Smaller Steps for Faster Algorithms: A New Approach to Solving Linear Systems

by

Aaron Daniel Sidford

B.S., Cornell University (2008)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2013

Author .....
Department of Electrical Engineering and Computer Science
August 26, 2013

Certified by ..
.....
Jonathan Kelner
Associate Professor
Thesis Supervisor

Accepted by .....
.....
Professor Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Smaller Steps for Faster Algorithms:

A New Approach to Solving Linear Systems

by

Aaron Daniel Sidford

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2013, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

In this thesis we study iterative algorithms with simple sublinear time update steps, and we show how a mix of of data structures, randomization, and results from numerical analysis allow us to achieve faster algorithms for solving linear systems in a variety of different regimes.

First we present a simple combinatorial algorithm for solving symmetric diagonally dominant (SDD) systems of equations that improves upon the best previously known running time for solving such system in the standard unit-cost RAM model. Then we provide a general method for convex optimization that improves this simple algorithm's running time as special case. Our results include the following:

- We achieve the best known running time of $O\big(m \log^{3/2} n \sqrt{\log \log n} \log(\varepsilon^{-1} \log n)\big)$ for solving Symmetric Diagonally Dominant (SDD) system of equations in the standard unit-cost RAM model.
- We obtain a faster asymptotic running time than conjugate gradient for solving a broad class of symmetric positive definite systems of equations.
- We achieve faster asymptotic convergence rates than the best known for Kaczmarz methods for solving overdetermined systems of equations, by accelerating an algorithm of Strohmer and Vershynin [55].

Beyond the independent interest of these solvers, we believe they highlight the versatility of the approach of this thesis and we hope that they will open the door for further algorithmic improvements in the future.

This work was done in collaboration with Jonathan Kelner, Yin Tat Lee, Lorenzo Orecchia, and Zeyuan Zhu, and is based on the content of [30] and [35].

Thesis Supervisor: Jonathan Kelner
Title: Associate Professor

Acknowledgments

First, I would like thank my advisor, supervisor, and mentor Professor Jonathan Kelner for his continuous support and guidance over the past two years. From our first meeting, when I was a prospective student, to our research discussions, when I was just getting acclimated, to the successful projects detailed in this thesis, and during all the time in between Jon has been a never-ending source of inspiration and wisdom. He keeps me focused while at the same time encouraging me to do the impossible with a kindness and enthusiasm that I cherish. I am forever grateful for all he taught me and for all he continues to teach.

Next, I would like to thank my collaborators Yin Tat Lee, Lorenzo Orecchia, and Zeyuan Zhu. It has been a thrill to work with them all and I thank them for the great ideas and long conversations that were the genesis of this thesis. I thank Zeyuan for his collaboration and for helping me to identify weaknesses in arguments that needed to be addressed. I thank Lorenzo for his patience, his thoughtfulness, and for encouraging me to look at the big picture and strike at the true heart of a problem. I thank Yin Tat for his enthusiasm, for his constant stream of ideas, and for encouraging me to look at problems in new ways and rise beyond all obstacles. I have been fortunate to have the opportunity to work with them all.

I would also like to thank my friends and professors at MIT for creating an exciting and enjoyable academic environment. I would like to express special thanks to Michael Forbes for being an excellent student buddy who helped me transition to MIT, to Professor David Karger for teaching my first class at MIT and delivering some of my favorite lectures in my research area, to Adam Bouland and Matt Coudron for their instrumental collaboration on my first research paper at MIT, to Adam, Abhishek, and Andreea for helping start "Not So Great Ideas" a Thursday night social forum, and to Gautam for organizing countless other social events. To the MIT graduate student community, I am grateful for all the CSC events, fun Thursday nights, hall adventures, dinners with professors, theory retreats, frisbee games, and more fond memories then I could possibly list.

Furthermore, I am very grateful for the constant support and encouragement from my girlfriend, Andreea. Through all the highs and lows of graduate school, from the all-nighters to the shattered theorems, as well as the eventual victories, Andreea continuously provided just the right mix of humor, sarcasm, love, and seriousness that has made every day a little brighter.

Finally, I would like to thank those not at MIT who have supported me over the past two years. I would like to thank Professor John Hopcroft, Professor Thorsten Joachims, and Jim Holt for their encouragement and recommendations that allowed me to start graduate school with excitement and enthusiasm. I would also like to thank my friends for helping me through the transition from work to graduate school. Lastly, I would like to thank my family. For my parents, Bill and Nancy Sidford and my siblings Rachel and Sarah Sidford, I have only the deepest gratitude. Without their limitless supply of encouragement, hope, and love I would be lost. I cannot possibly thank them enough.

This work was partially supported by an Akamai presidential fellowship, Hong Kong RGC grant 2150701, NSF awards 0843915 and 1111109, a NSF Graduate Research Fellowship (grant no. 1122374), and a Sloan Research Fellowship, and I thank them for their support.

Contents

Chapter 1

Introduction

In recent years iterative methods for convex optimization that make progress in sublinear time using only partial information about a function and its gradient have become of increased importance to both the theory and practice of computer science. From a practical perspective, the increasing volume and distributed nature of data are forcing efficient practical algorithms to be amenable to asynchronous and parallel settings where only a subset of the data is available to a single processor at any point in time. From a theoretical perspective, rapidly converging algorithms with sublinear time update steps create the hope for new faster algorithms for solving old problems.

In this thesis we present several results showing that by combining simple sublinear steps with a mix of data structures, randomization, and results from numerical analysis one can achieve faster and simpler algorithms for solving a wide array of linear systems. In the first half of this thesis we show how to use these techniques to produce a simple combinatorial algorithm that solves symmetric diagonally dominant (SDD) systems of equations in nearly linear time and improves upon the best previously known running times for solving SDD systems in the unit-cost RAM model.

In the second half of this thesis we generalize and improve upon the iterative framework applied by this SDD solver. We show to obtain a general first order method for convex optimization that can solve a broad class of linear systems faster than conjugate gradient, generically improve the convergence rate of randomized Kaczmarz [55], and obtain a faster SDD solver in the unit-cost RAM model.

1.1 SDD Systems

While the results in this thesis apply to general problems in convex optimization, they were motivated by the desire to achieve faster and simpler algorithms for solving *symmetric diagonal dominant (SDD)* systems of equations. A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is SDD if $\mathbf{A}^T = \mathbf{A}$ and $\mathbf{A}_{ii} \geq \sum_{j \neq i} |\mathbf{A}_{ij}|$ for all $i \in [n]$. While the best known algorithm for solving a general linear system takes time $O(n^{2.373})$ [61], a seminal paper by Spielman and Teng [50] showed that when $\mathbf{A}$ is SDD one can solve $\mathbf{A}\vec{x} = \vec{b}$ approximately in nearly linear time. [1]

Fast algorithms for solving SDD linear systems have found broad applications across both the theory and practice of computer science. They have long been central to scientific computing, where solving SDD systems is the main computational task in modeling of electrical networks of resistors and performing finite element simulations of a wide range of physical systems (see, e.g., [12]). Beyond this, SDD system solvers have been applied to foundational problems in a wide range of other fields, including machine learning, random processes, computer vision, image processing, network analysis, and computational biology (see, for example, [36, 33, 37, 60, 22]).

