-approximation (ESO), which arises in the context of randomized coordinate descent methods [Richtárik and Takáč, 2016]. Qu and Richtárik give an extensive discussion of how we might bound such quantities for certain problem structures [Qu and Richtárik, 2016]. In our experiments we also explored another simple choice that is inspired by the "simultaneous iterative reconstruction technique" (SIRT) from tomographic image reconstruction [Gregor and Fessler, 2015]. In this approach, we use a matrix upper bound M on the full Hessian $\nabla^2 f(x)$ (for all x) and set[15]

$$D_{b,i} = \sum_{j=1}^{n} |M_{i,j}|. \tag{5.23}$$

We found that this choice worked better when using gradient updates, although using the simpler $L_i\tau$ is less expensive and was more effective when doing matrix updates.

By using the relationship $L_b \leq \sum_{i\in b} L_i \leq |b| \max_{i\in b} L_i$, two other ways we might consider defining a more-tractable rule could be

$$b_k \in \operatorname*{argmax}_{b\in\mathcal{B}} \left\{ \frac{\sum_{i\in b} |\nabla_i f(x^k)|^2}{|b| \max_{i\in b} L_i} \right\}, \quad \text{or} \quad b_k \in \operatorname*{argmax}_{b\in\mathcal{B}} \left\{ \frac{\sum_{i\in b} |\nabla_i f(x^k)|^2}{\sum_{i\in b} L_i} \right\}.$$

The rule on the left can be computed using dynamic programming while the rule on the right can be computed using an algorithm of Megiddo [1979]. However, when using a step-size of $1/L_b$ we found both rules performed similarly or worse to using the GSD rule with $D_{b,i} = L_i$

[15]It follow that $D - M \succeq 0$ because it is symmetric and diagonally-dominant with non-negative diagonals.

(when paired with gradient or matrix updates).[16]

5.3.2 Tractable GSQ for Variable Blocks

In order to make the GSQ rule tractable with variable blocks, we could similarly require that the entries of H_b depend solely on the coordinates $i \in b$, so that $H_b = M_{b,b}$ where M is a fixed matrix (as above) and $M_{b,b}$ refers to the sub-matrix corresponding to the coordinates in b. Our restriction on the GSD rule in the previous section corresponds to the case where M is diagonal. In the full-matrix case, the block selected according to this rule is given by the coordinates corresponding to the non-zero variables of an L0-constrained quadratic minimization,

$$\underset{\|d\|_0 \leq \tau}{\operatorname{argmin}} \left\{ f(x^k) + \langle \nabla f(x^k), d \rangle + \frac{1}{2} d^T M d \right\}, \tag{5.24}$$

where $\|\cdot\|_0$ is the number of non-zeroes. This selection rule is discussed in Tseng and Yun [2009b], but in their implementation they use a diagonal M. Although this problem is NP-hard with a non-diagonal M, there is a recent wealth of literature on algorithms for finding approximate solutions. For example, one of the simplest local optimization methods for this problem is the iterative hard-thresholding (IHT) method [Blumensath and Davies, 2009]. Another popular method for approximately solving this problem is the orthogonal matching pursuit (OMP) method from signal processing which is also known as forward selection in statistics [Hocking, 1976, Pati et al., 1993]. Computing d via (5.24) is also equivalent to computing the MI rule for a quadratic function, and thus we could alternately use the approximation of Bo and Sminchisescu [2012] for this problem.

Although it appears quite general, note that the exact GSQ rule under this restriction on H_b does not guarantee as much progress as the more-general GSQ rule (if computed exactly) that we proposed in the previous section. For some problems we can obtain tighter matrix bounds over blocks of variables than are obtained by taking the sub-matrix of a fixed matrix-bound over all variables. We show this for the case of multi-class logistic regression in Appendix D.2. As a consequence of this result we conclude that there does not appear to be a reason to use this restriction in the case of fixed blocks.

Although using the GSQ rule with variable blocks forces us to use an approximation, these approximations might still select a block that makes more progress than methods based on diagonal approximations (which ignore the strengths of dependencies between variables). It is possible that approximating the GSQ rule does not necessarily lead to a bound of the form (5.18) as there may be no fixed norm for which this inequality holds. However, in this case we can initialize the algorithm with an update rule that does achieve such a bound in order to guarantee that Theorems 7 and 8 hold, since this initialization ensures that we do at least as well as this reference block selection rule.

[16] On the other hand, the rule on the right worked better if we forced the algorithms to use a step-size of $1/(\sum_{i \in b} L_i)$, but this led to worse performance overall than using the larger $1/L_b$ step-size.

The main disadvantage of this approach for large-scale problems is the need to deal with the full matrix M (which does not arise when using a diagonal approximation or using fixed blocks). In large-scale settings we would need to consider matrices M with special structures like the sum of a diagonal matrix with a sparse and/or a low-rank matrix.

5.3.3 Lipschitz Estimates for Fixed Blocks

Using the GSD rule with the choice of $D_{b,i} = L_i$ may also be useful in the case of fixed blocks. In particular, if it is easier to compute the single-coordinate L_i values than the block L_b values then we might prefer to use the GSD rule with this choice. On the other hand, an appealing alternative in the case of fixed blocks is to use an estimate of L_b for each block as in Nesterov's work [Nesterov, 2010]. In particular, for each L_b we could start with some small estimate (like $L_b = 1$) and then double it whenever the inequality (5.10) is not satisfied (since this indicates L_b is too small). Given some b, the bound obtained under this strategy is at most a factor of 2 slower than using the optimal values of L_b. Further, if our estimate of L_b is much smaller than the global value, then this strategy can actually guarantee much more progress than using the "correct" L_b value.[17]

In the case of matrix updates, we can use (5.14) to verify that an H_b matrix is valid [Fountoulakis and Tappenden, 2015]. Recall that (5.14) is derived by plugging the update (5.13) into the Lipschitz progress bound (5.12). Unfortunately, it is not obvious how to update a matrix H_b if we find that it is not a valid upper bound. One simple possibility is to multiply the elements of our estimate H_b by 2. This is equivalent to using a matrix update but with a scalar step-size α_k,

$$d^k = -\alpha_k (H_b)^{-1} \nabla_{b_k} f(x^k), \tag{5.25}$$

similar to the step-size in the Newton update (5.5).

5.3.4 Efficient Line Searches

The Lipschitz approximation procedures of the previous section do not seem practical when using variable blocks, since there are an exponential number of possible blocks. To use variable blocks for problems where we do not know L_b or H_b, a reasonable approach is to use a line search. For example, we can choose α_k in (5.25) using a standard line search like those that use the Armijo condition or Wolfe conditions [Nocedal and Wright, 1999]. When using large block sizes with gradient updates, line searches based on the Wolfe conditions are likely to make more progress than using the *true* L_b values (since for large block sizes the line search would tend to choose values of α_k that are much larger than $\alpha_k = 1/L_{b_k}$).

Further, the problem structures that lend themselves to efficient coordinate descent algorithms tend to lend themselves to efficient line search algorithms. For example, if our objective

[17] While it might be tempting to also apply such estimates in the case of variable blocks, a practical issue is that we would need a step-size for all of the exponential number of possible variable blocks.

has the form $f(Ax)$ then a line search would try to minimize the $f(Ax^k + \alpha_k AU_{b_k} d^k)$ in terms of α_k. Notice that the algorithm would already have access to Ax^k and that we can efficiently compute $AU_{b_k} d^k$ since it only depends on the columns of A that are in b_k. Thus, after (efficiently) computing $AU_{b_k} d^k$ once the line search simply involves trying to minimize $f(v_1 + \alpha_k v_2)$ in terms of α_k (for particular vectors v_1 and v_2). The cost of this approximate minimization will typically not add a large cost to the overall algorithm.

5.3.5 Block Partitioning with Fixed Blocks

Several prior works note that for fixed blocks the partitioning of coordinates into blocks can play a significant role in the performance of BCD methods. Thoppe et al. [2014] suggest trying to find a block-diagonally dominant partition of the coordinates, and experimented with a heuristic for quadratic functions where the coordinates corresponding to the rows with the largest values were placed in the same block. In the context of parallel BCD, Scherrer et al. [2012] consider a feature clustering approach in the context of problem h_1 that tries to minimize the spectral norm between columns of A from different blocks. Csiba and Richtárik [2016] discuss strategies for partitioning the coordinates when using randomized selection.

Based on the discussion in the previous sections, for greedy BCD methods it is clear that we guarantee the most progress if we can make the mixed norm $\|\nabla f(x^k)\|_{\mathcal{B}}$ as large as possible *across iterations* (assuming that the H_b give a valid bound). This supports strategies where we try to minimize the maximum Lipschitz constant across iterations. One way to do this is to try to ensure that the average Lipschitz constant across the blocks is small. For example, we could place the largest L_i value with the smallest L_i value, the second-largest L_i value with the second-smallest L_i value, and so on. While intuitive, this may be sub-optimal; it ignores that if we cleverly partition the coordinates we may force the algorithm to often choose blocks with very-small Lipschitz constants (which lead to much more progress in decreasing the objective function). In our experiments, similar to the method of Thoppe et. al. for quadratics, we explore the simple strategy of *sorting the L_i values and partitioning this list into equal-sized blocks*. Although in the worst case this leads to iterations that are not very productive since they update all of the largest L_i values, it also guarantees some very productive iterations that update none of the largest L_i values and leads to better overall performance in our experiments.

5.3.6 Newton Updates

Choosing the vector d^k that we use to update the block x_{b_k} would seem to be straightforward since in the previous section we derived the block selection rules in the context of specific block updates; the GSL rule is derived assuming a gradient update (5.9), the GSQ rule is derived assuming a matrix update (5.13), and so on. However, using the update d^k that leads to the selection rule can be highly sub-optimal. For example, we might make substantially more progress using the matrix update (5.13) even if we choose the block b_k based on the GSL rule. Indeed, given b_k the matrix update makes the optimal amount of progress for quadratic

functions, so in this case we should prefer the matrix update for all selection rules (including random and cyclic rules).

However, the matrix update in (5.13) can itself be highly sub-optimal for non-quadratic functions as it employs an upper-bound H_{b_k} on the sub-Hessian $\nabla^2_{b_k b_k} f(x)$ that must hold for *all* parameters x. For twice-differentiable non-quadratic functions, we could potentially make more progress by using classic Newton updates where we use the instantaneous Hessian $\nabla^2_{b_k b_k} f(x^k)$ with respect to the block. Indeed, considering the extreme case where we have one block containing all the coordinates, Newton updates can lead to superlinear convergence [Dennis and Moré, 1974] while matrix updates destroy this property. That being said, we should not expect superlinear convergence of BCD methods with Newton or even optimal updates.[18] Nevertheless, in Chapter 6 we show that for certain common problem structures it is possible to achieve superlinear convergence with Newton-style updates.

Fountoulakis & Tappenden recently highlight this difference between using matrix updates and using Newton updates [Fountoulakis and Tappenden, 2015], and propose a BCD method based on Newton updates. To guarantee progress when far from the solution classic Newton updates require safeguards like a line search or trust-region [Fountoulakis and Tappenden, 2015, Tseng and Yun, 2009b], but as we discussed in this section line searches tend not to add a significant cost to BCD methods. Thus, if we want to maximize the progress we make at each iteration we recommend to use one of the greedy rules to select the block to update, but then update the block using the Newton direction and a line search. In our implementation, we used a backtracking line search starting with $\alpha_k = 1$ and backtracking for the first time using quadratic Hermite polynomial interpolation and using cubic Hermite polynomial interpolation if we backtracked more than once (which rarely happened since $\alpha_k = 1$ or the first backtrack were typically accepted) [Nocedal and Wright, 1999].[19]

5.4 Message-Passing for Huge-Block Updates

Qu et al. [2016] discuss how in some settings increasing the block size with matrix updates does not necessarily lead to a performance gain due to the higher iteration cost. In the case of Newton updates the additional cost of computing the sub-Hessian $\nabla^2_{bb} f(x^k)$ may also be non-trivial. Thus, whether matrix and Newton updates will be beneficial over gradient updates will depend on the particular problem and the chosen block size. However, in this section we argue that in some cases matrix updates and Newton updates can be computed efficiently using huge blocks. In particular, we focus on the case where the dependencies between variables are sparse, and we will choose the structure of the blocks in order to guarantee that the matrix/Newton update can be computed efficiently.

[18] Consider a 2-variable quadratic objective where we use single-coordinate updates. The optimal update (which is equivalent to the matrix/Newton update) is easy to compute, but if the quadratic is non-separable then the convergence rate of this approach is only linear.

[19] We also explored a variant based on cubic regularization of Newton's method [Nesterov and Polyak, 2006], but were not able to obtain a significant performance gain with this approach.

The cost of using Newton updates with the BCD method depends on two factors: (i) the cost of calculating the sub-Hessian $\nabla^2_{b_k b_k} f(x^k)$ and (ii) the cost of solving the corresponding linear system. The cost of computing the sub-Hessian depends on the particular objective function we are minimizing. But, for the problems where coordinate descent is efficient (see Section 5.1.4), it is typically substantially cheaper to compute the sub-Hessian for a block than to compute the full Hessian. Indeed, for many cases where we apply BCD, computing the sub-Hessian for a block is cheap due to the sparsity of the Hessian. For example, in the graph-structured problems h_2 the edges in the graph correspond to the non-zeroes in the Hessian.

Although this sparsity and reduced problem size would seem to make BCD methods with exact Newton updates ideal, in the worst case the iteration cost would still be $O(|b_k|^3)$ using standard matrix factorization methods. A similar cost is needed using the matrix updates with fixed Hessian upper-bounds H_b and for performing an optimal update in the special case of quadratic functions. In some settings we can reduce this to $O(|b_k|^2)$ by storing matrix factorizations, but this cost is still prohibitive if we want to use large blocks (we can use $|b_k|$ in the thousands but not the millions).

An alternative to computing the exact Newton update is to use an approximation to the Newton update that has a runtime dominated by the sparsity level of the sub-Hessian. For example, we could use conjugate gradient methods or use randomized Hessian approximations [Dembo et al., 1982, Pilanci and Wainwright, 2017]. However, these approximations require setting an approximation accuracy and may be inaccurate if the sub-Hessian is not well-conditioned. In this section we consider an alternative approach: choosing blocks with a sparsity pattern that guarantees we can solve the resulting linear systems involving the sub-Hessian (or its approximation) in $O(|b_k|)$ using a "message-passing" algorithm. If the sparsity pattern is favourable, this allows us to update huge blocks at each iteration using exact matrix updates or Newton updates (which are the optimal updates for quadratic problems).

To illustrate the message-passing algorithm, we first consider the basic quadratic minimization problem

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} \frac{1}{2} x^T A x - c^T x,$$

where we assume the matrix $A \in \mathbb{R}^{n \times n}$ is positive-definite and sparse. By excluding terms not depending on the coordinates in the block, the optimal update for block b is given by the solution to the linear system

$$A_{bb} x_b = \tilde{c}, \tag{5.26}$$

where $A_{bb} \in \mathbb{R}^{|b| \times |b|}$ is the submatrix of A corresponding to block b, and $\tilde{c} = c_b - A_{b\bar{b}} x_{\bar{b}}$ is a vector with $\bar{b}$ defined as the complement of b and $A_{b\bar{b}} \in \mathbb{R}^{|b| \times |\bar{b}|}$ is the submatrix of A with rows from b and columns from $\bar{b}$. We note that in practice efficient BCD methods already need to track Ax so computing $\tilde{c}$ is efficient. Although we focus on solving (5.26) for simplicity, the message-passing solution we discuss here will also apply to matrix updates (which leads to a linear system involving H_b) and Newton updates (which leads to a linear system involving the

sub-Hessian).

Consider a pairwise undirected graph $G = (V, E)$, where the vertices V are the coordinates of our problem and the edges E are the non-zero off-diagonal elements of A. Thus, if A is diagonal then G has no edges, if A is dense then there are edges between all nodes (G is fully-connected), if A is tridiagonal then edges connect adjacent nodes (G is a chain-structured graph where $(1) - (2) - (3) - (4) - \dots$), and so on.

For BCD methods, unless we have a block size $|b| = n$, we only work with a subset of nodes b at each iteration. The graph obtained from the sub-matrix A_{bb} is called the *induced subgraph* G_b. Specifically, the nodes $V_b \in G_b$ are the coordinates in the set b, while the edges $E_b \in G_b$ are all edges $(i, j) \in E$ where $i, j \in V_b$ (edges between nodes in b). We are interested in the special case where the induced sub-graph G_b forms a *forest*, meaning that it has no cycles.[20] The idea of exploiting tree structures within BCD updates has previously been explored by Sontag and Jaakkola [2009]. However, unlike this work, where we use tree structured blocks to compute general Newton updates, Sontag and Jaakkola propose this idea in the context of linear programming.

In the special case of forest-structured induced subgraphs, we can compute the optimal update (5.26) in linear time using message passing [Shental et al., 2008] instead of the cubic worst-case time required by typical matrix factorization implementations. Indeed, in this case the message passing algorithm is equivalent to Gaussian elimination [Bickson, 2009, Prop. 3.4.1] where the amount of "fill-in" is guaranteed to be linear. This idea of exploiting tree structures within Gaussian elimination dates back over 50 years [Parter, 1961], and similar ideas have recently been explored by Srinivasan and Todorov [2015] for Newton methods. Their *graphical Newton* algorithm can solve the Newton system in $O(t^3)$ times the size of G, where t is the "treewidth" of the graph ($t = 1$ for forests). However, the tree-width of G is usually large while it is more reasonable to assume that we can find low-treewidth induced subgraphs G_b.

To illustrate the message-passing algorithm in the terminology of Gaussian elimination, we first need to divide the nodes $\{1, 2, \dots, |b|\}$ in the forest into sets $L\{1\}$, $L\{2\}, \dots, L\{T\}$, where $L\{1\}$ is an arbitrary node in graph G_b selected to be the root node, $L\{2\}$ is the set of all neighbours of the "root" node, $L\{3\}$ is the set of all neighbours of the nodes in $L\{2\}$ excluding parent nodes (nodes in $L\{1:2\}$), and so on until all nodes are assigned to a set (if the forest is made of disconnected trees, we need to do this for each tree). An example of this process is depicted in Figure 5.4. Once these sets are initialized, we start with the nodes furthest from the root node $L\{T\}$, and carry out the row operations of Gaussian elimination moving towards the root. Then we use backward substitution to solve the system $\tilde{A}x = \tilde{c}$. We outline the full procedure in Algorithm 1.

Whether or not message-passing is useful will depend on the sparsity pattern of A. Diagonal matrices correspond to disconnected graphs, which are clearly forests (they have no edges), and

[20] An undirected cycle is a sequence of adjacent nodes in V starting and ending at the same node, where there are no repetitions of nodes or edges other than the final node.

Algorithm 1 Message Passing for a Tree Graph

1. **Initialize**:
   Input: vector $\tilde{c}$, forest-structured matrix $\tilde{A}$, and levels $L\{1\}, L\{2\}, \ldots, L\{T\}$.
   for $i = 1, 2, \ldots, |b|$
   Set $P_{ii} \leftarrow \tilde{A}_{ii}$, $C_i \leftarrow \tilde{c}_i$. # P, C track row operations

2. **Gaussian Elimination**:
   for $t = T, T-1, \ldots, 1$ # start furthest from root
   for $i \in L\{t\}$
   if $t > 1$
   $J \leftarrow N\{i\} \backslash L\{1 : t-1\}$ # neighbours that are not parent node
   if $J = \emptyset$ # i corresponds to a leaf node
   continue # no updates
   else
   $J \leftarrow N\{i\}$ # root node has no parent node
   $P_{Ji} \leftarrow \tilde{A}_{Ji}$ # initialize off-diagonal elements
   $P_{ii} \leftarrow P_{ii} - \sum_{j \in J} \frac{P_{ji}^2}{P_{jj}}$ # update diagonal elements of P in $L\{t\}$
   $C_i \leftarrow C_i - \sum_{j \in J} \frac{P_{ji}}{P_{jj}} \cdot C_j$

3. **Backward Solve**:
   for $t = 1, 2, \ldots, T$ # start with root node
   for $i \in L\{t\}$
   if $t < T$
   $p \leftarrow N\{i\} \backslash L\{t+1 : T\}$ # parent node of i (empty for $t = 1$)
   else
   $p \leftarrow N\{i\}$ # only neighbour of leaf node is parent
   $x_i \leftarrow \frac{C_i - \tilde{A}_{ip} \cdot x_p}{P_{ii}}$ # solution to $\tilde{A}x = \tilde{c}$

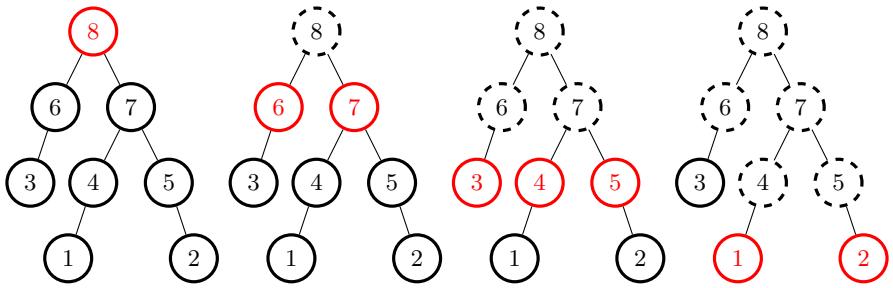


Figure 5.1: Process of partitioning nodes into level sets. For the above graph we have the following sets: $L\{1\} = \{8\}, L\{2\} = \{6, 7\}, L\{3\} = \{3, 4, 5\}$ and $L\{4\} = \{1, 2\}$.

thus we can use an optimal BCD update with a block size of n. Now consider a quadratic function with a lattice-structured non-zero pattern as in Figure 5.3. This graph is a bipartite

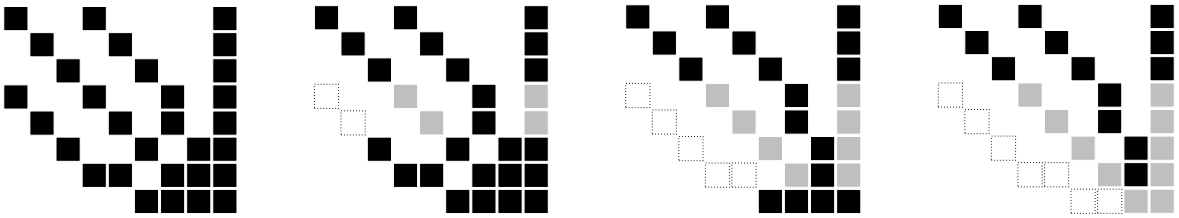
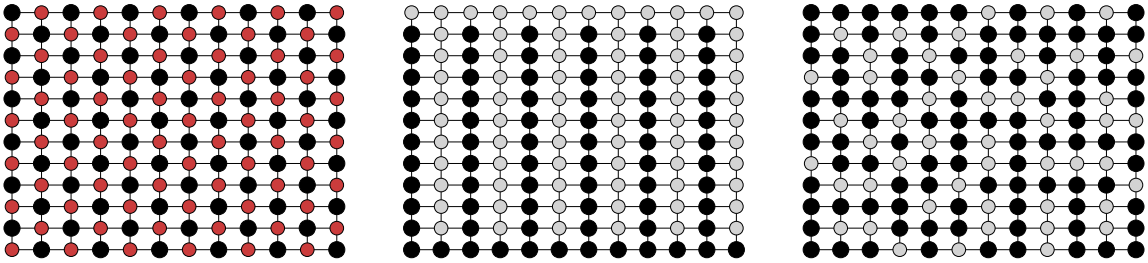


Figure 5.2: Illustration of Step 2 (row-reduction process) of Algorithm 1 for the tree in Figure 5.4. The matrix represents $[\tilde{A}|\tilde{c}]$. The black squares represent unchanged non-zero values of $\tilde{A}$ and the grey squares represent non-zero values that are updated at some iteration in Step 2. In the final matrix (far right), the values in the last column are the values assigned to the vector C in Steps 1 and 2 above, while the remaining columns that form an upper triangular matrix are the values corresponding to the constructed P matrix. The backward solve of Step 3 solves the linear system.



(a) Red-black, $|b| = n/2$. (b) Fixed block, $|b| = n/2$. (c) Variable block, $|b| \approx 2n/3$.

Figure 5.3: Partitioning strategies for defining forest-structured blocks.

graph, or a two-colourable graph, and a classic fixed partitioning strategy for problems with this common structure is to use a "red-black ordering" (see Figure 5.3a). Choosing this colouring makes the matrix A_{bb} diagonal when we update the red nodes (and similarly for the black nodes), allowing us to solve (5.26) in linear time. So this colouring scheme allows an optimal BCD update with a block size of $n/2$. The adjacency matrix of a graph like this is called consistently ordered. In general, an adjacency matrix is consistently ordered if the nodes of the corresponding graph can be partitioned into sets such that any two adjacent nodes belong to different, consecutive sets [Young, 1971, Def. 5.3.2]. In the case of red-black ordering, all odd numbered sets would make up one block and all even numbered sets would make up another block.

Message passing allows us to go beyond the red-black, and update *any* forest-structured block of size τ in linear time. For example, the partition given in Figure 5.3b also has blocks of size $n/2$ but these blocks include dependencies. Our experiments indicate that blocks that maintain dependencies, such as Figure 5.3b, make substantially more progress than using the red-black blocks or using smaller non-forest structured blocks. However, red-black blocks, or

more generally, consistently ordered matrices, are very well-suited for parallelization.

Unlike in Figure 5.3a, the colouring of a general graph or partitioning of a general adjacency matrix may lead to blocks of different sizes. Multi-colouring techniques are used to find graph colourings for general graphs, where colours are assigned to nodes such that no neighbouring nodes share the same colour [Saad, 2003, §12.4]. Finding the minimum number of "colours" (or the maximum size of blocks) for a given graph is exactly the NP-hard graph colouring problem. However, there are various heuristics that quickly give a non-minimal valid colouring of the nodes (for a survey of heuristic, meta-heuristic and hybrid methods for graph colouring, see Baghel et al. [2013]). If a graph is ν-colourable, then we can arrange the adjacency matrix such that it has ν diagonal blocks along its diagonal (the off diagonal blocks may be dense). In relation to BCD methods, this means that A_{bb} is diagonal for each block and we can update each of the ν blocks in linear time in the size of the block.

Alternatively, as we did for blocks of equal size in Figure 5.3b we can consider all blocks with a forest-structured induced subgraph, where the block size may vary at each iteration but restricting to forests still leads to a linear-time update. As seen in Figure 5.3c, by allowing variable block sizes we can select a forest-structured block of size $|b| \approx 2n/3$ in one iteration (black nodes) while still maintaining a linear-time update. If we further sample different random forests or blocks at each iteration, then the convergence rate under this strategy is covered by the arbitrary sampling theory [Qu et al., 2014]. Also note that the maximum of the gradient norms over all forests defines a valid norm, so our analysis of Gauss-Southwell can be applied to this case.

5.4.1 Partitioning into Forest-Structured Blocks

We can generalize the red-black approach to arbitrary graphs by defining our blocks such that no two neighbours are in the same block. While for lattice-structured graphs we only need two blocks to do this, for general graphs we may need a larger number of blocks. Finding the minimum number of blocks we need for a given graph is exactly the NP-hard graph colouring problem. Fortunately, there are various heuristics that quickly give a non-minimal valid colouring of the nodes. For example, in our experiments we used the following classic greedy algorithm [Welsh and Powell, 1967]:

1. Proceed through the vertices of the graph in some order $i = 1, 2, \dots, n$.
2. For each vertex i, assign it the smallest positive integer ("colour") such that it does not have the same colour as any of its neighbours among the vertices $\{1, 2, \dots, i-1\}$.

We can use all vertices assigned to the same integer as our blocks in the algorithm, and if we apply this algorithm to a lattice-structured graph (using row- or column-ordering of the nodes) then we obtain the classic red-black colouring of the graph.

Instead of disconnected blocks, in this work we instead consider forest-structured blocks. The size of the largest possible forest is related to the graph colouring problem [Esperet et al.,

2015], but we can consider a slight variation on the second step of the greedy colouring algorithm to find a set of forest-structured blocks:

1. Proceed through the vertices of the graph in some order $i = 1, 2, \ldots, n$.
2. For each vertex i, assign it the smallest positive integer ("forest") such that the nodes assigned to that integer among the set $\{1, 2, \ldots, i\}$ form a forest.

If we apply this to a lattice structured graph (in column-ordering), this generates a partition into two forest-structured graphs similar to the one in Figure 5.3b (only the bottom row is different). This procedure requires us to be able to test whether adding a node to a forest maintains the forest structure, and we show how to do this efficiently in Appendix D.4.

In the case of lattice-structured graph there is a natural ordering of the vertices, but for many graphs there is no natural ordering. In such we might simply consider a random ordering. Alternately, if we know the individual Lipschitz constants L_i, we could order by these values (with the largest L_i going first so that they are likely assigned to the same block if possible). In our experiments we found that this ordering improved performance for an unstructured dataset, and performed similarly to using the natural ordering in a lattice-structured dataset.

5.4.2 Approximate Greedy Rules with Forest-Structured Blocks

Similar to the problems of the previous section, computing the Gauss-Southwell rule over forest-structured variable blocks is NP-hard, as we can reduce the 3-satisfiability problem to the problem of finding a maximum-weight forest [Garey and Johnson, 1979]. However, we use a similar greedy method to approximate the greedy Gauss-Southwell rule over the set of trees:

1. Initialize b_k with the node i corresponding to the largest gradient, $|\nabla_i f(x^k)|$.
2. Search for the node i with the largest gradient that is not part of b_k and that maintains that b_k is a forest.
3. If such a node is found, add it to b_k and go back to step 2. Otherwise, stop.

Although this procedure does not yield the exact solution in general, it is appealing since (i) the procedure is efficient as it is easy to test whether adding a node maintains the forest property (see Appendix D.4), (ii) it outputs a forest so that the subsequent update is linear-time, (iii) we are guaranteed that the coordinate corresponding to the variable with the largest gradient is included in b_k, and (iv) we cannot add any additional node to the final forest and still maintain the forest property. A similar heuristic can be used to approximate the GSD rule under the restriction from Section 5.3.1 or to generate a forest randomly.

5.5 Numerical Experiments

We performed an extensive variety of experiments to evaluate the effects of the contributions listed in the previous sections. In this section we include several of these results that highlight

some key trends we observed, and in each subsection below we explicitly list the insights we obtained from the experiment. We considered five datasets that evaluate the methods in a variety of scenarios:

A Least-squares with a sparse data matrix.

B Binary logistic regression with a sparse data matrix.

C 50-class logistic regression problem with a dense data matrix.

D Lattice-structured quadratic objective as in Section 5.4.

E Binary label propagation problem (sparse but unstructured quadratic).

For interested readers, we give the full details of these datasets in Appendix D.5 where we have also included our full set of experiment results.

In our experiments we use the number of iterations as our measure of performance. This measure is far from perfect, especially when considering greedy methods, since it ignores the computational cost of each iteration. However, this measure of performance provides an problem- and implementation-independent measure of performance. We seek a problem-independent measure of performance since runtimes are highly dependent on the block size and the problem structure; as the block size grows, the extra computational cost of greedy methods may eventually be outweighed by the cost of computing the block update. We seek an implementation-independent measure of performance since the actual runtimes of different methods will vary wildly across applications. However, it is typically easy to estimate the per-iteration runtime when considering a new problem. Thus, we hope that our quantification of what can be gained from more-expensive methods gives guidance to readers about whether the more-expensive methods will lead to a performance gain on their applications. In any case, we are careful to qualify all of our claims with warnings in cases where the iteration costs differ.

5.5.1 Greedy Rules with Gradient Updates

Our first experiment considers gradient updates with a step-size of $1/L_b$, and seeks to quantify the effect of using fixed blocks compared to variable blocks (Section 5.2.1) as well as the effect of using the new GSL rule (Section 5.2.2). In particular, we compare selecting the block using Cyclic, Random, Lipschitz (sampling the elements of the block proportional to L_i), GS, and GSL rules. For each of these rules we implemented a fixed block (FB) and variable block (VB) variant. For VB using Cyclic selection, we split a random permutation of the coordinates into equal-sized blocks and updated these blocks in order (followed by using another random permutation). To approximate the seemingly-intractable GSL rule with VB, we used the GSD rule (Section 5.2.4) using the SIRT-style approximation (5.23) from Section 5.3.1. We used the bounds in Appendix D.2 to set the L_b values. To construct the partition of the coordinates

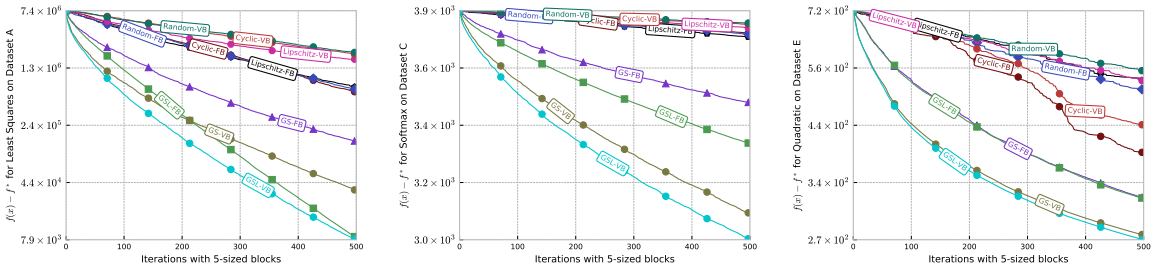


Figure 5.4: Comparison of different random and greedy block selection rules on three different problems when using gradient updates.

needed in the FB method, we sorted the coordinates according to their L_i values then placed the largest L_i values into the first block, the next set of largest in the second block, and so on.

We plot selected results in Figure 5.4, while experiments on all datasets and with other block sizes are given in Appendix D.5.2. Overall, we observed the following trends:

- **Greedy rules tend to outperform random and cyclic rules**, particularly with small block sizes. This difference is sometimes enormous, and this suggests we should prefer greedy rules when the greedy rules can be implemented with a similar cost to cyclic or random selection.

- **The variance of the performance between the rules becomes smaller as the block size increases**. This suggests that if we use very-large block sizes with gradient updates that we should prefer simple Cyclic or Random updates.

- **VB can substantially outperform FB when using GS** for certain problems. This is because FB are a subset of the VB, so we can make the progress bound better. Thus, we should prefer GS-VB for problems where this has a similar cost to GS-FB. We found *this trend was reversed for random rules*, where fixed blocks tended to perform better. We suspect this trend is due to the coupon collector problem: it takes FB fewer iterations than VB to select all variables at least once.

- **GSL consistently improved on the classic GS rule**, and in some cases the new rule with FB even outperformed the GS rule with VB. Interestingly, the performance gain was larger in the block case than in the single-coordinate case (see Section 2.8).

In Appendix D.5.2 we repeat this experiment for the FB methods but using the approximation to L_b discussed in Section 5.3.3. This sought to test whether this procedure, which may underestimate the true L_b and thus use larger step-sizes, would improve performance. This experiment lead to some additional insights:

- **Approximating L_b was more effective as the block size increases**. This makes sense, since with large block sizes there are more possible directions and we are unlikely to ever need to use a step-size as small as $1/L_b$ for the global L_b.

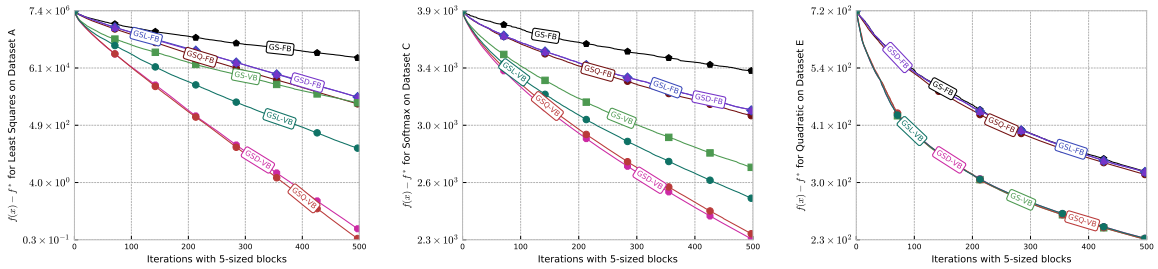


Figure 5.5: Comparison of different greedy block selection rules on three different problems when using matrix updates.

- **Approximating L_b is far more effective than using a loose bound**. We have relatively-good bounds for all problems except Problem C. On this problem the Lipschitz approximation procedure was much more effective even for small block sizes.

This experiment suggests that we should prefer to use an approximation to L_b (or an explicit line search) when using gradient updates unless we have a tight approximation to the true L_b *and* we are using a small block size. We also performed experiments with different block partitioning strategies for FB (see Appendix D.5.2). Although these experiments had some variability, we found that the block partitioning strategy did not make a large difference for cyclic and random rules. In contrast, when using greedy rules our sorting approach tended to outperform using random blocks or choosing the blocks to have similar average Lipschitz constants.

5.5.2 Greedy Rules with Matrix Updates

Our next experiment considers using matrix updates based on the matrices H_b from Appendix D.2, and quantifies the effects of the GSQ and GSD rules introduced in Sections 5.2.3-5.2.4 as well the approximations to these introduced in Sections 5.3.1-5.3.2. In particular, for FB we consider the GS rule and the GSL rule (from the previous experiment), the GSD rule (using the diagonal matrices from Section 5.3.1 with $D_{b,i} = L_i$), and the GSQ rule (which is optimal for the three quadratic objectives). For VB we consider the GS rule from the previous experiment as well as the GSD rule (using $D_{b,i} = L_i$), and the GSQ rule using the approximation from Section 5.3.2 and 10 iterations of iterative hard thresholding. Other than switching to matrix updates and focusing on these greedy rules, we keep all other experimental factors the same.