More recently, SDD solvers have emerged as a powerful tool in the design of graph algorithms. To every graph G, one can associate a SDD matrix $\mathbf{L}$ called its *Laplacian* (see Section 2.2) such that there are deep connections between the combinatorial properties of G and the linear algebraic properties of $\mathbf{L}$. By exploiting these connections, researchers have used nearly linear time algorithms for solving SDD systems to break longstanding barriers and provide new algorithms for a rapidly growing list of fundamental graph problems, including maximum flow problems [16], multi-commodity flow problems [28], generating random spanning tree [27], graph sparsification [48], lossy flow problems [17], sparsest cut [46], distributed routing [26], and balanced separator [44], as well as fundamental linear algebraic problems for SDD

[1] Throughout this thesis we are primarily interested in *approximate linear system solvers*, that is algorithms that compute $\vec{x} \in \mathbb{R}^n$ such that $\|\vec{x} - \vec{x}_{\text{opt}}\|_{\mathbf{A}} \leq \varepsilon \|\vec{x}_{\text{opt}}\|_{\mathbf{A}}$ for any $\varepsilon \in \mathbb{R} > 0$ where $x_{\text{opt}} \in \mathbb{R}^n$ is a vector such that $\mathbf{A}\vec{x}_{\text{opt}} = \vec{b}$. By a *nearly linear time SDD system solver* we mean an an algorithm that computes such a $\vec{x}$ in time $O(m \log^c n \log \varepsilon^{-1})$ where m is the number of nonzero entries in A and $c \geq 0 \in \mathbb{R}$ is a fixed constant.

matrices, including computing the matrix exponential [44] and the largest eigenvalue and corresponding eigenvector [53]. (See [49, 57, 59] for surveys of these solvers and their applications.)

1.1.1 Previous Nearly Linear Time SDD Solvers

The first nearly linear time algorithm for solving SDD systems was given by Spielman and Teng [50], building on a long line of previous work (e.g., [58, 20, 7, 13, 11]). Their algorithm and its analysis is a technical tour-de-force that required multiple fundamental innovations in spectral and combinatorial graph theory, graph algorithms, and computational linear algebra. Their work included the invention of spectral sparsification and ultra-sparsifiers, better and faster constructions of low-stretch spanning trees, and efficient local clustering algorithms, all of which was used to construct and analyze an intricate recursively preconditioned iterative solver. They divided this work into three papers totaling over 130 pages ([51, 53, 52]), each of which has prompted a new line of inquiry and substantial follow-up work. Their work was was followed by two beautifully insightful papers by Koutis, Miller, and Peng that simplified the SDD system solver while improving its running time to $\widetilde{O}(m \log n \log \varepsilon^{-1})$ [31, 32][2]. For a more in-depth discussion of the history of this work see [53].

These algorithms all rely on the same general framework. They reduce solving general SDD systems to solving systems in graph Laplacians. Given a graph, they show how to obtain a sequence of logarithmically many successively sparser graphs that approximate it, which they construct by adding carefully chosen sets of edges to a low-stretch spanning tree. They then show that the relationship between the combinatorial properties of a graph and the spectral properties of its Laplacian enables them to use the Laplacian of each graph in this sequence as a preconditioner for the one preceding it in a recursively applied iterative solver, such as the Preconditioned Chebyshev Method or Preconditioned Conjugate Gradient.

We remark that multigrid methods (see, e.g., [15]), which are widely used in

[2]Here and in the rest of the thesis we use $\widetilde{O}(.)$ to hide lower order log terms in n, e.g. $\widetilde{O}(m) = O(m \log^c n)$ and $\widetilde{O}(m \log^c n) = O(m \log^c n \log log^d n)$ for some constants $c, d \geq 0$.

practice on graphs with sufficiently nice topologies, can be thought of as following a similar multilevel recursively preconditioned iterative framework. Indeed, Koutis *et al.* have an algorithm and implementation based on related techniques that they refer to as "combinatorial multigrid" [9]. Even if one does not demand provable running time bounds, we are not aware of any algorithm whose running time empirically scales nearly linearly on large classes of input graphs that does not roughly follow this general structure.[3]

1.2 First Order Methods

The iterative framework applied by all known nearly linear time SDD solvers fall into the broad category of *first order methods* for convex optimization. That is, they all run combinatorial algorithms to set up convex optimization problems and then they solve these problems by repeatedly computing only values and gradients of the objective function being optimized. In fact, all iterative optimization algorithms in this thesis (including the new one we develop) can be similarly viewed as first order methods.

1.2.1 Previous First Order Methods

Gradient descent is one of the oldest and most fundamental first order methods in convex optimization. Given a convex differentiable function the *gradient descent method* is a simple greedy iterative method that computes the gradient at the current point and uses that information to perform an update and make progress. This method is central to much of scientific computing and from a theoretical perspective the standard method is well understood [41]. There are multiple more sophisticated variants of this method [25], but many of them have only estimates of local convergence rates which makes them difficult to be applied to theoretical problems and be compared in general.

[3]With the exception of the new algorithms considered in this thesis.

In 1983, Nesterov [40] proposed a way to accelerate the gradient descent method by iteratively developing an approximation to the function through what he calls an *estimate sequence.* This *accelerated gradient descent method* or *fast gradient method* has the same worst case running time as conjugate gradient method and it is applicable to general convex functions. Recently, this method has been used to improve the fastest known running time of some fundamental problems in computer science, such as compressive sensing [5, 6], undirected maximum flow [16, 34, 29], linear programming [42, 8].

The accelerated gradient descent method is known to achieve an optimal (up to constants) convergence rate among all first order methods, that is algorithm that only have access to the function's value and gradient [41]. Therefore, to further improve accelerated gradient descent one must either assume more information about the function or find a way to reduce the cost of each iteration. Using the idea of fast but crude iteration steps, Nesterov proposed a randomized coordinate descent method [43], which minimizes convex functions by updating one randomly chosen coordinate in each iteration.

Coordinate descent methods, which use gradient information about a single coordinate to update a single coordinate in each iteration, have been around for a long time [62]. Various schemes have been considered for picking the coordinate to update, such as cyclic coordinate update and the best coordinate update, however these schemes are either hard to estimate [38] or difficult to be implemented efficiently. Both the recent work of Strohmer and Vershynin [55] and Nesterov [43] (as well as the algorithms in this thesis) overcame these obstacles by showing that by performing particular randomized updates one can produce methods with provable global convergence rate and small costs per iteration.

Applying the similar ideas of accelerated gradient descent, Nesterov also proposed an accelerated variant called the *accelerated coordinate descent method (ACDM)* that achieves a faster convergence rate while still only considering a single coordinate of the gradient at a time. However, in both Nesterov's paper [43] and later work [45], this method was considered inefficient as the computational complexity of the

naive implementation of each iteration of ACDM requires $\Theta(n)$ time to update every coordinate of the input, at which point the accelerated gradient descent method would seem preferable.

1.3 Our Results

In this thesis we present both a simple, combinatorial algorithms that solves SDD systems in nearly linear time and a general first order method for convex optimization that improves the running time of this solver as a special case.

The SDD solver uses very little of the machinery that previously appeared to be necessary for a nearly linear time algorithm. It does not require spectral sparsifiers (or variants such as ultra-sparsifiers or incremental sparsifiers), recursive preconditioning, or even the Chebyshev Method or Conjugate Gradient. To solve a SDD system all the algorithm requires is a single low-stretch spanning tree[4] of G (not a recursive collection of subgraphs), and a straightforward data structure. Given these, the algorithm can be described in a few lines of pseudocode, and its analysis can be made to fit on a single blackboard.

We show how to cast this solver as an instance of the coordinate descent algorithm of Nesterov [43] and we provide a a first order method for convex optimization that both strengthens and unifies this result and the randomized Kaczmarz method of Strohmer and Vershynin [55]. In particular, we present a more general version of Nesterov's ACDM and show how to implement it so that each iteration has the same asymptotic runtime as its non-accelerated variants. We show that this method is numerically stable and we show how to use this method to outperform conjugate gradient in solving a general class of symmetric positive definite systems of equations. Furthermore, we show how to cast both randomized Kaczmarz and the simple SDD solver in this framework and achieve faster running times through the use of ACDM.[5]

[4]This can be obtained in nearly-linear time by a simple ball-growing algorithm [3]; the constructions with the best known parameters use a more intricate, but still nearly linear time, region growing technique [18, 1, 31, 2]. See [30] for a discussion of how even this requirement can be relaxed.

[5]This algorithm can also be shown to have an asymptotically tight running time in certain regimes and we refer the reader to [35] for a discussion of these lower bounds.

1.3.1 Comparison to SDD Solvers

Previous nearly-linear time SDD solvers relied on Preconditioned Chebyshev methods, whose numerical stability is quite difficult to analyze. At present, the best known results show that they can be implemented with finite-precision arithmetic, but the number of bits of precision required is $\log \kappa(\mathbf{L}) \log^c n \log \varepsilon^{-1}$, where $\kappa(\mathbf{L})$ is the condition number of $\mathbf{L}$, and c is some possibly large constant [53]. Therefore, while the stated running time of the best SDD solver is $\widetilde{O}(m \log n \log \varepsilon^{-1})$ [32], this is assuming arbitrary precision arithmetic. If one analyzes it in the more standard unit-cost RAM model, where one can perform operations only on $O(\log n)$-bit numbers in constant time, this introduces several additional logarithmic factors in the running time.

In contrast, we show that our algorithms are numerically stable and do not require this additional overhead in the bit precision. Our simple algorithm approximately solves both SDD systems and the dual electrical flow problem in time $O\big(m \log^2 n \log\log n \log(\varepsilon^{-1} n)\big)$ [6] and our accelerated solver solves these problems in time $O\big(m \log^{3/2} n \sqrt{\log\log n} \log(\varepsilon^{-1} \log n)\big)$. As such, our algorithm gives the fastest known algorithm for solving SDD systems in the unit-cost RAM model. If one allows infinite precision arithmetic to be performed in constant time, our fastest algorithm is slower than [32] by a factor of $\widetilde{O}(\log n)$ for solving SDD systems and actually faster than [32] for the dual problem of computing ε-approximate electric flows when ε is constant.[7]

Due to the complexity of previous nearly linear time solvers and the intricate and delicate nature of their analyses, it was necessary to apply them as a black box. By providing a new, easy-to-understand algorithms, it is our hope that algorithms that use SDD solvers can be improved by "opening up" this black box and modifying it to take advantage of specific features of the problem, and that similar techniques can be applied to related problems (e.g., ones with additional constraints or slight deviations from linearity or diagonal dominance). Furthermore, due to the lightweight nature of

[6] For a discussion of how to improve the running time to $O(m \log^2 n \log\log n \log(\varepsilon^{-1}))$ using a different technique we refer the reader to [30].

[7] To the best of our knowledge, to convert the SDD system solver of [32] to an ε'-approximate electrical flow solver, one needs to pick $\varepsilon = O(\varepsilon'^{-1} n)$.

the algorithm and data structure, we hope it to be fast in practice and adaptable to work in multicore, distributed, and even asynchronous settings.

1.3.2 Comparison to General Linear System Solvers

In some sense, the principle difference between the asymptotic running time of the efficient ACDM presented in this thesis and accelerated gradient descent (or conjugate gradient in the linear system case) is that as accelerated gradient descent depends on the maximum eigenvalue of the Hessian of the function being minimized, ACDM instead depends on the trace of the Hessian and has the possibility of each iteration costing a small fraction of the cost a single iteration of of accelerated gradient descent. As a result, any nontrivial bound on the trace of the Hessian and the computational complexity of performing a single coordinate update creates the opportunity for ACDM to yield improved running times.

Beyond demonstrating how ACDM yields a faster SDD solver we show that under mild assumptions ACDM solves positive definite systems with a faster asymptotic running time than conjugate gradient (and an even milder set of assumptions for Chebyshev method), and it is an asymptotically optimal algorithm for solving general systems in certain regimes.

Furthermore, consider over-constrained systems of equations where the randomized Kaczmarz method of Strohmer and Vershynin [55], which iteratively picks a random constraint and projects the current solution onto the plane corresponding to a random constraint, has been shown to have strong convergence guarantees and appealing practical performance. We show how to cast this method in the framework of coordinate descent and accelerate it using ACDM yielding improved asymptotic performance. Given the appeal of Kaczmarz methods for practical applications such as image reconstructions [21] , there is hope that this could yield improved performance in practice.

We remark that while our analysis of applications of ACDM focus on solving linear systems there is hope that our ACDM algorithm will have a broader impact in both theory and practice of efficient algorithms. Just as the accelerated gradient

descent method has improved the theoretical and empirical running time of various, both linear and nonlinear, gradient descent algorithms [19, 5], we hope that ACDM will improve the running time of various algorithms for which coordinate descent based approaches have proven effective. Given the generality of our analysis and the previous difficulty in analyzing such methods, we hope that this is just the next towards a new class of provably efficient algorithms with good empirical performance.

1.4 Thesis Organization

The rest of this thesis is organized as follows. In Chapter 2 we present and analyze our simplest nearly linear time SDD solver proving that it achieves a running time of $O\left(m \log^2 n \log\log n \log(\varepsilon^{-1} n)\right)$ in the unit cost RAM model. In Chapter 3 we take a more general approach and present a new first order method, the efficient accelerated coordinate descent method (ACDM) and show how ACDM yields faster algorithms for solving a variety of linear systems. In Appendix A we provide additional proof details for generalizing ACDM and showing its numerical stability.

The material in Chapter 2 is based on joint work with Jonathan Kelner, Lorenzo Orecchia, and Zeyuan Zhu [30] and the material in Chapter 3 and Appendix A is based on joint work with Yin Tat Lee [35].

Chapter 2

A Simple Nearly Linear Time Solver for SDD Systems

In this chapter we present a simple combinatorial algorithm for solving SDD systems in nearly linear time. The algorithm uses very little of the machinery that previously appeared to be necessary for a such an algorithm. It does not require recursive preconditioning, spectral sparsification, or even the Chebyshev Method or Conjugate Gradient. After constructing a "nice" spanning tree of a graph associated with the linear system, the algorithms consist of the repeated application of a simple (non-recursive) update rule, which can be implemented using a lightweight data structure. The algorithms is numerically stable and achieves a faster running time in the standard unit-cost RAM model than was known for previous solvers.