We plot selected results of doing this experiment in Figure 5.5. These experiments showed the following interesting trends:

- **There is a larger advantage to VB with matrix updates**. When using matrix updates, the basic GS-VB method outperformed even the most effective GSQ-FB rule for smaller block sizes.

- **There is little advantage to GSD/GSQ with FB**. Although the GSL rule consistently improved over the classic GS rule, we did not see any advantage to using the more-advanced GSD or GSQ rules when using FB.

- **GSD outperformed GS with VB**. Despite the use of a crude approximation to the GSD rule, the GSD rule consistently outperformed the classic GS rule.

- **GSQ slightly outperformed GSD with VB and large blocks**. Although the GSQ-VB rule performed the best across all experiments, the difference was more noticeable for large block sizes. However, this did not offset its high cost in any experiment. We also experimented with OMP instead of IHT, and found it gave a small improvement but the iterations were substantially more expensive.

Putting the above together, with matrix updates our experiments indicate that the GSL or GSD seem to both provide good performance for FB, while for VB the GSD rule should be preferred. We would only recommend using the GSQ rule in settings where we can use VB and where operations involving the objective f are much more expensive than running an IHT or OMP method. We performed experiments with different block partition strategies for FB, but found that when using matrix updates the partitioning strategy did not make a big difference for cyclic, random, or greedy rules.

In Appendix D.5.3 we repeat this experiment for the non-quadratic objectives using the Newton direction and a backtracking line search to set the step-size, as discussed in Sections 5.3.4 and 5.3.6. For both datasets, the Newton updates resulted in a significant performance improvement over the matrix updates. This indicates that we should prefer classic Newton updates over the more recent matrix updates for non-quadratic objectives where computing the sub-block of the Hessian is tractable.

5.5.3 Message-Passing Updates

We next seek to quantify the effect of using message-passing to efficiently implement exact updates for quadratic functions, as discussed in Section 5.4. For this experiment, we focused on the lattice-structured dataset D and the unstructured but sparse dataset E. These are both quadratic objectives with high treewidth, but that allow us to find large forest-structured induced subgraphs. We compared the following strategies to choose the block: greedily choosing the best general unstructured blocks using GS (General), cycling between blocks generated by the greedy graph colouring algorithm of Section 5.4.1 (Red Black), cycling between blocks generated by the greedy forest-partitioning algorithm of Section 5.4.1 (Tree Partitions), greedily choosing a tree using the algorithm of Section 5.4.2 (Greedy Tree), and growing a tree randomly using the same algorithm (Random Tree). For the lattice-structured Dataset D, the greedy partitioning algorithms proceed through the variables in order which generate partitions similar to those shown in in Figure 5.3b. For the unstructured Dataset E, we apply the greedy partitioning strategies of Section 5.4.1 using both a random ordering and by sorting the Lipschitz

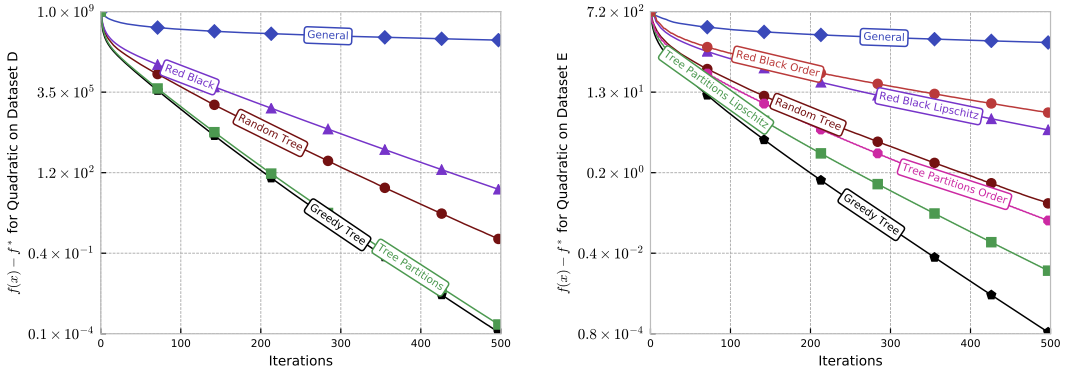


Figure 5.6: Comparison of different greedy block selection rules on two quadratic graph-structured problems when using optimal updates.

constants L_i. Since the cost of the exact update for tree-structured methods is $O(n)$, for the unstructured blocks we chose a block size of $b_k = n^{1/3}$ to make the costs comparable (since the exact solve is cubic in the block size for unstructured blocks).

We plot selected results of doing this experiment in Figure 5.6. Here, we see that even the classic red-black ordering outperforms using general unstructured blocks (since we must use such small block sizes). The tree-structured blocks perform even better, and in the unstructured setting our greedy approximation of the GS rule under variable blocks outperforms the other strategies. However, our greedy tree partitioning method also performs well. For the lattice-structured data (left) it performed similarly to the greedy approximation, while for the unstructured data (right) it outperformed all methods except greedy (and performed better when sorting by the Lipschitz constants than using a random order).

5.6 Discussion

In this chapter we focused on non-accelerated BCD methods. However, we expect that our conclusions are likely to also apply for accelerated BCD methods [Fercoq and Richtárik, 2015]. Similarly, while we focused on the setting of serial computation, we expect that our conclusions will give insight into developing more efficient parallel and distributed BCD methods [Richtárik and Takáč, 2016].

Although our experiments indicate that our choice of the diagonal matrices D within the GSD rule provides a consistent improvement, this choice is clearly sub-optimal. A future direction is to find a generic strategy to construct better diagonal matrices, and work on ESO methods could potentially be adapted for doing this [Qu and Richtárik, 2016]. This could be in the setting where we are given knowledge of the Lipschitz constants, but a more-interesting idea is to construct these matrices online as the algorithm runs.

The GSQ rule can be viewed as a greedy rule that incorporates more sophisticated second-

order information than the simpler GS and GSL rules. In preliminary experiments, we also considered selection rules based on the cubic regularization bound. However, these did not seem to converge more quickly than the existing rules in our experiments, and it is not obvious how one could efficiently implement such second-order rules.

We focused on BCD methods that approximate the objective function at each step by globally bounding higher-order terms in a Taylor expansion. However, we would expect more progress if we could bound these locally in a suitably-larger neighbourhood of the current iteration. Alternately, note that bounding the Taylor expansion is not the only way to upper bound a function. For example, Khan [2012] discusses a variety of strategies for bounding the binary logistic regression loss and indeed proves that other bounds are tighter than the Taylor expansion ("Bohning") bound that we use. It would be interesting to explore the convergence properties of BCD methods whose bounds do not come from a Taylor expansion.

While we focused on the case of trees, there are message-passing algorithms that allow graphs with cycles [Rose, 1970, Srinivasan and Todorov, 2015]. The efficiency of these methods depends on the "treewidth" of the induced subgraph, where if the treewidth is small (as in trees) then the updates are efficient, and if the treewidth is large (as in fully-connected graphs) then these do not provide an advantage. Treewidth is related to the notion of "chordal" graphs (trees are special cases of chordal graphs) and chordal embeddings which have been exploited for matrix problems like covariance estimation [Dahl et al., 2008] and semidefinite programming [Sun et al., 2014, Vandenberghe and Andersen, 2015]. Considering "treewidth 2" or "treewidth 3" blocks would give more progress than our tree-based updates, although it is NP-hard to compute the treewidth of a general graph (but it is easy to upper-bound this quantity by simply choosing a random elimination order).

As opposed to structural constraints like requiring the graph to be a tree, it is now known that message-passing algorithms can solve linear systems with other properties like diagonal dominance or "attractive" coefficients [Malioutov et al., 2006]. There also exist specialized linear-time solvers for Laplacian matrices [Kyng and Sachdeva, 2016], and it would be interesting to explore BCD methods based on these structures. It would also be interesting to explore whether approximate message-passing algorithms which allow general graphs [Malioutov et al., 2006] can be used to improve optimization algorithms.

Chapter 6

Active-Set Identification and Complexity

In this section we consider optimization problems of the form

$$\underset{x\in\mathbb{R}^n}{\operatorname{argmin}} f(x) + \sum_{i=1}^{n} g_i(x_i), \tag{6.1}$$

where ∇f is Lipschitz-continuous, that is for all $x, y \in \mathbb{R}^n$, we have

$$\|\nabla f(y) - \nabla f(x)\| \leq L\|y - x\|, \tag{6.2}$$

and each g_i only needs to be convex and lower semi-continuous (it may be non-smooth or infinite at some x_i). A classic example of a problem in this framework is optimization subject to non-negative constraints,

$$\underset{x\geq 0}{\operatorname{argmin}} f(x), \tag{6.3}$$

where in this case g_i is the indicator function on the non-negative orthant,

$$g_i(x_i) = \begin{cases} 0 & \text{if } x_i \geq 0, \\ \infty & \text{if } x_i < 0. \end{cases}$$

Another example that has received significant recent attention is the case of an ℓ_1-regularizer,

$$\underset{x\in\mathbb{R}^n}{\operatorname{argmin}} f(x) + \lambda\|x\|_1, \tag{6.4}$$

where in this case $g_i(x_i) = \lambda|x_i|$. Here, the ℓ_1-norm regularizer is used to encourage sparsity in the solution. A related problem is the group ℓ_1-regularization problem (5.6), where instead of being separable, g is block-separable.

Proximal gradient methods have become one of the default strategies for solving problem (6.1). Given the separability assumption we make on g in (6.1), the proximal gradient update is separable. The coordinate-wise proximal gradient update (using a step-size of $1/L$) is given by

$$x_i^{k+1} = \operatorname{prox}_{\frac{1}{L}g_i}\left(x_i^k - \frac{1}{L}\nabla_i f(x^k)\right), \tag{6.5}$$

where the coordinate-wise proximal operator is defined as

$$\text{prox}_{\frac{1}{L}g_i}(y) = \underset{x_i \in \mathbb{R}}{\text{argmin}} \frac{1}{2}|x_i - y|^2 + \frac{1}{L}g_i(x_i).$$

Whether we are using the coordinate-wise or full proximal gradient method, this update form holds for the individual coordinates.

In the special case of non-negative constraints like (6.3) the update in (6.5) is given by

$$\begin{aligned} x_i^{k+\frac{1}{2}} &= x_i - \frac{1}{L}\nabla_i f(x^k) \\ x_i^{k+1} &= \left[x_i^{k+\frac{1}{2}}\right]^+, \end{aligned}$$

which we have written as a gradient update followed by the projection $[\beta]^+ = \max\{0, \beta\}$ onto the non-negative orthant (see Figure 6.1a). For ℓ_1-regularization problems (6.4) the update reduces to an element-wise soft-thresholding step,

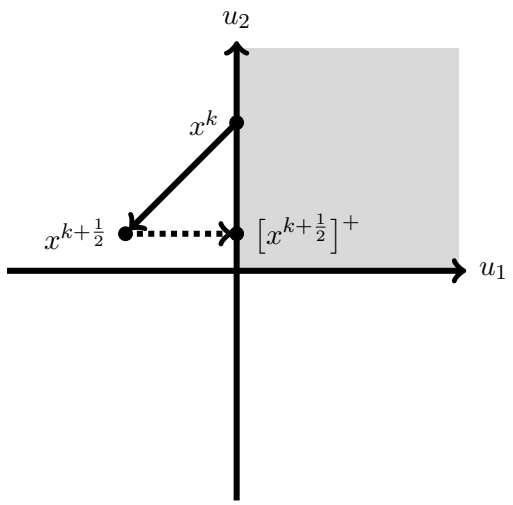
$$\begin{aligned} x_i^{k+\frac{1}{2}} &= x^k - \frac{1}{L}\nabla_i f(x^k), \\ x_i^{k+1} &= \frac{x_i^{k+\frac{1}{2}}}{\left|x_i^{k+\frac{1}{2}}\right|}\left[\left|x_i^{k+\frac{1}{2}}\right| - \frac{\lambda}{L}\right]^+, \end{aligned} \tag{6.6}$$

which we have written as a gradient update followed by the soft-threshold operator (shown in Figure 6.1b).

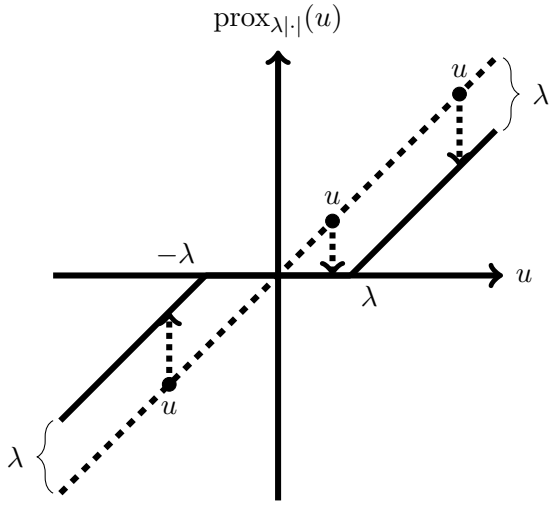
It has been established that (block) coordinate descent methods based on the update (6.5) for problem (6.1) obtain similar convergence rates to the case where we do not have a non-smooth term g (see Section 2.7 and [Nesterov, 2012, Richtárik and Takáč, 2014]). The main focus of this chapter is to show that *the non-smoothness of g can actually lead to a faster convergence rate*.

This idea dates back at least 40 years to the work of Bertsekas [1976].[21] For the case of non-negative constraints, he shows that the sparsity pattern of x^k generated by the projected-gradient method matches the sparsity pattern of the solution x^* for all sufficiently large k. Thus, after a finite number of iterations the projected-gradient method will "identify" the final set of non-zero variables. Once these values are identified, Bertsekas suggests that we can fix the zero-valued variables and apply an unconstrained Newton update to the set of non-zero variables to obtain superlinear convergence. Even without switching to a superlinearly-convergent method, the convergence rate of the projected-gradient method can be faster once the set of non-zeroes is identified since it is effectively optimizing in the lower-dimensional space corresponding to the non-zero variables.

[21] A similar property was shown for proximal point methods in a more general setting around the same time, [Rockafellar, 1976].



(a) For problem (6.3) each iteration of the proximal gradient method takes a gradient descent step, $x^{k+\frac{1}{2}}$, and then projects this point onto the non-negative orthant.



(b) For problem (6.4), the proximal operator or "soft-threshold" operator shrinks the value of u by the regularization constant, λ. If $|u| > \lambda$, then the resulting value is $u - \text{sign}(u) \cdot \lambda$. Otherwise, if $|u| \leq \lambda$, then the proximal operator sets u to 0.

Figure 6.1: Visualization of (a) the proximal gradient update for a non-negatively constrained optimization problem (6.3); and (b) the proximal operator (soft-threshold) used in the proximal gradient update for an ℓ_1-regularized optimization problem (6.4).

This idea of identifying a smooth "manifold" containing the solution x^* has been generalized to allow polyhedral constraints [Burke and Moré, 1988], general convex constraints [Wright, 1993], and even non-convex constraints [Hare and Lewis, 2004]. Similar results exist in the proximal gradient setting. For example, it has been shown that the proximal gradient method identifies the sparsity pattern in the solution of ℓ_1-regularized problems after a finite number of iterations [Hare, 2011]. The active-set identification property has also been shown for other algorithms like certain coordinate descent and stochastic gradient methods [Lee and Wright, 2012, Mifflin and Sagastizábal, 2002, Wright, 2012]. Specifically, Wright shows that BCD also has this manifold identification property for separable g [Wright, 2012], provided that the coordinates are chosen in an essentially-cyclic way (or provided that we can simultaneously choose to update all variables that do not lie on the manifold). Wright also shows that superlinear convergence is possible if we use a Newton update on the manifold, assuming the Newton update does not leave the manifold.

In this chapter, we show active-set identification for the full proximal gradient method. This is a well-known characteristic of the full proximal gradient method but given our assumption on the separability of g our proof analysis is much simpler than these existing analyses. We then extend this result to the proximal coordinate descent case for general separable g. We follow a similar argument to Bertsekas [1976], which yields a simple proof that holds for many possible selection rules including greedy rules (which may not be essentially-cyclic). When using greedy BCD methods with variable blocks we show this leads to superlinear convergence for

problems with sufficiently-sparse solutions (when we use updates incorporating second-order information). In the special case of LASSO and SVM problems, we further show that optimal updates are possible. This leads to finite convergence for SVM and LASSO problems with sufficiently-sparse solutions when using greedy selection and sufficiently-large variable blocks.

Most prior works show the active-set identification happens asymptotically. In Section 6.3, we introduce the notion of the "active-set complexity" of an algorithm, which we define as the number of iterations required before an algorithm is guaranteed to have reached the active-set. Our active-set identification arguments lead to bounds on the *active-set complexity* of the full and BCD variants of the proximal gradient method. We are only aware of one previous work giving such bounds, the work of Liang et al. who included a bound on the active-set complexity of the proximal gradient method [Liang et al., 2017, Proposition 3.6]. Unlike this work, their result does not evoke strong-convexity. Instead, their work applies an inclusion condition on the local subdifferential of the regularization term that ours does not require. By focusing on the strongly-convex case in Section 6.3 (which is common in machine learning due to the use of regularization), we obtain a simpler analysis and a much tighter bound than in this previous work. Specifically, both rates depend on the "distance to the subdifferential boundary", but in our analysis this term only appears inside of a logarithm rather than outside of it. As examples, we consider problems (6.3) and (6.4), and show explicit bounds for the active-set complexity in both the full and BCD proximal gradient methods.

6.1 Notation and Assumptions

By our separability assumption on g, the subdifferential of g can be expressed as the concatenation of the individual subdifferential of each g_i, where the subdifferential of g_i at any $x_i \in \mathbb{R}$ is defined by

$$\partial g_i(x_i) = \{v \in \mathbb{R} : g_i(y) \geq g_i(x_i) + v \cdot (y - x_i), \text{ for all } y \in \mathbf{dom}\ g_i\}.$$

This implies that the subdifferential of each g_i is just an interval on the real line. In particular, the interior of the subdifferential of each g_i at a non-differentiable point x_i can be written as an open interval,

$$\operatorname{int} \partial g_i(x_i) \equiv (l_i, u_i), \tag{6.7}$$

where $l_i \in \mathbb{R} \cup \{-\infty\}$ and $u_i \in \mathbb{R} \cup \{\infty\}$ (the ∞ values occur if x_i is at its lower or upper bound, respectively). The *active-set* at a solution x^* for a separable g is then defined by

$$\mathcal{Z} = \{i : \partial g_i(x_i^*) \text{ is not a singleton}\}.$$

By (6.7), the set $\mathcal{Z}$ includes indices i where x_i^* is equal to the lower bound on x_i, is equal to the upper bound on x_i, or occurs at a non-smooth value of g_i. In our examples of non-negative constraints or ℓ_1-regularization, $\mathcal{Z}$ is the set of coordinates that are zero at the solution x^*.

With this definition, we can formally define the manifold identification property.

Definition 1. *The manifold identification property for problem* (6.1) *is satisfied if for all sufficiently large k, we have that $x_i^k = x_i^*$ for some solution x^* for all $i \in \mathcal{Z}$.*

In order to prove the manifold identification property, in addition to assuming that ∇f is L-Lipschitz continuous (6.2), we require two assumptions. Our first assumption is that the iterates of the algorithm converge to a solution x^*.

Assumption 1. *The iterates converge to an optimal solution x^* of problem* (6.1)*, that is $x^k \to x^*$ as $k \to \infty$.*

Our second assumption is a *nondegeneracy* condition on the solution x^* that the algorithm converges to. Below we write the standard nondegeneracy condition from the literature for our special case of (6.1).

Assumption 2. *We say that x^* is a nondegenerate solution for problem* (6.1) *if it holds that*

$$\begin{cases} -\nabla_i f(x^*) = \nabla_i g(x_i^*) & \text{if } \partial g_i(x_i^*) \text{ is a singleton } (g_i \text{ is smooth at } x_i^*) \\ -\nabla_i f(x^*) \in int\, \partial g_i(x_i^*) & \text{if } \partial g_i(x_i^*) \text{ is not a singleton } (g_i \text{ is non-smooth at } x_i^*). \end{cases}$$

This condition states that $-\nabla f(x^*)$ must be in the "relative interior" (see [Boyd and Vandenberghe, 2004, Section 2.1.3]) of the subdifferential of g at the solution x^*. In the case of the non-negative bound constrained problem (6.3), this requires that $\nabla_i f(x^*) > 0$ for all variables i that are zero at the solution ($x_i^* = 0$). For the ℓ_1-regularization problem (6.4), this requires that $|\nabla_i f(x^*)| < \lambda$ for all variables i that are zero at the solution.[22]

6.2 Manifold Identification for Separable g

In this section we show that the full and BCD variants of the proximal gradient method identify the active-set of (6.1) in a finite number of iterations. Although this result follows from the more general results in the literature for the full proximal gradient method, by focusing on (6.1) we give a substantially simpler proof that will allow us to bound the active-set iteration complexity of the method.

6.2.1 Proximal Gradient Method

We first note that if we assume f is strongly convex, then the iterates converge to a (unique) solution x^* with a linear rate [Schmidt et al., 2011],

$$\|x^k - x^*\| \leq \left(1 - \frac{1}{\kappa}\right)^k \|x^0 - x^*\|, \tag{6.8}$$

[22]Note that $|\nabla_i f(x^*)| \leq \lambda$ for all i with $x_i^* = 0$ follows from the optimality conditions, so this assumption simply rules out the case where $|\nabla_i f(x_i^*)| = \lambda$. We note that in this case the nondegeneracy condition is a strict complementarity condition [De Santis et al., 2016].

where κ is the condition number of f. This implies that Assumption 1 holds. We give a simple result that follows directly from Assumption 1 and establishes that for any $\beta > 0$ there exists a finite iteration $\bar{k}$ such that the distance from the iterate x^k to the solution x^* for all iterations $k \geq \bar{k}$ is bounded above by β.

Lemma 1. *Let Assumption 1 hold. For any β, there exists some minimum finite $\bar{k}$ such that $\|x^k - x^*\| \leq \beta$ for all $k \geq \bar{k}$.*

An important quantity in our analysis is the minimum distance to the nearest boundary of the subdifferential (6.7) among indices $i \in \mathcal{Z}$. This quantity is given by

$$\delta = \min_{i \in \mathcal{Z}} \left\{ \min\{-\nabla_i f(x^*) - l_i, u_i + \nabla_i f(x^*)\} \right\}. \tag{6.9}$$

Our argument essentially states that once Lemma 1 is satisfied for some finite $\bar{k}$ and a particular $\beta > 0$, then at this point the algorithm always sets x_i^k to x_i^* for all $i \in \mathcal{Z}$. In the next result, we prove that this happens for a value β depending on δ as defined in (6.9).

Lemma 2. *Consider problem (6.1), where f is convex with L-Lipschitz continuous gradient and the g_i are proper convex functions (not necessarily smooth). Let Assumption 1 be satisfied and Assumption 2 be satisfied for the particular x^* that the algorithm converges to. Then for the proximal gradient method with a step-size of $1/L$ there exists a $\bar{k}$ such that for all $k > \bar{k}$ we have $x_i^k = x_i^*$ for all $i \in \mathcal{Z}$.*

Proof. By the definition of the proximal gradient step and the separability of g, for all i we have

$$x_i^{k+1} \in \operatorname*{argmin}_y \left\{ \frac{1}{2} \left| y - \left(x_i^k - \frac{1}{L} \nabla_i f(x^k) \right) \right|^2 + \frac{1}{L} g_i(y) \right\}.$$

This problem is strongly-convex, and its unique solution satisfies

$$0 \in y - x_i^k + \frac{1}{L} \nabla_i f(x^k) + \frac{1}{L} \partial g_i(y),$$

or equivalently that

$$L(x_i^k - y) - \nabla_i f(x^k) \in \partial g_i(y). \tag{6.10}$$

By Lemma 1, there exists a minimum finite iterate $\bar{k}$ such that $\|x^{\bar{k}} - x^*\| \leq \delta/2L$. Since $|x_i^k - x_i^*| \leq \|x^k - x^*\|$, this implies that for all $k \geq \bar{k}$ we have

$$-\delta/2L \leq x_i^k - x_i^* \leq \delta/2L, \quad \text{for all } i. \tag{6.11}$$

Further, the Lipschitz continuity of ∇f in (6.2) implies that we also have

$$\begin{aligned}|\nabla_i f(x^k) - \nabla_i f(x^*)| &\leq \|\nabla f(x^k) - \nabla f(x^*)\| \\ &\leq L\|x^k - x^*\| \\ &\leq \delta/2,\end{aligned}$$

which implies that

$$-\delta/2 - \nabla_i f(x^*) \leq -\nabla_i f(x^k) \leq \delta/2 - \nabla_i f(x^*). \tag{6.12}$$

To complete the proof it is sufficient to show that for any $k \geq \bar{k}$ and $i \in \mathcal{Z}$ that $y = x_i^*$ satisfies (6.10). Since the solution to (6.10) is unique, this will imply the desired result. We first show that the left-side is less than the upper limit u_i of the interval $\partial g_i(x_i^*)$,

$$\begin{aligned}L(x_i^k - x_i^*) - \nabla_i f(x^k) &\leq \delta/2 - \nabla_i f(x^k) && \text{(right-side of (6.11))} \\ &\leq \delta - \nabla_i f(x^*) && \text{(right-side of (6.12))} \\ &\leq (u_i + \nabla_i f(x^*)) - \nabla_i f(x^*) && \text{(definition of } \delta\text{, (6.9))} \\ &\leq u_i.\end{aligned}$$

We can use the left-sides of (6.11) and (6.12) and an analogous sequence of inequalities to show that $L(x_i^k - x_i^*) - \nabla_i f(x^k) \geq l_i$, implying that x_i^* solves (6.10). □

Both problems (6.3) and (6.4) satisfy the manifold identification result. By the definition of δ in (6.9), we have that $\delta = \min_{i \in \mathcal{Z}}\{\nabla_i f(x^*)\}$ for problem (6.3). We note that if $\delta = 0$, then we may approach the manifold through the interior of the domain and the manifold may never be identified (this is the purpose of the nondegeneracy condition). For problem (6.4), we have that $\delta = \lambda - \max_{i \in \mathcal{Z}}\{|\nabla_i f(x^*)|\}$. From these results, we are able to define explicit bounds on the number of iterations required to reach the manifold, a new result that we explore in Section 6.3.

6.2.2 Proximal Coordinate Descent Method

We note that Assumption 1 holds for the proximal coordinate descent method if we assume that f is strongly convex and that we use cyclic or greedy selection. Specifically, by existing works on cyclic [Beck and Tetruashvili, 2013] and greedy selection (Section 2.7) of i_k within proximal coordinate descent methods, we have that

$$F(x^k) - F(x^*) \leq \rho^k[F(x^0) - F(x^*)], \tag{6.13}$$

for some $\rho < 1$ when f is strongly convex. Note that strong convexity of f implies the strong convexity of F, so we have

$$F(y) \geq F(x) + \langle s, y - x \rangle + \frac{\mu}{2}\|y - x\|^2,$$

where μ is the strong convexity constant of f and s is any subgradient of F at x. Taking $y = x^k$ and $x = x^*$ we obtain that

$$F(x^k) \geq F(x^*) + \frac{\mu}{2}\|x^k - x^*\|^2, \tag{6.14}$$

which uses that $0 \in \partial F(x^*)$. Thus we have that

$$\|x^k - x^*\|^2 \leq \frac{2}{\mu}[F(x^k) - F(x^*)] \leq \frac{2}{\mu}\rho^k[F(x^0) - F(x^*)], \tag{6.15}$$

which implies Assumption 1. However, Assumption 1 will also hold under a variety of other scenarios.

There are three results that we require in order to prove the manifold identification property for proximal coordinate descent methods. The first result is Lemma 1 and follows directly from Assumption 1. The second result we require is that for any $i \in \mathcal{Z}$ such that $x_i^k \neq x_i^*$, eventually coordinate i is selected at some finite iteration.

Lemma 3. *Let Assumption 1 hold. If $x_i^k \neq x_i^*$ for some $i \in \mathcal{Z}$, then coordinate i will be selected by the proximal coordinate descent method after a finite number of iterations.*

Proof. For eventual contradiction, suppose we did not select such an i after iteration k'. Then for all $k \geq k'$ we have that

$$|x_i^{k'} - x_i^*| = |x_i^k - x_i^*| \leq \|x^k - x^*\|. \tag{6.16}$$

By Assumption 1 the right-hand side is converging to 0, so it will eventually be less than $|x_i^{k'} - x_i^*|$ for some $k \geq k'$, contradicting the inequality. Thus after a finite number of iterations we must have that $x_i^k \neq x_i^{k'}$, which can only be achieved by selecting i. □

The third result we require is an adaptation of Lemma 2 to the proximal coordinate descent setting. It states that once Lemma 1 is satisfied for some finite $\bar{k}$ and a particular $\beta > 0$ (depending on δ), then for the coordinate $i \in \mathcal{Z}$ selected at some iteration $k' \geq \bar{k}$ by the proximal coordinate descent method, we have $x_i^{k'} = x_i^*$.

Lemma 4. *Consider problem* (6.1)*, where f is convex with L-Lipschitz continuous gradient and the g_i are proper convex functions (not necessarily smooth). Let Assumption 1 be satisfied and Assumption 2 be satisfied for the particular x^* that the algorithm converges to. Then for the proximal coordinate descent method with a step-size of $1/L$, if $\|x^k - x^*\| \leq \delta/2L$ holds and $i \in \mathcal{Z}$ is selected at iteration k, then $x_i^{k+1} = x_i^*$.*

Proof. The proof is identical to Lemma 2, but restricting to the update of the single coordinate. □

With the above results we next have the manifold identification property for the proximal coordinate descent method.

Theorem 9. *Consider problem* (6.1)*, where f is convex with L-Lipschitz continuous gradient and the g_i are proper convex functions. Let Assumption 1 be satisfied and Assumption 2 be satisfied for the particular x^* that the algorithm converges to. Then for the proximal coordinate descent method with a step-size of $1/L$ there exists a finite k such that $x_i^k = x_i^*$ for all $i \in \mathcal{Z}$.*

Proof. Lemma 1 implies that the assumptions of Lemma 4 are eventually satisfied, and combining this with Lemma 3 we have our result. □

While the above result considers single-coordinate updates, it can trivially be modified to show that the proximal BCD method has the manifold identification property. The only change is that once $\|x^k - x^*\| \leq \delta/2L$, we have that $x_i^{k+1} = x_i^*$ for all $i \in b_k \cap \mathcal{Z}$. Thus, BCD methods can simultaneously move many variables onto the optimal manifold. In Section 6.4 we show how the active-set identification results presented in this section can lead to superlinear or finite convergence of proximal BCD methods.

Instead of using a step-size of $1/L$, it is more common to use a bigger step-size of $1/L_i$ within coordinate descent methods, where L_i is the coordinate-wise Lipschitz constant. In this case, the results of Lemma 4 hold for $\beta = \delta/(L + L_i)$. This is a larger region since $L_i \leq L$, so with this standard step-size the iterates can move onto the manifold from further away and we expect to identify the manifold earlier. The argument can also be modified to use other step-size selection methods, provided that we can write the algorithm in terms of a step-size α_k that is guaranteed to be bounded from below.

6.3 Active-Set Complexity

The manifold identification property presented in the previous section can be shown using the more sophisticated tools of related works [Burke and Moré, 1988, Hare and Lewis, 2004]. However, an appealing aspect of the simple argument in Section 6.2 is that it can be combined with non-asymptotic convergence rates of the iterates to bound the *number of iterations required to reach the manifold*. We call this the "active-set complexity" of the method. Given any method with an iterate bound of the form,

$$\|x^k - x^*\| \leq \gamma \left(1 - \frac{1}{\kappa}\right)^k, \tag{6.17}$$

for some $\kappa \geq 1$, the next result uses that $(1 - 1/\kappa)^k \leq \exp(-k/\kappa)$ to bound the number of iterations it will take to identify the active-set, and thus reach the manifold.

Theorem 10. *Consider any method that achieves an iterate bound* (6.17)*. For* δ *as defined in* (6.9)*, we have* $\|x^{\bar{k}} - x^*\| \leq \delta/2L$ *after at most* $\kappa \log(2L\gamma/\delta)$ *iterations. Further, we will identify the active-set after an additional* t *iterations, where* t *is the number of additional iterations required to select all suboptimal* x_i *with* $i \in \mathcal{Z}$*.*

For the full proximal gradient method, if we assume f is strongly convex, then by the rate in 6.8 we have that $\gamma = \|x^0 - x^*\|$ and κ is the condition number of f. Further, all variables are updated at each iteration so it will only take a single additional iteration ($t = 1$) to ensure all suboptimal x_i with $i \in \mathcal{Z}$ are optimal. Therefore, we will identify the active-set after at most $\kappa \log(2L\|x^0 - x^*\|/\delta)$ iterations. Nutini et al. [2017b] extend these results to show active-set identification and active-set complexity for the full proximal gradient method when using a general constant step-size. Their results include proving a generalized convergence rate bound for the proximal gradient method.

For the proximal BCD case, if we assume that f is strongly convex and that we are using cyclic or greedy selection, then we are guaranteed the linear convergence rate in (6.15) for $\gamma = \frac{2}{\mu}[F(x^0) - F(x^*)]$ and some $\kappa \geq 1$. (We note that this type of rate also holds for a variety of other types of selection rules). Unlike the full proximal gradient case, the active-set complexity is complicated by the fact that not all coordinates are updated on each iteration for proximal BCD methods; the value of t depends on the selection rule we use. If we use cyclic selection we will require at most $t = n$ additional iterations to select all suboptimal coordinates $i \in \mathcal{Z}$ and thus, to reach the optimal manifold. To bound the active-set complexity for general rules like greedy rules, we cannot guarantee that all coordinates will be selected after n iterations once we are close to the solution. In the case of non-negative constraints (6.3), the number of additional iterations depends on a quantity we will call ϵ, which is the smallest non-zero variable $x_i^{\bar{k}}$ for $i \in \mathcal{Z}$ and $\bar{k}$ satisfying the first part of Theorem 10. It follows from (6.15) that we require at most $\kappa \log(\gamma/\epsilon)$ iterations beyond $\bar{k}$ to select all non-zero $i \in \mathcal{Z}$. Thus, the active-set complexity for greedy rules for problem (6.3) is bounded above by $\kappa(\log(2L\gamma/\delta) + \log(\gamma/\epsilon))$. Based on this bound, greedy rules (which yield a smaller κ) may identify the manifold more quickly than cyclic rules in cases where ϵ is large. However, if ϵ is very small then greedy rules may take a larger number of iterations to reach the manifold.[23]

Finally, it is interesting to note that the bound we prove in Theorem 10 only depends logarithmically on $1/\delta$, and that if δ (as defined in (6.9)) is quite large then we can expect to identify the active-set very quickly. This $O(\log(1/\delta))$ dependence is in contrast to the previous result of Liang et al. who give a bound of the form $O(1/\sum_{i=1}^n \delta_i^2)$ where δ_i is the distance of $\nabla_i f$ to the boundary of the subdifferential ∂g_i at x^* [Liang et al., 2017, Proposition 3.6]. Thus, our bound is much tighter as it only depends logarithmically on the single largest δ_i (though we make the extra assumption of strong-convexity).

[23] If this is a concern, the implementer could consider a safeguard ensuring that the method is essentially-cyclic. Alternately, we could consider rules that prefer to include variables that are near the manifold and have the appropriate gradient sign.

6.4 Superlinear and Finite Convergence of Proximal BCD

Most of the issues discussed in Chapter 5 for smooth BCD methods carry over in a straightforward way to the proximal setting; we can still consider fixed or variable blocks, there exist matrix and Newton updates, and we can still consider cyclic, random, or greedy selection rules. One subtle issue is that, as presented in Section 2.7, there are many generalizations of the GS rule to the proximal setting. However, the GS-q rule defined by Tseng and Yun [2009a] seems to be the generalization of GS with the best theoretical properties. A GSL variant of this rule in the notation of Chapter 5 would take the form

$$b_k \in \underset{b\in\mathcal{B}}{\operatorname{argmin}} \left\{ \min_d \left\{ \langle \nabla_b f(x^k), d\rangle + \frac{L_b}{2}\|d\|^2 + \sum_{i\in b} g_i(x_i + d_i) - \sum_{i\in b} g_i(x_i) \right\} \right\}, \tag{6.18}$$

where we assume that the gradient of f is L_b-Lipschitz continuous with respect to block b. A generalization of the GS rule is obtained if we assume that the L_b are equal across all blocks.

In the next three subsections, we discuss the consequences of our active-set identification results when using the proximal BCD method. Specifically, once we have identified the active-set, we can achieve superlinear convergence for certain problems with sparse solutions (and in some cases finite termination at an optimal solution).

6.4.1 Proximal-Newton Updates and Superlinear Convergence

Once we have identified the optimal manifold, we can think about switching from using the proximal BCD method to using an unconstrained optimizer on the coordinates $i \notin \mathcal{Z}$. The unconstrained optimizer can be a Newton update, and thus under the appropriate conditions can achieve superlinear convergence. However, a problem with such "2-phase" approaches is that we do not know the exact time at which we reach the optimal manifold. This can make the approach inefficient: if we start the second phase too early, then we sub-optimize over the wrong manifold, while if we start the second phase too late, then we waste iterations performing first-order updates when we should be using second-order updates. Wright proposes an interesting alternative where at each iteration we consider replacing the proximal gradient block update with a Newton block update on the current manifold [Wright, 2012]. This has the advantage that the manifold can continue to be updated, and that Newton updates are possible as soon as the optimal manifold has been identified. However, note that the dimension of the current manifold might be larger than the block size and the dimension of the optimal manifold, so this approach can significantly increase the iteration cost for some problems.

Rather than "switching" to an unconstrained Newton update, we can alternately take advantage of the superlinear converge of proximal-Newton updates [Lee et al., 2012]. For example, in this section we consider Newton proximal-BCD updates as in several recent works [Fountoulakis and Tappenden, 2015, Qu et al., 2016, Tappenden et al., 2016]. For a block b these

updates have the form

$$x_b^{k+1} \in \underset{y \in \mathbb{R}^{|b|}}{\operatorname{argmin}} \left\{ \langle \nabla_b f(x_b^k), y - x_b^k \rangle + \frac{1}{2\alpha_k} \|y - x_b^k\|_{H_b^k}^2 + \sum_{i \in b} g_i(y_i) \right\}, \tag{6.19}$$

where H_b^k is the matrix corresponding to block b at iteration k (which can be the sub-Hessian $\nabla_{bb}^2 f(x^k)$) and α_k is the step-size. As before if we set $H_b^k = H_b$ for some fixed matrix H_b, then we can take $\alpha_k = 1$ if block b of f is 1-Lipschitz continuous in the H_b-norm.

In the next section, we give a practical variant on proximal-Newton updates that also has the manifold identification property under standard assumptions.[24] An advantage of this approach is that the block size typically restricts the computational complexity of the Newton update (which we discuss further in the next sections). Further, superlinear convergence is possible in the scenario where the coordinates $i \notin \mathcal{Z}$ are chosen as part of the block b_k for all sufficiently large k. However, note that this superlinear scenario only occurs in the special case where we use a *greedy rule with variable blocks* and where the *size of the blocks is at least as large as the dimension of the optimal manifold*. With variable blocks, the GS-q and GSL-q rules (6.18) will no longer select coordinates $i \in \mathcal{Z}$ since their optimal d_i value is zero when close to the solution and on the manifold. Thus, these rules will only select $i \notin \mathcal{Z}$ once the manifold has been identified.[25] In contrast, we would not expect superlinear convergence for fixed blocks unless all $i \notin \mathcal{Z}$ happen to be in the same partition. While we could show superlinear convergence of subsequences for random selection with variable blocks, the number of iterations between elements of the subsequence may be prohibitively large.

6.4.2 Practical Proximal-Newton Methods

A challenge with using the update (6.19) in general is that the optimization is non-quadratic (due to the g_i terms) and non-separable (due to the H_b^k-norm). If we make the H_b^k diagonal, then the objective is separable but this destroys the potential for superlinear convergence. Fortunately, a variety of strategies exist in the literature to allow non-diagonal H_b^k.

For example, for bound constrained problems we can apply two-metric projection (TMP) methods, which use a modified H_b^k and allow the computation of a (cheap) projection under the Euclidean norm [Gafni and Bertsekas, 1984]. This method splits the coordinates into an "active" set and a "working" set, where the active-set $\mathcal{A}$ for non-negative constraints would be

$$\mathcal{A} = \{i \mid x_i < \epsilon, \nabla_i f(x) > 0\},$$

[24] A common variation of the proximal-Newton method solves (6.19) with $\alpha_k = 1$ and then sets x^{k+1} based on a search along the line segment between x^k and this solution [Fountoulakis and Tappenden, 2015, Schmidt, 2010]. This variation does *not* have the manifold identification property; only when the line search is on α_k do we have this property.

[25] A subtle issue is the case where $d_i = 0$ in (6.18) but $i \notin \mathcal{Z}$. In such cases we can break ties by preferring coordinates i, where g_i is differentiable so that the $i \notin \mathcal{Z}$ are included.

for some small ϵ while the working-set $\mathcal{W}$ is the compliment of this set. So the active-set contains the coordinates corresponding to the variables that we expect to be zero while the working-set contains the coordinates corresponding to the variables that we expect to be unconstrained. The TMP method can subsequently use the update

$$x_{\mathcal{W}} \leftarrow \text{proj}_C\left(x_{\mathcal{W}} - \alpha H_{\mathcal{W}}^{-1}\nabla_{\mathcal{W}} f(x)\right)$$
$$x_{\mathcal{A}} \leftarrow \text{proj}_C\left(x_{\mathcal{A}} - \alpha\nabla_{\mathcal{A}} f(x)\right).$$

This method performs a gradient update on the active-set and a Newton update on the working-set. Gafni and Bertsekas [1984] show that this preserves many of the essential properties of projected-Newton methods like giving a descent direction, converging to a stationary point, and superlinear convergence if we identify the correct set of non-zero variables. Also note that for indices $i \in \mathcal{Z}$, this eventually only takes gradient steps so our analysis of the previous section applies (it identifies the manifold in a finite number of iterations). As opposed to solving the block-wise proximal-Newton update in (6.19), in our experiments we explore simply using the TMP update applied to the block and found that it gave nearly identical performance for a much lower cost.

TMP methods have also been generalized to settings like ℓ_1-regularization [Schmidt, 2010] and they can essentially be used for any separable g function. Another widely-used strategy is to inexactly solve (6.19) [Fountoulakis and Tappenden, 2015, Lee et al., 2012, Schmidt, 2010]. This has the advantage that it can still be used in the group ℓ_1-regularization setting or other group-separable settings.

6.4.3 Optimal Updates for Quadratic f and Piecewise-Linear g

Two of the most well-studied optimization problems in machine learning are the SVM and LASSO problems. The LASSO problem is given by an ℓ_1-regularized quadratic objective

$$\underset{x}{\text{argmin}}\ \frac{1}{2}\|Ax - b\|^2 + \lambda\|x\|_1,$$

while the dual of the (non-smooth) SVM problem has the form of a bound-constrained quadratic objective

$$\underset{x\in[0,U]}{\text{argmin}}\ \frac{1}{2}x^T Mx - \sum_i x_i, \tag{6.20}$$

for a particular positive semi-definite matrix M and constant U. In both cases we typically expect the solution to be sparse, and identifying the optimal manifold has been shown to improve practical performance of BCD methods [De Santis et al., 2016, Joachims, 1999].

Both problems have a set of g_i that are piecewise-linear over their domain, implying that the they can be written as a maximum over univariate linear functions on the domain of

each variable. Although we can still consider TMP or inexact proximal-Newton updates for these problems, this special structure actually allows us to compute the exact minimum with respect to a block (which is efficient when considering medium-sized blocks). Indeed, for SVM problems the idea of using exact updates in BCD methods dates back to the sequential minimal optimization (SMO) method [Platt, 1998], which uses exact updates for blocks of size 2. In this section we consider methods that work for blocks of arbitrary size.[26]

While we could write the optimal update as a quadratic program, the special structure of the LASSO and SVM problems lends well to exact homotopy methods. These methods date back to Osborne and Turlach [2011], Osborne et al. [2000] who proposed an exact homotopy method that solves the LASSO problem for all values of λ. This type of approach was later popularized under the name "least angle regression" (LARS) [Efron et al., 2004]. Since the solution path is piecewise-linear, given the output of a homotopy algorithm we can extract the exact solution for our given value of λ. Hastie et al. [2004] derive an analogous homotopy method for SVMs, while Rosset and Zhu [2007] derive a generic homotopy method for the case of piecewise-linear g_i functions.

The cost of each iteration of a homotopy method on a problem with $|b|$ variables is $O(|b|^2)$. It is known that the worst-case runtime of these homotopy methods can be exponential [Mairal and Yu, 2012]. However, the problems where this arises are somewhat unstable, and in practice the solution is typically obtained after a linear number of iterations. This gives a runtime in practice of $O(|b|^3)$, which does not allow enormous blocks but does allow us to efficiently use block sizes in the hundreds or thousands. That being said, since these methods compute the exact block update, in the scenario where we previously had superlinear convergence, we now obtain *finite* convergence. That is, the algorithm will stop in a finite number of iterations with the exact solution *provided that* it has identified the optimal manifold, uses a greedy rule with variable blocks, and the block size is larger than the dimension of the manifold. This finite termination is also guaranteed under similar assumptions for TMP methods, and although TMP methods may make less progress per-iteration than exact updates, they may be a cheaper alternative to homotopy methods as the cost is explicitly restricted to $O(|b|^3)$.

6.5 Numerical Experiments

In this section we demonstrate the manifold identification and superlinear/finite convergence properties of the greedy BCD method as discussed in Section 6.4 for a sparse non-negative constrained ℓ_1-regularized least-squares problem using Dataset A (see Appendix D.5.1). In particular, we compare the performance of a projected gradient update with L_b step-size, a projected Newton (PN) solver with line search as discussed in Section 6.4.1 and the two-metric projection (TMP) update as discussed in Section 6.4.2 when using fixed (FB) and variable (VB)

[26] The methods discussed in this section can also be used to compute exact Newton-like updates in the case of a non-quadratic f, but where the g_i are still piecewise-linear.

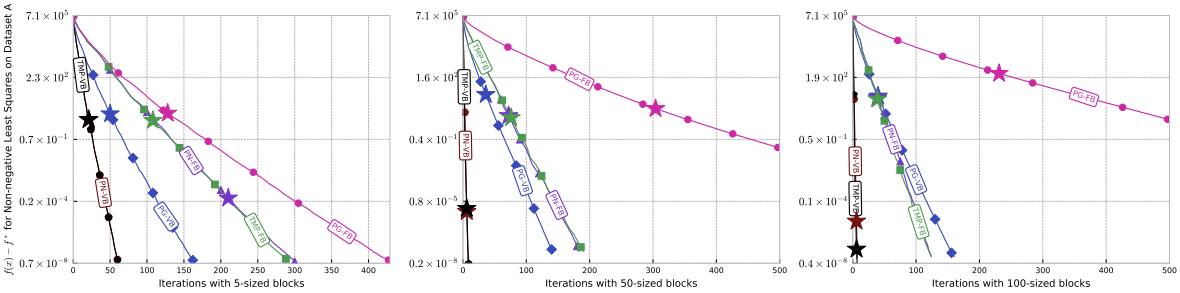


Figure 6.2: Comparison of different updates when using greedy fixed and variable blocks of different sizes.

blocks of different sizes ($|b_k| \in 5, 50, 100$). We use a regularization constant of $\lambda = 50,000$ to encourage a high level of sparsity resulting in an optimal solution x^* with 51 non-zero variables.

In Figure 6.2 we indicate active-set identification with a star and show that all approaches eventually identify the active-set. We see that TMP does as well as projected Newton for all block sizes, while both do better than gradient updates. For a block size of 100, we get finite convergence using projected Newton and TMP updates. We repeat this experiment for random block selection in Figure 6.3 and show that for such a sparse problem multiple iterations are often required before progress is made due to the repetitive selection of variables that are already zero/active.

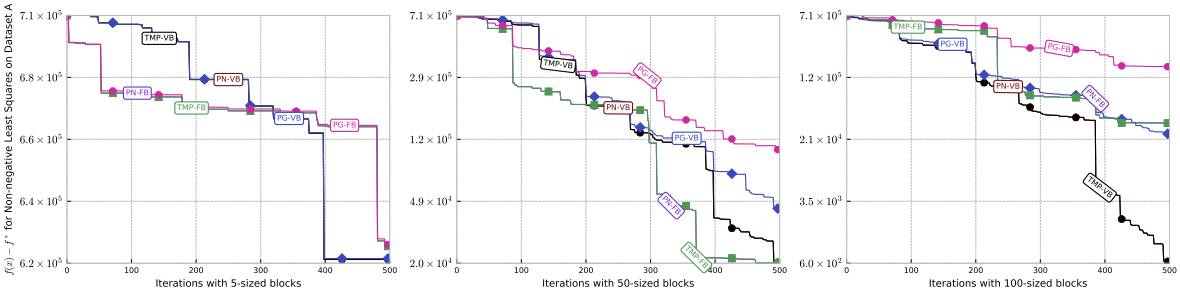


Figure 6.3: Comparison of different updates when using random fixed and variable blocks of different sizes.

6.6 Discussion

In this chapter we showed that greedy BCD methods have a finite-time manifold identification property for problems with separable non-smooth structures like bound constraints or ℓ_1-regularization. Our analysis notably leads to bounds on the number of iterations required to reach the optimal manifold, or the "active-set complexity", for the full proximal gradient method as well as BCD variants. Further, when using greedy rules with variable blocks this leads to superlinear or finite convergence for problems with sufficiently-sparse solutions.

While we made the assumption of strong convexity (or similar), it would be useful to relax this condition while still establishing active-set identification results. Further directions include allowing linear equalities between blocks or extending the results to accelerated proximal methods. Finally, although it is easy to extend the results in this section to the case of a group-separable g, a very useful extension would be to consider the non-separable case.

Chapter 7

Discussion

In this chapter we discuss interesting extensions of the work presented in this dissertation, as well as issues that we did not consider and several future directions.

- **Revitalization of greedy coordinate descent methods.** Since the publication of our work in Chapter 2, there has been a resurgence of greedy coordinate descent methods in the literature for various applications. We list several below that cite our work:
  - Wang [2017] considers model-based iterative reconstruction methods for image reconstruction applications. They propose using a combination of greedy and random coordinate descent methods, and show that the best performance is obtained using a hybrid of 20% random updates and 80% greedy updates.
  - Gsponer et al. [2017] present a new learning algorithm for learning a sequence regression function, which uses greedy coordinate descent method and exact optimization to find α_k. They exploit the sparsity and nested structure of the feature space to ensure an efficient calculation of the greedy Gauss-Southwell rule. They improve the efficiency further by proving an upper bound on the best coordinate and then using a branch-and-bound algorithm to calculate the best coordinate. Their method compares to state-of-the-art, while requiring little to no pre-processing or domain knowledge.
  - Massias et al. [2017] consider working-set methods, where only a subset of constraints are considered at each iteration leading to simpler problems of reduced size. The authors propose a new batch version of the GS-r rule and show that their new greedy active-set method achieves state-of-the-art performance on sparse learning problems with respect to floating point calculations (and time) on LASSO and multi-task LASSO estimators.
  - Stich et al. [2017] propose an approximate greedy coordinate descent method, where a gradient oracle is used to approximate the gradient. They show that the approximate gradient can be updated cheaply at no extra cost, making their method efficient for a more general set of problems with less structure than we assume in this work.
- **Accelerated methods**: We focused on non-accelerated greedy (block) coordinate descent methods in Chapters 2 and 5. However, we expect our conclusions are likely to hold in the case of accelerated methods. In fact, since the publication of our work in

Chapter 2, Lau and Yao [2017] proposed an accelerated greedy block coordinate proximal gradient method using the GS-r selection rule. They assume the Kurdyka-Łojasiewicz inequality (see Chapter 4) and exploit the results of recent works to show convergence of their method for non-convex problems. Their method is shown to beat state-of-the-art solvers on sparse linear regression problems with separable or block-separable regularizers.

- **Parallel methods**: Another extension that we did not consider in this work is parallelization. For iterative methods like coordinate descent, parallel implementations are suitable when the dependency graph is sparse (see Bertsekas and Tsitsiklis [1989, §1.2.4]). For example, if the objective function is separable, then coordinate descent methods are "embarrassingly parallel", meaning the speedup achieved is directly proportional to the number of processors used. When we talk about coordinate descent for truly huge-scale problems, it is impractical to consider a serial implementation. Lots of work has been done on parallel randomized coordinate descent methods (see Richtárik and Takáč [2016] for summary). Since the publication of our work the following parallel greedy coordinate descent methods have been proposed:

  - You et al. [2016] considered smooth functions with bound constraints and showed linear convergence of their proposed asynchronous parallel greedy coordinate descent method when using the GS-r selection rule.
  - Moreau and Oudre [2017] considered the specific problem of convolutional sparse coding, which is designed to build sparse linear representations of datasets. They exploited the specific structure of the convolutional problem and showed that their proposed asynchronous parallel version of greedy coordinate descent using the GS-r rule scales superlinearly with the number of cores (up to a point), making it an efficient option compared to other state-of-the-art methods. Unlike [You et al., 2016] their results do not require centralized communication and a finely tuned step size.

- **Non-convex problems**: While we focused on convex problems, since our work in Chapter 2 was published other authors have shown that our methods are useful for non-convex problems. For example, although it is well-known that the Gauss-Southwell rule works well for the non-convex PageRank problem [Berkhin, 2006, Bonchi et al., 2012, Jeh and Widom, 2003, Lei et al., 2016, McSherry, 2005, Nassar et al., 2015], following the publication of our work, Wang et al. [2017] showed that the Gauss-Southwell-Lipschitz rule is also very useful for this problem.

  We have also seen extensions of the PL work that was presented Chapter 4.

  - Zhang et al. [2016] show that principle component analysis satisfies the PL inequality on a Riemann manifold.
  - Reddi et al. [2016b] show linear convergence of proximal versions of both the stochastic variance reduced gradient and SAGA algorithms when assuming our proximal

PL-inequality.

- Joulani et al. [2017] show that if a function is star-strongly-convex, then that implies that the function satisfies the PL inequality.
- Roulet and d'Aspremont [2017] use the PL inequality to ensure linear convergence of an accelerated method with restart.
- [Yin et al., 2017] analyze the convergence of mini-batch stochastic gradient descent methods under the PL inequality, specifically when the batch-size is proportional to a measure of the "gradient diversity".
- The PL inequality under the name of "gradient dominance condition" has also been show to hold for phase retrieval problems [Zhou et al., 2016], blind deconvolution [Li et al., 2016], and linear residual neural networks [Hardt and Ma, 2016, Zhou and Liang, 2017], while Zhou and Liang [2017] also show it holds for the square loss function of linear and one-hidden-layer nonlinear neural networks.
- Csiba and Richtárik [2017] introduce the "Weak Polyak-Łojasiewicz" condition. Similar to how we introduced the PL inequality as a generalization of strong-convexity to a class of non-convex problems in Chapter 4, Csiba and Richtárik generalize the weakly convex case, providing convergence theory for a new class of non-convex problems.

An elegant consequence of the work in Chapter 4 is that we are starting to see more connections drawn between existing works when it comes to relaxing strong-convexity. This allows authors to exploit the "best" or weakest assumption in the easiest form for their given problem setting.

- **Other methods**: Greedy variations of several other methods have also been proposed since our work, including a greedy primal-dual method [Lei et al., 2017] and a greedy direction method of multiplier (uses our analysis from Chapter 2) [Huang et al., 2017].

  We mentioned in Chapter 3 that the Kaczmarz method could be used for piecewise-linear objectives. Since the publication of our work, Yang and Lin [2015] developed an SGD method that has a linear convergence rate (like stochastic average gradient) for piecewise-linear objectives.

Future extensions that were not considered in this dissertation are:

- **Successive over-relaxation methods.** Successive over-relaxation (SOR) [Frankel, 1950, Young, 1950] is an extrapolation of the cyclic Gauss-Seidel method. It is defined, depending on the extrapolation factor $\omega > 0$, by replacing the iterate x^{k+1} following a full sweep through the coordinates by the following modification,

$$x^{k+1} = \omega x^{k+1} + (1 - \omega)x^k.$$

This scheme can significantly accelerate the convergence rate obtained by the Gauss-Seidel (cyclic CD) method for an optimal value of ω where $1 < \omega < 2$. The optimal value of ω is known in some cases but it is not known for general matrices.

When we apply SOR to the coordinate-wise update in coordinate descent methods (or Gauss-Seidel assuming cyclic selection), we obtain the following

$$\begin{aligned} x^{k+1} &= \omega x^{k+1} + (1-\omega)x^k \\ &= \omega(x^k - \alpha \nabla_i f(x^k) e_{i_k}) + (1-\omega)x^k \\ &= x^k - \omega\alpha \nabla_i f(x^k) e_{i_k}, \end{aligned}$$

which translates to the adjustment of the step size by a constant factor. It would be interesting to see if a similar speed-up as is seen for the Gauss-Seidel method can be obtained for coordinate descent or Kaczmarz methods when using greedy selection rules. It is possible that in this setting, there is a connection between SOR and a simpler version of some well-known first-order acceleration technique like Nesterov's accelerated gradient descent method or the heavy-ball method. We may be able to exploit or generalize the cases where ω is known to see if these connections exist.

- **Successive Projection Methods.** We did not talk about the successive projection method of Censor [1981] in this work, which is a generalization of the Kaczmarz method. Recently, Tibshirani [2017] analyzed the similarities between Dykstra's algorithm (equivalently, successive projection method), alternating direction method of multipliers and coordinate descent, showing connections and equivalences between these methods when applied to the primal and dual regularized regression problem (under some assumptions). These connections could lead to various new analyses and extensions for coordinate descent methods, including new parallel methods and extensions to infinite-dimensional function spaces.

- **BCD methods with constraints between blocks.** This is an important setting for variational inference in graphical models, which is a very important topic in machine learning [Bishop, 2006]. A recent result gives a convergence rate for a variation inference method [Khan et al., 2016] but this result is not for the standard coordinate descent method. The difficulty with using BCD for problems with constraints between blocks is that related subproblems (blocks with constraints between them) are solved independently. Zhang et al. [2014] show that it is possible to strategically partition blocks and then use a message passing algorithm (which allows you to account for the constraints) so as to significantly improve the empirical efficiency of alternating minimization techniques. More recently, She and Schmidt [2017] show linear convergence of the 2-coordinate sequential minimal optimization methods for application to SVMs with unregularized bias. This is a case where the constraints are not completely separable. Earlier works on this

topic include Tseng and Yun [2009a] who considered linearly constrained nonsmooth separable functions, Necoara et al. [2011] who considered random block coordinate descent for general large-scale convex objectives with linearly coupled constraints, and Necoara and Patrascu [2014] who propose a variant of random block coordinate descent for composite objective functions with linearly coupled constraints. It would be interesting to see if there is potential to exploit simpler analysis/better convergence rates for randomized methods, analyze these methods for a more general class of constrained problems and possibly extend the analysis to greedy BCD methods.

Bibliography

A. Agarwal, S. N. Negahban, and M. J. Wainwright. Fast global convergence rates of gradient methods for high-dimensional statistical recovery. *Ann. Statist.*, pages 2452–2482, 2012.

M. Anitescu. Degenerate nonlinear programming with a quadratic growth condition. *SIAM J. Optim.*, pages 1116–1135, 2000.

H. Attouch and J. Bolte. On the convergence of the proximal algorithm for nonsmooth functions involving analytic features. *Math. Program., Ser. B*, pages 5–16, 2009.

F. Bach and E. Moulines. Non–asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems 24*, pages 451–459, 2011.

M. Baghel, S. Agrawal, and S. Silakari. Recent trends and developments in graph coloring. In *Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications*, pages 431–439. Springer Berlin Heidelberg, 2013.

S. Bakin. *Adaptive regression and model selection in data mining problems.* PhD thesis, Australian National University, Canberra, Australia, 1999.

A. Beck and M. Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM J. Imaging Sci.*, 2(1):183–202, 2009.

A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM J. Optim.*, 23(4):2037–2060, 2013.

Y. Bengio, O. Delalleau, and N. Le Roux. Label propagation and quadratic criterion. In O. Chapelle, B. Schölkopf, and A. Zien, editors, *Semi-Supervised Learning*, chapter 11, pages 193–216. MIT Press, 2006.

P. Berkhin. Bookmark-coloring algorithm for personalized PageRank computing. *Internet Mathematics*, 3(1):41–62, 2006.

D. P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21(2):174–184, 1976.

D. P. Bertsekas. *Convex Optimization Algorithms.* Athena Scientific Belmont, 2015.

D. P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 3rd edition, 2016.

D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*, volume 23. Englewood Cliffs: Prentice Hall, NJ, 1989.

D. Bickson. *Gaussian Belief Propagation: Theory and Application.* PhD thesis, The Hebrew University of Jerusalem, Jerusalem, Israel, 2009.

C. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

T. Blumensath and M. E. Davies. Iterative hard thresholding for compressed sensing. *Appl. Comput. Harmon. Anal.*, 27(3):265–274, 2009.

L. Bo and C. Sminchisescu. Greedy block coordinate descent for large scale Gaussian process regression. *arXiv:1206.3238*, 2012.

D. Böhning. Multinomial logistic regression algorithm. *Ann. Inst. Stat. Math.*, 44(1):197–200, 1992.

J. Bolte, T. P. Nguyen, J. Peypouquet, and B. W. Suter. From error bounds to the complexity of first-order descent methods for convex functions. *arXiv:1510.08234*, 2015.

F. Bonchi, P. Esfandiar, D. F. Gleich, C. Greif, and L. V. S. Lakshmanan. Fast matrix computations for pairwise and columnwise commute times and Katz scores. *Internet Mathematics*, 8(1-2):73–112, 2012.

L. Bottou, F. E. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *arXiv:1606.04838*, 2016.

N. Boulanger-Lewandowski, Y. Bengio, and P. Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv:1206.6392*, 2012.

S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

J. V. Burke and J. J. Moré. On the identification of active constraints. *SIAM J. Numer. Anal.*, 25(5):1197–1211, 1988.

A. Cauchy. Méthode générale pour la résolution des systèmes d'équations simultanées. *Comptes Rendus Hebd. Séances Acad. Sci.*, 25:536–538, 1847.

Y. Censor. Row-action methods for huge and sparse systems and their applications. *SIAM Rev.*, 23(4):444–466, 1981.

Y. Censor, P. B. Eggermont, and D. Gordon. Strong underrelaxation in Kaczmarz's method for inconsistent systems. *Numer. Math.*, 41:83–92, 1983.

Y. Censor, G. T. Herman, and M. Jiang. A note on the behaviour of the randomized Kaczmarz algorithm of Strohmer and Vershynin. *J. Fourier Anal. Appl.*, 15:431–436, 2009.

V. Cevher, S. Becker, and M. Schmidt. Convex optimization for big data: Scalable, randomized, and parallel algorithms for big data analytics. *IEEE Signal Processing Magazine*, 31:32–43, 2014.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, 2011. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

B. Chen, S. He, Z. Li, and S. Zhang. Maximum block improvement and polynomial optimization. *SIAM J. Optim.*, 22(1):87–107, 2012.

S. Chen and D. Donoho. Basis pursuit. In *28th Asilomar Conf. Signals, Systems Computers*. Asilomar, 1994.

S. S. Chen, D. L. Donoho, and M. A. Saunders. Atomic decomposition by basis pursuit. *SIAM Rev.*, 43(1):129–159, 2001.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press Cambridge, second edition, 2001.

C. Cortes and V. N. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

B. D. Craven and B. M. Glover. Invex functions and duality. *J. Austral. Math. Soc. (Series A)*, pages 1–20, 1985.

D. Csiba and P. Richtárik. Importance sampling for minibatches. *arXiv:1602.02283*, 2016.

D. Csiba and P. Richtárik. Global convergence of arbitrary-block gradient methods for generalized Polyak-Łojasiewicz functions. *arXiv:1709.03014*, 2017.

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

J. Dahl, L. Vandenberghe, and V. Roychowdhury. Covariance selection for nonchordal graphs via chordal embedding. *Optim. Methods Softw.*, 23(4):501–520, 2008.

M. De Santis, S. Lucidi, and F. Rinaldi. A fast active set block coordinate descent algorithm for ℓ_1-regularized least squares. *SIAM J. Optim.*, 26(1):781–809, 2016.

S. Della Pietra, V. Della Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

R. S. Dembo, S. C. Eisenstat, and T. Steihaug. Inexact Newton methods. *SIAM J. Numer. Anal.*, 19(2):400–408, 1982.

J. E. Dennis and J. J. Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Math. Comput.*, 28(126):549–560, 1974.

F. Deutsch. Rate of convergence of the method of alternating projections. *Internat. Schriftenreihe Numer. Math.*, 72:96–107, 1985.

F. Deutsch and H. Hundal. The rate of convergence for the method of alternating projections, II. *J. Math. Anal. Appl.*, 205:381–405, 1997.

I. S. Dhillon, P. K. Ravikumar, and A. Tewari. Nearest neighbor based greedy coordinate descent. In *Advances in Neural Information Processing Systems 24*, pages 2160–2168, 2011.

F. Dinuzzo, C. S. Ong, P. Gehler, and G. Pillonetto. Learning output kernels with block coordinate descent. In *Proceedings of the 28th International Conference on Machine Learning*, pages 49–56, 2011.

D. Drusvyatskiy and A. S. Lewis. Error bounds, quadratic growth, and linear convergence of proximal methods. *arXiv:1602.06661*, 2016.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Ann. Stat.*, 32(2): 407–451, 2004.

Y. C. Eldar and D. Needell. Acceleration of randomized Kaczmarz methods via the Johnson-Lindenstrauss Lemma. *Numer. Algor.*, 58:163–177, 2011.

A. Ene and H. L. Nguyen. Random coordinate descent methods for minimizing decomposable submodular functions. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 787–795, 2015.

L. Esperet, L. Lemoine, and F. Maffray. Equitable partition of graphs into induced forests. *Discrete Math.*, 338:1481–1483, 2015.

H. G. Feichtinger, C. Cenker, M. Mayer, H. Steier, and T. Strohmer. New variants of the POCS method using affine subspaces of finite codimension with applications to irregular sampling. *SPIE: VCIP*, pages 299–310, 1992.

O. Fercoq and P. Richtárik. Accelerated, parallel and proximal coordinate descent. *SIAM J. Optim.*, 25(4):1997–2023, 2015.

W. F. Ferger. The nature and use of the harmonic mean. *Journal of the American Statistical Association*, 26(173):36–40, 1931.

K. Fountoulakis and R. Tappenden. A flexible coordinate descent method. *arXiv:1507.03713*, 2015.

K. Fountoulakis, F. Roosta-Khorasani, J. Shun, X. Cheng, and M. W. Mahoney. Exploiting optimization for local graph clustering. *arXiv:1602.01886*, 2016.

S. P. Frankel. Convergence rates of iterative treatments of partial differential equations. *Math. Tables Aids Comput.*, 4(30):65–75, 1950.

M. P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM Journal on Scientific Computing*, 34(3):A1380–A1405, 2012.

W. J. Fu. Penalized regressions: the bridge versus the lasso. *J. Comput. Graph. Stat.*, 7(3): 397–416, 1998.

E. M. Gafni and D. P. Bertsekas. Two-metric projection methods for constrained optimization. *SIAM J. Control Optim.*, 22(6):936–964, 1984.

A. Galántai. On the rate of convergence of the alternating projection method in finite dimensional spaces. *J. Math. Anal. Appl.*, 310:30–44, 2005.

D. Garber and E. Hazan. Faster rates for the Frank-Wolfe method over strongly-convex sets. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 541–549, 2015.

M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

T. Glasmachers and U. Dogan. Accelerated coordinate descent with adaptive coordinate frequencies. In *Proceedings of the 5th Asian Conference on Machine Learning*, pages 72–86, 2013.

P. Gong and J. Ye. Linear convergence of variance-reduced stochastic gradient without strong convexity. *arXiv:1406.1102*, 2014.

R. Gordon, R. Bender, and G. T. Herman. Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and x-ray photography. *J. Theor. Biol.*, 29(3):471–481, 1970.

R. M. Gower and P. Richtárik. Randomized iterative methods for linear systems. *SIAM J. Matrix Anal. Appl.*, 36(4):1660–1690, 2015.

J. Gregor and J. A. Fessler. Comparison of SIRT and SQS for regularized weighted least squares image reconstruction. *IEEE Trans. Comput. Imaging*, 1(1):44–55, 2015.

M. Griebel and P. Oswald. Greedy and randomized versions of the multiplicative Schwartz method. *Lin. Alg. Appl.*, 437:1596–1610, 2012.

S. Gsponer, B. Smyth, and G. Ifrim. Efficient sequence regression by learning linear models in all-subsequence space. In *Machine Learning and Knowledge Discovery in Databases*, pages 37–52, 2017.

M. Gu, L.-H. Lim, and C. J. Wu. ParNes: A rapidly convergent algorithm for accurate recovery of sparse and approximately sparse signals. *Numer. Algor.*, pages 321–347, 2013.

M. Hanke and W. Niethammer. On the acceleration of Kaczmarz's method for inconsistent linear systems. *Lin. Alg. Appl.*, 130:83–98, 1990.

M. A. Hanson. On sufficiency of the Kuhn-Tucker conditions. *J. Math. Anal. Appl.*, pages 545–550, 1981.

M. Hardt and T. Ma. Identity matters in deep learning. *arXiv:1611.04231*, 2016.

W. L. Hare. Identifying active manifolds in regularization problems. In H. H. Bauschke, R. S. Burachik, P. L. Combettes, V. Elser, D. R. Luke, and H. Wolkowicz, editors, *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 261–271. Springer New York, New York, NY, 2011.

W. L. Hare and A. S. Lewis. Identifying active constraints via partial smoothness and prox-regularity. *J. Convex Analysis*, 11(2):251–266, 2004.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*, volume 1. Springer Series in Statistics, New York, 2nd edition, 2001.

T. Hastie, S. Rosset, R. Tibshirani, and J. Zhu. The entire regularization path for the support vector machine. *J. Mach. Learn. Res.*, 5:1391–1415, 2004.

G. T. Herman and L. B. Meyer. Algebraic reconstruction techniques can be made computationally efficient. *IEEE Trans. Medical Imaging*, 12(3):600–609, 1993.

R. R. Hocking. A biometrics invited paper. The analysis and selection of variables in linear regression. *Biometrics*, 32(1):1–49, 1976.

A. J. Hoffman. On approximate solutions of systems of linear inequalities. *J. Res. Nat. Bur. Stand.*, 49(4):263–265, 1952.

K. Hou, Z. Zhou, A. M.-C. So, and Z.-Q. Luo. On the linear convergence of the proximal gradient method for trace norm regularization. In *Advances in Neural Information Processing Systems 26*, pages 710–718, 2013.

C.-J. Hsieh, K.-W. Chang, C.-J. Lin, S. S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, pages 408–415, 2008.

C.-J. Hsieh, M. A. Sustik, I. S. Dhillon, P. K. Ravikumar, and R. Poldrack. BIG & QUIC: Sparse inverse covariance estimation for a million variables. In *Advances in Neural Information Processing Systems 26*, pages 3165–3173, 2013.

X. Huang, I. E. H. Yen, R. Zhang, Q. Huang, P. Ravikumar, and I. S. Dhillon. Greedy direction method of multiplier for MAP inference of large output domain. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 1550–1559, 2017.

D. Hush, P. Kelly, C. Scovel, and I. Steinwart. QP algorithms with guaranteed accuracy and run time for support vector machines. *J. Mach. Learn. Res.*, pages 733–769, 2006.

P. Jain, S. M. Kakade, R. Kidambi, P. Netrapalli, and A. Sidford. Accelerating stochastic gradient descent. *arXiv:1704.08227*, 2017.

S. Jegelka, F. Bach, and S. Sra. Reflection methods for user-friendly submodular optimization. In *Advances in Neural Information Processing Systems 26*, pages 1313–1321, 2013.

G. Jeh and J. Widom. Scaling personalized web search. In *Proceedings of the 12th International Conference on World Wide Web*, pages 271–279. ACM, 2003.

T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 169–184. MIT Press, 1999.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems 26*, pages 315–323, 2013.

W. B. Johnson and J. Lindenstrauss. Extensions of Lipchitz mappings into a Hilbert space. *Contemp. Math.*, 26:189–206, 1984.

P. Joulani, A. György, and C. Szepesvári. A modular analysis of adaptive (non-) convex optimization: Optimism, composite objectives, and variational bounds. *Proceedings of Machine Learning Research*, 1:1–40, 2017.

S. Kaczmarz. Angenäherte Auflösung von Systemen linearer Gleichungen, Bulletin International de l'Académie Polonaise des Sciences et des Letters. *Classe des Sciences Mathématiques et Naturelles. Série A, Sciences Mathématiques*, 35:355–357, 1937.

M. Kadkhodaie, M. Sanjabi, and Z.-Q. Luo. On the linear convergence of the approximate proximal splitting method for non-smooth convex optimization. *arXiv:1404.5350v1*, 2014.

H. Karimi, J. Nutini, and M. Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-Łojasiewicz condition. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Proceedings, Part I*, pages 795–811, 2016.

A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

M. E. Khan. *Variational Learning for Latent Gaussian Model of Discrete Data*. PhD thesis, The University of British Columbia, Vancouver, Canada, 2012.

M. E. Khan, R. Babanezhad, W. Lin, M. Schmidt, and M. Sugiyama. Faster stochastic variational inference using proximal gradient methods with general divergence functions. In *Proceedings of the 32nd Conference on Uncertainty in Artificial Intelligence*, pages 319–328, 2016.

R. Kyng and S. Sachdeva. Approximate Gaussian elimination for Laplacians - fast, sparse, and simple. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 573–582. IEEE, 2016.

T. K. Lau and Y. Yao. Accelerated block coordinate proximal gradients with applications in high dimensional statistics. *arXiv:1710.05338*, 2017.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *Advances in Neural Information Processing Systems 25*, pages 2663–2671, 2012.

C.-P. Lee and S. J. Wright. Random permutations fix a worst case for cyclic coordinate descent. *arXiv:1607.08320*, 2016.

J. D. Lee, Y. Sun, and M. A. Saunders. Proximal Newton-type methods for convex optimization. In *Advances in Neural Information Processing Systems 25*, pages 827–835, 2012.

S. Lee and S. J. Wright. Manifold identification in dual averaging for regularized stochastic online learning. *J. Mach. Learn. Res.*, 13(1):1705–1744, 2012.