The rest of this chapter is organized as follows. In Section 2.1 we provide an overview of our approach, in Section 2.2 we provide technical background that needed for our analysis, in Section 2.3 we present the algorithm, in Section 2.4 we prove the algorithm's convergence rate of the algorithm, in Section 2.5 we present a data structure allowing us to implement the algorithm efficiently, in Section 2.6 we prove the asymptotic running time of the algorithm, and in Section 2.7 we prove that the algorithm is numerically stable and achieves the desired running time in the unit-cost RAM model.

2.1 Overview of our Approach

Here we provide a basic overview of our simple algorithm in order to motivate the definitions and analysis that follow. Using standard reductions techniques we reduce solving arbitrary SDD systems to solving $\mathbf{L}\vec{v} = \vec{\chi}$, where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is the Laplacian of a weighted connected graph $G = (V, E, \vec{w})$ with n vertices and m edges and $\vec{\chi} \in \mathbb{R}^n$, (see Appendix A of [30] for details). Just to simplify the overview we suppose that $\vec{\chi} = \vec{e}_s - \vec{e}_t$ and $\vec{e}_s$ and $\vec{e}_t$ are the unit basis vectors corresponding to arbitrary $s, t \in V$.

Such Laplacian systems $\mathbf{L}\vec{v} = \vec{\chi}$ can be viewed as *electrical flow problems*: each edge $e \in E$ can be viewed as a resistor of resistance $r_e = 1/w_e$, where w_e is the weight of this edge, and one unit of electric current needs to be sent from s to t. Now, if $\vec{v}$ is a valid solution to $\mathbf{L}\vec{v} = \vec{\chi}$, then the entries of $\vec{v}$ can be viewed as the *electrical potentials* of this system, and the amount of *electric current or flow* $\vec{f}(e)$ on an edge $e = (i, j)$ from i to j is given by $(\vec{v}_i - \vec{v}_j)/r_e$. The fact that electric flows are induced by vertex potential differences is a crucial property of electrical systems that our algorithm is going to exploit.

While previous solvers worked with the potential vector $\vec{v}$, our algorithm works with the flow vector $\vec{f}$. Our algorithm begins with any arbitrary unit s-t flow (e.g., a path from s to t) and maintains its feasibility throughout the execution. If $\vec{f}$ were a valid electrical flow, then it would be induced by some potential vector $\vec{v} \in \mathbb{R}^V$ satisfying $\vec{f}(e) = (\vec{v}_i - \vec{v}_j)/r_e$ for all edges $e = (i, j) \in E$, and in particular, for any cycle C in G, we would have the potential drop $\sum_{e \in C} \vec{f}(e) r_e = 0$. Exploiting this fact, our entire algorithm consists of simply repeating the following two steps.

- Randomly sample a cycle C from some probability distribution.
- Compute $\sum_{e \in C} \vec{f}(e) r_e$ and add a multiple of C to $\vec{f}$ to make it zero.

To turn this into a provably-correct and efficient algorithm we need to do the following:

- **Specify the cycle distribution.** The cycles are those found by adding edges to a low-stretch spanning tree. They are sampled proportional to their stretch (See Section 2.2.2 and 2.3).

- **Bound the number of iterations.** In Section 2.4, we show that repeating this process a nearly linear number of times yields an ε-approximate solution.

- **Implement the iterations efficiently.** Since a cycle may contain a large number of edges, we cannot simply update the flow edge-by-edge. In Section 2.5, we give a data structure that allows each iteration to take $O(\log n)$ time.

While in the next chapter we show how this algorithm can be viewed as an instantiation of coordinate descent, here we note that this algorithm has a geometric interpretation and can be viewed as an application of the *randomized Kaczmarz* method of Strohmer and Vershynin [55] (see Section 3.4), which solves a linear system by randomly picking a constraint and projecting the current solution onto its feasible set. For a more detailed exposition on this fact and discussion of how the algorithm can be viewed as providing a linear approximation to $\mathbf{L}^\dagger$ we refer the reader to [30].

2.2 Preliminaries

For the remainder of this chapter we let $G = (V, E, w)$ be a weighted, connected, undirected graph with $n = |V|$ vertices, $m = |E|$ edges and edge weights $w_e > 0$. We think of w_e as the *conductance* of e, and we define the *resistance* of e by $r_e \stackrel{\text{def}}{=} 1/w_e$. For notational convenience, we fix an orientation of the edges so that for any vertices a and b connected by an edge, exactly one of $(a, b) \in E$ or $(b, a) \in E$ holds.

We make extensive use of the following matrices associated with G:

Definition 2.2.1 (Incidence Matrix). *The* incidence matrix $\mathbf{B} \in \mathbb{R}^{E \times V}$ *is defined by*

$$\forall (a,b) \in \mathbb{E} \ , \ \forall c \in V \ : \ \mathbf{B}_{(a,b),c} = \begin{cases} 1 & a = c \\ -1 & b = c \\ 0 & \textit{otherwise} \end{cases}$$

Definition 2.2.2 (Resistance Matrix). *The* resistance matrix, $\mathbf{R} \in \mathbb{R}^{E \times E}$, *is the diagonal matrix where* $\mathbf{R}_{e,e} = r_e$ *for all* $e \in E$.

Definition 2.2.3 (Laplacian Matrix). *The* Laplacian matrix, $\mathbf{L} \in \mathbb{R}^{V \times V}$, *is defined for all* $a, b \in V$ *by*

$$\mathbf{L}_{a,b} = \begin{cases} \sum_{\{a,u\} \in E} w_{a,u} & a = b \\ -w_{a,b} & \{a,b\} \in E \\ 0 & \textit{otherwise} \end{cases}$$

For a vector $\vec{f} \in \mathbb{R}^E$ and an edge $e = (a,b) \in E$, we write $\vec{f}(e) = \vec{f}(a,b)$ for the coordinate of $\vec{f}$ corresponding to e, and we adopt the convention $\vec{f}(b,a) = -\vec{f}(a,b)$. This allows us to think of $\vec{f}$ as a flow on the graph (not necessarily obeying any conservation constraints) that sends $\vec{f}(e)$ units of flow from a to b, and thus $-\vec{f}(e)$ units of flow from b to a.

The following facts follow by simple manipulations of the above definitions:

Claim 2.2.4. *For all* $\vec{f} \in \mathbb{R}^E$, $x \in \mathbb{R}^V$, $a \in V$ *and* $(a,b) \in E$*:*

- $\left[\mathbf{B}^T \vec{f}\right]_a = \sum_{(b,a) \in E} \vec{f}(b,a) - \sum_{(a,b) \in E} \vec{f}(a,b)$,
- $\mathbf{L} = \mathbf{B}^T \mathbf{R}^{-1} \mathbf{B}$,
- $[\mathbf{B}x]_{(a,b)} = x(a) - x(b)$, *and*
- $x^T \mathbf{L} x = \sum_{(a,b) \in E} \frac{(x_a - x_b)^2}{r_{a,b}}$.