S.-i. Lee, V. Ganapathi, and D. Koller. Efficient structure learning of Markov networks using ℓ_1-regularization. In *Advances in Neural Information Processing Systems 19*, pages 817–824, 2006.

Y. T. Lee and A. Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. *arXiv:1305.1922v1*, 2013.

Q. Lei, K. Zhong, and I. S. Dhillon. Coordinate-wise power method. In *Advances in Neural Information Processing Systems 29*, pages 2064–2072, 2016.

Q. Lei, I. E.-H. Yen, C.-y. Wu, I. S. Dhillon, and P. Ravikumar. Doubly greedy primal-dual coordinate descent for sparse empirical risk minimization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2034–2042, 2017.

L. Leventhal and A. S. Lewis. Randomized methods for linear constraints: Convergence rates and conditioning. *Math. Oper. Res.*, 35(3):641–654, 2010.

E. S. Levitin and B. T. Polyak. Constrained minimization methods. *USSR Comput. Math. Math. Phys.*, 6:1–50, 1966.

G. Li and T. K. Pong. Calculus of the exponent of Kurdyka-Lojasiewicz inequality and its applications to linear convergence of first-order methods. *arXiv:1602.02915v1*, 2016.

X. Li, S. Ling, T. Strohmer, and K. Wei. Rapid, robust, and reliable blind deconvolution via nonconvex optimization. *arXiv:1606.04933*, 2016.

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.

Z. Li, A. Uschmajew, and S. Zhang. On convergence of the maximum block improvement method. *SIAM J. Optim.*, 25(1):210–233, 2015.

J. Liang, J. Fadili, and G. Peyré. Activity identification and local linear convergence of forward–backward-type methods. *SIAM J. Optim.*, 27(1):408–437, 2017.

J. Liu and S. J. Wright. An accelerated randomized Kaczmarz method. *arXiv:1310.2887v2*, 2014.

J. Liu and S. J. Wright. Asynchronous stochastic coordinate descent: Parallelism and convergence properties. *SIAM J. Optim.*, pages 351–376, 2015.

J. Liu, S. J. Wright, C. Ré, V. Bittorf, and S. Sridhar. An asynchronous parallel stochastic coordinate descent algorithm. *arXiv:1311.1873v3*, 2014.

S. Łojasiewicz. A topological property of real analytic subsets (in French). *Coll. du CNRS, Les équations aux dérivées partielles*, pages 87–89, 1963.

Z.-Q. Luo and P. Tseng. Error bounds and convergence analysis of feasible descent methods: A general approach. *Ann. Oper. Res.*, 46(1):157–178, 1993.

A. Ma, D. Needell, and A. Ramdas. Convergence properties of the randomized extended Gauss-Seidel and Kaczmarz methods. *arXiv:1503.08235v2*, 2015a.

C. Ma, T. Tappenden, and M. Takáč. Linear convergence of the randomized feasible descent method under the weak strong convexity assumption. *arXiv:1506.02530*, 2015b.

J. Mairal and B. Yu. Complexity analysis of the lasso regularization path. In *Proceedings of the 29th International Conference on Machine Learning*, pages 353–360, 2012.

D. M. Malioutov, J. K. Johnson, and A. S. Willsky. Walk-sums and belief propagation in Gaussian graphical models. *J. Mach. Learn. Res.*, 7(Oct):2031–2064, 2006.

M. Massias, A. Gramfort, and J. Salmon. From safe screening rules to working sets for faster Lasso-type solvers. *arXiv:1703.07285*, 2017.

F. McSherry. A uniform approach to accelerated PageRank computation. In *Proceedings of the 14th International Conference on World Wide Web*, pages 575–582. ACM, 2005.

N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res.*, 4(4):414–424, 1979.

L. Meier, S. Van De Geer, and P. Bühlmann. The group lasso for logistic regression. *J. R. Stat. Soc. Series B Stat. Methodol.*, 70(1):53–71, 2008.

R. Meir and G. Rätsch. *An Introduction to Boosting and Leveraging*, pages 118–183. Springer, Heidelberg, 2003.

O. Meshi, T. Jaakkola, and A. Globerson. Convergence rate analysis of MAP coordinate minimization algorithms. In *Advances in Neural Information Processing Systems 25*, pages 3014–3022, 2012.

R. Mifflin and C. Sagastizábal. Proximal points are on the fast track. *J. Convex Anal.*, 9(2): 563–579, 2002.

P. W. Mirowski, Y. LeCun, D. Madhavan, and R. Kuzniecky. Comparing SVM and convolutional networks for epileptic seizure prediction from intracranial EEG. In *IEEE Workshop on Machine Learning for Signal Processing*, pages 244–249. IEEE, 2008.

A.-r. Mohamed, G. Dahl, and G. Hinton. Deep belief networks for phone recognition. In *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, page 39, 2009.

T. Moreau and N. Oudre, L. amd Vayatis. Distributed convolutional sparse coding. *arXiv:1705.10087*, 2017.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

H. Namkoong, A. Sinha, S. Yadlowsky, and J. C. Duchi. Adaptive sampling probabilities for non-smooth optimization. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2574–2583, 2017.

H. Nassar, K. Kloster, and D. F. Gleich. Strong localization in personalized PageRank vectors. In *International Workshop on Algorithms and Models for the Web-Graph*, pages 190–202. Springer, 2015.

I. Necoara and D. Clipici. Parallel random coordinate descent method for composite minimization: Convergence analysis and error bounds. *SIAM J. Optim.*, pages 197–226, 2016.

I. Necoara and A. Patrascu. A random coordinate descent algorithm for optimization problems with composite objection function and linear coupled constraints. *Comput. Optim. Appl.*, pages 307–337, 2014.

I. Necoara, Y. Nesterov, and F. Glineur. A random coordinate descent method on large optimization problems with linear constraints. *Technical Report*, 2011.

I. Necoara, Y. Nesterov, and F. Glineur. Linear convergence of first order methods for non-strongly convex optimization. *arXiv:1504.06298v3*, 2015.

D. Needell. Randomized Kaczmarz solver for noisy linear systems. *BIT Numer. Math.*, 50: 395–403, 2010.

D. Needell and J. A. Tropp. Paved with good intentions: analysis of a randomized block Kaczmarz method. *Linear Algebra Appl.*, 441:199–221, 2014.

D. Needell, N. Srebro, and R. Ward. Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm. *arXiv:1310.5715v5*, 2013.

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

Y. Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. In *Soviet Mathematics Doklady*, volume 27, pages 372–376, 1983.

Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 2004.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *CORE Discussion Paper*, 2010.

Y. Nesterov. Efficiency of coordinate descent methods on huge-scale optimization problems. *SIAM J. Optim.*, 22(2):341–362, 2012.

Y. Nesterov. Gradient methods for minimizing composite functions. *Math. Program.*, 140: 125–161, 2013.

Y. Nesterov and B. T. Polyak. Cubic regularization of Newton method and its global performance. *Math. Program.*, pages 177–205, 2006.

A. Y. Ng. Feature selection, l_1 vs. l_2 regularization, and rotational invariance. In *Proceedings of the 21st International Conference on Machine Learning*, page 78. ACM, 2004.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

A. B. J. Novikoff. On convergence proofs for perceptrons. *Symp. Math. Theory Automata*, 12: 615–622, 1962.

J. Nutini, M. Schmidt, I. H. 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*, pages 1632–1641, 2015.

J. Nutini, B. Sepehry, I. Laradji, M. Schmidt, H. Koepke, and A. Virani. Convergence rates for greedy Kaczmarz algorithms, and faster randomized Kaczmarz rules using the orthogonality graph. *arXiv:1612.07838*, 2016.

J. Nutini, I. Laradji, M. Schmidt, and W. Hare. Let's make block coordinate descent go fast: Faster greedy rules, message- passing, active-set complexity, and superlinear convergence. *submitted for publication*, 2017a.

J. Nutini, M. Schmidt, and W. Hare. "Active-set complexity" of proximal gradient: How long does it take to find the sparsity pattern? *submitted for publication*, 2017b.

S. M. Omohundro. Five balltree construction algorithms. Technical report, International Computer Science Institute, Berkeley, 1989.

M. R. Osborne and B. A. Turlach. A homotopy algorithm for the quantile regression lasso and related piecewise linear problems. *J. Comput. Graph. Stat.*, 20(4):972–987, 2011.

M. R. Osborne, B. Presnell, and B. A. Turlach. A new approach to variable selection in least squares problems. *IMA J. Numer. Anal.*, 20(3):389–403, 2000.

P. Oswald and W. Zhou. Convergence analysis for Kaczmarz-type methods in a Hilbert space framework. *Lin. Alg. Appl.*, 478:131–161, 2015.

S. Parter. The use of linear graphs in Gauss elimination. *SIAM Rev.*, 3(2):119–130, 1961.

Y. C. Pati, R. Rezaiifar, and P. S. Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of the 27th Annu. Asilomar Conf. Signals, Systems and Computers*, pages 40–44. IEEE, 1993.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

M. Pilanci and M. J. Wainwright. Newton sketch: A linear-time optimization algorithm with linear-quadratic convergence. *SIAM J. Optim.*, 27(1):205–245, 2017.

J. C. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. Technical report, Microsoft Research, 1998.

B. T. Polyak. Gradient methods for minimizing functionals (in Russian). *Zh. Vychisl. Mat. Mat. Fiz.*, pages 643–653, 1963.

Z. Qin, K. Scheinberg, and D. Goldfarb. Efficient block-coordinate descent algorithms for Group Lasso. *Mathematical Programming Computation*, 5:143–169, 2013.

Z. Qu and P. Richtárik. Coordinate descent with arbitrary sampling I: Algorithms and complexity. *Optim. Methods Softw.*, 31(5):829–857, 2016.

Z. Qu and P. Richtárik. Coordinate descent with arbitrary sampling II: Expected separable overapproximation. *Optim. Methods Softw.*, 31(5):858–884, 2016.

Z. Qu, P. Richtárik, and T. Zhang. Randomized dual coordinate ascent with arbitrary sampling. *arXiv:1411.5873*, 2014.

Z. Qu, P. Richtárik, M. Takáč, and O. Fercoq. SDNA: Stochastic dual Newton ascent for empirical risk minimization. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 1823–1832, 2016.

G. Rätsch, S. Mika, and M. K. Warmuth. On the convergence of leveraging. In *Advances in Neural Information Processing Systems 14*, pages 487–494, 2001.

S. J. Reddi, S. Sra, B. Poczos, and A. Smola. Fast stochastic methods for nonsmooth nonconvex optimization. *arXiv:1605.06900*, 2016a.

S. J. Reddi, S. Sra, B. Póczos, and A. J. Smola. Proximal stochastic methods for nonsmooth nonconvex finite-sum optimization. In *Advances in Neural Information Processing Systems 29*, pages 1145–1153, 2016b.

P. Richtárik and M. Takáč. Parallel coordinate descent methods for big data optimization. *Math. Prog.*, 156(1-2):433–484, 2016.

P. Richtárik and M. Takáč. Iteration complexity of randomized block-coordinate descent methods for minimizing a composite function. *Math. Program.*, 144:1–38, 2014.

P. Richtárik and M. Takáč. On optimal probabilities in stochastic coordinate descent methods. *Optimization Letters*, 10(6):1233–1243, 2016.

M. Riedmiller and H. Braun. RPROP - A fast adaptive learning algorithm. *In: Proc. of ISCIS VII*, 1992.

H. Robbins and S. Monro. A stochastic approximation method. *Ann. Math. Statistics*, 22(3): 400–407, 1951.

R. T. Rockafellar. Monotone operators and the proximal point algorithm. *SIAM J. Control Optim.*, 14(5):877898, 1976.

L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

D. J. Rose. Triangulated graphs and the elimination process. *J. Math. Anal. Appl.*, 32(3): 597–609, 1970.

S. Rosset and J. Zhu. Piecewise linear regularized solution paths. *Ann. Stat.*, 35(3):1012–1030, 2007.

V. Roulet and A. d'Aspremont. Sharpness, restart and acceleration. In *Advances in Neural Information Processing Systems 30*, pages 1119–1129, 2017.

H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*. CRC Press, 2005.

Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, 1959.

S. Sardy, A. G. Bruce, and P. Tseng. Block coordinate relaxation methods for nonparametric wavelet denoising. *J. Comput. Graph. Stat.*, 9(2):361–379, 2000.

K. Scheinberg and I. Rish. SINCO - a greedy coordinate ascent method for sparse inverse covariance selection problem. *Optimization Online*, 2009.

C. Scherrer, A. Tewari, M. Halappanavar, and D. J. Haglin. Feature clustering for accelerating parallel coordinate descent. In *Advances in Neural Information Processing Systems 25*, pages 28–36, 2012.

M. Schmidt. *Graphical Model Structure Learning with ℓ_1-Regularization*. PhD thesis, The University of British Columbia, Vancouver, BC, Canada, 2010.

M. Schmidt, N. Le Roux, and F. Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. In *Advances in Neural Information Processing Systems 24*, pages 1458–1466, 2011.

M. Schmidt, N. Le Roux, and F. Bach. Minimizing finite sums with the stochastic average gradient. *Math. Program.*, 162(1-2):83–112, 2017.

L. Seidel. *Über ein Verfahren die Gleichungen, auf welche die Methode der kleinsten Quadrate führt, sowie lineäre Gleichungen überhaupt durch successive Annäherung aufzulösen*. Verlag der k. Akademie, 1874.

B. Sepehry. Finding a maximum weight sequence with dependency constraints. Master's thesis, University of British Columbia, Vancouver, BC, Canada, 2016.

S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *J. Mach. Learn. Res.*, pages 567–599, 2013.

J. She and M. Schmidt. Linear convergence and support vector identification of sequential minimal optimization. In *10th NIPS Workshop on Optimization for Machine Learning*, page 5, 2017.

O. Shental, P. H. Siegel, J. K. Wolf, D. Bickson, and D. Dolev. Gaussian belief propagation solver for systems of linear equations. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, pages 1863–1867. IEEE, 2008.

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.

H.-J. M. Shi, S. Tu, Y. Xu, and W. Yin. A primer on coordinate descent algorithms. *arXiv:1610.00040*, 2016.

A. Shrivastava and P. Li. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). In *Advances in Neural Information Processing Systems 27*, pages 2321–2329, 2014.

C. Song, S. Cui, Y. Jiang, and S.-T. Xia. Accelerated stochastic greedy coordinate descent by soft thresholding projection onto simplex. In *Advances in Neural Information Processing Systems 30*, pages 4841–4850, 2017.

D. Sontag and T. Jaakkola. Tree block coordinate descent for MAP in graphical models. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics*, pages 544–551, 2009.

A. Srinivasan and E. Todorov. Graphical Newton. *arXiv:1508.00952*, 2015.

S. U. Stich, A. Raj, and M. Jaggi. Approximate steepest coordinate descent. *arXiv:1706.08427*, 2017.

T. Strohmer and R. Vershynin. A randomized Kaczmarz algorithm with exponential convergence. *J. Fourier Anal. Appl.*, 15:262–278, 2009.

Y. Sun, M. S. Andersen, and L. Vandenberghe. Decomposition in conic optimization with partially separable structure. *SIAM J. Optim.*, 24(2):873–897, 2014.

K. Tanabe. Projection method for solving a singular system of linear equations and its applications. *Numer. Math.*, 17:203–214, 1971.

R. Tappenden, P. Richtárik, and J. Gondzio. Inexact coordinate descent: Complexity and preconditioning. *J. Optim. Theory Appl.*, pages 144–176, 2016.

M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun. MultiNet: Real-time joint semantic reasoning for autonomous driving. *arXiv:1612.07695*, 2016.

G. Thoppe, V. S. Borkar, and D. Garg. Greedy block coordinate descent (GBCD) method for high dimensional quadratic programs. *arXiv:1404.6635*, 2014.

R. J. Tibshirani. Dykstra's algorithm, ADMM, and coordinate descent: Connections, insights, and extensions. In *Advances in Neural Information Processing Systems 30*, pages 517–528, 2017.

P. Tseng. Approximation accuracy, gradient methods, and error bound for structured convex optimization. *Math. Program., Ser. B*, pages 263–295, 2010.

P. Tseng and S. Yun. Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization. *J. Optim. Theory Appl.*, pages 513–535, 2009a.

P. Tseng and S. Yun. A coordinate gradient descent method for nonsmooth separable minimization. *Math. Program.*, 117:387–423, 2009b.

A. van der Sluis. Condition numbers and equilibrium of matrices. *Numer. Math.*, 14:14–23, 1969.

L. Vandenberghe and M. S. Andersen. Chordal graphs and semidefinite optimization. *Found. Trends Optim.*, 1(4):241–433, 2015.

N. K. Vishnoi. $Lx = b$ Laplacian solvers and their algorithmic applications. *Found. Trends Theoretical Computer Science*, 8(1-2):1–141, 2013.

J. von Neumann. *Functional Operators (AM-22), Volume 2: The Geometry of Orthogonal Spaces. (AM-22).* Princeton University Press, 1950.

M. J. Wainwright. Structured regularizers for high-dimensional problems: Statistical and computational issues. *Ann. Rev. Stat. Appl.*, 1(1):233–253, Jan 2014.

J. Wang, W. Wang, D. Garber, and N. Srebro. Efficient coordinate-wise leading eigenvector computation. *arXiv:1702.07834*, 2017.

P.-W. Wang and C.-J. Lin. Iteration complexity of feasible descent methods for convex optimization. *J. Mach. Learn. Res.*, pages 1523–1548, 2014.

X. Wang. *High Performance Tomography.* PhD thesis, Purdue University, West Lafayette, IN, USA, 2017.

D. J. A. Welsh and M. B. Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

T. Whitney and R. Meany. Two algorithms related to the method of steepest descent. *SIAM J. Numer. Anal.*, 4(1):109–118, 1967.

S. J. Wright. Identifiable surfaces in constrained optimization. *SIAM J. Control Optim.*, 31(4): 1063–1079, 1993.

S. J. Wright. Accelerated block-coordinate relaxation for regularized optimization. *SIAM J. Optim.*, 22(1):159–186, 2012.

S. J. Wright. Coordinate descent algorithms. *arXiv:1502.04759v1*, 2015.

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

L. Xiao and T. Zhang. A proximal-gradient homotopy method for the sparse least-squares problem. *SIAM J. Optim.*, 23(2):1062–1091, 2013.

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

T. Yang and Q. Lin. RSG: Beating subgradient method without smoothness and strong convexity. *arXiv:1512.03107*, 2015.

D. Yin, A. Pananjady, M. Lam, D. Papailiopoulos, K. Ramchandran, and P. Bartlett. Gradient diversity empowers distributed learning. *arXiv:1706.05699*, 2017.

Y. You, X. Lian, J. Liu, H.-F. Yu, I. S. Dhillon, J. Demmel, and C.-J. Hsieh. Asynchronous parallel greedy coordinate descent. In *Advances in Neural Information Processing Systems 29*, pages 4682–4690, 2016.

D. M. Young. *Iterative Methods for Solving Partial Difference Equations of Elliptic Type*. PhD thesis, Harvard University, Cambridge, MA, USA, 1950.

D. M. Young. *Iterative Solution of Large Linear Systems*. New York: Academic Press, 1971.

H.-F. Yu, C.-J. Hsieh, S. Si, and I. S. Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 765–774. IEEE, 2012.

H. Zhang. The restricted strong convexity revisited: Analysis of equivalence to error bound and quadratic growth. *arXiv:1511.01635*, 2015.

H. Zhang. New analysis of linear convergence of gradient-type methods via unifying error bound conditions. *arXiv:1606.00269*, 2016.

H. Zhang and W. Yin. Gradient methods for convex minimization: Better rates under weaker conditions. *arXiv:1303.4645*, 2013.

H. Zhang, J. Jiang, and Z.-Q. Luo. On the linear convergence of a proximal gradient method for a class of nonsmooth convex minimization problems. *J. Oper. Res. Soc. China*, 1(2):163–186, 2013.

H. Zhang, S. J. Reddi, and S. Sra. Riemannian SVRG: Fast stochastic optimization on Riemannian manifolds. In *Advances in Neural Information Processing Systems 29*, pages 4592–4600, 2016.

J. Zhang, A. G. Schwing, and R. Urtasun. Message passing inference for large scale graphical models with high order potentials. In *Advances in Neural Information Processing Systems 27*, pages 1134–1142, 2014.

D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *Advances in Neural Information Processing Systems 16*, pages 321–328, 2003.

Y. Zhou and Y. Liang. Characterization of gradient dominance and regularity conditions for neural networks. *arXiv:1710.06910*, 2017.

Y. Zhou, H. Zhang, and Y. Liang. Geometrical properties and accelerated gradient solvers of non-convex phase retrieval. In *54th Annual Allerton Conference on Communication, Control, and Computing*, pages 331–335. IEEE, 2016.

Z. Zhou and A. M.-C. So. A unified approach to error bounds for structured convex optimization problems. *arXiv:1512.03518*, 2015.

A. Zouzias and N. M. Freris. Randomized extended Kaczmarz for solving least-squares. *arXiv:1205.5770v3*, 2013.

Appendix A

Chapter 2 Supplementary Material

A.1 Efficient Calculation of GS Rules for Sparse Problems

We first give additional details on how to calculate the GS rule efficiently for sparse instances of problems h_1 and h_2. We will consider the case where each g_i is smooth, but the ideas can be extended to allow a non-smooth g_i. Further, note that the efficient calculation does not rely on convexity, so these strategies can also be used for non-convex problems.

A.1.1 Problem h_2

Problem h_2 has the form

$$h_2(x) := \sum_{i \in V} g_i(x_i) + \sum_{(i,j) \in E} f_{ij}(x_i, x_j),$$

where each g_i and f_{ij} are differentiable and $G = \{V, E\}$ is a graph where the number of vertices $|V|$ is the same as the number of variables n. If all nodes in the graph have a degree (number of neighbours) bounded above by some constant d, we can implement the GS rule in $O(d \log n)$ after an $O(n + |E|)$ time initialization by maintaining the following information about x^k:

1. A vector containing the values $\nabla_i g_i(x_i^k)$.
2. A matrix containing the values $\nabla_i f_{ij}(x_i^k, x_j^k)$ in the first column and $\nabla_j f_{ij}(x_i^k, x_j^k)$ in the second column.
3. The elements of the gradient vector $\nabla h_2(x^k)$ stored in a binary max heap data structure [see Cormen et al., 2001, Chapter 6].

Given the heap structure, we can compute the GS rule in $O(1)$ by simply reading the index value of the root node in the max heap. The costs for initializing these structures are:

1. $O(n)$ to compute $g_i(x_i^0)$ for all n nodes.
2. $O(|E|)$ to compute $\nabla_{ij} f_{ij}(x_i^0, x_j^0)$ for all $|E|$ edges.
3. $O(n + |E|)$ to sum the values in the above structures to compute $\nabla h(x^0)$, and $O(n)$ to construct the initial max heap.

Thus, the one-time initialization cost is $O(n + |E|)$. The costs of updating the data structures after we update $x_{i_k}^k$ to $x_{i_k}^{k+1}$ for the selected coordinate i_k are:

1. $O(1)$ to compute $g_{i_k}(x_{i_k}^{k+1})$.
2. $O(d)$ to compute $\nabla_{ij} f_{ij}(x_i^{k+1}, x_j^{k+1})$ for $(i,j) \in E$ and $i = i_k$ or $j = i_k$ (only d such values exist by assumption, and all other $\nabla_{ij} f_{ij}(x_i, x_j)$ are unchanged).
3. $O(d)$ to update up the d elements of $\nabla h(x^{k+1})$ that differ from $\nabla h(x^k)$ by using the differences in changed values of g_i and f_{ij}, followed by $O(d \log n)$ to perform d updates of the heap at a cost of $O(\log n)$ for each update.

The most expensive part of the update is modifying the heap, and thus the total cost is $O(d \log n)$.[27]

A.1.2 Problem h_1

Problem h_1 has the form

$$h_1(x) := \sum_{i=1}^{n} g_i(x_i) + f(Ax),$$

where g_i and f are differentiable, and A is an m by n matrix where we denote column i by a_i and row j by a_j^T. Note that f is a function from $\mathbb{R}^m$ to $\mathbb{R}$, and we assume $\nabla_j f$ only depends on $a_j^T x$. While this is a strong assumption (e.g., it rules out f being the product function), this class includes a variety of notable problems like the least squares and logistic regression models from our experiments. If A has z non-zero elements, with a maximum of c non-zero elements in each column and r non-zero elements in each row, then with a pre-processing cost of $O(z)$ we can implement the GS rule in this setting in $O(cr \log n)$ by maintaining the following information about x^k:

1. A vector containing the values $\nabla_i g_i(x_i^k)$.
2. A vector containing the product Ax^k.
3. A vector containing the values $\nabla f(Ax^k)$.
4. A vector containing the product $A^T \nabla f(Ax^k)$.
5. The elements of the gradient vector $\nabla h_1(x^k)$ stored in a binary max heap data structure.

The heap structure again allows us to compute the GS rule in $O(1)$, and the costs of initializing these structures are:

1. $O(n)$ to compute $g_i(x_i^0)$ for all n variables.

[27] For less-sparse problems where $n < d \log n$, using a heap is actually inefficient and we should simply store $\nabla h(x^k)$ as a vector. The initialization cost is the same, but we can then perform the GS rule in $O(n)$ by simply searching through the vector for the maximum element.

2. $O(z)$ to compute the product Ax^0.

3. $O(m)$ to compute $\nabla f(Ax^0)$ (using that $\nabla_j f$ only depends on $a_j^T x^0$).

4. $O(z)$ to compute $A^T \nabla f(Ax^0)$.

5. $O(n)$ to add the $\nabla_i g_i(x_i^0)$ to the above product to obtain $\nabla h_1(x^0)$ and construct the initial max heap.

As it is reasonable to assume that $z \geq m$ and $z \geq n$ (e.g., we have at least one non-zero in each row and column), the cost of the initialization is thus $O(z)$. The costs of updating the data structures after we update $x_{i_k}^k$ to $x_{i_k}^{k+1}$ for the selected coordinate i_k are:

1. $O(1)$ to compute $g_{i_k}(x_{i_k}^{k+1})$.

2. $O(c)$ to update the product using $Ax^{k+1} = Ax^k + (x_{i_k}^{k+1} - x_{i_k}^k)a_i$, since a_i has at most c non-zero values.

3. $O(c)$ to update up to c elements of $\nabla f(Ax^{k+1})$ that have changed (again using that $\nabla_j f$ only depends on $a_j^T x^{k+1}$).

4. $O(cr)$ to perform up to c updates of the form

$$A^T \nabla f(Ax^{k+1}) = A^T \nabla f(Ax^k) + (\nabla_j f(Ax^{k+1}) - \nabla_j f(Ax^k))(a_i)^T,$$

   where each update costs $O(r)$ since each a_i has at most r non-zero values.

5. $O(cr \log n)$ to update the gradients in the heap.

The most expensive part is again the heap update, and thus the total cost is $O(cr \log n)$.

A.2 Relationship Between μ_1 and μ

We can establish the relationship between μ and μ_1 by using the known relationship between the 2-norm and the 1-norm,

$$\|x\|_1 \geq \|x\| \geq \frac{1}{\sqrt{n}}\|x\|_1.$$

In particular, if we assume that f is μ-strongly convex in the 2-norm, then for all x and y we have

$$\begin{aligned} f(y) &\geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|^2 \\ &\geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2n}\|y - x\|_1^2, \end{aligned}$$

implying that f is at least $\frac{\mu}{n}$-strongly convex in the 1-norm. Similarly, if we assume that a given f is μ_1-strongly convex in the 1-norm then for all x and y we have

$$\begin{aligned} f(y) &\geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu_1}{2}\|y - x\|_1^2 \\ &\geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu_1}{2}\|y - x\|^2, \end{aligned}$$

implying that f is at least μ_1-strongly convex in the 2-norm. Summarizing these two relationships, we have

$$\frac{\mu}{n} \leq \mu_1 \leq \mu.$$

A.3 Analysis for Separable Quadratic Case

We first establish an equivalent definition of strong convexity in the 1-norm, along the lines of Nesterov [2004, Theorem 2.1.9]. Subsequently, we use this equivalent definition to derive μ_1 for a separable quadratic function.

A.3.1 Equivalent Definition of Strong Convexity

Assume that f is μ_1-strongly convex in the 1-norm, so that for any $x, y \in \mathbb{R}^n$ we have

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu_1}{2}\|y - x\|_1^2.$$

Reversing x and y in the above gives

$$f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle + \frac{\mu_1}{2}\|x - y\|_1^2,$$

and adding these two together yields

$$\langle \nabla f(y) - \nabla f(x), y - x \rangle \geq \mu_1 \|y - x\|_1^2. \tag{A.1}$$

Conversely, assume that for all x and y we have

$$\langle \nabla f(y) - \nabla f(x), y - x \rangle \geq \mu_1 \|y - x\|_1^2,$$

and consider the function $g(\tau) = f(x + \tau(y - x))$ for $\tau \in \mathbb{R}$. Then

$$\begin{aligned}
f(y) - f(x) - \langle \nabla f(x), y - x \rangle &= g(1) - g(0) - \langle \nabla f(x), y - x \rangle \\
&= \int_0^1 \frac{dg}{d\tau}(\tau) - \langle \nabla f(x), y - x \rangle \, d\tau \\
&= \int_0^1 \langle \nabla f(x + \tau(y - x)), y - x \rangle - \langle \nabla f(x), y - x \rangle \, d\tau \\
&= \int_0^1 \langle \nabla f(x + \tau(y - x)) - \nabla f(x), y - x \rangle \, d\tau \\
&\geq \int_0^1 \frac{\mu_1}{\tau} \|\tau(y - x)\|_1^2 \, d\tau \\
&= \int_0^1 \mu_1 \tau \|y - x\|_1^2 \, d\tau \\
&= \frac{\mu_1}{2} \tau^2 \|y - x\|_1^2 \Big|_0^1 \\
&= \frac{\mu_1}{2} \|y - x\|_1^2.
\end{aligned}$$

Thus, μ_1-strong convexity in the 1-norm is equivalent to having

$$\langle \nabla f(y) - \nabla f(x), y - x \rangle \geq \mu_1 \|y - x\|_1^2 \quad \forall \; x, y. \tag{A.2}$$

A.3.2 Strong Convexity Constant μ_1 for Separable Quadratic Functions

Consider a strongly convex quadratic function f with a diagonal Hessian $H = \nabla^2 f(x) = \text{diag}(\lambda_1, \ldots, \lambda_n)$, where $\lambda_i > 0$ for all $i = 1, \ldots, n$. We show that in this case

$$\mu_1 = \left(\sum_{i=1}^n \frac{1}{\lambda_i} \right)^{-1}.$$

From the previous section, μ_1 is the minimum value such that (A.2) holds,

$$\mu_1 = \inf_{x \neq y} \frac{\langle \nabla f(y) - \nabla f(x), y - x \rangle}{\|y - x\|_1^2}.$$

Using $\nabla f(x) = Hx + b$ for some b and letting $z = y - x$, we get

$$\begin{aligned}
\mu_1 &= \inf_{x \neq y} \frac{\langle (Hy - b) - (Hx - b), y - x \rangle}{\|y - x\|_1^2} \\
&= \inf_{x \neq y} \frac{\langle H(y - x), y - x \rangle}{\|y - x\|_1^2} \\
&= \inf_{z \neq 0} \frac{z^T H z}{\|z\|_1^2} \\
&= \min_{\|z\|_1 = 1} z^T H z \\
&= \min_{e^T z = 1} \sum_{i=1}^{n} \lambda_i z_i^2,
\end{aligned}$$

where the last two lines use that the objective is invariant to scaling of z and to the sign of z (respectively), and where e is a vector containing a one in every position. This is an equality-constrained strictly-convex quadratic program, so its solution is given as a stationary point (z^*, η^*) of the Lagrangian,

$$\Lambda(z, \eta) = \sum_{i=1}^{n} \lambda_i z_i^2 + \eta(1 - e^T z).$$

Differentiating with respect to each z_i for $i = 1, \ldots, n$ and equating to zero, we have for all i that $2\lambda_i z_i^* - \eta^* = 0$, or

$$z_i^* = \frac{\eta^*}{2\lambda_i}. \tag{A.3}$$

Differentiating the Lagrangian with respect to η and equating to zero we obtain $1 - e^T z^* = 0$, or equivalently

$$1 = e^T z^* = \frac{\eta^*}{2} \sum_j \frac{1}{\lambda_j},$$

which yields

$$\eta^* = 2 \left(\sum_j \frac{1}{\lambda_j} \right)^{-1}.$$

Combining this result for η^* with equation (A.3), we have

$$z_i^* = \frac{1}{\lambda_i} \left(\sum_j \frac{1}{\lambda_j} \right)^{-1}.$$

This gives the minimizer, so we evaluate the objective at this point to obtain μ_1,

$$\begin{aligned}
\mu_1 &= \sum_{i=1}^{n} \lambda_i (z_i^*)^2 \\
&= \sum_{i=1}^{n} \lambda_i \left(\frac{1}{\lambda_i} \left(\sum_{j=1}^{n} \frac{1}{\lambda_j} \right)^{-1} \right)^2 \\
&= \sum_{i=1}^{n} \frac{1}{\lambda_i} \left(\sum_{j=1}^{n} \frac{1}{\lambda_j} \right)^{-2} \\
&= \left(\sum_{j=1}^{n} \frac{1}{\lambda_j} \right)^{-2} \left(\sum_{i=1}^{n} \frac{1}{\lambda_i} \right) \\
&= \left(\sum_{j=1}^{n} \frac{1}{\lambda_j} \right)^{-1}.
\end{aligned}$$

A.4 Gauss-Southwell-Lipschitz Rule: Convergence Rate

The coordinate-descent method with a constant step-size of L_{i_k} uses the iteration

$$x^{k+1} = x^k - \frac{1}{L_{i_k}} \nabla_{i_k} f(x^k) e_{i_k}.$$

Because f is coordinate-wise L_{i_k}-Lipschitz continuous, we obtain the following bound on the progress made by each iteration:

$$\begin{aligned}
f(x^{k+1}) &\leq f(x^k) + \nabla_{i_k} f(x^k)(x^{k+1} - x^k)_{i_k} + \frac{L_{i_k}}{2}(x^{k+1} - x^k)_{i_k}^2 \\
&= f(x^k) - \frac{1}{L_{i_k}} (\nabla_{i_k} f(x^k))^2 + \frac{L_{i_k}}{2} \left[\frac{1}{L_{i_k}} \nabla_{i_k} f(x^k) \right]^2 \\
&= f(x^k) - \frac{1}{2L_{i_k}} [\nabla_{i_k} f(x^k)]^2 \\
&= f(x^k) - \frac{1}{2} \left[\frac{\nabla_{i_k} f(x^k)}{\sqrt{L_{i_k}}} \right]^2.
\end{aligned} \tag{A.4}$$

By choosing the coordinate to update according to the Gauss-Southwell-Lipchitz (GSL) rule,

$$i_k \in \operatorname*{argmax}_i \frac{|\nabla_i f(x^k)|}{\sqrt{L_i}},$$

we obtain the tightest possible bound on (A.4). We define the following norm,

$$\|x\|_L = \sum_{i=1}^{n} \sqrt{L_i} |x_i|, \tag{A.5}$$

which has a dual norm of

$$\|x\|_L^* = \max_i \frac{1}{\sqrt{L_i}}|x_i|.$$

Under this notation, and using the GSL rule, (A.4) becomes

$$f(x^{k+1}) \leq f(x^k) - \frac{1}{2}\big(\|\nabla f(x^k)\|_L^*\big)^2,$$

Measuring strong convexity in the norm $\|\cdot\|_L$ we get

$$f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\mu_L}{2}\|y - x\|_L^2.$$

Minimizing both sides with respect to y we get

$$\begin{aligned} f(x^*) &\geq f(x) - \sup_y\{\langle -\nabla f(x), y - x\rangle - \frac{\mu_L}{2}\|y - x\|_L^2\} \\ &= f(x) - \Big(\frac{\mu_L}{2}\|\cdot\|_L^2\Big)^*(-\nabla f(x)) \\ &= f(x) - \frac{1}{2\mu_L}\big(\|\nabla f(x)\|_L^*\big)^2. \end{aligned}$$

Putting these together yields

$$f(x^{k+1}) - f(x^*) \leq (1 - \mu_L)[f(x^k) - f(x^*)]. \tag{A.6}$$

A.5 Comparing μ_L to μ_1 and μ

By the logic Appendix A.2, to establish a relationship between different strong convexity constants under different norms, it is sufficient to establish the relationships between the squared norms. In this section, we use this to establish the relationship between μ_L defined in (A.5) and both μ_1 and μ.

A.5.1 Relationship Between μ_L and μ_1

We have

$$c\|x\|_1 - \|x\|_L = c\sum_i |x_i| - \sum_i \sqrt{L_i}|x_i| = \sum_i (c - \sqrt{L_i})|x_i|,$$

Assuming $c \geq \sqrt{L}$, where $L = \max_i\{L_i\}$, the expression is non-negative and we get

$$\|x\|_L \leq \sqrt{L}\|x\|_1.$$

By using

$$c\|x\|_L - \|x\|_1 = \sum_i (c\sqrt{L_i} - 1)|x_i|,$$

and assuming $c \geq \frac{1}{\sqrt{L_{min}}}$, where $L_{min} = \min_i\{L_i\}$, this expression is nonnegative and we get

$$\|x\|_1 \leq \frac{1}{\sqrt{L_{min}}}\|x\|_L.$$

The relationship between μ_L and μ_1 is based on the squared norm, so in summary we have

$$\frac{\mu_1}{L} \leq \mu_L \leq \frac{\mu_1}{L_{min}}.$$

A.5.2 Relationship Between μ_L and μ

Let $\vec{L}$ denote a vector with elements $\sqrt{L_i}$, and we note that

$$\|\vec{L}\| = \left(\sum_i (\sqrt{L_i})^2\right)^{1/2} = \left(\sum_i L_i\right)^{1/2} = \sqrt{n\bar{L}}, \quad \text{where } \bar{L} = \frac{1}{n}\sum_i L_i.$$

Using this, we have

$$\|x\|_L = x^T(\text{sign}(x) \circ \vec{L}) \leq \|x\|\|\,\text{sign}(x) \circ \vec{L}\| = \sqrt{n\bar{L}}\|x\|.$$

This implies that

$$\frac{\mu}{n\bar{L}} \leq \mu_L.$$

Note that we can also show that $\mu_L \leq \frac{\mu}{L_{min}}$, but this is less tight than the upper bound from the previous section because $\mu_1 \leq \mu$.

A.6 Approximate Gauss-Southwell with Additive Error

In the additive error regime, the approximate Gauss-Southwell rule chooses an i_k satisfying

$$|\nabla_{i_k} f(x^k)| \geq \|\nabla f(x^k)\|_\infty - \epsilon_k, \quad \text{where } \epsilon_k \geq 0 \;\; \forall k,$$

and we note that we can assume $\epsilon_k \leq \|\nabla f(x^k)\|_\infty$ without loss of generality because we must always choose an i with $|\nabla_{i_k} f(x^k)| \geq 0$. Applying this to our bound on the iteration progress, we get

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2L}\left[\nabla_{i_k} f(x^k)\right]^2 \\ &\leq f(x^k) - \frac{1}{2L}\left(\|\nabla f(x^k)\|_\infty - \epsilon_k\right)^2 \\ &= f(x^k) - \frac{1}{2L}\left(\|\nabla f(x^k)\|_\infty^2 - 2\epsilon_k\|\nabla f(x^k)\|_\infty + \epsilon_k^2\right) \\ &= f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L}\|\nabla f(x^k)\|_\infty - \frac{\epsilon_k^2}{2L} \end{aligned} \tag{A.7}$$

We first give a result that assumes f is L_1-Lipschitz continuous in the ℓ_1-norm. This implies an inequality that we prove next, followed by a convergence rate that depends on L_1. However, note that $L \leq L_1 \leq Ln$, so this potentially introduces a dependency on n. We subsequently give a slightly less concise result that has a worse dependency on ϵ but does not rely on L_1.

A.6.1 Gradient Bound in Terms of L_1

We say that ∇f is L_1-Lipschitz continuous in the ℓ_1-norm if we have for all x and y that

$$\|\nabla f(x) - \nabla f(y)\|_\infty \leq L_1 \|x - y\|_1.$$

Similar to Nesterov [2004, Theorem 2.1.5], we now show that this implies

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2L_1} \|\nabla f(y) - \nabla f(x)\|_\infty^2, \tag{A.8}$$

and subsequently that

$$\begin{aligned} \|\nabla f(x^k)\|_\infty &= \|\nabla f(x^k) - \nabla f(x^*)\|_\infty \\ &\leq \sqrt{2L_1(f(x^k) - f(x^*))} \\ &\leq \sqrt{2L_1(f(x^0) - f(x^*))}, \end{aligned} \tag{A.9}$$

where we have used that $f(x^k) \leq f(x^{k-1})$ for all k and any choice of i_{k-1} (this follows from the basic bound on the progress of coordinate descent methods).

We first show that ∇f being L_1-Lipschitz continuous in the 1-norm implies that

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L_1}{2} \|y - x\|_1^2,$$

for all x and y. Consider the function $g(\tau) = f(x + \tau(y - x))$ with $\tau \in \mathbb{R}$. Then

$$\begin{aligned}
f(y) - f(x) - &\langle \nabla f(x), y - x \rangle \\
&= g(1) - g(0) - \langle \nabla f(x), y - x \rangle \\
&= \int_0^1 \frac{dg}{d\tau}(\tau) - \langle \nabla f(x), y - x \rangle \; d\tau \\
&= \int_0^1 \langle \nabla f(x + \tau(y - x)), y - x \rangle - \langle \nabla f(x), y - x \rangle \; d\tau \\
&= \int_0^1 \langle \nabla f(x + \tau(y - x)) - \nabla f(x), y - x \rangle \; d\tau \\
&\leq \int_0^1 \|\nabla f(x + \tau(y - x)) - \nabla f(x)\|_\infty \|y - x\|_1 \; d\tau \\
&\leq \int_0^1 L_1 \tau \|y - x\|_1^2 \; d\tau \\
&= \frac{L_1}{2} \tau^2 \|y - x\|_1^2 \Big|_0^1 \\
&= \frac{L_1}{2} \|y - x\|_1^2.
\end{aligned}$$

To subsequently show (A.8), fix $x \in \mathbb{R}^n$ and consider the function

$$\phi(y) = f(y) - \langle \nabla f(x), y \rangle,$$

which is convex on $\mathbb{R}^n$ and also has an L_1-Lipschitz continuous gradient in the 1-norm, as

$$\begin{aligned}
\|\phi'(y) - \phi'(x)\|_\infty &= \|(\nabla f(y) - \nabla f(x)) - (\nabla f(x) - \nabla f(x))\|_\infty \\
&= \|\nabla f(y) - \nabla f(x)\|_\infty \\
&\leq L_1 \|y - x\|_1.
\end{aligned}$$

As the minimizer of ϕ is x (i.e., $\phi'(x) = 0$), for any $y \in \mathbb{R}^n$ we have

$$\begin{aligned}
\phi(x) = \min_v \phi(v) &\leq \min_v \phi(y) + \langle \phi'(y), v - y \rangle + \frac{L_1}{2} \|v - y\|_1^2 \\
&= \phi(y) - \sup_v \langle -\phi'(y), v - y \rangle - \frac{L_1}{2} \|v - y\|_1^2 \\
&= \phi(y) - \frac{1}{2L_1} \|\phi'(y)\|_\infty^2.
\end{aligned}$$

Substituting in the definition of ϕ, we have

$$\begin{aligned} f(x) - \langle \nabla f(x), x \rangle &\leq f(y) - \langle \nabla f(x), y \rangle - \frac{1}{2L_1} \|\nabla f(y) - \nabla f(x)\|_\infty^2 \\ &\iff f(x) \leq f(y) + \langle \nabla f(x), x - y \rangle - \frac{1}{2L_1} \|\nabla f(y) - \nabla f(x)\|_\infty^2 \\ &\iff f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2L_1} \|\nabla f(y) - \nabla f(x)\|_\infty^2. \end{aligned}$$

A.6.2 Additive Error Bound in Terms of L_1

Using (A.9) in (A.7) and noting that $\epsilon_k \geq 0$, we obtain

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L} \|\nabla f(x^k)\|_\infty - \frac{\epsilon_k^2}{2L} \\ &\leq f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L} \sqrt{2L_1 (f(x^0) - f(x^*))} - \frac{\epsilon_k^2}{2L} \\ &\leq f(x^k) - \frac{1}{2L} \|\nabla f(x^k)\|_\infty^2 + \epsilon_k \frac{\sqrt{2L_1}}{L} \sqrt{f(x^0) - f(x^*)}. \end{aligned}$$

Applying strong convexity (taken with respect to the 1-norm), we get

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \frac{\mu_1}{L}\right) \left[f(x^k) - f(x^*)\right] + \epsilon_k \frac{\sqrt{2L_1}}{L} \sqrt{f(x^0) - f(x^*)},$$

which implies

$$\begin{aligned} f(x^{k+1}) - f(x^*) &\leq \left(1 - \frac{\mu_1}{L}\right)^k \left[f(x^0) - f(x^*)\right] \\ &\quad + \sum_{i=1}^{k} \left(1 - \frac{\mu_1}{L}\right)^{k-i} \epsilon_i \frac{\sqrt{2L_1}}{L} \sqrt{f(x^0) - f(x^*)} \\ &= \left(1 - \frac{\mu_1}{L}\right)^k \left[f(x^0) - f(x^*) + \sqrt{f(x^0) - f(x^*)} A_k\right], \end{aligned}$$

where

$$A_k = \frac{\sqrt{2L_1}}{L} \sum_{i=1}^{k} \left(1 - \frac{\mu_1}{L}\right)^{-i} \epsilon_i.$$

A.6.3 Additive Error Bound in Terms of L

By our additive error inequality, we have

$$|\nabla_{i_k} f(x^k)| + \epsilon_k \geq \|\nabla f(x^k)\|_\infty.$$

Using this again in (A.7) we get

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L}\|\nabla f(x^k)\|_\infty - \frac{\epsilon_k^2}{2L} \\ &\leq f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L}\big(|\nabla_{i_k} f(x^k)| + \epsilon_k\big) - \frac{\epsilon_k^2}{2L} \\ &= f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L}|\nabla_{i_k} f(x^k)| + \frac{\epsilon_k^2}{2L}. \end{aligned}$$

Further, from our basic progress bound that holds for any i_k we have

$$f(x^*) \leq f(x^{k+1}) \leq f(x^k) - \frac{1}{2L}\left[\nabla_{i_k} f(x^k)\right]^2 \leq f(x^0) - \frac{1}{2L}\left[\nabla_{i_k} f(x^k)\right]^2,$$

which implies

$$|\nabla_{i_k} f(x^k)| \leq \sqrt{2L(f(x^0) - f(x^*))}.$$

and thus that

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|_\infty^2 + \frac{\epsilon_k}{L}\sqrt{2L(f(x^0) - f(x^*))} + \frac{\epsilon_k^2}{2L} \\ &= f(x^k) - \frac{1}{2L}\|\nabla f(x^k)\|_\infty^2 + \epsilon_k\sqrt{\frac{2}{L}}\sqrt{f(x^0) - f(x^*)} + \frac{\epsilon_k^2}{2L}. \end{aligned}$$

Applying strong convexity and applying the inequality recursively we obtain

$$\begin{aligned} f(x^{k+1}) - f(x^*) &\leq \left(1 - \frac{\mu_1}{L}\right)^k \left[f(x^0) - f(x^*)\right] \\ &\quad + \sum_{i=1}^{k} \left(1 - \frac{\mu_1}{L}\right)^{k-i} \left(\epsilon_i\sqrt{\frac{2}{L}}\sqrt{f(x^0) - f(x^*)} + \frac{\epsilon_i^2}{2L}\right) \\ &= \left(1 - \frac{\mu_1}{L}\right)^k \left[f(x^0) - f(x^*) + A_k\right], \end{aligned}$$

where

$$A_k = \sum_{i=1}^{k} \left(1 - \frac{\mu_1}{L}\right)^{-i} \left(\sqrt{\frac{2}{L}}\epsilon_i\sqrt{f(x^0) - f(x^*)} + \frac{\epsilon_i^2}{2L}\right).$$

Although uglier than the expression depending on L_1, this expression will tend to be smaller unless ϵ_k is not small.

A.7 Convergence Analysis of GS-s, GS-r, and GS-q Rules

In this section, we consider problems of the form

$$\min_{x \in \mathbb{R}^n} F(x) = f(x) + g(x) = f(x) + \sum_{i=1}^{n} g_i(x_i),$$

where f satisfies our usual assumptions, but the g_i can be non-smooth. We first introduce some notation that will be needed to state our result for the GS-q rule, followed by stating the result and then showing that it holds in two parts. We then turn to showing that the rate cannot hold in general for the GS-s and GS-r rules.

A.7.1 Notation and Basic Inequality

To analyze this case, an important inequality we will use is that the L-Lipschitz-continuity of $\nabla_i f$ implies that for all x, i, and d that

$$\begin{aligned} F(x + de_i) &= f(x + de_i) + g(x + de_i) \\ &\leq f(x) + \langle \nabla f(x), de_i \rangle + \frac{L}{2} d^2 + g(x + de_i) \\ &= f(x) + g(x) + \langle \nabla f(x), de_i \rangle + \frac{L}{2} d^2 + g_i(x_i + d) - g_i(x_i) \\ &= F(x) + V_i(x, d), \end{aligned} \tag{A.10}$$

where

$$V_i(x, d) \equiv \langle \nabla f(x), de_i \rangle + \frac{L}{2} d^2 + g_i(x_i + d) - g_i(x_i).$$

Notice that the GS-q rule is defined by

$$i_k \in \operatorname*{argmin}_i \{\min_d V_i(x, d)\}.$$

We use the notation $d_i^k \in \operatorname{argmin}_d V_i(x^k, d)$ and we will use d^k to denote the vector containing these values for all i. When using the GS-q rule, the iteration is defined by

$$\begin{aligned} x^{k+1} &= x^k + d_{i_k} e_{i_k} \\ &= x^k + \operatorname*{argmin}_d \{V_{i_k}(x, d)\} e_{i_k}. \end{aligned} \tag{A.11}$$

In this notation the GS-r rule is given by

$$j_k \in \operatorname*{argmax}_i |d_i^k|.$$

We will use the notation x_+^k to be the step that would be taken at x_k if we update coordinate j_k according the GS-r rule

$$x_+^k = x^k + d_{j_k} e_{j_k}.$$

From the optimality of d_i^k, we have for any i that

$$-L[(x_i^k - \frac{1}{L}\nabla_i f(x^k)) - (x_i^k + d_i^k)] \in \partial g_i(x_i^k + d_i^k), \tag{A.12}$$

and we will use the notation s_j^k for the unique element of $\partial g_j(x_j^k + d_j^k)$ satisfying this relationship. We use s^k to denote the vector containing these values.

A.7.2 Convergence Bound for GS-q Rule

Under this notation, we can show that coordinate descent with the GS-q rule satisfies the bound

$$\begin{aligned} &F(x^{k+1}) - F(x^*) \\ &\quad \leq \min\left\{\left(1 - \frac{\mu}{Ln}\right)[f(x^k) - f(x^*)], \left(1 - \frac{\mu_1}{L}\right)[f(x^k) - f(x^*)] + \epsilon_k\right\}, \end{aligned} \tag{A.13}$$

where

$$\epsilon_k \leq \frac{\mu_1}{L}\left(g(x_+^k) - g(x^k + d^k) + \langle s^k, (x^k + d^k) - x_+^k\rangle\right).$$

We note that if g is linear then $\epsilon^k = 0$ and this convergence rate reduces to

$$F(x^{k+1}) - F(x^*) \leq \left(1 - \frac{\mu_1}{L}\right)\Big[F(x^k) - F(x^*)\Big].$$

Otherwise, ϵ_k depends on how far $g(x_+^k)$ lies above a particular linear underestimate extending from $(x^k + d^k)$, as well as the conditioning of f. We show this result by first showing that the GS-q rule makes at least as much progress as randomized selection (first part of the min), and then showing that the GS-q rule also makes at least as much progress as the GS-r rule (second part of the min).

A.7.3 GS-q is at Least as Fast as Random

Our argument in this section follows a similar approach to Richtárik and Takáč [2014]. In particular, combining (A.10) and (A.11) we have the following upper bound on the iteration

progress

$$
\begin{aligned}
&F(x^{k+1}) \\
&\leq F(x^k) + \min_{i\in\{1,2,\ldots,n\}} \left\{ \min_{d\in\mathbb{R}} V_i(x^k, d) \right\}, \\
&= F(x^k) + \min_{i\in\{1,2,\ldots,n\}} \left\{ \min_{y\in\mathbb{R}^n} V_i(x^k, y_i - x_i^k) \right\}, \\
&= F(x^k) + \min_{y\in\mathbb{R}^n} \left\{ \min_{i\in\{1,2,\ldots,n\}} V_i(x^k, y_i - x_i^k) \right\}, \\
&\leq F(x^k) + \min_{y\in\mathbb{R}^n} \left\{ \frac{1}{n}\sum_{i=1}^n V_i(x^k, y_i - x^k) \right\} \\
&= F(x^k) + \frac{1}{n} \min_{y\in\mathbb{R}^n} \left\{ \langle \nabla f(x^k), y - x^k\rangle + \frac{L}{2}\|y - x^k\|^2 + g(y) - g(x^k) \right\} \\
&= \left(1 - \frac{1}{n}\right) F(x^k) + \frac{1}{n} \min_{y\in\mathbb{R}^n} \left\{ f(x^k) + \langle \nabla f(x^k), y - x^k\rangle + \frac{L}{2}\|y - x^k\|^2 + g(y) \right\}.
\end{aligned}
$$

From strong convexity of f, we have that F is also μ-strongly convex and that

$$
\begin{aligned}
f(x^k) &\leq f(y) - \langle \nabla f(x^k), y - x^k)\rangle - \frac{\mu}{2}\|y - x^k\|^2, \\
F(\alpha x^* + (1-\alpha)x^k) &\leq \alpha F(x^*) + (1-\alpha)F(x^k) - \frac{\alpha(1-\alpha)\mu}{2}\|x^k - x^*\|^2,
\end{aligned}
$$

for any $y \in \mathbb{R}^n$ and any $\alpha \in [0, 1]$ [see Nesterov, 2004, Theorem 2.1.9]. Using these gives us

$$
\begin{aligned}
&F(x^{k+1}) \\
&\leq \left(1 - \frac{1}{n}\right) F(x^k) + \frac{1}{n} \min_{y\in\mathbb{R}^n} \left\{ f(y) - \frac{\mu}{2}\|y - x\|^2 + \frac{L}{2}\|y - x^k\|^2 + g(y) \right\} \\
&= \left(1 - \frac{1}{n}\right) F(x^k) + \frac{1}{n} \min_{y\in\mathbb{R}^n} \left\{ F(y) + \frac{L-\mu}{2}\|y - x^k\|^2 \right\} \\
&\leq \left(1 - \frac{1}{n}\right) F(x^k) + \\
&\quad \frac{1}{n} \min_{\alpha\in[0,1]} \left\{ F(\alpha x^* + (1-\alpha)x^k) + \frac{\alpha^2(L-\mu)}{2}\|x^k - x^*\|^2 \right\} \\
&\leq \left(1 - \frac{1}{n}\right) F(x^k) + \\
&\quad \frac{1}{n} \min_{\alpha\in[0,1]} \left\{ \alpha F(x^*) + (1-\alpha)F(x^k) + \frac{\alpha^2(L-\mu) - \alpha(1-\alpha)\mu}{2}\|x^k - x^*\|^2 \right\} \\
&\leq \left(1 - \frac{1}{n}\right) F(x^k) + \frac{1}{n}\left[\alpha^* F(x^*) + (1-\alpha^*)F(x^k) \right] \left(\text{with } \alpha^* = \frac{\mu}{L} \in (0, 1] \right) \\
&= \left(1 - \frac{1}{n}\right) F(x^k) + \frac{\alpha^*}{n} F(x^*) + \frac{(1-\alpha^*)}{n} F(x^k) \\
&= F(x^k) - \frac{\alpha^*}{n}[F(x^k) - F(x^*)].
\end{aligned}
$$

Subtracting $F(x^*)$ from both sides of this inequality gives us

$$F(x^{k+1}) - F(x^*) \leq \left(1 - \frac{\mu}{nL}\right)[F(x^k) - F(x^*)].$$

A.7.4 GS-q is at Least as Fast as GS-r

In this section we derive the right side of the bound (A.13) for the GS-r rule, but note it also applies to the GS-q rule because from (A.10) and (A.11) we have

$$\begin{aligned} F(x^{k+1}) &\leq F(x^k) + \min_i V_i(x, d_i^k) \quad \text{(GS-}q\text{ rule)} \\ &\leq F(x^k) + V_{j_k}(x, d_{j_k}^k) \quad (j_k \text{ selected by the GS-}r\text{ rule}). \end{aligned}$$

Note that we lose progress by considering a bound based on the GS-r rule, but its connection to the ∞-norm will make it easier to derive an upper bound.

By the convexity of g_{j_k} we have

$$\begin{aligned} g_{j_k}(x_{j_k}^k) &\geq g_{j_k}(x_{j_k}^k + d_{j_k}^k) + s_{j_k}^k(x_{j_k}^k - (x_{j_k}^k + d_{j_k}^k)) \\ &= g_{j_k}(x_{j_k}^k + d_{j_k}^k) - (-Ld_{j_k}^k - \nabla_{j_k} f(x^k))(d_{j_k}^k) \\ &= g_{j_k}(x_{j_k}^k + d_{j_k}^k) + \nabla_{j_k} f(x^k) d_{j_k}^k + L(d_{j_k}^k)^2, \end{aligned}$$

where s_i^k is defined by (A.12). Using this we have that

$$\begin{aligned} F(x^{k+1}) &\leq F(x^k) + V_j(x, d_{j_k}^k) \\ &= F(x^k) + \nabla_j f(x^k)(d_{j_k}^k) + \frac{L}{2}(d_{j_k}^k)^2 + g_i(x_{j_k}^k + d_{j_k}^k) - g_i(x_{j_k}^k) \\ &\leq F(x^k) + \nabla_j f(x^k)(d_{j_k}^k) + \frac{L}{2}(d_{j_k}^k)^2 - \nabla_{j_k} f(x^k) d_{j_k}^k - L(d_{j_k}^k)^2 \\ &= F(x^k) - \frac{L}{2}(d_{j_k}^k)^2. \end{aligned}$$

Adding and subtracting $F(x^*)$ and noting that j_k is selected using the GS-r rule, we obtain the upper bound

$$F(x^{k+1}) - F(x^*) \leq F(x^k) - F(x^*) - \frac{L}{2}||d^k||_\infty^2. \tag{A.14}$$

Recall that we use x_+^k to denote the iteration that would result if we chose j_k and actually performed the GS-r update. Using the Lipschitz continuity of the gradient and definition of

the GS-q rule again, we have

$$\begin{aligned}
&F(x^{k+1}) \\
&\quad \le F(x^k) + \nabla f(x^k)^T(x^{k+1} - x^k) + \frac{L}{2}||x^{k+1} - x^k||^2 + g(x^{k+1}) - g(x^k) \\
&\quad \le F(x^k) + \nabla f(x^k)^T(x_+^k - x^k) + \frac{L}{2}||x_+^k - x^k||^2 + g(x_k^+) - g(x^k) \\
&\quad = f(x^k) + \nabla f(x^k)^T(x_+^k - x^k) + \frac{L}{2}\|d^k\|_\infty^2 + g(x_+^k)
\end{aligned}$$

By the strong convexity of f, for any $y \in \mathbb{R}^N$ we have

$$f(x^k) \le f(y) - \nabla f(x^k)^T(y - x^k) - \frac{\mu_1}{2}\|y - x^k\|_1^2,$$

and using this we obtain

$$F(x^{k+1}) \le f(y) + \nabla f(x^k)^T(x_+^k - y) - \frac{\mu_1}{2}\|y - x^k\|_1^2 + \frac{L}{2}\|d^k\|_\infty^2 + g(x_+^k). \tag{A.15}$$

By the convexity of g and $s^k \in \partial g(x^k + d^k)$, we have

$$g(y) \ge g(x^k + d^k) + \langle s^k, y - (x^k + d^k)\rangle.$$

Combining (A.15) with the above inequality, we have

$$\begin{aligned}
F(x^{k+1}) - F(y) \le \langle \nabla f(x^k), x_+^k - y\rangle - \frac{\mu_1}{2}\|y - x^k\|_1^2 + \frac{L}{2}\|d^k\|_\infty^2 \\
+ g(x_+^k) - g(x^k + d^k) + \langle s^k, (x^k + d^k) - y\rangle.
\end{aligned}$$

We add and subtract $\langle s^k, x_+^k\rangle$ on the right-hand side to get

$$\begin{aligned}
F(x^{k+1}) - F(y) \le \langle \nabla f(x^k) + s^k, x_+^k - y\rangle - \frac{\mu_1}{2}\|y - x^k\|_1^2 + \frac{L}{2}\|d^k\|_\infty^2 \\
+ g(x_+^k) - g(x^k + d^k) + \langle s^k, (x^k + d^k) - x_+^k\rangle.
\end{aligned}$$

Let $c^k = g(x_+^k) - g(x^k + d^k) + \langle s^k, (x^k + d^k) - x_+^k\rangle$, which is non-negative by the convexity g. Making this substitution, we have

$$F(y) \ge F(x^{k+1}) + \langle -Ld^k, y - x_+^k\rangle + \frac{\mu_1}{2}\|y - x^k\|_1^2 - \frac{L}{2}\|d^k\|_\infty^2 - c^k.$$

Now add and subtract $\langle -Ld^k, x^k\rangle$ to the right-hand side and use (A.12) to get

$$F(y) \ge F(x^{k+1}) + \langle -Ld^k, y - x^k\rangle + \frac{\mu_1}{2}\|y - x^k\|_1^2 - \frac{L}{2}\|d^k\|_\infty^2 - L\langle d^k, x^k - x_+^k\rangle - c^k.$$

Minimizing both sides with respect to y results in

$$\begin{aligned} F(x^*) &\geq F(x^{k+1}) - \frac{L^2}{2\mu_1}\|d^k\|_\infty^2 - \frac{L}{2}\|d^k\|_\infty^2 - L\langle d^k, x^k - x_+^k\rangle - c^k \\ &\geq F(x^{k+1}) - \frac{L^2}{2\mu_1}\|d^k\|_\infty^2 - \frac{L}{2}\|d^k\|_\infty^2 + L\|d^k\|_\infty^2 - c^k \\ &= F(x^{k+1}) - \frac{L(L-\mu_1)}{2\mu_1}\|d^k\|_\infty^2 - c^k, \end{aligned}$$

where we have used that $x_+^k = x^k + d_{j_k}^k e_{j_k}$ and $|d_{j_k}^k| = \|d^k\|_\infty$. Combining this with equation (A.14), we get

$$\begin{aligned} &F(x^{k+1}) - F(x^*) \\ &\quad\leq F(x^k) - F(x^*) - \frac{L}{2}\|d^k\|_\infty^2 \\ &\quad\leq F(x^k) - F(x^*) - \frac{\mu_1}{(L-\mu_1)}\left[F(x^{k+1}) - F(x^*) - c^k\right], \end{aligned}$$

and with some manipulation and simplification, we have

$$\begin{aligned} \left(1 + \frac{\mu_1}{(L-\mu_1)}\right)\left[F(x^{k+1}) - F(x^*)\right] &\leq F(x^k) - F(x^*) + \epsilon^k \frac{\mu_1}{(L-\mu_1)} \\ F(x^{k+1}) - F(x^*) &\leq \frac{(L-\mu_1)}{L}\left[F(x^k) - F(x^*)\right] + c^k\frac{\mu_1}{L} \\ F(x^{k+1}) - F(x^*) &\leq \left(1 - \frac{\mu_1}{L}\right)\left[F(x^k) - F(x^*)\right] + c^k\frac{\mu_1}{L}. \end{aligned}$$

A.7.5 Lack of Progress of the GS-s Rule

We now show that the rate $(1 - \mu_1/L)$, and even the slower rate $(1 - \mu/Ln)$, cannot hold for the GS-s rule. We do this by constructing a problem where an iteration of the GS-s method does not make sufficient progress. In particular, consider the bound-constrained problem

$$\min_{x \in C} f(x) = \frac{1}{2}\|Ax - b\|_2^2,$$

where $C = \{x \ : \ x \geq 0\}$, and

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 0.7 \end{pmatrix}, \quad b = \begin{pmatrix} -1 \\ -3 \end{pmatrix}, \quad x^0 = \begin{pmatrix} 1 \\ 0.1 \end{pmatrix}, \quad x^* = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

We thus have that

$$f(x^0) = \frac{1}{2}((1+1)^2 + (.07+3)^2) \approx 6.7$$
$$f(x^*) = \frac{1}{2}((-1)^2 + (-3)^2) = 5$$
$$\nabla f(x^0) = A^T(Ax_0 - b) \approx \begin{pmatrix} 2.0 \\ \\ 2.1 \end{pmatrix}$$
$$\nabla^2 f(x) = A^T A = \begin{pmatrix} 1 & 0 \\ \\ 0 & 0.49 \end{pmatrix}.$$

The parameter values for this problem are

$$n = 2$$
$$\mu = \lambda_{min} = 0.49$$
$$L = \lambda_{max} = 1$$
$$\mu_1 = \left(\frac{1}{\lambda_1} + \frac{1}{\lambda_2}\right)^{-1} = 1 + \frac{1}{0.49} \approx 0.33,$$

where the λ_i are the eigenvalues of $A^T A$, and μ and μ_1 are the corresponding strong convexity constants for the 2-norm and 1-norm, respectively.

The proximal operator of the indicator function is the projection onto the set C, which involves setting negative elements to zero. Thus, our iteration update is given by

$$x^{k+1} = \text{prox}_{\delta_C}[x^k - \frac{1}{L}\nabla_{i_k} f(x^k)e_{i_k}] = \max(x^k - \frac{1}{L}\nabla_{i_k} f(x^k)e_{i_k}, 0),$$

For this problem, the GS-s rule is given by

$$i \in \underset{i}{\text{argmax}} \, |\eta_i^k|,$$

where

$$\eta_i^k = \begin{cases} \nabla_i f(x^k), & \text{if } x_i^k \neq 0 \text{ or } \nabla_i f(x^k) < 0 \\ 0, & \text{otherwise} \end{cases}.$$

Based on the value of $\nabla f(x^0)$, the GS-s rule thus chooses to update coordinate 2, setting it to zero and obtaining

$$f(x^1) = \frac{1}{2}((1+1)^2 + (-3)^2) = 6.5.$$

Thus we have

$$\frac{f(x^1) - f(x^*)}{f(x^0) - f(x^*)} \approx \frac{6.5 - 5}{6.7 - 5} \approx 0.88,$$

even though the bounds obtain the faster rates of

$$\left(1 - \frac{\mu}{Ln}\right) = \left(1 - \frac{0.49}{2}\right) \approx 0.76,$$
$$\left(1 - \frac{\mu_1}{L}\right) \approx (1 - 0.33) = 0.67.$$

Thus, the GS-s rule does not satisfy either bound. On the other hand, the GS-r and GS-q rules are given in this context by

$$i_k \in \underset{i}{\operatorname{argmax}} \left| \max\left(x^k - \frac{1}{L}\nabla_i f(x^k) e_i, 0\right) - x^k \right|,$$

and thus both these rules choose to update coordinate 1, setting it to zero to obtain $f(x^1) \approx 5.2$ and a progress ratio of

$$\frac{f(x^1) - f(x^*)}{f(x^0) - f(x^*)} \approx \frac{5.2 - 5}{6.7 - 5} \approx 0.12,$$

which clearly satisfies both bounds.

A.7.6 Lack of Progress of the GS-r Rule

We now turn to showing that the GS-r rule does not satisfy these bounds in general. It will not be possible to show this for a simple bound-constrained problem since the GS-r and GS-q rules are equivalent for these problems. Thus, we consider the following ℓ_1-regularized problem

$$\min_{x \in \mathbb{R}^2} \frac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1 \equiv F(x).$$

We use the same A as the previous section, so that n, μ, L, and μ_1 are the same. However, we now take

$$b = \begin{pmatrix} 2 \\ \\ -1 \end{pmatrix}, \quad x_0 = \begin{pmatrix} 0.4 \\ \\ 0.5 \end{pmatrix}, \quad x_* = \begin{pmatrix} 1 \\ \\ 0 \end{pmatrix}, \quad \lambda = 1,$$

so we have

$$f(x_0) \approx 3.1, \quad f(x_*) = 2$$

The proximal operator of the absolute value function is given by the soft-threshold function, and our coordinate update of variable i_k is given by

$$x_{i_k}^{k+1} = \text{prox}_{\lambda|\cdot|}[x_{i_k}^{k+\frac{1}{2}}] = \text{sgn}(x_{i_k}^{k+\frac{1}{2}}) \cdot \max(x_{i_k}^{k+\frac{1}{2}} - \lambda/L, 0),$$

where we have used the notation

$$x_i^{k+\frac{1}{2}} = x_i^k - \frac{1}{L}\nabla_i f(x^k)e_i.$$

The GS-r rule is defined by

$$i_k \in \underset{i}{\operatorname{argmax}} |d_i^k|,$$

where $d_i^k = \text{prox}_{\lambda|\cdot|}[x_i^{k+\frac{1}{2}}] - x_i^k$ and in this case

$$d^0 = \begin{pmatrix} 0.6 \\ \\ -0.5 \end{pmatrix}.$$

Thus, the GS-r rule chooses to update coordinate 1. After this update the function value is

$$F(x^1) \approx 2.9,$$

so the progress ratio is

$$\frac{F(x^1) - F(x^*)}{F(x^0) - F(x^*)} \approx \frac{2.9 - 2}{3.1 - 2} \approx 0.84.$$

However, the bounds suggest faster progress ratios of

$$\left(1 - \frac{\mu}{Ln}\right) \approx 0.76,$$

$$\left(1 - \frac{\mu_1}{L}\right) \approx 0.67,$$

so the GS-r rule does not satisfy either bound. In contrast, in this setting the GS-q rule chooses to update coordinate 2 and obtains $F(x^1) \approx 2.2$, obtaining a progress ratio of

$$\frac{F(x^1) - F(x^*)}{F(x^0) - F(x^*)} \approx \frac{2.2 - 2}{3.1 - 2} \approx 0.16,$$

which satisfies both bounds by a substantial margin. Indeed, we used a genetic algorithm to search for a setting of the parameters of this problem (values of x^0, λ, b, and the diagonals of A) that would make the GS-q not satisfy the bound depending on μ_1, and it easily found counter-examples for the GS-s and GS-r rules but was not able to produce a counter example for the GS-q rule.

A.8 Proximal Gradient in the ℓ_1-Norm

Our argument in this section follows a similar approach to Richtárik and Takáč [2014]. Assuming L_1-Lipschitz continuity of ∇f, we have

$$\begin{aligned}
F(x+d) &= f(x+d) + g(x+d) \\
&\leq f(x) + \langle \nabla f(x), d \rangle + \frac{L_1}{2}\|d\|_1^2 + g(x+d) \\
&= F(x) + \langle \nabla f(x), d \rangle + \frac{L_1}{2}\|d\|_1^2 + g(x+d) - g(x) \\
&= F(x) + V(x,d),
\end{aligned}$$

where

$$V(x,d) \equiv \langle \nabla f(x), d \rangle + \frac{L_1}{2}\|d\|_1^2 + g(x+d) - g(x).$$

Generalizing the GS-q rule defined by Song et al. [2017], we have

$$\begin{aligned}
d^k &\in \operatorname*{argmin}_{d \in \mathbb{R}^n} V(x,d), \\
x^{k+1} &= x^k + \operatorname*{argmin}_{d \in \mathbb{R}^n} V(x,d).
\end{aligned}$$

Plugging this update into the above inequality with a change of variable, we have the following upper bound on the iteration progress

$$\begin{aligned}
F(x^{k+1}) &\leq F(x^k) + \left\{ \min_{d \in \mathbb{R}^n} V(x^k, d) \right\}, \\
&= F(x^k) + \left\{ \min_{y \in \mathbb{R}^n} V(x^k, y - x^k) \right\}, \\
&= F(x^k) + \min_{y \in \mathbb{R}^n} \left\{ \langle \nabla f(x^k), y - x^k \rangle + \frac{L_1}{2}\|y - x^k\|_1^2 + g(y) - g(x^k) \right\} \\
&= \min_{y \in \mathbb{R}^n} \left\{ f(x^k) + \langle \nabla f(x^k), y - x^k \rangle + \frac{L_1}{2}\|y - x^k\|_1^2 + g(y) \right\}.
\end{aligned}$$

By the strong convexity of f, we have that F is also μ_1-strongly convex,

$$\begin{aligned}
f(x^k) &\leq f(y) - \langle \nabla f(x^k), y - x^k) \rangle - \frac{\mu_1}{2}\|y - x^k\|_1^2, \\
F(\alpha x^* + (1-\alpha)x^k) &\leq \alpha F(x^*) + (1-\alpha)F(x^k) - \frac{\alpha(1-\alpha)\mu_1}{2}\|x^k - x^*\|_1^2,
\end{aligned}$$

for any $y \in \mathbb{R}^n$ and $\alpha \in [0,1]$ [see Nesterov, 2004, Theorem 2.1.9]. Therefore,

$$\begin{aligned}
F(x^{k+1}) &\leq \min_{y \in \mathbb{R}^n} \left\{ f(y) - \frac{\mu_1}{2}\|y - x^k\|_1^2 + \frac{L_1}{2}\|y - x^k\|_1^2 + g(y) \right\} \\
&= \min_{y \in \mathbb{R}^n} \left\{ F(y) + \frac{(L_1 - \mu_1)}{2}\|y - x^k\|_1^2 \right\} \\
&\leq \min_{\alpha \in [0,1]} \left\{ F(\alpha x^* + (1-\alpha)x^k) + \frac{\alpha^2(L_1 - \mu_1)}{2}\|x^k - x^*\|_1^2 \right\} \\
&\leq \min_{\alpha \in [0,1]} \left\{ \alpha F(x^*) + (1-\alpha)F(x^k) + \frac{\alpha^2(L_1 - \mu_1) - \alpha(1-\alpha)\mu_1}{2}\|x^k - x^*\|_1^2 \right\} \\
&\leq \left[\alpha^* F(x^*) + (1-\alpha^*)F(x^k) \right] \qquad \left(\text{choosing } \alpha^* = \frac{\mu_1}{L_1} \in (0,1] \right) \\
&= F(x^k) - \alpha^*[F(x^k) - F(x^*)].
\end{aligned}$$

Subtracting $F(x^*)$ from both sides of this inequality gives us

$$F(x^{k+1}) - F(x^*) \leq \left(1 - \frac{\mu_1}{L_1}\right)[F(x^k) - F(x^*)].$$

Appendix B

Chapter 3 Supplementary Material

B.1 Efficient Calculations for Sparse A

To compute the MR rule efficiently for sparse $A \in \mathbb{R}^{m \times n}$, we need to store and update the residuals $r_i = (a_i^T x^k - b_i)$ for all i. If we initialize with $x^0 = 0$, then the initial values of the residuals are simply the corresponding b_i values. Given the initial residuals, we can construct a max-heap structure on these residuals in $O(m)$ time. The max-heap structure lets us compute the MR rule in $O(1)$ time. After an iteration of the Kaczmarz method, we can update the max-heap efficiently as follows:

For each j where $x_j^{k+1} \neq x_j^k$:

- **For each i with $a_{ij} \neq 0$:**
  - Update r_i using $r_i \leftarrow r_i - a_{ij}x_j^k + a_{ij}x_j^{k+1}$.
  - Update max-heap using the new value of $|r_i|$.