One can interpret the first assertion in Claim 2.2.4 as saying that $\mathbf{B}^T \vec{f}$ is a vector in $\mathbb{R}^V$ whose a-th coordinate indicates how much flow $\vec{f}$ leaves (or enters, if it is negative) the graph G at vertex a. We say that $\vec{f}$ is a *circulation* if $\mathbf{B}^T \vec{f} = 0$.

2.2.1 Electrical Flow

For any vector $\vec{f} \in \mathbb{R}^E$, we define its *energy* $\xi(\vec{f})$ by

$$\xi(\vec{f}) \stackrel{\text{def}}{=} \sum_{e \in E} r_e \vec{f}(e)^2 = \vec{f}^T \mathbf{R} \vec{f} = \|\vec{f}\|_{\mathbf{R}}^2 .$$

We fix a *demand vector* $\vec{\chi} \in \mathbb{R}^V$ and we say a flow $\vec{f} \in \mathbb{R}^E$ is *feasible (with respect to* $\vec{\chi}$*)*, or that it *meets the demands*, if $\mathbf{B}^T \vec{f} = \vec{\chi}$. Since G is connected it is straightforward to check that there exists a feasible flow with respect to $\vec{\chi}$ if and only if $\sum_{v \in V} \vec{\chi}(v) = 0$.

Definition 2.2.5 (Electrical Flow). *For a demand vector* $\vec{\chi} \in \mathbb{R}^V$ *that satisfies* $\sum_{a \in V} \vec{\chi}(a) = 0$, *the* electrical flow satisfying $\vec{\chi}$ *is the unique minimizer to*

$$\vec{f}_{\text{opt}} \stackrel{\text{def}}{=} \underset{\vec{f} \in \mathbb{R}^E \ : \ \mathbf{B}^T \vec{f} = \vec{\chi}}{\arg\min} \xi(\vec{f}) \ . \tag{2.1}$$

This quadratic program describes a natural physical problem. Given an electric circuit with nodes in V, for each undirected edge $e = \{a, b\} \in E$ we connect nodes a and b with a resistor of resistance r_e. Next, we fix the amount of current entering and leaving each node and denote this by demand vector $\vec{\chi}$. Recalling from physics that the energy of sending i units of current over a resistor of resistance r is $i^2 \cdot r$ the amount of electric current on each resistor is given by $\vec{f}_{\text{opt}}$. The central problem of this chapter is to efficiently compute an ε-approximate electrical flow.

Definition 2.2.6 (ε-Approximate Electrical Flow). *For* $\varepsilon \in \mathbb{R}^{\geq 0}$, *we say* $\vec{f} \in \mathbb{R}^E$ *is an* ε-approximate electric flow satisfying $\vec{\chi}$ *if* $\mathbf{B}^T \vec{f} = \vec{\chi}$, *and* $\xi(\vec{f}) \leq (1+\varepsilon) \cdot \xi(\vec{f}_{\text{opt}})$.

Duality

The electric flow problem is dual to solving $\mathbf{L}\vec{x} = \vec{\chi}$ when $\mathbf{L}$ is the Laplacian for the same graph G. To see this, we note that the Lagrangian dual of (2.1) is given by

$$\max_{\vec{v} \in \mathbb{R}^V} 2\vec{v}^T \vec{\chi} - \vec{v}^T \mathbf{L} \vec{v} \ . \tag{2.2}$$

For symmetry we define the *(dual) energy* of $\vec{v} \in \mathbb{R}^V$ as $\zeta(\vec{v}) \stackrel{\text{def}}{=} 2\vec{v}^T \vec{\chi} - \vec{v}^T \mathbf{L} \vec{v}$. Setting the gradient of (2.2) to 0 we see that (2.2) is minimized by $\vec{v} \in \mathbb{R}^V$ satisfying $\mathbf{L}\vec{v} = \vec{\chi}$. Let $\mathbf{L}^\dagger$ denote the *Moore-Penrose pseduoinverse* of $\mathbf{L}$ and let $\vec{v}_{\text{opt}} \stackrel{\text{def}}{=} \mathbf{L}^\dagger \vec{\chi}$ denote a particular set of optimal voltages . Since the primal program (2.1) satisfies Slater's condition, we have strong duality, so for all $\vec{v} \in \mathbb{R}^V$

$$\zeta(\vec{v}) \leq \zeta(\vec{v}_{\text{opt}}) = \xi(\vec{f}_{\text{opt}}) \ .$$

Therefore, for feasible $\vec{f} \in \mathbb{R}^E$ and $\vec{v} \in \mathbb{R}^V$ the *duality gap*, $\text{gap}(\vec{f}, \vec{v}) \stackrel{\text{def}}{=} \xi(\vec{f}) - \zeta(\vec{v})$ is an upper bound on both $\xi(\vec{f}) - \xi(\vec{f}_{\text{opt}})$ and $\zeta(\vec{v}_{\text{opt}}) - \zeta(\vec{v})$.

In keeping with the electrical circuit interpretation we refer to a candidate dual solution $\vec{v} \in \mathbb{R}^V$ as *voltages* or *vertex potentials* and we define $\Delta_{\vec{v}}(a, b) \stackrel{\text{def}}{=} \vec{v}(a) - \vec{v}(b)$ to be the *potential drop* of $\vec{v}$ across (a, b). By the KKT conditions we know that

$$\vec{f}_{\text{opt}} = \mathbf{R}^{-1}\mathbf{B}\vec{v}_{\text{opt}} \quad \text{i.e.} \quad \forall e \in E \ : \ \vec{f}_{\text{opt}}(e) = \frac{\Delta_{\vec{v}_{\text{opt}}}(e)}{r_e} \ .$$

For $e \in E$ we call $\frac{\Delta_{\vec{v}}(e)}{r_e}$ the *flow induced by $\vec{v}$ across e* and we call $\vec{f}(e)r_e$ *the potential drop induced by $\vec{f}$ across e.*

The optimality conditions $\vec{f}_{\text{opt}} = \mathbf{R}^{-1}\mathbf{B}\vec{v}_{\text{opt}}$ can be restated solely in terms of flows in a well known variant of *Kirchoff's Potential Law* (KPL) as follows

Lemma 2.2.7 (KPL). *Feasible $\vec{f} \in \mathbb{R}^E$ is optimal if and only if $\vec{f}^T\mathbf{R}\vec{c} = 0$ for all circulations $\vec{c} \in \mathbb{R}^E$.*

2.2.2 Spanning Trees and Cycle Space

Let $T \subseteq E$ be a spanning tree of G and let us call the edges in T the *tree edges* and the edges in $E \setminus T$ the *off-tree edges.* By the fact that G is connected and T spans G we know that for every $a, b \in V$ there is a unique path connecting a and b using only tree edges.