The cost of each update to an r_i is $O(1)$ and the cost of each heap update is $O(\log m)$. If each row of A has at most r non-zeroes and each column has at most c non-zeroes, then the outer loop is run at most r times while the inner loop is run at most c times for each outer loop iteration. Thus, in the worst case the total cost is $O(cr \log m)$, although it might be much faster if we have particularly sparse rows or columns. Thus, if c and r are sufficiently small, the MR rule is not much more expensive than non-uniform random selection which costs $O(\log m)$. For the MD rule, the cost is the same except there is an extra one-time cost to pre-compute the row norms $\|a_i\|$.

Now consider the case where A may be dense but each row is orthogonal to all but at most g other rows. In this setting it would be too slow to implement the above update of the residuals, since the cost would be $O(mn \log(m))$. In this setting, it makes more sense to use the following alternative approach to update the max-heap after we have updated row i_k:

For each i that is a neighbour of i_k in the orthogonality graph:

- Compute the residual $r_i = a_i^T x^k - b_i$.
- Update max-heap using the new value of $|r_i|$.

We can find the set of neighbours for each node in constant time by keeping a list of the neighbours of each node. This loop would run at most g times and the cost of each iteration would be $O(n)$ to update the residual and $O(\log m)$ to update the heap. Thus, the cost to

track the residuals using this alternative approach would be $O(gn + g\log(m))$ or the faster $O(gr + g\log(m))$ if each row has at most r non-zeros.

B.2 Randomized and Maximum Residual

In this section, we provide details of the convergence rate derivations for the non-uniform and maximum residual (MR) selection rules. All the convergence rates we discuss use the following relationship,

$$\begin{aligned}\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \|x^{k+1} - x^k\|^2 \\ &= \|x^k - x^*\|^2 - \left\| \frac{(b_i - a_i^T x^k)}{\|a_i\|^2} \cdot a_i \right\|^2 \\ &= \|x^k - x^*\|^2 - \frac{\left(a_i^T x^k - b_i\right)^2}{\|a_i\|^2}, \end{aligned} \tag{B.1}$$

which is equation (3.6).

Non-Uniform

We first review how the steps discussed by Vishnoi [2013] that can be used to derive the convergence rate bound of Strohmer and Vershynin [2009] for non-uniform random selection when row i is chosen according to the probability distribution determined by $\|a_i\|/\|A\|_F$. Taking the expectation of (B.1) with respect to i, we have

$$\begin{aligned}\mathbb{E}[\|x^{k+1} - x^*\|^2] &= \|x^k - x^*\|^2 - \mathbb{E}\left[\frac{(a_i^T x^k - b_i)^2}{\|a_i\|^2}\right] \\ &= \|x^k - x^*\|^2 - \sum_{i=1}^{m} \frac{\|a_i\|^2}{\|A\|_F^2} \frac{(a_i^\top (x^k - x^*))^2}{\|a_i\|^2} \\ &= \|x^k - x^*\|^2 - \frac{\|A(x^k - x^*)\|^2}{\|A\|_F^2} \\ &\leq \left(1 - \frac{\sigma(A,2)^2}{\|A\|_F^2}\right) \|x^k - x^*\|^2, \end{aligned} \tag{B.2}$$

where $\sigma(A, 2)$ is the Hoffman [1952] constant, which can be defined as the largest value such that for any x that is not a solution to the linear system we have

$$\sigma(A,2)\|x - x^*\| \leq \|A(x - x^*)\|, \tag{B.3}$$

where x^* is the projection of x onto the set of solutions S. In other words, we can write it as

$$\sigma(A,2) := \inf_{x \notin S} \frac{\|A(x - x^*)\|}{\|x - x^*\|}.$$

Strohmer and Vershynin [2009] consider the special case where A has independent columns, and this result yields their rate in this special case since under this assumption $\sigma(A, 2)$ is given by the nth singular value of A. For general matrices, $\sigma(A, 2)$ is given by the smallest non-zero singular value of A.

Maximum Residual

We use a similar analysis to prove a convergence rate bound for the MR rule,

$$i_k \in \underset{i}{\operatorname{argmax}} \ |a_i^T x^k - b_i|. \tag{B.4}$$

Assuming that i is selected according to (B.4), then starting from (B.1) we have

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \frac{\max_i (a_i^T x^k - b_i)^2}{\|a_i\|^2} \\
&\leq \|x^k - x^*\|^2 - \frac{1}{\|A\|_{\infty,2}^2} \max_i \ (a_i^T (x^k - x^*))^2 \\
&= \|x^k - x^*\|^2 - \frac{\|A(x^k - x^*)\|_\infty^2}{\|A\|_{\infty,2}^2} \\
&\leq \left(1 - \frac{\sigma(A, \infty)^2}{\|A\|_{\infty,2}^2}\right) \|x^k - x^*\|^2,
\end{aligned} \tag{B.5}$$

where $\|A\|_{\infty,2}^2 := \max_i\{\|a_i\|^2\}$ and $\sigma(A, \infty)$ is the largest value such that

$$\sigma(A, \infty)\|x - x^*\| \leq \|A(x - x^*)\|_\infty, \tag{B.6}$$

or equivalently

$$\sigma(A, \infty) := \inf_{x \notin S} \frac{\|A(x - x^*)\|_\infty}{\|x - x^*\|}.$$

The existence of such a Hoffman-like constant follows from the existence of the Hoffman constant and the equivalence between norms. Applying the norm equivalence $\|\cdot\|_\infty \geq \frac{1}{\sqrt{m}}\|\cdot\|$ to equation (B.3) we have

$$\sigma(A, 2)\|x - x^*\| \leq \|A(x - x^*)\| \leq \sqrt{m}\|A(x - x^*)\|_\infty,$$

which implies that $\sigma(A, 2)/\sqrt{m} \leq \sigma(A, \infty)$. Similarly, applying $\|\cdot\|_\infty \leq \|\cdot\|$ to (B.6) we have

$$\sigma(A, \infty)\|x - x^*\| \leq \|A(x - x^*)\|_\infty \leq \|A(x - x^*)\|,$$

which implies that $\sigma(A, \infty)$ cannot be larger than $\sigma(A, 2)$. Thus, $\sigma(A, \infty)$ satisfies the relationship

$$\frac{\sigma(A, 2)}{\sqrt{m}} \leq \sigma(A, \infty) \leq \sigma(A, 2). \tag{B.7}$$

B.3 Tighter Uniform and MR Analysis

To avoid using the inequality $\|a_i\| \leq \|A\|_{\infty,2}$ for all i, we want to 'absorb' the individual row norms into the bound. We start with uniform selection.

Uniform

Consider the diagonal matrix $D = \text{diag}(\|a_1\|^2, \|a_2\|^2, \ldots, \|a_m\|^2)$. By taking the expectation of (B.1), we have

$$\begin{aligned}
\mathbb{E}[\|x^{k+1} - x^*\|^2] &= \|x^k - x^*\|^2 - \mathbb{E}\left[\frac{\left(a_i^T x^k - b_i\right)^2}{\|a_i\|^2}\right] \\
&= \|x^k - x^*\|^2 - \sum_{i=1}^{m} \frac{1}{m} \frac{(a_i^T x^k - b_i)^2}{\|a_i\|^2} \\
&= \|x^k - x^*\|^2 - \frac{1}{m} \sum_{i=1}^{m} \left(\left[\frac{a_i}{\|a_i\|}\right]^T (x^k - x^*) \right)^2 \\
&= \|x^k - x^*\|^2 - \frac{\|D^{-1}A(x^k - x^*)\|^2}{m} \\
&\leq \left(1 - \frac{\sigma(\bar{A}, 2)^2}{m}\right) \|x^k - x^*\|^2, \qquad \text{(B.8)}
\end{aligned}$$

where recall that $\bar{A} = D^{-1}A$, and we used that $Ax = b$ and $D^{-1}Ax = D^{-1}b$ have the same solution set.

Maximum Residual

For the tighter analysis of the MR rule we do not want to alter the selection rule. Thus, we first evaluate the MR rule and then divide by the corresponding $\|a_{i_k}\|^2$ for the selected i_k at iteration k. Starting from (B.1), this gives us

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \frac{1}{\|a_{i_k}\|^2} \max_i (a_i^T (x^k - x^*))^2 \\
&= \|x^k - x^*\|^2 - \frac{\|A(x^k - x^*)\|_\infty^2}{\|a_{i_k}\|^2} \\
&\leq \left(1 - \frac{\sigma(A, \infty)^2}{\|a_{i_k}\|^2}\right) \|x^k - x^*\|^2. \qquad \text{(B.9)}
\end{aligned}$$

Applying this recursively over all k iterations yields the rate

$$\|x^k - x^*\|^2 \leq \prod_{j=1}^{k} \left(1 - \frac{\sigma(A, \infty)^2}{\|a_{i_j}\|^2}\right) \|x^0 - x^*\|^2. \qquad \text{(B.10)}$$

B.4 Maximum Distance Rule

If we can only perform one iteration of the Kaczmarz method, the *optimal* rule with respect to iterate progress is the maximum distance (MD) rule,

$$i_k \in \operatorname*{argmax}_i \left| \frac{a_i^T x^k - b_i}{\|a_i\|} \right|. \tag{B.11}$$

Starting again from (B.1) and using D as defined in the tight analysis for the U rule, we have

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \max_i \left(\frac{a_i^T x^k - b_i}{\|a_i\|} \right)^2 \\
&= \|x^k - x^*\|^2 - \max_i \left(\left[\frac{a_i}{\|a_i\|} \right]^T (x^k - x^*) \right)^2 \\
&= \|x^k - x^*\|^2 - \|D^{-1} A (x^k - x^*)\|_\infty^2 \\
&\leq \left(1 - \sigma(\bar{A}, \infty)^2\right) \|x^k - x^*\|^2.
\end{aligned} \tag{B.12}$$

We now show that

$$\max \left\{ \frac{\sigma(\bar{A}, 2)}{\sqrt{m}}, \frac{\sigma(A, 2)}{\|A\|_F}, \frac{\sigma(A, \infty)}{\|A\|_{\infty,2}} \right\} \leq \sigma(\bar{A}, \infty) \leq \sigma(\bar{A}, 2). \tag{B.13}$$

To derive the upper bound on $\sigma(\bar{A}, \infty)$, and to derive the lower bound in terms of $\sigma(\bar{A}, 2)$, we can use norm equivalence arguments as we did for $\sigma(A, \infty)$. This yields

$$\frac{\sigma(\bar{A}, 2)}{\sqrt{m}} \leq \sigma(\bar{A}, \infty) \leq \sigma(\bar{A}, 2).$$

The last argument in the maximum in (B.13), corresponding to the MR_∞ rate, holds because $\|A\|_{\infty,2} \geq \|a_i\|$ for all i so we have

$$\begin{aligned}
\frac{\sigma(A, \infty)}{\|A\|_{\infty,2}} \|x - x^*\| &\leq \frac{\|A(x - x^*)\|_\infty}{\|A\|_{\infty,2}} \\
&= \max_i \left\{ \frac{|a_i^T (x - x^*)|}{\|A\|_{\infty,2}} \right\} \\
&\leq \max_i \left\{ \frac{|a_i^T (x - x^*)|}{\|a_i\|} \right\} \\
&= \|\bar{A}(x - x^*)\|_\infty.
\end{aligned}$$

For the second argument in the maximum in (B.13), the NU rate, we have

$$\begin{aligned}\frac{\sigma(A,2)^2}{\|A\|_F^2}\|x-x^*\|^2 &\leq \frac{\|A(x-x^*)\|^2}{\|A\|_F^2}\\ &= \frac{\sum_i (a_i^T(x-x^*))^2}{\sum_i \|a_i\|^2}\\ &\leq \max_i \left\{\frac{(a_i^T(x-x^*))^2}{\|a_i\|^2}\right\}\\ &= \|\bar{A}(x-x^*)\|_\infty.\end{aligned}$$

The second inequality is true by noting that it is equivalent to the inequality

$$1 \leq \max_i \left\{\frac{(a_i^T(x-x^*)^2/\sum_j (a_j^T(x-x^*))^2}{\|a_i\|^2/\sum_j \|a_j\|^2}\right\},$$

and this true because the maximum ratio between two probability mass functions must be at least 1,

$$1 \leq \max_i \frac{p_i/\sum_j p_j}{q_i/\sum_j q_j}, \quad \text{with all } p_i \geq 0, q_i \geq 0.$$

Finally, we note that the MD rule obtains the tightest bound in terms of performing one step. This follows from (B.1),

$$\|x^{k+1}-x^*\|^2 = \|x^k-x^*\|^2 - \|x^{k+1}-x^k\|^2 = \|x^k-x^*\|^2 - \frac{\left(a_i^T x^k - b_i\right)^2}{\|a_i\|^2},$$

and noting that the MD rule maximizes $\|x^{k+1}-x^k\|$ and thus it maximizes how much smaller $\|x^{k+1}-x^*\|$ is than $\|x^k-x^*\|$.

B.5 Kaczmarz and Coordinate Descent

Consider the Kaczmarz update:

$$x^{k+1} = x^k - \frac{(a_i^T x^k - b_i)}{\|a_i\|^2} a_i.$$

This update is equivalent to one step of coordinate descent (CD) with step length $1/\|a_i\|^2$ applied to the dual problem,

$$\min_y \frac{1}{2}\|A^T y\|^2 - b^T y, \tag{B.14}$$

see Wright [2015]. Using the primal-dual relationship $A^T y = x$, we can show the relationship between the greedy Kaczmarz selection rules and applying greedy coordinate descent rules to

this dual problem. Consider the gradient of the dual problem,

$$\nabla f(y) = AA^T y - b.$$

The Gauss-Southwell (GS) rule for CD on the dual problem is equivalent to the MR rule for Kaczmarz on the primal problem since

$$i_k \in \underbrace{\operatorname*{argmax}_i |\nabla_i f(y^k)|}_{\text{Gauss-Southwell rule}} \equiv \operatorname*{argmax}_i |a_i^T(A^T y^k) - b_i| \equiv \underbrace{\operatorname*{argmax}_i |a_i^T x^k - b_i|}_{\text{Maximum residual rule}}$$

where a_i^T is the ith row of A. Similarly, the Gauss-Southwell-Lipschitz (GSL) rule applied to the dual is equivalent to applying a Kaczmarz iteration with the MD rule,

$$i_k \in \underbrace{\operatorname*{argmax}_i \frac{|\nabla_i f(y^k)|}{\sqrt{L_i}}}_{\text{Gauss-Southwell-Lipschitz rule}} \equiv \operatorname*{argmax}_i \frac{|a_i^T(A^T y^k) - b_i|}{\|a_i\|} \equiv \underbrace{\operatorname*{argmax}_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|}_{\text{Maximum distance rule}},$$

as the Lipschitz constants for the dual problem are $L_i = \|a_i\|^2$.

Figure B.1 shows the results of running Kaczmarz compared to using CD (on the least-squares primal problem) for our 3 datasets from Section 3.9. In this figure we measure the performance in terms of the number of "effective passes" through the data (one "effective" pass would be the number of iterations needed for the cyclic variant of the algorithm to visit the entire dataset). In the first experiment Kaczmarz and CD methods perform similarly, while Kaczmarz methods work better in the second experiment and CD methods work better in the third experiment.

B.6 Example: Diagonal A

Consider a square diagonal matrix A with $a_{ii} > 0$ for all i. In this case, the diagonal entries are the eigenvalues λ_i of the A and $\sigma(A, 2) = \lambda_{\min}$. We give the convergence rate constants for such a diagonal A in Table B.1, and in this section we show how to arrive at these rates. We use U_∞ for the slower uniform rate to differentiate from U (tight uniform) for rate (B.8), and we use MR_∞ for rate (B.5) to differentiate it from MR (tight) rate (B.9).

For U_∞, the rate follows straight from $\|A\|_{\infty,2} = \max_i \|a_i\| = \max_i \lambda_i = \lambda_{\max}$. For U, we note that the weighted matrix $\bar{A} := D^{-1}A$ is simply the identity matrix. The NU rate uses that $\|A\|_F^2 = \sum_i \lambda_i^2$. For both MR_∞ and MR, we have

$$\sigma(A, \infty)^2 := \inf_{y \neq z} \frac{\|A(y-z)\|_\infty^2}{\|y-z\|^2} = \inf_{\|w\|=1} \|Aw\|_\infty^2.$$

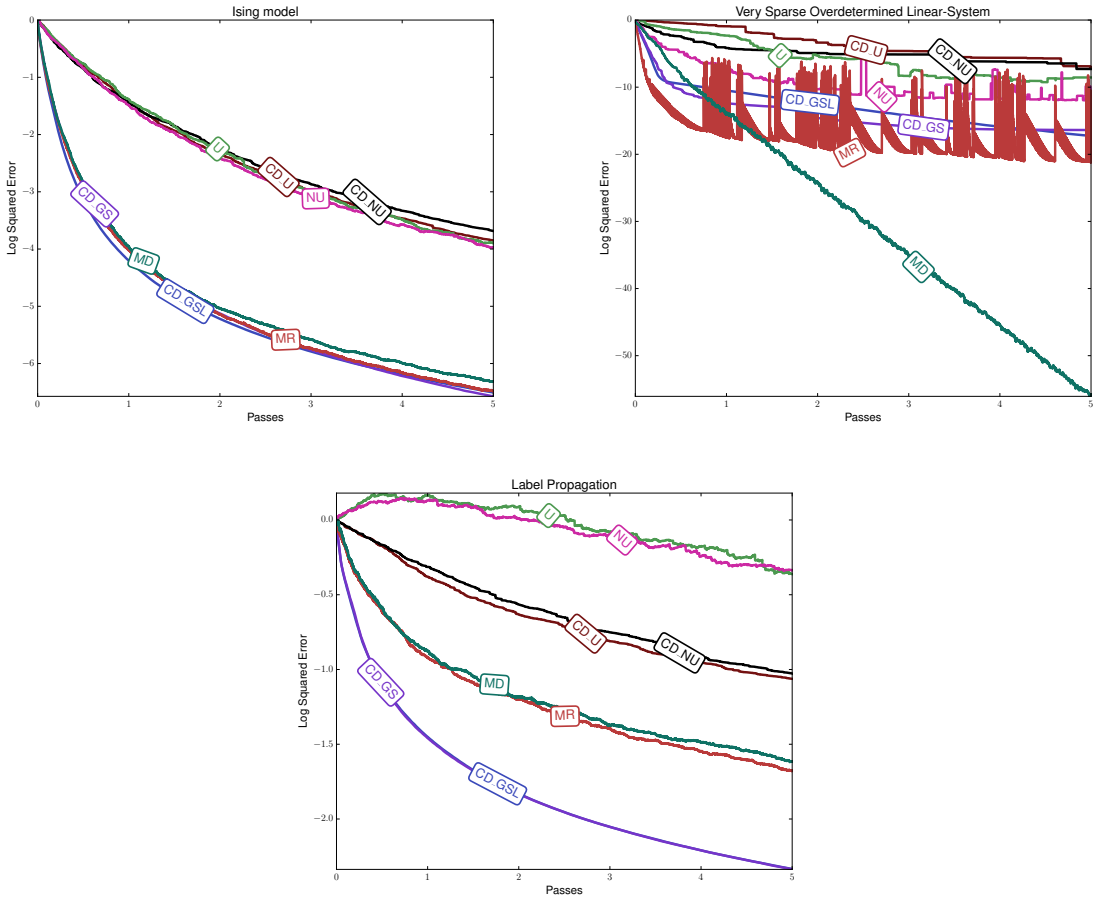


Figure B.1: Comparison of Kaczmarz and Coordinate Descent.

Consider the equivalent problem

$$\begin{aligned} \min_{w \in \mathbb{R}^m,\, y \in R} \quad & y \\ \text{s.t.} \quad & -y \leq \lambda_i^2 w_i^2 \leq y \text{ for all } i, \\ & \|w\| = 1, \end{aligned}$$

From the first inequality, we get

$$-\frac{y}{\lambda_i^2} \leq w_i^2 \leq \frac{y}{\lambda_i^2} \quad \forall i \quad \Rightarrow \quad (w_i)^2 \leq \frac{y}{\lambda_i^2} \quad \forall i.$$

It follows that

$$\|w\|^2 = \sum_{i=1}^m w_i^2 \leq \sum_{i=1}^m \frac{y}{\lambda_i^2},$$

Table B.1: Convergence Rate Constants for Diagonal A

Rule	Rate	Diagonal A
U_∞	$\left(1 - \frac{\sigma(A,2)^2}{m\|A\|_{\infty,2}^2}\right)$	$\left(1 - \frac{\lambda_{\min}^2}{m\lambda_{\max}^2}\right)$
U	$\left(1 - \frac{\sigma(\bar{A},2)^2}{m}\right)$	$\left(1 - \frac{1}{m}\right)$
NU	$\left(1 - \frac{\sigma(A,2)^2}{\|A\|_F^2}\right)$	$\left(1 - \frac{\lambda_{\min}^2}{\sum_i \lambda_i^2}\right)$
MR_∞	$\left(1 - \frac{\sigma(A,\infty)^2}{\|A\|_{\infty,2}^2}\right)$	$\left(1 - \frac{1}{\lambda_1^2}\left[\sum_i \frac{1}{\lambda_i^2}\right]^{-1}\right)$
MR	$\left(1 - \frac{\sigma(A,\infty)^2}{\|a_{i_k}\|^2}\right)$	$\left(1 - \frac{1}{\lambda_{i_k}^2}\left[\sum_i \frac{1}{\lambda_i^2}\right]^{-1}\right)$
MD	$\left(1 - \sigma(\bar{A},\infty)^2\right)$	$\left(1 - \frac{1}{m}\right)$

which is equivalent to

$$y \geq \frac{\|w\|^2}{\sum_{i=1}^m \frac{1}{\lambda_i^2}}.$$

Because we are minimizing y this must hold with equality at a solution, and because of the constraints $\|w\| = 1$ we have

$$\sigma(A,\infty)^2 = \left(\sum_i \frac{1}{\lambda_i^2}\right)^{-1}.$$

For the MR_∞ rate, we divide $\sigma(A,\infty)^2$ by the maximum eigenvalue squared. For the MR rate, we divide by the specific $\lambda_{i_k}^2$ corresponding to the row i_k selected at iteration k.

For the MD rule, following the argument we did to derive $\sigma(A,\infty)^2$ and using that $\bar{A} = I$ gives us

$$\sigma(\bar{A},\infty)^2 = \frac{1}{m}.$$

B.7 Multiplicative Error

Suppose we have approximated the MR selection rule such that there is a multiplicative error in our selection of i_k,

$$|a_{i_k}^T x^k - b_{i_k}| \geq \max_i |a_i^T x^k - b_i|(1 - \epsilon_k),$$

for some $\epsilon_k \in [0,1)$. In this scenario, we have

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \frac{1}{\|a_{i_k}\|^2}\left(\left|a_{i_k}^T x^k - b_{i_k}\right|^2\right) \\
&\leq \|x^k - x^*\|^2 - \frac{1}{\|a_{i_k}\|^2}\left(\max_i \left|a_i^T x^k - b_i\right|(1-\epsilon_k)\right)^2 \\
&= \|x^k - x^*\|^2 - \frac{(1-\epsilon_k)^2}{\|a_{i_k}\|^2}\|A(x^k - x^*)\|_\infty^2 \\
&\leq \left(1 - \frac{(1-\epsilon_k)^2\sigma(A,\infty)^2}{\|a_{i_k}\|^2}\right)\|x^k - x^*\|^2.
\end{aligned}$$

We define a multiplicative approximation to the MD rule as an i_k satisfying

$$\left|\frac{a_{i_k}^T x^k - b_{i_k}}{\|a_{i_k}\|}\right| \geq \max_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|(1-\bar{\epsilon}_k),$$

for some $\bar{\epsilon}_k \in [0,1)$. With such a rule we have

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \left(\left|\frac{a_{i_k}^T x^k - b_{i_k}}{\|a_{i_k}\|}\right|^2\right) \\
&\leq \|x^k - x^*\|^2 - \left(\max_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|(1-\bar{\epsilon}_k)\right)^2 \\
&= \|x^k - x^*\|^2 - (1-\bar{\epsilon}_k)^2 \max_i \left|\frac{a_i^T (x^k - x^*)}{\|a_i\|}\right|^2 \\
&= \|x^k - x^*\|^2 - (1-\bar{\epsilon}_k)^2\|D^{-1}A(x^k - x^*)\|_\infty^2 \\
&\leq \left(1 - (1-\bar{\epsilon}_k)^2\sigma(\bar{A},\infty)^2\right)\|x^k - x^*\|^2.
\end{aligned}$$

B.8 Additive Error

Suppose we select i_k using an approximate MR rule where

$$|a_{i_k}^T x^k - b_{i_k}|^2 \geq \max_i |a_i^T x^k - b_i|^2 - \epsilon_k,$$

for some $\epsilon_k \geq 0$. Then we have the following convergence rate,

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \frac{1}{\|a_{i_k}\|^2}\left|a_{i_k}^T x^k - b_{i_k}\right|^2 \\
&\leq \|x^k - x^*\|^2 - \frac{1}{\|a_{i_k}\|^2}\left(\max_i \left|a_i^T x^k - b_i\right|^2 - \epsilon_k\right) \\
&= \|x^k - x^*\|^2 - \frac{\|A(x^k - x^*)\|_\infty^2}{\|a_{i_k}\|^2} + \frac{\epsilon_k}{\|a_{i_k}\|^2} \\
&\leq \left(1 - \frac{\sigma(A,\infty)^2}{\|a_{i_k}\|^2}\right)\|x^k - x^*\|^2 + \frac{\epsilon_k}{\|a_{i_k}\|^2}.
\end{aligned}$$

For the MD rule with additive error, i_k is selected such that

$$\left|\frac{a_{i_k}^T x^k - b_{i_k}}{\|a_{i_k}\|}\right|^2 \geq \max_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|^2 - \bar{\epsilon}_k,$$

for some $\bar{\epsilon_k} \geq 0$. Then we have

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \left|\frac{a_{i_k}^T x^k - b_{i_k}}{\|a_{i_k}\|}\right|^2 \\
&\leq \|x^k - x^*\|^2 - \left(\max_i \left|\frac{a_i^T x^k - b_i}{\|a_i\|}\right|^2 - \bar{\epsilon_k}\right) \\
&= \|x^k - x^*\|^2 - \|D^{-1}A(x^k - x^*)\|_\infty^2 + \bar{\epsilon_k} \\
&\leq \left(1 - \sigma(\bar{A},\infty)^2\right)\|x^k - x^*\|^2 + \bar{\epsilon_k}.
\end{aligned}$$

B.9 Comparison of Rates for the Maximum Distance Rule and the Randomized Kaczmarz via Johnson-Lindenstrauss Method

In Eldar and Needell [2011], the authors assume that the rows of A are normalized and that we are dealing with a homogeneous system ($Ax = 0$), which is not particularly interesting since we can solve it in $O(1)$ by setting $x = 0$. Their main convergence result is stated in Theorem 1. Note that *RKJL* stands for *Randomized Kaczmarz via Johnson-Lindenstrauss*, which is a hybrid technique using both random selection and an approximate MD rule using the dimensionality reduction technique of Johnson and Lindenstrauss [1984]. In their work they give the result below.

Theorem 1 *Fix an estimation x^k and denote by x^{k+1} and x_{RK}^{k+1} the next estimations using the RKJL and the standard RK method, respectively. Define $\gamma_j = |\langle a_j, x^k\rangle|^2$ and ordering these so that $\gamma_1 \geq \gamma_2 \geq \cdots \geq \gamma_m$. Then, with δ being a constant affecting the error due to the JL*

approximation we have

$$\mathbb{E}\|x^{k+1} - x^*\|^2 \leq \min\left[\mathbb{E}\|x_{RK}^{k+1} - x\|^2 - \sum_{j=1}^{m}\left(p_j - \frac{1}{m}\right)\gamma_j + 2\delta, \quad \mathbb{E}\|x_{RK}^{k+1} - x^*\|^2\right],$$

where

$$p_j = \begin{cases} \frac{\binom{m-j}{n-1}}{\binom{m}{n}}, & j \leq m-n+1 \\ 0, & j > m-n+1 \end{cases}$$

are non-negative values satisfying $\sum_{j=1}^{m} p_j = 1$ *and* $p_1 \geq p_2 \geq \cdots \geq p_m = 0$.

First, we simplify this bound. Applying the nonuniform random rate of Strohmer and Vershynin [2009] to the result of Theorem 1, we get

$$\begin{aligned}
&\mathbb{E}\left[\|x^{k+1} - x\|^2\right] \\
&\leq \min\left[\mathbb{E}\left[\|x_{RK}^{k+1} - x^*\|^2\right] - \sum_{j=1}^{m}\left(p_j - \frac{1}{m}\right)\gamma_j + 2\delta, \quad \mathbb{E}\left[\|x_{RK}^{k+1} - x^*\|^2\right]\right] \\
&= \min\left[\|x^k - x^*\|^2 - \frac{1}{\|A\|_F^2}\sum_{j=1}^{m}\gamma_j - \sum_{j=1}^{m} p_j\gamma_j \;+\right. \\
&\qquad\qquad \left.\sum_{j=1}^{m}\frac{1}{m}\gamma_j + 2\delta, \|x^k - x^*\|^2 - \frac{1}{\|A\|_F^2}\sum_{j=1}^{m}\gamma_j\right] \\
&= \min\left[\|x^k - x^*\|^2 - \sum_{j=1}^{m} p_j\gamma_j + 2\delta, \quad \|x^k - x^*\|^2 - \frac{1}{m}\sum_{j=1}^{m}\gamma_j\right], \qquad \text{(B.15)}
\end{aligned}$$

where in the last line we use $\|A\|_F^2 = m$ for a matrix A with normalized rows (in this case of normalized rows non-uniform selection is simply uniform random selection). To compare this to our rate in the setting of an additive error, suppose we define ϵ_k such that the i_k selected satisfies

$$\gamma_{i_k} \geq \max_i \gamma_i - \bar{\epsilon}_k.$$

Then, noting that $\|a_i\| = 1$ for all i, our convergence rate with additive error is based on the bound

$$\begin{aligned}
\|x^{k+1} - x^*\|^2 &= \|x^k - x^*\|^2 - \gamma_{i_k} \\
&\leq \|x^k - x^*\|^2 - \max_i \gamma_i + \bar{\epsilon}_k. \qquad \text{(B.16)}
\end{aligned}$$

Comparing the bounds (B.15) and (B.16), we see that our MD bound is always faster in the case of exact optimization ($\bar{\epsilon}_k = \delta = 0$), as the average and the weighted sum of the absolute

inner products squared is less than the maximum inner product squared,

$$\max\left\{\frac{1}{m}\sum_{j=1}^{m}\gamma_j, \sum_{j=1}^{m}p_j\gamma_j\right\} \leq \max_i \gamma_i.$$

If there is error present, then our rate is faster when

$$\max_i \gamma_i - \epsilon_k \geq \max\left\{\frac{1}{m}\sum_{j=1}^{m}\gamma_j, \sum_{j=1}^{m}p_j\gamma_j - 2\delta\right\}.$$

We note that even if our approximation is worse than the error resulting from the RKJL method, $\epsilon_k \geq 2\delta$, it is possible that $\max_i \gamma_i$ is significantly larger than $\frac{1}{m}\sum_{j=1}^{m}\gamma_j$ and $\sum_{j=1}^{m}p_j\gamma_j$ and in this case our rate would be tighter. Further, our rate is more general as it does not specifically assume the Johnson-Lindenstrauss dimensionality reduction technique, that the rows of A are normalized, or that the linear system is homogeneous.

B.10 Systems of Linear Inequalities

Consider the system of linear equalities and inequalities,

$$\begin{cases} a_i^T x \leq b_i & (i \in I_{\leq}) \\ a_i^T x = b_i & (i \in I_{=}). \end{cases} \tag{B.17}$$

where the disjoint index sets $I_{\leq}$ and $I_{=}$ partition the set $\{1, 2, \ldots, m\}$. As presented by Leventhal and Lewis [2010], a generalization of the Kaczmarz algorithm that accommodates linear inequalities is given by

$$\beta_{i_k}^k = \begin{cases} (a_{i_k}^T x^k - b_{i_k})^+ & (i_k \in I_{\leq}) \\ a_{i_k}^T x^k - b_{i_k} & (i_k \in I_{=}), \end{cases}$$
$$x^{k+1} = x^k - \frac{\beta_{i_k}^k}{\|a_{i_k}\|^2} a_{i_k},$$

where for $x \in \mathbb{R}^n$ we define x^+ element-wise by

$$(x^+)_i = \max\{x_i, 0\}.$$

This leads to the following generalization of the MR and MD rules, respectively,

$$i_k = \max\left|\beta_i^k\right| = \|\beta^k\|_\infty, \quad \text{and} \quad i_k = \max\left|\frac{\beta_i^k}{\|a_i\|}\right| = \|D^{-1}\beta^k\|_\infty. \tag{B.18}$$

Unlike for equalities where the Kaczmarz method converges to the projection of the initial

iterate x^0 onto the intersection of the constraints, for inequalities we can only guarantee that the Kaczmarz method converges to a point in the feasible set. Thus, in convergence rates involving inequalities it is standard to use a bound for the distance from the current iterate x^k to the feasible region,

$$d(x, S) = \min_{z \in S} \|x - z\|_2 = \|x - P_S(x)\|_2,$$

where $P_S(x)$ is the projection of x onto the feasible set S.

Following closely the arguments of Leventhal and Lewis [2010] for systems of inequalities, we next give the following result which they credit to Hoffman [1952].

Theorem 11. *Let* (B.17) *be a consistent system of linear equalities and inequalities, then there exists a constant* $\sigma(A, \infty)$ *such that*

$$x \in \mathbb{R}^n \text{ and } S \neq \emptyset \quad \Rightarrow \quad d(x, S) \leq \frac{1}{\sigma(A, \infty)} \|e(Ax - b)\|_\infty,$$

where S *is the set of feasible solutions and where the function* $e : \mathbb{R}^m \mapsto \mathbb{R}^m$ *is defined by*

$$e(y)_i = \begin{cases} y_i^+ & (i \in I_{\leq}) \\ y_i & (i \in I_{=}). \end{cases}$$

From Leventhal and Lewis [2010], combining both cases ($i_k \in I_{\leq}$ or $i_k \in I_{=}$), the following relationship holds with respect to the distance measure $d(x, S)$,

$$d(x^{k+1}, S)^2 \leq d(x^k, S) - \frac{e(Ax^k - b)_{i_k}^2}{\|a_{i_k}\|^2}. \tag{B.19}$$

Following from this bound and Theorem 11, it is straightforward to derive analogous results for all greedy selection rates derived in this chapter. For example, if we select i_k according to the generalized MR rule (B.18) then the analogous tight rate for the MR rule is given by

$$\begin{aligned} d(x^{k+1}, S)^2 &\leq d(x^k, S)^2 - \frac{e(Ax^k - b)_{i_k}^2}{\|a_{i_k}\|^2} \\ &= d(x^k, S)^2 - \frac{\|\beta^k\|_\infty^2}{\|a_{i_k}\|^2} \\ &\leq \left(1 - \frac{\sigma(A, \infty)^2}{\|a_{i_k}\|^2}\right) d(x^k, S)^2. \end{aligned}$$

B.11 Faster Randomized Kaczmarz Using the Orthogonality Graph of A

In order for the adaptive methods to be efficient, we must be able to efficiently update the set of selectable nodes at each iteration. To do this we use a tree structure that keeps track of

the number of selectable children in the tree (for uniform random selection) or the cumulative sums of the selectable row norms of A (for non-uniform random selection). A similar structure is used in the non-uniform sampling code of Schmidt et al. [2017].

Recall that the standard inverse-transform approach approach to sampling from a non-uniform discrete probability distribution over m variables:

1. Compute the cumulative probabilities, $c_i = \sum_{j=1}^{i} p_j$ for each i from 1 to m.
2. Generate a random number u uniformly distributed over $[0, 1]$.
3. Return the smallest i such that $c_i \geq u$.

We can compute all m values of c_i in Step 1 at a cost of $O(m)$ by maintaining the running sum. We assume that Step 2 costs $O(1)$ and we can implement Step 3 in $O(\log(m))$ using a binary search. If we are sampling from a fixed distribution, then we only need to perform Step 1 once and from that point we can generate samples from the distribution at a cost of $O(\log(m))$.