Definition 2.2.8 (Tree Path). *For $a, b \in V$, we define the* tree path $P_{(a,b)} \subseteq V \times V$ *to be the unique path from a to b using edges from T.*[1] *In vector form we let $\vec{p}_{(a,b)} \in \mathbb{R}^E$ denote the unique flow sending 1 unit of flow from a to b, that is nonzero only on T.*

For the off-tree edges we similarly define *tree cycles.*

Definition 2.2.9 (Tree Cycle). *For $(a, b) \in E \setminus T$, we define the* tree cycle $C_{(a,b)} \stackrel{\text{def}}{=} \{(a, b)\} \cup P_{(b,a)}$ *to be the unique cycle consisting of edge (a, b) and $P_{(b,a)}$. In vector form we let $\vec{c}_{(a,b)}$ denote the unique circulation sending 1 unit of flow on $C_{(a,b)}$.*

[1] Note that edges of $P_{(a,b)}$ are oriented with respect to the path, not the natural orientation of G.

Cycle Space

The tree cycles form a complete characterization of circulations in a graph. The set of all circulations $\{\vec{c} \in \mathbb{R}^E \mid \mathbf{B}^T\vec{c} = 0\}$ is a well-known subspace called *cycle space* [10] and the tree cycles $\{\vec{c}_e \mid e \in E \setminus T\}$ form a basis. This yields an even more succinct description of the KPL optimality condition (Lemma 2.2.7). A feasible $\vec{f} \in \mathbb{R}^E$ is optimal if and only if $\vec{f}^T R\vec{c}_e = 0$ for all $e \in E \setminus T$.

We can think of each tree cycle C_e as a long resistor consisting of its edges in series with total resistance $\sum_{e \in C_e} r_e$ and flow induced potential drop of $\sum_{e \in C_e} \vec{f}(e) r_e$. KPL optimality then states that $\vec{f} \in \mathbb{R}^E$ is optimal if and only if the potential drop across each of these resistors is 0. Here we define two key quantities relevant to this view.

Definition 2.2.10 (Cycle Quantities). *For $e \in E \setminus T$ and $\vec{f} \in \mathbb{R}^E$ the* resistance of *C_e, R_e, and the* flow induced potential across *C_e, $\Delta_{c_e}(\vec{f})$, are given by*

$$R_e \stackrel{\text{def}}{=} \sum_{e' \in C_e} r_{e'} = \vec{c}_e^T \mathbf{R} \vec{c}_e \quad \text{and} \quad \Delta_{c_e}(\vec{f}) \stackrel{\text{def}}{=} \sum_{e \in C_e} r_e \vec{f}(e) = \vec{f}^T \mathbf{R} \vec{c}_e \ .$$

Low-Stretch Spanning Trees

Starting with a feasible flow, our algorithm computes an approximate electrical flow by fixing violations of KPL on randomly sampled tree cycles. How well this algorithm performs is determined by how well the resistances of the off-tree edges are approximated by their corresponding cycle resistances. This quantity, which we refer to as as the *tree condition number*, is in fact a certain condition number of a matrix whose rows are properly normalized instances of $\vec{c}_e$ (see Section 3.4.3).

Definition 2.2.11 (Tree Condition Number). *The* tree condition number *of spanning tree T is given by $\tau(T) \stackrel{\text{def}}{=} \sum_{e \in E \setminus T} \frac{R_e}{r_e}$, and we abbreviate it as τ when the underlying tree T is clear from context.*

This is closely related to a common quantity associated with a spanning tree called *stretch.*

Definition 2.2.12 (Stretch). *The* stretch *of* $e \in E$, $\mathrm{st}(e)$, *and the* total stretch *of* T, $\mathrm{st}(T)$, *are*

$$\mathrm{st}(e) \stackrel{\text{def}}{=} \frac{\sum_{e' \in P_e} r_{e'}}{r_e} \quad \textit{and} \quad \mathrm{st}(T) \stackrel{\text{def}}{=} \sum_{e \in E} \mathrm{st}(e) \ .$$

Since $R_e = r_e \cdot (1 + \mathrm{st}(e))$ we see that these quantities are related by $\tau(T) = \mathrm{st}(T) + m - 2n + 2$.

Efficient algorithms for computing spanning trees with low total or average stretch, i.e. *low-stretch spanning trees*, have found numerous applications [3, 18] and all previous nearly-linear-time SDD-system solvers [50, 53, 31, 32], including the most efficient form of the SDD solver presented in this thesis, make use of such trees. There have been multiple breakthroughs in the efficient construction of *low-stretch spanning trees* [3, 18, 1, 31, 2] and the latest such result is used in this thesis and stated below.

Theorem 2.2.13 ([2]). *In* $O(m \log n \log\log n)$ *time we can compute a spanning tree* T *with total stretch* $\mathrm{st}(T) = O(m \log n \log\log n)$.

2.3 The Algorithm

Given a SDD system $\mathbf{A}\vec{x} = \vec{b}$ we wish to efficiently compute $\vec{x}$ such that $\|\vec{x} - \mathbf{A}^\dagger \vec{b}\|_{\mathbf{A}} \leq \varepsilon \|\mathbf{A}^\dagger \vec{b}\|_A$. Using standard reduction techniques (see Appendix A of [30]) we can reduce solving such SDD systems to solving Laplacian systems corresponding to connected graphs without a loss in asymptotic run time. Therefore it suffices to solve $\mathbf{L}\vec{v} = \vec{\chi}$ in nearly-linear time when $\mathbf{L}$ is the Laplacian matrix for some connected graph G. Here we provide an algorithm that both solves such systems and computes the corresponding ε-approximate electric flows in $O\big(m \log^2 n \log\log n \log(\varepsilon^{-1} n)\big)$ time.

2.3.1 A Simple Iterative Approach

Our algorithm focuses on the electric flow problem. First, we compute a low stretch spanning tree, T, and a crude initial feasible $\vec{f}_0 \in \mathbb{R}^E$ taken to be the unique $\vec{f}_0$ that meets the demands and is nonzero only on tree edges. Next, for a fixed number of iterations, K, we perform simple iterative steps, referred to as *cycle updates*, in

which we compute a new feasible $\vec{f}_i \in \mathbb{R}^E$ from the previous feasible $\vec{f}_{i-1} \in \mathbb{R}^E$ while attempting to decrease energy. Each iteration, i, consists of sampling an $e \in E \setminus T$ proportional to $\frac{R_e}{r_e}$, checking if $\vec{f}_{i-1}$ violates KPL on C_e (i.e. $\Delta_{c_e}(\vec{f}_i) \neq 0$) and adding a multiple of $\vec{c}_e$ to make KPL hold on C_e (i.e. $\vec{f}_i = \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e}\vec{c}_e$). Since, $\mathbf{B}^T\vec{c}_e = 0$, this operation preserves feasibility. We show that in expectation $\vec{f}_K$ is an ε-approximate electrical flow.

To solve $\mathbf{L}\vec{v} = \vec{\chi}$ we show how to use an ε-approximate electrical flow to derive a candidate solution to $\mathbf{L}\vec{v} = \vec{\chi}$ of comparable quality. In particular, we use the fact that a vector $\vec{f} \in \mathbb{R}^E$ and a spanning tree T induce a natural set of voltages, $\vec{v} \in \mathbb{R}^V$ which we call the *tree induced voltages.*

Definition 2.3.1 (Tree Induced Voltages). *For $\vec{f} \in \mathbb{R}^E$ and an arbitrary (but fixed) $s \in V$,*[2] *we define the* tree induced voltages $\vec{v} \in \mathbb{R}^V$ *by $\vec{v}(a) \stackrel{\text{def}}{=} \sum_{e \in P_{(a,s)}} \vec{f}(e) r_e$ for $\forall a \in V$.*

Our algorithm simply returns the tree induced voltages for $\vec{f}_K$, denoted $\vec{v}_K$, as the approximate solution to $\mathbf{L}\vec{v} = \vec{\chi}$. The full pseudocode for the algorithm is given in Algorithm 1.

Algorithm 1: `SimpleSolver`

Input : $G = (V, E, r)$, $\vec{\chi} \in \mathbb{R}^V$, $\varepsilon \in \mathbb{R}^+$
Output: $\vec{f} \in \mathbb{R}^E$ and $\vec{v} \in \mathbb{R}^V$

$T :=$ low-stretch spanning tree of G;
$\vec{f}_0 :=$ unique flow on T such that $\mathbf{B}^T\vec{f}_0 = \vec{\chi}$;
$\vec{p}_e := \frac{1}{\tau(T)} \cdot \frac{R_e}{r_e}$ for all $e \in E \setminus T$;
$K = \lceil \tau \log\left(\frac{\mathrm{st}(T) \cdot \tau(T)}{\varepsilon}\right) \rceil$;
for $i = 1$ *to* K **do**
  Pick random $e_i \in E \setminus T$ by probability distribution $\vec{p}$;
  $\vec{f}_i = \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e}\vec{c}_e$;
end
return $\vec{f}_K$ *and its tree induced voltages* $\vec{v}_K$

[2] We are primarily concerned with difference between potentials which are invariant under the choice $s \in V$.

2.3.2 Algorithm Guarantees

In the next few sections we prove that `SimpleSolver` both computes an ε-approximate electric flow and solves the corresponding Laplacian system in nearly linear time:

Theorem 2.3.2 (`SimpleSolver`). *The output of* ***SimpleSolver*** *satisfies* [3]

$$\mathbb{E}\left[\xi\left(\vec{f}\right)\right] \leq (1+\varepsilon)\cdot\xi\left(\vec{f}_{\text{opt}}\right) \quad \textit{and} \quad \mathbb{E}\left\|\vec{v}-\mathbf{L}^{\dagger}\vec{\chi}\right\|_{\mathbf{L}} \leq \sqrt{\varepsilon}\cdot\left\|\mathbf{L}^{\dagger}\vec{\chi}\right\|_{\mathbf{L}}$$

and ***SimpleSolver*** *can be implemented to run in time* $O\left(m\log^2 n\log\log n\log(\varepsilon^{-1}n)\right)$.

By construction `SimpleSolver` outputs a feasible flow and, by choosing T to be a low stretch spanning tree with properties guaranteed by Theorem 2.2.13, we know that the number of iterations of simple solver is bounded by $O(m\log n\log\log n\log(\varepsilon^{-1}n))$. However, in order to prove the theorem we still need to show that (1) each iteration makes significant progress, (2) each iteration can be implemented efficiently, and (3) the starting flow and final voltages are appropriately related to $\xi\left(\vec{f}_{\text{opt}}\right)$. In particular we show:

1. Each iteration of `SimpleSolver` decreases the energy of the current flow by at least an expected $\left(1-\frac{1}{\tau}\right)$ fraction of the energy distance to optimality (see Section 2.4).

2. Each iteration of `SimpleSolver` can be implemented to take $O(\log n)$ time (see Section 2.5).[4]

3. $\xi\left(\vec{f}_0\right)$ is sufficiently bounded, the quality of tree voltages is sufficiently bounded, and the entire algorithm can be implemented efficiently (see Section 2.6).

[3] Although the theorem is stated as an expected guarantee on $\vec{f}$ and $\vec{v}$, one can easily use Markov bound and Chernoff bound to provide a probabilistic but exact guarantee.

[4] Note that a naïve implementation of cycle updates does not necessarily run in sublinear time. In particular, updating $\vec{f}_i := \vec{f}_{i-1} - \frac{\Delta_{c_e}(\vec{f}_{i-1})}{R_e}\vec{c}_e$ by walking C_e and updating flow values one by one may take more than (even amortized) sublinear time, even though T may be of low total stretch. Since st (T) is defined with respect to cycle resistances but not with respect to the number of edges in these cycles, it is possible to have a low-stretch tree where each tree cycle still has $\Omega(|V|)$ edges on it. Furthermore, even if all edges have resistances 1 and therefore the average number of edges in a tree cycle is $\tilde{O}(\log n)$, since `SimpleSolver` samples off-tree edges with higher stretch with higher probabilities, the expected number of edges in a tree cycle may still be $\Omega(|V|)$.

2.4 Convergence Rate Analysis

In this section we analyze the convergence rate of `SimpleSolver`. The central result is as follows.

Theorem 2.4.1 (Convergence). *Each iteration i of* `SimpleSolver` *computes feasible $\vec{f}_i \in \mathbb{R}^E$ such that*

$$\mathbb{E}[\xi(\vec{f}_i)] - \xi(\vec{f}_{\text{opt}}) \leq \left(1 - \frac{1}{\tau}\right)^i \left(\xi(\vec{f}_0) - \xi(\vec{f}_{\text{opt}})\right) .$$

Our proof is divided into three steps. In Section 2.4.1 we analyze the energy gain of a single algorithm iteration, in Section 2.4.2 we bound the distance to optimality in a single algorithm iteration, and in Section 2.4.3 we connect these to prove the theorem.

2.4.1 Cycle Update Progress

For feasible $\vec{f} \in \mathbb{R}^E$, we can decrease $\xi(\vec{f})$ while maintaining feasibility, by adding a multiple of a circulation $\vec{c} \in \mathbb{R}^E$ to $\vec{f}$. In fact, we can easily optimize to pick the best multiple.

Lemma 2.4.2 (Energy Improvement). *For $\vec{f} \in \mathbb{R}^E$, $\vec{c} \in \mathbb{R}^E$, and $\alpha^* = -\frac{\vec{f}^T\mathbf{R}\vec{c}}{\vec{c}^T\mathbf{R}\vec{c}} \in \mathbb{R}$ we have*

$$\arg\min_{\alpha \in \mathbb{R}} \xi(\vec{f} + \alpha\vec{c}) = -\frac{\vec{f}^T\mathbf{R}\vec{c}}{\vec{c}^T\mathbf{R}\vec{c}} \quad \text{and} \quad \xi(\vec{f} + \alpha^*\vec{c}) - \xi(\vec{f}) = -\frac{\left(\vec{f}^T\mathbf{R}\vec{c}\right)^2}{\vec{c}^T\mathbf{R}\vec{c}} .$$

Proof. By definition of energy $\xi(\vec{f}+\alpha\vec{c}) = (\vec{f}+\alpha\vec{c})^T\mathbf{R}(\vec{f}+\alpha\vec{c}) = \vec{f}^T\mathbf{R}\vec{f} + 2\alpha\vec{f}^T\mathbf{R}\vec{c} + \alpha^2\vec{c}^T\mathbf{R}\vec{c}$. Setting the derivative with respect to α to 0 and substituting in $\alpha = \alpha^*$ yields the results. □

In the special case where $\vec{c} = \vec{c}_e$ is a tree cycle for some off-tree edge $e \in E \setminus T$, since $R_e = \vec{c}_e^T\mathbf{R}\vec{c}_e$ and $\Delta_{c_e}(\vec{f}) = \vec{f}^T\mathbf{R}\vec{c}_e$, this procedure is precisely the iterative step of `SimpleSolver`, i.e. a cycle update. The following lemma follows immediately and

states that the energy decrease of a cycle update is exactly the energy of a resistor with resistance R_e and potential drop $\Delta_{c_e}(\vec{f})$.