In the adaptive randomized selection rules, the probabilities p_j change at each iteration and hence the c_i values also change. This means we we cannot skip Step 1 as we can for fixed probabilities. However, if the orthogonality graph is sparse then it is still possible to efficiently implement these strategies. To do this, we consider a binary tree-structure that has the probabilities p_j as leaf nodes while each internal node is the *sum* of its two descendants (and thus the root node has a value of 1). Given this structure, we can find the smallest $c_i \geq u$ in $O(\log(m))$ by traversing the tree. Further, if we update one of the p_j values then we can update this data structure in $O(\log(m))$ time since this only requires changing one node at each depth of the tree. If each node has at most g neighbours in the orthogonality graph, then we need to update g probabilities in the binary tree, leading to a cost of $O(g \log(m))$ to update the tree structure on each iteration.

Note that the above structure can be modified to work with *unnormalized probabilities* at the leaf nodes, since the root node will contain the normalizing constant required to make these unnormalized probabilities into a valid probability mass function. Using this, we can implement the adaptive uniform method by setting the leaf nodes to 1 for selectable nodes and 0 for non-selectable nodes. To implement the adaptive non-uniform method, we set the leaf nodes to 0 for non-selectable nodes and $\|a_i\|^2$ for selectable nodes.

B.12 Additional Experiments

Formulating the Semi-Supervised Label Propagation Problem as a Linear System

Our third experiment solves a label propagation problem for semi-supervised learning in the 'two moons' dataset [Zhou et al., 2003]. We use a variant of the quadratic labelling criterion of

Bengio et al. [2006],

$$\min_{y_i \in S'} \quad f(y) \equiv \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(y_i - y_j)^2,$$

where y is our label vector (each y_i can take one of 2 values), S is the set of labels that we do know, S' is the set of labels that we do not know and $w_{ij} \geq 0$ are the weights assigned to each y_i describing how strongly we want the labels y_i and y_j to be similar. We assume without loss of generality that $w_{ii} = 0$ (since it does not affect the objective) and that $w_{ij} = w_{ji}$ for all i, j because by the symmetry in the objective the model only depends on these terms through $(w_{ij} + w_{ji})$. We can express this quadratic problem as a linear system that is consistent by construction. In other words, we can define A and b such that

$$\nabla f(y) = 0 \iff Ay = b, \quad \text{with } y \in S'.$$

Differentiating f with respect to some $y_k \in S'$, we have

$$\begin{aligned}
\nabla_k f(y) &= \underbrace{\sum_{j \neq k} w_{kj}(y_k - y_j)}_{i=k,\ j \neq k} - \underbrace{\sum_{i \neq k} w_{ik}(y_i - y_k)}_{i \neq k,\ j=k} + \underbrace{\sum_{i=k} w_{kk}(y_k - y_k)}_{i=k,\ j=k} \\
&= \sum_{i=1}^{n} w_{ki}(y_k - y_i) - \sum_{i=1}^{n} w_{ik}(y_i - y_k) \\
&= 2\sum_{i=1}^{n} w_{ki} y_k - 2\sum_{i=1}^{n} w_{ki} y_i.
\end{aligned}$$

Setting this equal to zero and splitting the summation over S and S' separately, we have

$$\sum_{i=1}^{n} w_{ki} y_k - \sum_{i \in S'} w_{ki} y_i = \sum_{i \in S} w_{ki} y_i.$$

Assuming the elements of S' form the first $|S'|$ elements of the matrix A, the above formulation yields the $|S'| \times |S'|$ matrix with entries

$$A_{k,i} = \begin{cases} \sum_{j=1}^{n} w_{kj} & \text{if } i = k, \\ -w_{ki} & \text{if } i \neq k, \end{cases}$$

where k and $i \in S'$ and

$$b_k = \sum_{i \in S} w_{ki} y_i.$$

Hybrid Methods

For the very sparse overdetermined dataset, we see very different performances between the MR and MD rules with respect to squared error and distance. We see that the MR rule outperforms

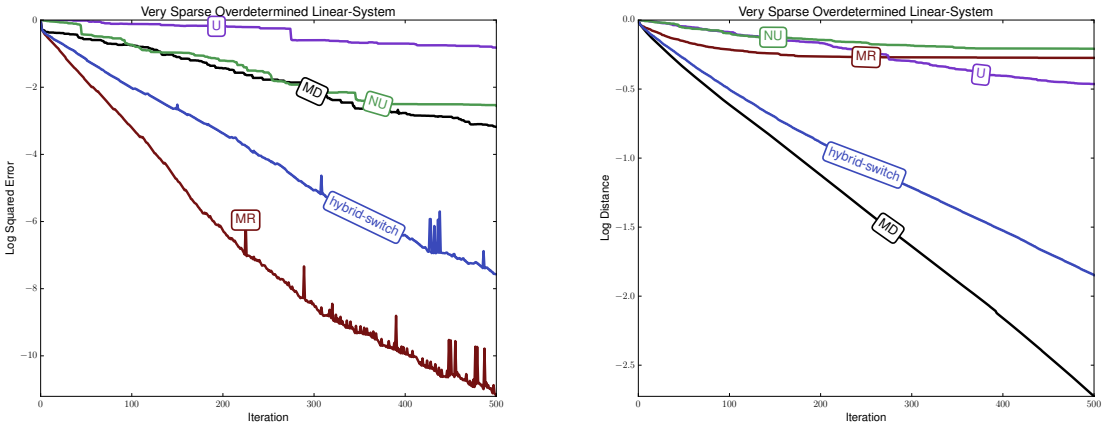


Figure B.2: Comparison of MR, MD and Hybrid Method for Very Sparse Dataset.

the MD rule in the beginning with respect to squared-error and the MD rule outperforms the MR rule significantly with respect to distance. These observations align with the respective definitions of each greedy rule. However, if we want a method that converges well with respect to *both* of these objectives, then we could consider 'hybrid' greedy rule. For example, we could simply alternate between using the MR rule and the MD rule. As we see in Figure B.2, this approach simultaneously exploits the convergence of the MR rule in terms of squared error and the MD rule in terms of distance to the solution. However, computationally this approach requires the maintenance of two max-heap structures.

Appendix C

Chapter 4 Supplementary Material

C.1 Relationships Between Conditions

Below we prove a subset of the implications in Theorem 2. The remaining relationships in Theorem 2 follow from these results and transitivity.

- **SC** $\rightarrow$ **ESC**: The SC assumption implies that the ESC inequality is satisfied for all x and y, so it is also satisfied under the constraint $x_p = y_p$.
- **ESC** $\rightarrow$ **WSC**: Take $y = x_p$ in the ESC inequality (which clearly has the same projection as x) to get WSC with the same μ as a special case.
- **WSC** $\rightarrow$ **RSI**: Re-arrange the WSC inequality to

  $$\langle \nabla f(x), x - x_p \rangle \geq f(x) - f^* + \frac{\mu}{2}\|x_p - x\|^2.$$

  Since $f(x) - f^* \geq 0$, we have RSI with $\frac{\mu}{2}$.
- **RSI** $\rightarrow$ **EB**: Using Cauchy-Schwartz on the RSI we have

  $$\|\nabla f(x)\|\|x - x_p\| \geq \langle \nabla f(x), x - x_p \rangle \geq \mu\|x_p - x\|^2,$$

  and dividing both sides by $\|x - x_p\|$ (assuming $x \neq x_p$) gives EB with the same μ (while EB clearly holds if $x = x_p$).
- **EB** $\rightarrow$ **PL**: By Lipschitz continuity we have

  $$f(x) \leq f(x_p) + \langle \nabla f(x_p), x - x_p \rangle + \frac{L}{2}\|x_p - x\|^2,$$

  and using EB along with $f(x_p) = f^*$ and $\nabla f(x_p) = 0$ we have

  $$f(x) - f^* \leq \frac{L}{2}\|x_p - x\|^2 \leq \frac{L}{2\mu^2}\|\nabla f(x)\|^2,$$

  which is the PL inequality with constant $\frac{\mu^2}{L}$.
- **PL** $\rightarrow$ **EB**: Below we show that PL implies QG. Using this result, while denoting the PL

constant with μ_p and the QG constant with μ_q, we get

$$\frac{1}{2}\|\nabla f(x)\|^2 \geq \mu_p(f(x) - f^*) \geq \frac{\mu_p\mu_q}{2}\|x - x_p\|^2,$$

which implies that EB holds with constant $\sqrt{\mu_p\mu_q}$.

- **QG + Convex → RSI**: By convexity we have

$$f(x_p) \geq f(x) + \langle \nabla f(x), x_p - x\rangle.$$

Re-arranging and using QG we get

$$\langle \nabla f(x), x - x_p\rangle \geq f(x) - f^* \geq \frac{\mu}{2}\|x_p - x\|^2,$$

which is RSI with constant $\frac{\mu}{2}$.

- **PL → QG**: Define the function

$$g(x) = \sqrt{f(x) - f^*}.$$

If we assume that f satisfies the PL inequality then for any $x \notin \mathcal{X}^*$ we have

$$\|\nabla g(x)\|^2 = \frac{\|\nabla f(x)\|^2}{f(x) - f^*} \geq 2\mu,$$

or that

$$\|\nabla g(x)\| \geq \sqrt{2\mu}. \tag{C.1}$$

By the definition of g, to show QG it is sufficient to show that

$$g(x) \geq \sqrt{2\mu}\|x - x_p\|. \tag{C.2}$$

As f is assumed to satisfy the PL inequality we have that f is an invex function and thus by definition g is a positive invex function ($g(x) \geq 0$) with a closed optimal solution set $\mathcal{X}^*$ such that for all $y \in \mathcal{X}^*$, $g(y) = 0$. For any point $x_0 \notin \mathcal{X}^*$, consider solving the following differential equation:

$$\begin{aligned} \frac{dx(t)}{dt} &= -\nabla g(x(t)) \\ x(t=0) &= x_0, \end{aligned} \tag{C.3}$$

for $x(t) \notin \mathcal{X}^*$. (This is a flow orbit starting at x_0 and flowing along the gradient of g.) By (C.1), ∇g is bounded from below, and as g is a positive invex function g is also bounded from below. Thus, by moving along the path defined by (C.3) we are sufficiently reducing the function and will eventually reach the optimal set. Thus there exists a T such that

$x(T) \in \mathcal{X}^*$ (and at this point the differential equation ceases to be defined). We can show this by starting from the gradient theorem for line integrals,

$$\begin{aligned}
g(x_0) - g(x_t) &= \int_{x_t}^{x_0} \langle \nabla g(x), dx \rangle \\
&= -\int_{x_0}^{x_t} \langle \nabla g(x), dx \rangle && \text{(flipping integral bounds)} \\
&= -\int_0^T \langle \nabla g(x(t)), \frac{dx(t)}{dt} \rangle \, dt && \text{(reparameterization)} \\
(*) \qquad &= \int_0^T \|\nabla g(x(t))\|^2 \, dt && \text{(from (C.3))} \\
&\geq \int_0^T 2\mu dt && \text{(from (C.1))} \\
&= 2\mu T.
\end{aligned}$$

As $g(x_t) \geq 0$, this shows we need to have $T \leq g(x_0)/2\mu$, so there must be a T with $x(T) \in \mathcal{X}^*$.

The *length* of the orbit $x(t)$ starting at x_0, which we denote by $\mathcal{L}(x_0)$, is given by

$$\mathcal{L}(x_0) = \int_0^T \|dx(t)/dt\| dt = \int_0^T \|\nabla g(x(t))\| \, dt \geq \|x_0 - x_p\|, \tag{C.4}$$

where x_p is the projection of x_0 onto $\mathcal{X}^*$ and the inequality follows because the orbit is a path from x_0 to a point in $\mathcal{X}^*$ (and thus it must be at least as long as the projection distance).

Starting from the line marked $(*)$ above we have

$$\begin{aligned}
g(x_0) - g(x_T) &= \int_0^T \|\nabla g(x(t))\|^2 \, dt \\
&\geq \sqrt{2\mu} \int_0^T \|\nabla g(x(t))\| \, dt && \text{(by (C.1))} \\
&\geq \sqrt{2\mu} \|x_0 - x_p\|. && \text{(by (C.4))}
\end{aligned}$$

As $g(x_T) = 0$, this yields our result (C.2), or equivalently

$$f(x) - f^* \geq 2\mu \|x - x_p\|^2,$$

which is QG with a different constant.

C.2 Relevant Problems

Strongly convex:
By minimizing both sides of the strong convexity inequality with respect to y we get

$$f(x^*) \geq f(x) - \frac{1}{2\mu}||\nabla f(x)||^2,$$

which implies the PL inequality holds with the same value μ. Thus, Theorem 1 exactly matches the known rate for gradient descent with a step-size of $1/L$ for a μ-strongly convex function.

Strongly convex composed with linear:
To show that this class of functions satisfies the PL inequality, we first define $f(x) := g(Ax)$ for a σ-strongly convex function g. For arbitrary x and y, we define $u := Ax$ and $v := Ay$. By the strong convexity of g, we have

$$g(v) \geq g(u) + \nabla g(u)^T(v-u) + \frac{\sigma}{2}\|v-u\|^2.$$

By our definitions of u and v, we get

$$g(Ay) \geq g(Ax) + \nabla g(Ax)^T(Ay - Ax) + \frac{\sigma}{2}\|Ay - Ax\|^2,$$

where we can write the middle term as $(A^T\nabla g(Ax))^T(y-x)$. By the definition of f and its gradient being $\nabla f(x) = A^T\nabla g(Ax)$ by the multivariate chain rule, we obtain

$$f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma}{2}||A(y-x)||^2.$$

Using x_p to denote the projection of x onto the optimal solution set $\mathcal{X}^*$, we have

$$\begin{aligned}
f(x_p) &\geq f(x) + \langle \nabla f(x), x_p - x\rangle + \frac{\sigma}{2}||A(x_p - x)||^2 \\
&\geq f(x) + \langle \nabla f(x), x_p - x\rangle + \frac{\sigma\theta(A)}{2}||x_p - x||^2 \\
&\geq f(x) + \min_y \left[\langle \nabla f(x), y - x\rangle + \frac{\sigma\theta(A)}{2}||y-x||^2\right] \\
&= f(x) - \frac{1}{2\theta(A)\sigma}||\nabla f(x)||^2.
\end{aligned}$$

In the second line we use that $\mathcal{X}^*$ is polyhedral, and use the theorem of Hoffman [1952] to obtain a bound in terms of $\theta(A)$ (the smallest non-zero singular value of A). This derivation implies that the PL inequality is satisfied with $\mu = \sigma\theta(A)$.

C.3 Sign-Based Gradient Methods

Defining a diagonal matrix Λ with $1/\sqrt{L_i}$ along the diagonal, the update can be written as

$$x^{k+1} = x^k - \|\nabla f(x^k)\|_{L^{-1}[1]} \Lambda \circ \operatorname{sign} \nabla f(x^k).$$

Consider the function $g(\tau) = f(x + \tau(y - x))$ with $\tau \in \mathbb{R}$. Then

$$\begin{aligned}
f(y) - f(x) - \langle \nabla f(x), y - x \rangle \\
&= g(1) - g(0) - \langle \nabla f(x), y - x \rangle \\
&= \int_0^1 \frac{dg}{d\tau}(\tau) - \langle \nabla f(x), y - x \rangle \; d\tau \\
&= \int_0^1 \langle \nabla f(x + \tau(y - x)), y - x \rangle - \langle \nabla f(x), y - x \rangle \; d\tau \\
&= \int_0^1 \langle \nabla f(x + \tau(y - x)) - \nabla f(x), y - x \rangle \; d\tau \\
&\leq \int_0^1 \|\nabla f(x + \tau(y - x)) - \nabla f(x)\|_{L^{-1}[1]} \|y - x\|_{L[\infty]} \; d\tau \\
&\leq \int_0^1 \tau \|y - x\|^2_{L[\infty]} \; d\tau \\
&= \tau^2 \frac{1}{2} \|y - x\|^2_{L[\infty]} \Big|_0^1 \\
&= \frac{1}{2} \|y - x\|^2_{L[\infty]} \\
&= \frac{1}{2} |y - x\|^2_{L[\infty]}.
\end{aligned}$$

where the second inequality uses the Lipschitz assumption, and in the first inequality we have used the Cauchy-Schwarz inequality and that the dual norm of the $L^{-1}[1]$ norm is the $L[\infty]$ norm. The above gives an upper bound on the function in terms of this $L[\infty]$-norm,

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{1}{2} \|y - x\|^2_{L[\infty]}.$$

Plugging in our iteration update we have

$$\begin{aligned}
&f(x^{k+1}) \\
&\leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k \rangle + \frac{1}{2}\|x^{k+1} - x^k\|^2_{L[\infty]} \\
&= f(x^k) - \|\nabla f(x^k)\|_{L^{-1}[1]} \langle \nabla f(x^k), \Lambda \circ \operatorname{sign} \nabla f(x^k) \rangle \; + \\
&\qquad\qquad \frac{\|\nabla f(x^k)\|^2_{L^{-1}[1]}}{2} \|\Lambda \circ \operatorname{sign} \nabla f(x^k)\|^2_{L[\infty]} \\
&= f(x^k) - \|\nabla f(x^k)\|^2_{L^{-1}[1]} + \frac{\|\nabla f(x^k)\|^2_{L^{-1}[1]}}{2} \left(\max_i \frac{1}{\sqrt{L_i}} \sqrt{L_i} |\operatorname{sign} \nabla_i f(x^k)| \right)^2 \\
&= f(x^k) - \frac{1}{2}\|\nabla f(x^k)\|^2_{L^{-1}[1]}.
\end{aligned}$$

Subtracting f^* from both sides yields

$$f(x^{k+1}) - f(x^*) \leq f(x^k) - f(x^*) - \frac{1}{2}\|\nabla f(x^k)\|^2_{L^{-1}[1]}.$$

Applying the PL inequality with respect to the $L^{-1}[1]$-norm (which, if the PL inequality is satisfied, holds for some $\mu_{L[\infty]}$ by the equivalence between norms),

$$\frac{1}{2}\|\nabla f(x^k)\|^2_{L^{-1}[1]} \geq \mu_{L[\infty]} \left(f(x^k) - f^* \right),$$

we have

$$f(x^{k+1}) - f(x^*) \leq \left(1 - \mu_{L[\infty]}\right) \left(f(x^k) - f(x^*) \right).$$

C.4 Proximal-PL Lemma

In this section we give a useful property of the function $\mathcal{D}_g$.

Lemma 5. *For any differentiable function f and any convex function g, given $\mu_2 \geq \mu_1 > 0$ we have*

$$\mathcal{D}_g(x, \mu_2) \geq \mathcal{D}_g(x, \mu_1).$$

We will prove Lemma 5 as a corollary of a related result. We first restate the definition

$$\mathcal{D}_g(x, \lambda) = -2\lambda \min_y \left[\langle \nabla f(x), y - x \rangle + \frac{\lambda}{2}||y - x||^2 + g(y) - g(x) \right], \tag{C.5}$$

and we note that we require $\lambda > 0$. By completing the square, we have

$$\begin{aligned}\mathcal{D}_g(x,\lambda) = -\min_y \Big[&- \|\nabla f(x)\|^2 + \|\nabla f(x)\|^2 + 2\lambda\langle \nabla f(x), y-x\rangle + \\ &\lambda^2||y-x||^2 + 2\lambda(g(y)-g(x))\Big] \\ = ||\nabla f(x)||^2 &- \min_y \left[||\lambda(y-x)+\nabla f(x)||^2 + 2\lambda(g(y)-g(x))\right].\end{aligned}$$

Notice that if $g = 0$, then $\mathcal{D}_g(x,\lambda) = ||\nabla f(x)||^2$ and the proximal-PL inequality reduces to the PL inequality. We will define the *proximal residual* function as the second part of the above equality,

$$\mathcal{R}_g(\lambda, x, a) \triangleq \min_y \left[||\lambda(y-x)+a||^2 + 2\lambda(g(y)-g(x)\right]. \tag{C.6}$$

Lemma 6. *If g is convex then for any x and a, and for $0 < \lambda_1 \leq \lambda_2$ we have*

$$\mathcal{R}_g(\lambda_1, x, a) \geq \mathcal{R}_g(\lambda_2, x, a). \tag{C.7}$$

Proof. Without loss of generality, assume $x = 0$. Then we have

$$\begin{aligned}\mathcal{R}_g(\lambda, a) &= \min_y \left[||\lambda y + a||^2 + 2\lambda(g(y)-g(0)\right] \\ &= \min_{\bar{y}} \left[||\bar{y}+a||^2 + 2\lambda(g(\bar{y}/\lambda)-g(0)\right],\end{aligned} \tag{C.8}$$

where in the second line we used a changed of variables $\bar{y} = \lambda y$ (note that we are minimizing over the whole space of $\mathbb{R}^n$). By the convexity of g, for any $\alpha \in [0,1]$ and $z \in \mathbb{R}^n$ we have

$$\begin{aligned}& g(\alpha z) \leq \alpha g(z) + (1-\alpha)g(0) \\ \iff \quad & g(\alpha z) - g(0) \leq \alpha(g(z)-g(0)).\end{aligned} \tag{C.9}$$

By using $0 < \lambda_1/\lambda_2 \leq 1$ and using the choices $\alpha = \frac{\lambda_1}{\lambda_2}$ and $z = \bar{y}/\lambda_1$ we have

$$\begin{aligned}& g(\bar{y}/\lambda_2) - g(0) \leq \frac{\lambda_1}{\lambda_2}(g(\bar{y}/\lambda_1) - g(0)) \\ \iff \quad & \lambda_2(g(\bar{y}/\lambda_2) - g(0)) \leq \lambda_1(g(\bar{y}/\lambda_1) - g(0)),\end{aligned} \tag{C.10}$$

Adding $||\bar{y}+a||^2$ to both sides, we get

$$||\bar{y}+a||^2 + \lambda_2(g(\bar{y}/\lambda_2) - g(0)) \leq ||\bar{y}+a||^2 + \lambda_1(g(\bar{y}/\lambda_1) - g(0)). \tag{C.11}$$

Taking the minimum over both sides with respect to $\bar{y}$ yields Lemma 6 due to (C.8). □

Corollary 2. *For any differentiable function f and convex function g, given $\lambda_1 \leq \lambda_2$, we have*

$$\mathcal{D}_g(x, \lambda_2) \geq \mathcal{D}_g(x, \lambda_1). \tag{C.12}$$

By using $\mathcal{D}_g(x, \lambda) = ||\nabla f(x)||^2 - \mathcal{R}_g(\lambda, x, \nabla f(x))$, Corollary 2 is exactly Lemma 5.

C.5 Relevant Problems

In this section we prove that the three classes of functions listed in Section 4.3.1 satisfy the proximal-PL inequality condition. Note that while we prove these hold for $\mathcal{D}_g(x, \lambda)$ for $\lambda \leq L$, by Lemma 5 above they also hold for $\mathcal{D}_g(x, L)$.

1. *$f(x)$, where f satisfies the PL inequality (g is constant)*:
   As g is assumed to be constant, we have $g(y) - g(x) = 0$ and the left-hand side of the proximal-PL inequality simplifies to

$$\begin{aligned}
\mathcal{D}_g(x, \mu) &= -2\mu \min_y \left\{ \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} \|y - x\|^2 \right\} \\
&= -2\mu \left(-\frac{1}{2\mu} \|f(x)\|^2 \right) \\
&= \|\nabla f(x)\|^2,
\end{aligned}$$

   Thus, the proximal PL inequality simplifies to f satisfying the PL inequality,

$$\frac{1}{2} \|\nabla f(x)\|^2 \geq \mu \left(f(x) - f^* \right),$$

   as we assumed.

2. *$F(x) = f(x) + g(x)$ and f is strongly convex*:
   By the strong convexity of f we have

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} ||y - x||^2, \tag{C.13}$$

   which leads to

$$F(y) \geq F(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} ||y - x||^2 + g(y) - g(x). \tag{C.14}$$

   Minimizing both sides respect to y,

$$\begin{aligned}
F^* &\geq F(x) + \min_y \langle \nabla f(x), y - x \rangle + \frac{\mu}{2} ||y - x||^2 + g(y) - g(x) \\
&= F(x) - \frac{1}{2\mu} \mathcal{D}_g(x, \mu).
\end{aligned} \tag{C.15}$$

   Rearranging, we have our result.

3. *$F(x) = f(Ax) + g(x)$ and f is strongly convex, g is the indicator function for a polyhedral set $\mathcal{X}$, and A is a linear transformation*:
By defining $\tilde{f}(x) = f(Ax)$ and using strong convexity of f, we have

$$\tilde{f}(y) \geq \tilde{f}(x) + \langle \nabla \tilde{f}(x), y - x \rangle + \frac{\mu}{2}||A(y-x)||^2, \tag{C.16}$$

which leads to

$$F(y) \geq F(x) + \langle \nabla \tilde{f}(x), y - x \rangle + \frac{\mu}{2}||A(y-x)||^2 + g(y) - g(x). \tag{C.17}$$

Since $\mathcal{X}$ is polyhedral, it can be written as a set $\{x : Bx \leq c\}$ for a matrix B and a vector c. As before, assume that x_p is the projection of x onto the optimal solution set $\mathcal{X}^*$ which in this case is $\{x : Bx \leq c, Ax = z\}$ for some z.

$$\begin{aligned}
&F^* - F(x) = F(x_p) - F(x) \\
&\geq \langle \nabla \tilde{f}(x), x_p - x \rangle + \frac{\mu}{2}||A(x - x_p)||^2 + g(x_p) - g(x) \\
&= \langle \nabla \tilde{f}(x), x_p - x \rangle + \frac{\mu}{2}||Ax - z||^2 + g(x_p) - g(x) \\
&= \langle \nabla \tilde{f}(x), x_p - x \rangle + \frac{\mu}{2}||\{Ax - z\}_+ + \{-Ax + z\}_+||^2 + g(x_p) - g(x) \\
&= \langle \nabla \tilde{f}(x), x_p - x \rangle + \frac{\mu}{2}\left\| \left\{ \begin{bmatrix} A \\ -A \\ B \end{bmatrix} x - \begin{bmatrix} z \\ -z \\ c \end{bmatrix} \right\}_+ \right\|^2 + g(x_p) - g(x) \\
&\geq \langle \nabla \tilde{f}(x), x_p - x \rangle + \frac{\mu\theta(A,B)}{2}||x - x_p||^2 + g(x_p) - g(x) \\
&\geq \min_y \left[\langle \nabla \tilde{f}(x), y - x \rangle + \frac{\mu\theta(A,B)}{2}||y - x||^2 + g(y) - g(x) \right] \\
&= -\frac{1}{2\mu\ \theta(A)}\mathcal{D}_g(x, \mu\theta(A,B)).
\end{aligned}$$

where we have used the notation that $\{\cdot\}_+ = \max\{0, \cdot\}$, the fourth equality follows because x was projected onto $\mathcal{X}$ in the previous iteration (so $Bx - c \leq 0$), and the line after that uses Hoffman's bound [Hoffman, 1952].

4. *$F(x) = f(x) + g(x)$, f is convex, and F satisfies the quadratic growth (QG) condition*:
A function F satisfies the QG condition if

$$F(x) - F^* \geq \frac{\mu}{2}||x - x_p||^2. \tag{C.18}$$

For any $\lambda > 0$ we have,

$$\begin{aligned}
& \min_y \left[\langle \nabla f(x), y - x \rangle + \frac{\lambda}{2} ||y - x||^2 + g(y) - g(x) \right] \\
\leq \; & \langle \nabla f(x), x_p - x \rangle + \frac{\lambda}{2} ||x_p - x||^2 + g(x_p) - g(x) \\
\leq \; & f(x_p) - f(x) + \frac{\lambda}{2} ||x_p - x||^2 + g(x_p) - g(x) \\
= \; & \frac{\lambda}{2} ||x_p - x||^2 + F^* - F(x) \\
\leq \; & \left(1 - \frac{\lambda}{\mu}\right) (F^* - F). \qquad \text{(C.19)}
\end{aligned}$$

The third line follows from the convexity of f, and the last inequality uses the QG condition of F. Multiplying both sides by -2λ, we have

$$\begin{aligned}
\mathcal{D}_g(x, \lambda) &= -2\lambda \min_y \left[\langle \nabla \tilde{f}(x), y - x \rangle + \frac{\lambda}{2} ||y - x||^2 + g(y) - g(x) \right] \\
&\geq 2\lambda \left(1 - \frac{\lambda}{\mu}\right) (F(x) - F^*). \qquad \text{(C.20)}
\end{aligned}$$

This is true for any $\lambda > 0$, and by choosing $\lambda = \mu/2$ we have

$$\mathcal{D}_g(x, \mu/2) \geq \frac{\mu}{2}(F(x) - F^*). \qquad \text{(C.21)}$$

C.6 Proximal Coordinate Descent

In this section, we show linear convergence of randomized coordinate descent for $F(x) = f(x) + g(x)$ assuming that F satisfies the proximal PL inequality, ∇f is coordinate-wise Lipschitz continuous, and g is a separable convex function ($g(x) = \sum_i g_i(x_i)$).

From coordinate-wise Lipschitz continuity of ∇f and separability of g, we have

$$F(x + y_i e_i) - F(x) \leq y_i \nabla_i f(x) + \frac{L}{2} y_i^2 + g_i(x_i + y_i) - g(x_i). \qquad \text{(C.22)}$$

Given a coordinate i the coordinate descent step chooses y_i to minimize this upper bound on the improvement in F,

$$y_i = \underset{t_i \in \mathbb{R}}{\operatorname{argmin}} \left\{ t_i \nabla_i f(x) + \frac{L}{2} t_i^2 + g_i(x_i + t_i) - g(x_i). \right\}$$

We next use an argument similar to Richtárik and Takáč [2014] to relate the expected improve-

ment (with random selection of the coordinates) to the function $\mathcal{D}_g$,

$$\begin{aligned}
&\mathbb{E}\left\{\min_{t_i} t_i\nabla_i f(x) + \frac{L}{2}t_i^2 + g_i(x_i+t_i) - g_i(x_i)\right\} \\
&= \frac{1}{n}\sum_i \min_{t_i} t_i\nabla_i f(x) + \frac{L}{2}t_i^2 + g_i(x_i+t_i) - g_i(x_i) \\
&= \frac{1}{n}\min_{t_1,\cdots,t_n}\sum_i t_i\nabla_i f(x) + \frac{L}{2}t_i^2 + g_i(x_i+t_i) - g_i(x_i) \\
&= \frac{1}{n}\min_{y\equiv x+(t_1,\cdots,t_n)} \langle\nabla f(x), y-x\rangle + \frac{L}{2}||y-x||^2 + g(y) - g(x) \\
&= -\frac{1}{2Ln}\mathcal{D}_g(L,x).
\end{aligned}$$

(Note that separability allows us to exchange the summation and minimization operators.) By using this and taking the expectation of (C.22) we get

$$\mathbb{E}\left[F(x^{k+1})\right] \leq F(x^k) - \frac{1}{2Ln}\mathcal{D}_g(L,x). \tag{C.23}$$

Subtracting F^* from both sides and applying the proximal-PL inequality yields a linear convergence rate of $\left(1-\frac{\mu}{nL}\right)$.

Appendix D

Chapter 5 Supplementary Material

D.1 Cost of Multi-Class Logistic Regression

The typical setting where we expect coordinate descent to outperform gradient descent is when the cost of one gradient descent iteration is similar to the cost of updating all variables via coordinate descent. It is well known that for the binary logistic regression objective, one of the most ubiquitous models in machine learning, coordinate descent with uniform random selection satisfies this property. As seen in Appendix A.1.2 this property is also satisfied for the GS rule in the case of logistic regression, provided that the data is sufficiently sparse.

In this section we consider *multi-class* logistic regression. We first analyze the cost of gradient descent on this objective and how randomized coordinate descent is efficient for any sparsity level. Then we show that a high sparsity level is not sufficient for the GS rule to be efficient for this problem, but that it is efficient if we use a particular set of fixed blocks.

D.1.1 Cost of Gradient Descent

The likelihood for a single training example i with features $a_i \in \mathbb{R}^d$ and a label $b_i \in \{1, 2, \dots, k\}$ is given by

$$p(b_i|a_i, X) = \frac{\exp(x_{b_i}^T a_i)}{\sum_{c=1}^k \exp(x_c^T a_i)},$$

where x_c is column c of our matrix of parameters $X \in \mathbb{R}^{d\times k}$ (so the number of parameters n is dk). To maximize the likelihood over m independent and identically-distributed training examples we minimize the negative log-likelihood,

$$f(X) = \sum_{i=1}^m \left[-x_{b_i}^T a_i + \log\left(\sum_{c=1}^k \exp(x_c^T a_i) \right) \right], \tag{D.1}$$

which is a convex function. The partial derivative of this objective with respect to a particular X_{jc} is given by

$$\frac{\partial}{\partial X_{jc}} f(X) = -\sum_{i=1}^m a_{ij} \left[I(b_i = c) - \frac{\exp(x_c^T a_i)}{\sum_{c'=1}^k \exp(x_{c'}^T a_i)} \right], \tag{D.2}$$

where I is a 0/1 indicator variable and a_{ij} is feature j for training example i. We use A to denote a matrix where row i is given by a_i^T. To compute the full gradient, the operations which

depend on the size of the problem are:

1. Computing $x_c^T a_i$ for all values of i and c.
2. Computing the sums $\sum_{c=1}^{k} \exp(x_c^T a_i)$ for all values of i.
3. Computing the partial derivative sums (D.2) for all values of j and c.

The first step is the result of the matrix multiplication AX, so if A has z non-zeroes then this has a cost of $O(zk)$ if we compute it using k matrix-vector multiplications. The second step costs $O(mk)$, which under the reasonable assumption that $m \leq z$ (since each row usually has at least one non-zero) is also in $O(zk)$. The third step is the result of a matrix multiplication of the form $A^T R$ for a (dense) m times k matrix R (whose elements have a constant-time cost to compute given the results of the first two steps), which also costs $O(zk)$ giving a final cost of $O(zk)$.

D.1.2 Cost of Randomized Coordinate Descent

Since there are $n = dk$ variables, we want our coordinate descent iterations to be dk-times faster than the gradient descent cost of $O(zk)$. Thus, we want to be able to implement coordinate descent iterations for a cost of $O(z/d)$ (noting that we always expect $z \geq d$ since otherwise we could remove some columns of A that only have zeroes). The key to doing this for randomized coordinate descent is to track two quantities:

1. The values $x_c^T a_i$ for all i and c.
2. The values $\sum_{c'=1}^{k} \exp(x_{c'}^T a_i)$ for all i.

Given these values we can compute the partial derivative in $O(z/d)$ in expectation, because this is the expected number of non-zero values of a_{ij} in the partial derivative sum (D.2) (A has z total non-zeroes and we are randomly choosing one of the d columns). Further, after updating a particular X_{jc} we can update the above quantities for the same cost:

1. We need to update $x_c^T a_i$ for the particular c we chose for the examples i where a_{ij} is non-zero for the chosen value of j. This requires an $O(1)$ operation (subtract the old $x_{jc} a_{ij}$ and add the new value) for each non-zero element of column j of A. Since A has z non-zeroes and d columns, the expected number of non-zeroes is z/d so this has a cost of $O(z/d)$.
2. We need to update $\sum_{c'=1}^{k} \exp(x_{c'}^T a_i)$ for all i where a_{ij} is non-zero for our chosen j. Since we expect z/d non-zero values of a_{ij}, the cost of this step is also $O(z/d)$.

Note that BCD is also efficient since if we update τ elements, the cost is $O(z\tau/d)$ by just applying the above logic τ times. In fact, step 2 and computing the final partial derivative has some redundant computation if we update multiple X_{jc} with the same c, so we might have a small performance gain in the block case.

D.1.3 Cost of Greedy Coordinate Descent (Arbitrary Blocks)

The cost of greedy coordinate descent is typically higher than randomized coordinate descent since we need to track *all* partial derivatives. However, consider the case where each row has at most z_r non-zeroes and each column has at most z_c non-zeroes. In this setting we previously showed that for binary logistic regression it is possible to track all partial derivatives for a cost of $O(z_r z_c)$, and that we can track the maximum gradient value at the cost of an additional logarithmic factor (see Appendix A.1).[28] Thus, greedy coordinate selection has a similar cost to uniform selection when the sparsity pattern makes $z_r z_c$ similar to z/d (as in the case of a grid-structured dependency graph like Figure 5.3).

Unfortunately, having $z_r z_c$ similar to z/d is not sufficient in the multi-class case. In particular, the cost of tracking all the partial derivatives after updating an X_{jc} in the multi-class case can be broken down as follows:

1. We need to update $x_c^T a_i$ for the examples i where a_{ij} is non-zero. Since there are at most z_c non-zero values of a_{ij} over all i the cost of this is $O(z_c)$.

2. We need to update $\sum_{c=1}^{k} \exp(x_c^T a_i)$ for all i where a_{ij} is non-zero. Since there are at most z_c non-zero values of a_{ij} the cost of this is $O(z_c)$.

3. We need to update the partial derivatives $\partial f / \partial X_{jc}$ for all j and c. Observe that each time we have a_{ij} non-zero, we change the partial derivative with respect to all features j' that are non-zero in the example i and we must update all classes c' for these examples. Thus, for the $O(z_c)$ examples with a non-zero feature j we need to update up to $O(z_r)$ other features for that example and for each of these we need to update all k classes. This gives a cost of $O(z_r z_c k)$.

So while in the binary case we needed $O(z_r z_c)$ to be comparable to $O(z/d)$ for greedy coordinate descent to be efficient, in the multi-class case we now need $O(z_r z_c k)$ to be comparable to $O(z/d)$ in the multi-class case. This means that not only do we need a high degree of sparsity but we also need the number of classes k to be small for greedy coordinate descent to be efficient.

D.1.4 Cost of Greedy Coordinate Descent (Fixed Blocks)

Greedy rules are more expensive in the multi-class case because whenever we change an individual variable X_{jc}, it changes the partial derivative with respect to $X_{j'c'}$ for a set of j' values and for *all* c'. But we can improve the efficiency of greedy rules by using a special choice of fixed blocks that reduces the number of j' values. In particular, BCD is more efficient for the multi-class case if we put $X_{jc'}$ for all c' into the same block. In other words, we ensure that each row of X is part of the same block so that we apply BCD to rows rather than in an

[28] Note that the purpose of the quantity $z_r z_c$ is to serve as a potentially-crude upper bound on the maximum degree in the dependency graph we describe in Section 5.4. Any tighter bound on this degree would yield a tighter upper bound on the runtime.

unstructured way. Below we consider the cost of updating the needed quantities after changing an entire row of X_{jc} values:

1. Since we are updating k elements, the cost of updating the $x_c^T a_i$ is k-times larger giving $O(z_c k)$ when we update a row.

2. Similarly, the cost of updating the sums $\sum_{c=1}^{k} \exp(x_c^T a_i)$ is k-times larger also giving $O(z_c k)$.

3. Where we gain in computation is the cost of computing the changed values of the partial derivatives $\partial f / \partial X_{jc}$. As before, each time we have a_{ij} non-zero for our particular row j, we change the partial derivative with respect to all other j' for this example and with respect to each class c' for these j'. Thus, for the $O(z_c)$ examples with a non-zero feature j we need to update up to $O(z_r)$ other features for that example and for each of these we need to update all k classes. But since j is the same for each variable we update, we only have to do this once which gives us a cost of $O(z_r z_c k)$.

So the cost to update a row of the matrix X is $O(z_r z_c k)$, which is the same cost as only updating a single element. Considering the case of updating individual rows, this gives us d blocks so in order for BCD to be efficient it must be d-times faster than the gradient descent cost of $O(zk)$. Thus, we need a cost of $O(zk/d)$ per iteration. This is achieved if $O(z_r z_c)$ to be similar to $O(z/d)$, which is the same condition we needed in the binary case.

D.2 Blockwise Lipschitz Constants

In this section we show how to derive lower-bounds on the block-Lipschitz constants of the gradient and Hessian for several common problem settings. We will use that a twice-differentiable function has an L-Lipschitz continuous gradient if and only if the absolute eigenvalues of its Hessian are upper-bounded by L,

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\| \iff \nabla^2 f(x) \preceq LI.$$

This implies that when considering blockwise constants we have

$$\|\nabla_b f(x + U_b d) - \nabla_b f(x)\| \iff \nabla^2_{bb} f(x) \preceq L_b I.$$

Thus, bounding the blockwise eigenvalues of the Hessian bounds the blockwise Lipschitz constants of the gradient. We also use that this equivalence extends to the case of general quadratic norms,

$$\|\nabla_b f(x + U_b d) - \nabla_b f(x)\|_{H_b^{-1}} \leq \|d\|_{H_b} \iff \nabla^2_{bb} f(x) \preceq H_b.$$

D.2.1 Quadratic Functions

Quadratic functions have the form

$$f(x) = \frac{1}{2}x^T Ax + c^T x,$$

for a positive semi-definite matrix A and vector c. For all x the Hessian with respect to block b is given by the sub-matrix of A,

$$\nabla^2_{bb} f(x) = A_{bb}.$$

Thus, we have that L_b is given by the maximum eigenvalue of the submatrix, $L_b = \|A_b\|$ (the operator norm of the submatrix). In the special case where b only contains a single element i, we have that L_i is given by the absolute value of the diagonal element, $L_i = |A_{ii}|$. If we want to use a general quadratic norm we can simply take $H_b = A_{bb}$, which is cheaper to compute than the L_b (since it does not require an eigenvalue calculation).

D.2.2 Least Squares

The least squares objective has the form

$$f(x) = \frac{1}{2}\|Ax - c\|^2,$$

for a matrix A and vector c. This is a special case of a quadratic function, where the Hessian is given by

$$\nabla^2 f(x) = A^T A.$$

This gives us that $L_b = \|A_b\|^2$ (where A_b is the matrix containing the columns b of A). In the special case where the block has a single element j, observe that $L_j = \sum_{i=1}^m a_{ij}^2$ (sum of the squared values in column j) so we do not need to solve an eigenvalue problem. When using a quadratic norm we can take $H_b = A_b^T A_b$ which similarly does not require solving an eigenvalue problem.

D.2.3 Logistic Regression

The likelihood of a single example in a logistic regression model is given by

$$p(b_i|a_i, x) = \frac{1}{1 + \exp(-b_i x^T a_i)},$$

where each $a_i \in \mathbb{R}^d$ and $b_i \in \{-1, 1\}$. To maximize the likelihood over m examples (sampled independently) we minimize the negative log-likelihood,

$$f(x) = \sum_{i=1}^m \log(1 + \exp(-b_i x^T a_i)).$$

Using A as a matrix where row i is given by a_i^T and defining $h_i(x) = p(b_i|a_i, x)$, we have that

$$\begin{aligned}\nabla^2 f(x) &= \sum_{i=1}^{m} h_i(x)(1 - h_i(x))a_i a_i^T \\ &\preceq 0.25 \sum_{i=1}^{m} a_i a_i^T \\ &= 0.25 A^T A.\end{aligned}$$

The generalized inequality above is the binary version of the Bohning bound [Böhning, 1992]. This bound can be derived by observing that $h_i(x)$ is in the range $(0, 1)$, so the quantity $h_i(x)(1-h_i(x))$ has an upper bound of 0.25. This result means that we can use $L_b = 0.25\|A_b\|^2$ for block b, $L_j = 0.25\sum_{i=1}^{m} a_{ij}^2$ for single-coordinate blocks, and $H_b = 0.25 A_b^T A_b$ if we are using a general quadratic norm (notice that computing H_b is again cheaper than computing L_b).

D.2.4 Multi-Class Logistic Regression

The Hessian of the multi-class logistic regression objective (D.1) with respect to parameter vectors x_c and $x_{c'}$ can be written as

$$\frac{\partial^2}{\partial x_c \partial x_c'} f(X) = \sum_{i=1}^{m} h_{i,c}(X)(I(c = c') - h_{i,c'}(X))a_i a_i^T,$$

where similar to the binary logistic regression case we have defined $h_{i,c} = p(c|a_i, X)$. This gives the full Hessian the form

$$\nabla^2 f(X) = \sum_{i=1}^{m} H_i(X) \otimes a_i a_i^T,$$

where we used $\otimes$ to denote the Kronecker product and where element (c, c') of the k by k matrix $H_i(X)$ is given by $h_{i,c}(X)(I(c = c') - h_{i,c'}(X))$. Bohning's bound [Böhning, 1992] on this matrix is that

$$H_i(X) \preceq \frac{1}{2}\left(I - \frac{1}{k} 1 1^T\right),$$

where 1 is a vector of ones while recall that k is the number of classes. Using this we have

$$\begin{aligned}\nabla^2 f(X) &\preceq \sum_{i=1}^{m} \frac{1}{2}\left(I - \frac{1}{k} 1 1^T\right) \otimes a_i a_i^T \\ &= \frac{1}{2}\left(I - \frac{1}{k} 1 1^T\right) \otimes \sum_{i=1}^{m} a_i a_i^T \\ &= \frac{1}{2}\left(I - \frac{1}{k} 1 1^T\right) \otimes A^T A.\end{aligned}$$

As before we can take submatrices of this expression as our H_b, and we can take eigenvalues of the submatrices as our L_b. However, due to the $1/k$ factor we can actually obtain tighter bounds for sub-matrices of the Hessian that do not involve at least two of the classes. In particular, consider a sub-Hessian involving the variables only associated with k' classes for $k' < k$. In this case we can replace the k by k matrix $(I - (1/k)11^T)$ with the k' by k' matrix $(I - (1/(k'+1))11^T)$. The "+1" added to k' in the second term effectively groups all the other classes (whose variables are fixed) into a single class (the "+1" is included in Bohning's original paper as he fixes $x_k = 0$ and defines k to be one smaller than the number of classes). This means (for example) that we can take $L_j = 0.25\sum_{i=1}^m a_{ij}^2$ as in the binary case rather than the slightly-larger diagonal element $0.5(1-1/k)\sum_{i=1}^m a_{ij}^2$ in the matrix above.[29]

D.3 Derivation of GSD Rule

In this section we derive a progress bound for twice-differentiable convex functions when we choose and update the block b_k according to the GSD rule with $D_{b,i} = L_i\tau$ (where τ is the maximum block size). We start by using the Taylor series representation of $f(x^{k+1})$ in terms of $f(x^k)$ and some z between x^{k+1} and x^k (keeping in mind that these only differ along coordinates in b_k),

$$\begin{aligned}
f(x^{k+1}) &= f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k\rangle + \frac{1}{2}(x^{k+1} - x^k)^T \nabla^2_{b_k b_k} f(z)(x^{k+1} - x^k) \\
&\leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k\rangle + \frac{|b_k|}{2}\sum_{i\in b_k} \nabla^2_{ii} f(z)(x_i^{k+1} - x_i^k)^2 \\
&\leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k\rangle + \frac{\tau}{2}\sum_{i\in b_k} \nabla^2_{ii} f(z)(x_i^{k+1} - x_i^k)^2 \\
&\leq f(x^k) + \langle \nabla f(x^k), x^{k+1} - x^k\rangle + \frac{\tau}{2}\sum_{i\in b_k} L_i(x_i^{k+1} - x_i^k)^2,
\end{aligned}$$

where the first inequality follows from convexity of f which implies that $\nabla^2_{b_k b_k} f(x^k)$ is positive semi-definite and by Lemma 1 of Nesterov's coordinate descent paper [Nesterov, 2010]. The second inequality follows from the definition of τ and the third follows from the definition of L_i. Now using our choice of $D_{b,i} = L_i\tau$ in the update we have for $i \in b_k$ that

$$x_i^{k+1} = x_i^k - \frac{1}{L_i\tau}\nabla_i f(x^k),$$

[29] The binary logistic regression case can conceptually be viewed as a variation on the softmax loss where we fix $x_c = 0$ for one of the classes and thus are always only updating variables from class. This gives the special case of $0.5(I - 1/(k+1)11^T)A^TA = 0.5(1-0.5)A^TA = 0.25A^TA$, the binary logistic regression bound from the previous section.

which yields

$$\begin{aligned} f(x^{k+1}) &\leq f(x^k) - \frac{1}{2\tau} \sum_{i \in b_k} \frac{|\nabla_i f(x^k)|^2}{L_i} \\ &= f(x^k) - \frac{1}{2} \max_b \sum_{i \in b} \frac{|\nabla_i f(x^k)|^2}{L_i \tau} \\ &= f(x^k) - \|\nabla f(x^k)\|_{\mathcal{B}}^2. \end{aligned}$$

The first equality uses that we are selecting b_k using the GSD rule with $D_{b,i} = L_i\tau$ and the second inequality follows from the definition of of the mixed norm $\|\cdot\|_{\mathcal{B}}$ from Section 5.2.5 with $H_b = D_b$. This progress bound implies that the convergence rate results in that section also hold.

D.4 Efficiently Testing the Forest Property

In this section we give a method to test whether adding a node to an existing forest maintains the forest property. In this setting our input is an undirected graph G and a set of nodes b whose induced subgraph G_b forms a forest (has no cycles). Given a node i, we want to test whether adding i to b will maintain that the induced subgraph is acyclic. In this section we show how to do this in $O(p)$, where p is the degree of the node i.

The method is based on the following simple observations:

- If the new node i introduces a cycle, then it must be part of the cycle. This follows because G_b is assumed to be acyclic, so no cycles can exist that do not involve i.

- If i introduces a cycle, we can arbitrarily choose i to be the start and end point of the cycle.

- If the new node i has 1 or fewer neighbours in b, then it does not introduce a cycle. With no neighbours it clearly can not be part of a cycle. With one neighbour, we would have to traverse its one edge more than once to have it start and end a path.

- If the new node i has at least 2 neighbours in b that are part of the same tree, then i introduces a cycle. Specifically, we can construct a cycle as follows: we start at node i, go to one of its neighbours, follow a path through the tree to another one of its neighbours in the same tree (such a path exists because trees are connected by definition), and then return to node i.

- If the new node i has at least 2 neighbours in b but they are all in different trees, then i does not introduce a cycle. This is similar to the case where i has only one edge: any path that starts and ends at node i would have to traverse one of its edges more than once (because the disjoint trees are not connected to each other).

The above cases suggest that to determine whether adding node i to the forest b maintains the forest property, we only need to test whether node i is connected to two nodes that are part of the same tree in the existing forest. We can do this in $O(p)$ using the following data structures:

1. For each of the n nodes, a list of the adjacent nodes in G.
2. A set of n labels in $\{0, 1, 2, \ldots, t\}$, where t is the number of trees in the existing forest. This number is set to 0 for nodes that are not in b, is set to 1 for nodes in the first tree, is set to 2 for nodes in the second tree, and so on.

Note that there is no ordering to the labels $\{1, 2, \ldots, t\}$, each tree is just assigned an arbitrary number that we will use to determine if nodes are in the same tree. We can find all neighbours of node i in $O(p)$ using the adjacency list, and we can count the number of neighbours in each tree in $O(p)$ using the tree numbers. If this count is at least 2 for any tree then the node introduces a cycle, and otherwise it does not.

In the algorithm of Section 5.4.2, we also need to update the data structures after adding a node i to b that maintains the forest property. For this update we need to consider three scenarios:

- If the node i has one neighbour in b, we assign it the label of its neighbour.
- If the node i has no neighbours in b, we assign it the label $(t + 1)$ since it forms a new tree.
- If the node i has multiple neighbours in b, we need to merge all the trees it is connected to.

The first two steps cost $O(1)$, but a naive implementation of the third step would cost $O(n)$ since we could need to re-label almost all of the nodes. Fortunately, we can reduce the cost of this merge step to $O(p)$. This requires a relaxation of the condition that the labels represent disjoint trees. Instead, we only require that nodes with the same label are part of the same tree. This allows multiple labels to be associated with each tree, but using an extra data structure we can still determine if two labels are part of the same tree:

3. A list of t numbers, where element j gives the *minimum node number* in the tree that j is part of.

Thus, given the labels of two nodes we can determine whether they are part of the same tree in $O(1)$ by checking whether their minimum node numbers agree. Given this data structure, the merge step is simple: we arbitrarily assign the new node i to the tree of one of its neighbours, we find the minimum node number among the p trees that need to be merged, and then we use this as the minimum node number for all p trees. This reduces the cost to $O(p)$.

Giving that we can efficiently test the forest property in $O(p)$ for a node with p neighbours, it follows that the total cost of the greedy algorithm from Section 5.4.2 is $O(n \log n + |E|)$ given

the gradient vector and adjacency lists. The $O(n \log n)$ factor comes from sorting the gradient values, and the number of edges $|E|$ is 2 times the number of p values. If this cost is prohibitive, one could simply restrict the number of nodes that we consider adding the forest to reduce this time.

D.5 Full Experimental Results

In this section we first provide details on the datasets, and then we present our complete set of experimental results.

D.5.1 Datasets

We considered these five datasets:

A A least squares problem with a data matrix $A \in \mathbb{R}^{m \times n}$ and target $b \in \mathbb{R}^m$,

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2.$$

We set A to be an m by n matrix with entries sampled from a $\mathcal{N}(0,1)$ distribution (with $m = 1000$ and $n = 10000$). We then added 1 to each entry (to induce a dependency between columns), multiplied each column by a sample from $\mathcal{N}(0,1)$ multiplied by ten (to induce different Lipschitz constants across the coordinates), and only kept each entry of A non-zero with probability $10\log(m)/m$. We set $b = Ax + e$, where the entries of e were drawn from a $\mathcal{N}(0,1)$ distribution while we set 90% of x to zero and drew the remaining values from a $\mathcal{N}(0,1)$ distribution.

B A binary logistic regression problem of the form

$$\operatorname*{argmin}_{x \in \mathbb{R}^n} \sum_{i=1}^{n} \log(1 + \exp(-b_i x^T a_i)).$$

We use the data matrix A from the previous dataset (setting row i of A to a_i^T), and b_i to be the sign of $x^T a_i$ using the x used in the generating the previous dataset. We then flip the sign of each entry in b with probability 0.1 to make the dataset non-separable.

C A multi-class logistic regression problem of the form

$$\operatorname*{argmin}_{x \in \mathbb{R}^{d \times k}} \sum_{i=1}^{m} \left[-x_{b_i}^T a_i + \log\left(\sum_{c=1}^{k} \exp(x_c^T a_i) \right) \right],$$

see (D.1). We generate a 1000 by 1000 matrix A as in the previous two cases. To generate the $b_i \in \{1, 2, \ldots, k\}$ (with $k = 50$), we compute $AX + E$ where the elements of

the matrices $X \in \mathbb{R}^{d\times k}$ and $E \in \mathbb{R}^{m\times k}$ are sampled from a standard normal distribution. We then compute the maximum index in each row of that matrix as the class labels.

D A label propagation problem of the form

$$\min_{x_i \in S'} \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(x_i - x_j)^2,$$

where x is our label vector, S is the set of labels that we do know (these x_i are set to a sample from a normal distribution with a variance of 100), S' is the set of labels that we do not know, and $w_{ij} \geq 0$ are the weights assigned to each x_i describing how strongly we want the labels x_i and x_j to be similar. We set the non-zero pattern of the w_{ij} so that the graph forms a 50 by 50 lattice-structure (setting the non-zero values to 10000). We labeled 100 points, leading to a problem with 2400 variables but where each variable has at most 4 neighbours in the graph.

E Another label propagation problem for semi-supervised learning in the 'two moons' dataset [Zhou et al., 2003]. We generate 2000 samples from this dataset, randomly label 100 points in the data, and connect each node to its five nearest neighbours (using $w_{ij} = 1$). This results in a very sparse but unstructured graph.

D.5.2 Greedy Rules with Gradients Updates

In Figure D.1 we show the performance of the different methods from Section 5.5.1 on all five datasets with three different block sizes. In Figure D.2 we repeat the experiment but focusing only on the FB methods. For each FB method, we plot the performance using our upper bounds on L_b as the step-size (Lb) and using the Lipschitz approximation procedure from Section 5.3.3 (LA). Here we see the LA methods improves performance when using large block sizes and in cases where the global L_b bound is not tight.

Our third experiment also focused on the FB methods, but considered different ways to partition the variables into fixed blocks. We considered three approaches:

1. Order: just using the variables in their numerical order (which is similar to using a random order for dataset except Dataset D, where this method groups variables that adjacent in the lattice).

2. Avg: we compute the coordinate-wise Lipschitz constants L_i, and place the largest L_i with the smallest L_i values so that the average L_i values are similar across the blocks.

3. Sort: we sort the L_i values and place the largest values together (and the smallest values together).

We compared many variations on cyclic/random/greedy rules with gradient or matrix updates. In the case of greedy rules with gradient updates, we found that the Sort method tended to

perform the best while the Order method tended to perform the worst (see Figure D.3). When using matrix updates or when using cyclic/randomized rules, we found that no partitioning strategy dominated other strategies.

D.5.3 Greedy Rules with Matrix and Newton Updates

In Figure D.4 we show the performance of the different methods from Section 5.5.2 on all five datasets with three different block sizes. In Figure D.5 we repeat this experiment on the two non-quadratic problems, using the Newton direction and a line search rather than matrix updates. We see that using Newton's method significantly improves performance over matrix updates.

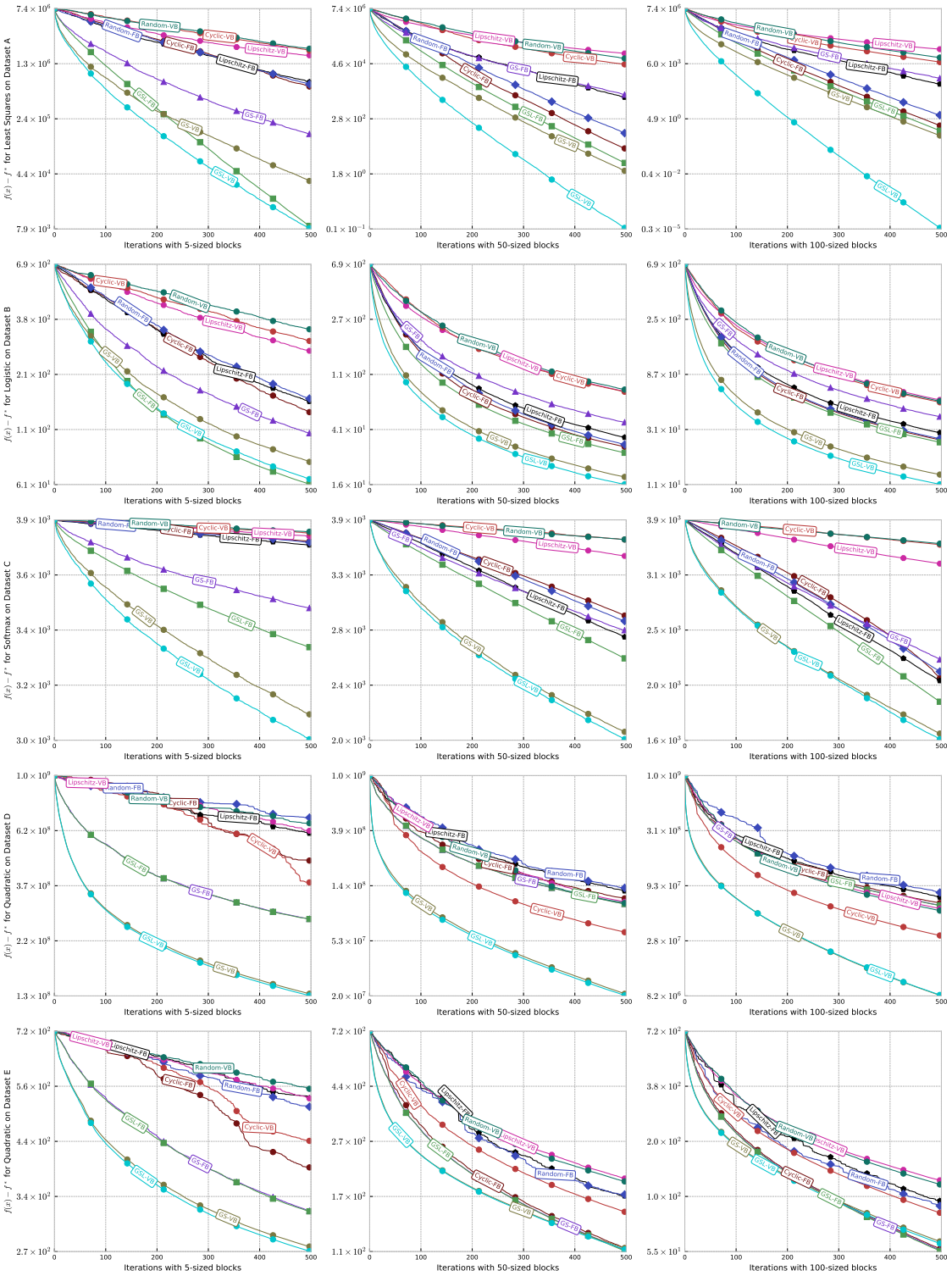


Figure D.1: Comparison of different random and greedy block selection rules on five different problems (rows) with three different blocks (columns) when using gradient updates.

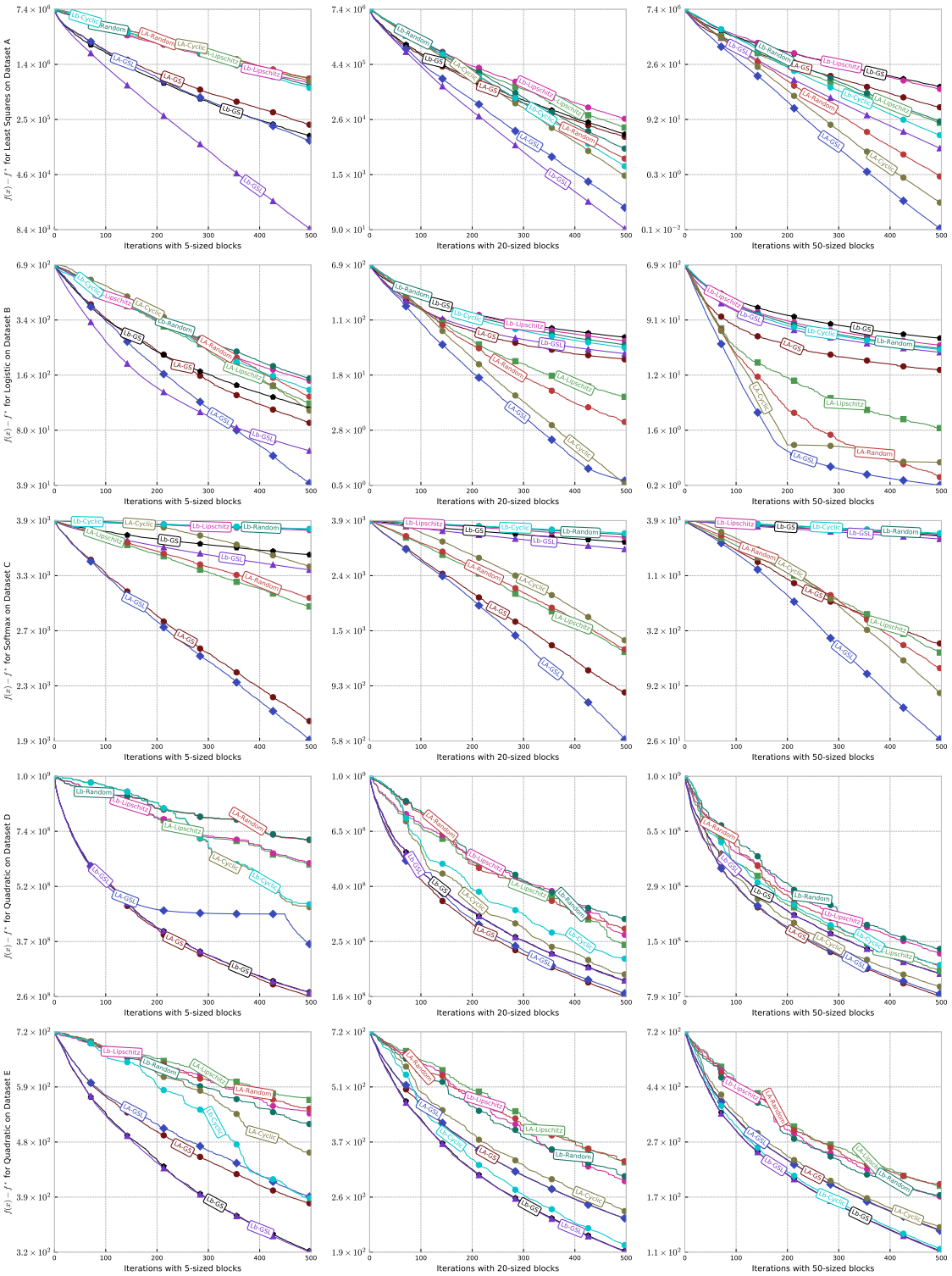


Figure D.2: Comparison of different random and greedy block selection rules with gradient updates and fixed blocks, using two different strategies to estimate L_b.

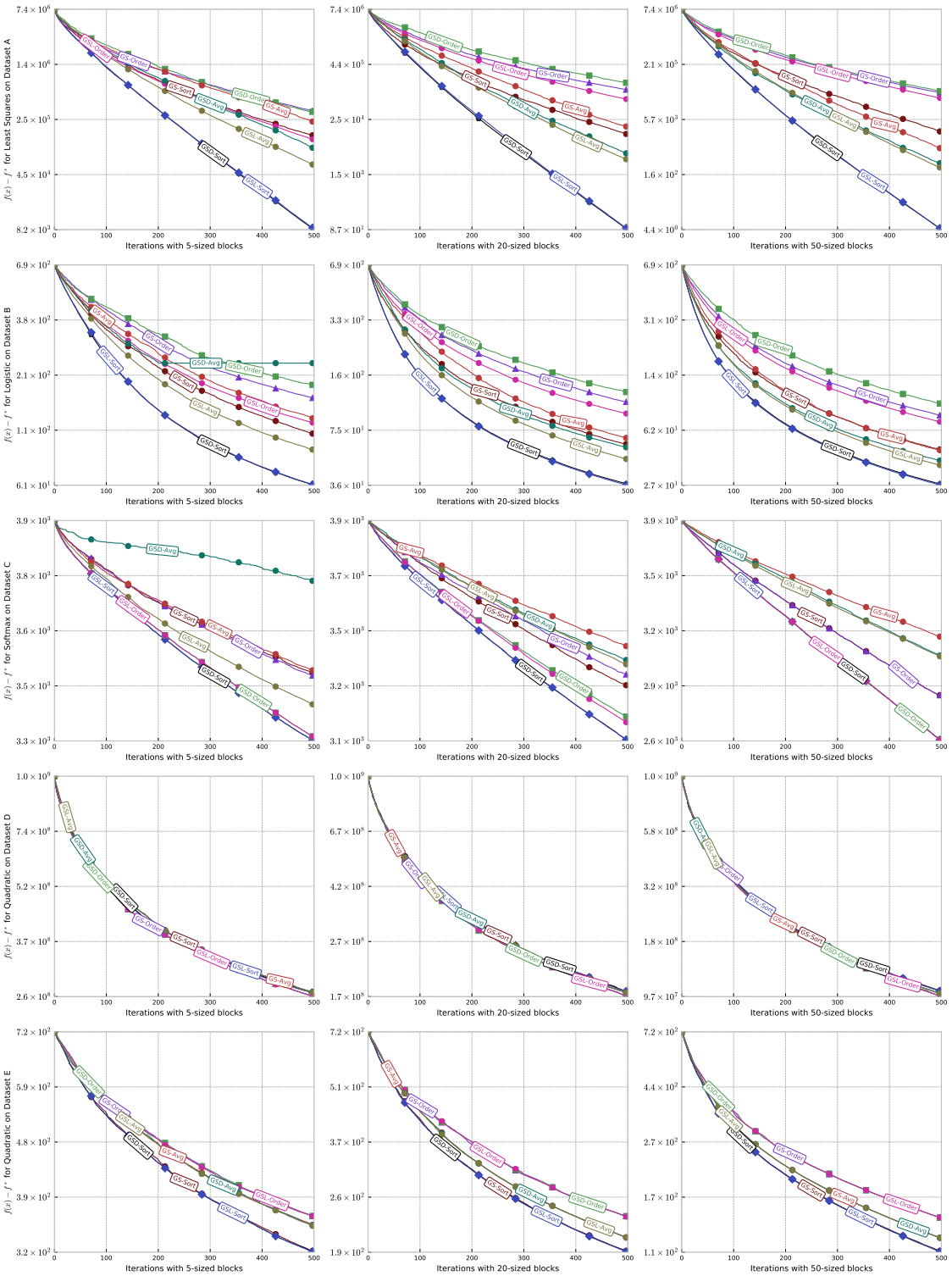


Figure D.3: Comparison of different random and greedy block selection rules with gradient updates and fixed blocks, using three different ways to partition the variables into blocks.

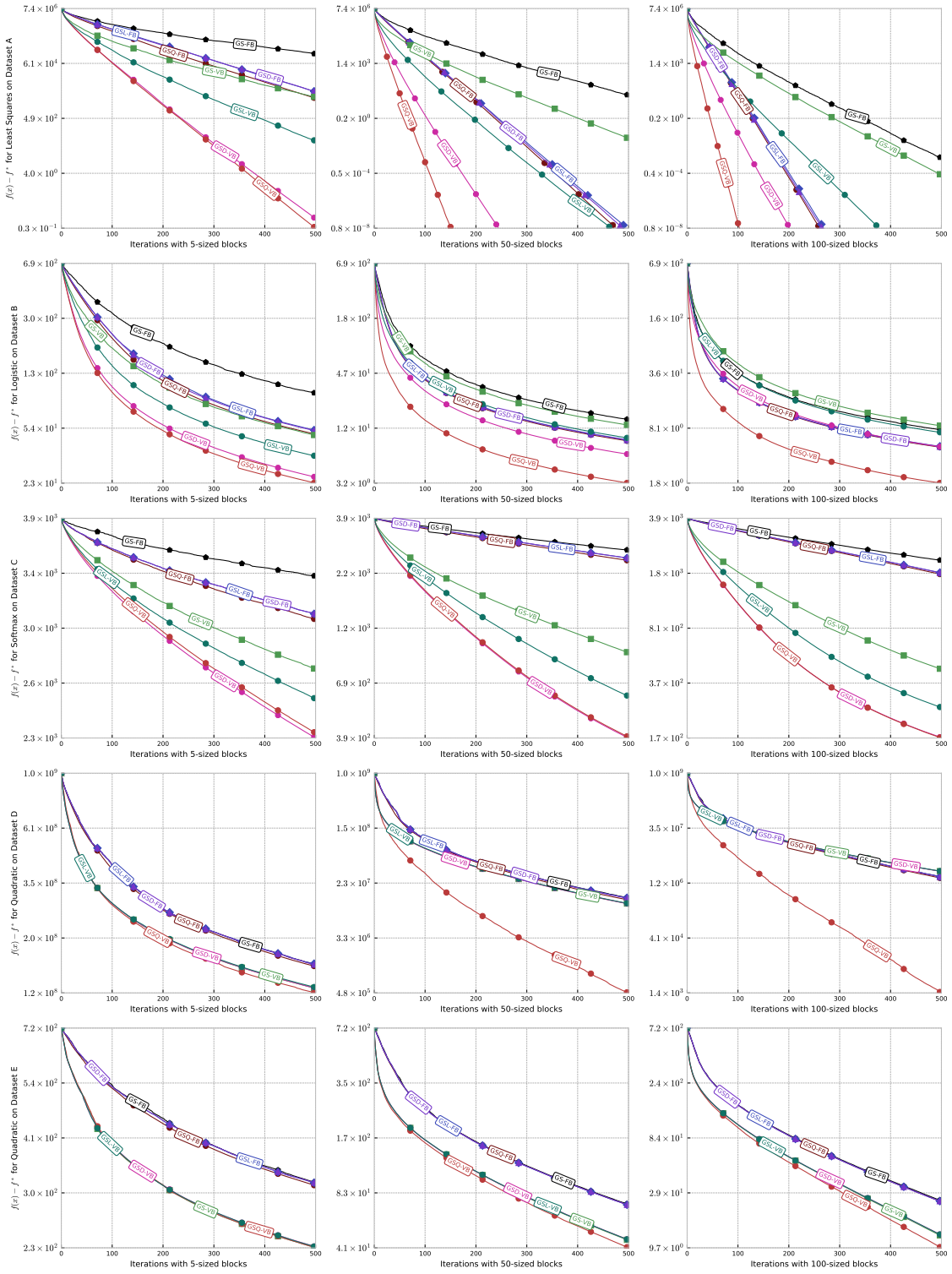


Figure D.4: Comparison of different greedy block selection rules when using matrix updates.

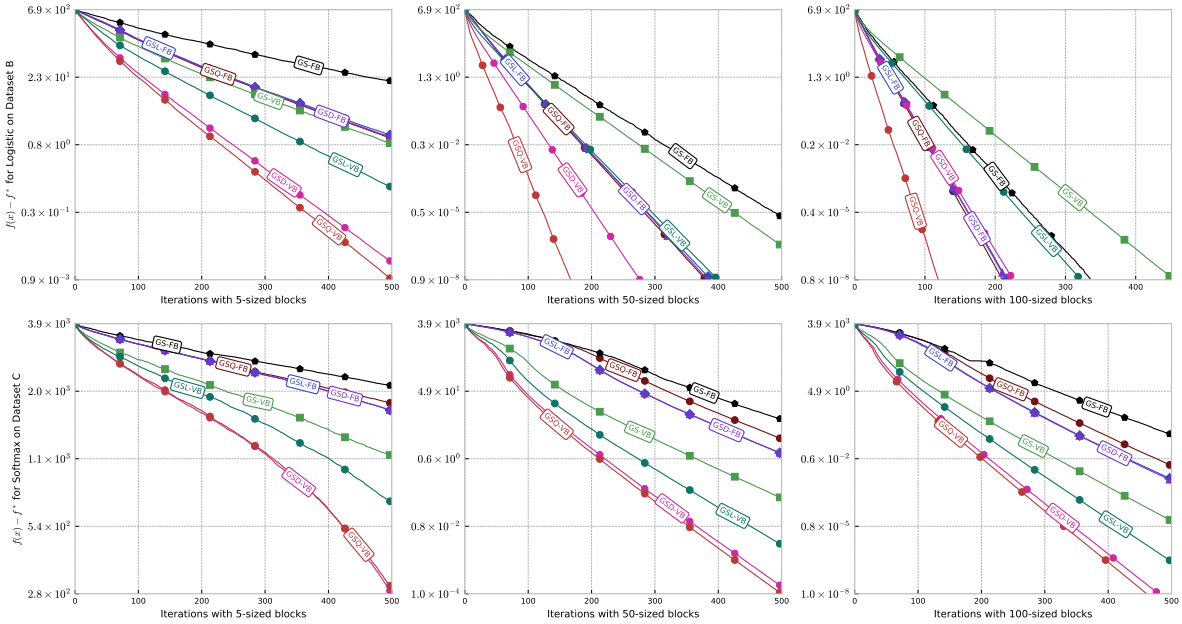


Figure D.5: Comparison of different greedy block selection rules when using Newton updates and a line search.