Lemma 2.4.3 (Cycle Update). *For feasible $\vec{f} \in \mathbb{R}^E$ and $e \in E \setminus T$ we have*

$$\xi\left(\vec{f} - \frac{\Delta_{c_e}(\vec{f})}{R_e}\vec{c}_e\right) - \xi(\vec{f}) = -\frac{\Delta_{c_e}(\vec{f})^2}{R_e} \ .$$

2.4.2 Distance to Optimality

To bound how far $\vec{f}_i$ in `SimpleSolver` is from optimality we use that the duality gap between $\vec{f}_i$ and its tree induced voltages $\vec{v}_i$ is an upper bound on this distance. Here we derive a simple expression for this quantity in terms of cycle potentials.

Lemma 2.4.4 (Tree Gap). *For feasible $\vec{f} \in \mathbb{R}^E$ and tree induced voltages $\vec{v} \in \mathbb{R}^V$ we have*

$$\text{gap}(\vec{f}, \vec{v}) = \sum_{e \in E \setminus T} \frac{\Delta_{c_e}(\vec{f})^2}{r_e} \ .$$

Proof. By definition of primal and dual energy we have $\text{gap}(\vec{f}, \vec{v}) = \vec{f}^T\mathbf{R}\vec{f} - (2\vec{v}^T\vec{\chi} - \vec{v}^T\mathbf{L}\vec{v})$. Using that $\mathbf{B}^T\vec{f} = \vec{\chi}$ and $\mathbf{L} = \mathbf{B}^T\mathbf{R}^{-1}\mathbf{B}$ we get

$$\text{gap}(\vec{f}, \vec{v}) = \vec{f}^T\mathbf{R}\vec{f} - 2\vec{v}^T\mathbf{B}^T\vec{f} - \vec{v}^T\mathbf{B}^T\mathbf{R}^{-1}\mathbf{B}\vec{v} = \left(\mathbf{R}\vec{f} - \mathbf{B}\vec{v}\right)^T \mathbf{R}^{-1}\left(\mathbf{R}\vec{f} - \mathbf{B}\vec{v}\right) \ .$$

Therefore $\text{gap}(\vec{f}, \vec{v}) = \sum_{e \in E} \frac{1}{r_e}\left(\vec{f}(e)r_e - \Delta_{\vec{v}}(e)\right)^2$. However, by the uniqueness of tree paths, the antisymmetry of $\vec{f} \in \mathbb{R}^E$, and the definition of tree voltages we have

$$\forall a, b \in V \ : \ \Delta_{\vec{v}}(a, b) = \vec{v}(a) - \vec{v}(b) = \sum_{e \in P_{as}} \vec{f}(e)r_e + \sum_{e \in P_{sb}} \vec{f}(e)r_e = \sum_{e \in P_{ab}} \vec{f}(e)r_e \ .$$

Therefore, $e \in T \Rightarrow \vec{f}(e)r_e - \Delta_{\vec{v}}(e) = 0$ and $e \in E \setminus T \Rightarrow \vec{f}(e)r_e - \Delta_{\vec{v}}(e) = \Delta_{c_e}(\vec{f})$. □

2.4.3 Convergence Proof

Here we connect the energy decrease of a cycle update (Lemma 2.4.3) and the duality gap formula (Lemma 2.4.4) to bound the the convergence of `SimpleSolver`.

Throughout this section we use $\vec{v}_i$ to denote the tree induced voltages for flow $\vec{f}_i$.

First, we show that in expectation each iteration i of `SimpleSolver` decreases $\xi(\vec{f}_{i-1})$ by a $\frac{1}{\tau}$ fraction of the duality gap.

Lemma 2.4.5 (Expected Progress). *For iteration i of `SimpleSolver` we have*

$$\mathbb{E}\left[\xi(\vec{f}_i) - \xi(\vec{f}_{i-1}) \middle| \text{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right] = -\frac{\text{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})}{\tau} .$$

Proof. In each iteration i, `SimpleSolver` picks a random $e_i \in E \setminus T$ with probability p_{e_i} and adds a multiple of $\vec{c}_{e_i}$ which by Lemma 2.4.3 decreases the energy by $\frac{\Delta_{c_e}(\vec{f}_{i-1})^2}{R_e}$. Therefore

$$\mathbb{E}\left[\xi(\vec{f}_i) - \xi(\vec{f}_{i-1}) \middle| \text{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right] = \mathbb{E}\left[\sum_{e \in E \setminus T} p_e \left(\frac{-\Delta_{c_e}(\vec{f}_{i-1})^2}{R_e}\right) \middle| \text{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})\right]$$

Using that $p_e = \frac{1}{\tau} \cdot \frac{R_e}{r_e}$ and applying Lemma 2.4.4 yields the result. □

Next, we show that this implies that each iteration decreases the expected energy difference between the current flow and the optimal flow by a multiplicative $\left(1 - \frac{1}{\tau}\right)$.

Lemma 2.4.6 (Convergence Rate). *For all $i \geq 0$ let random variable $D_i \stackrel{\text{def}}{=} \xi(\vec{f}_i) - \xi(\vec{f}_{\text{opt}})$. Then for all iterations $i \geq 1$ we have $\mathbb{E}[D_i] \leq \left(1 - \frac{1}{\tau}\right)\mathbb{E}[D_{i-1}]$.*

Proof. Since in each iteration of `SimpleSolver` one of a finite number of edges is chosen, clearly D_i is a discrete random variable and by law of total expectation we have

$$\mathbb{E}[D_i] = \sum_c \mathbb{E}[D_i | D_{i-1} = c] \Pr[D_{i-1} = c]$$

However, we know $D_{i-1} \leq \text{gap}(\vec{f}_{i-1}, \vec{v}_{i-1})$ so by Lemma 2.4.5, $\mathbb{E}[D_i | D_{i-1} = c] \leq c - \frac{c}{\tau}$. Therefore:

$$\mathbb{E}[D_i] \leq \sum_c \left[\left(1 - \frac{1}{\tau}\right) c \cdot \Pr[D_{i-1} = c]\right] = \left(1 - \frac{1}{\tau}\right) \mathbb{E}[D_{i-1}] .$$

□

Finally, by induction on Lemma 2.4.6 and the definition of D_i we prove Theorem 2.4.1.

2.5 Cycle Update Data Structure

In this section we show how to implement each iteration (i.e. cycle update) of `SimpleSolver` in $O(\log n)$ time. In Section 2.5.1 we formulate this as a data structure problem, in Section 2.5.2 we present a recursive solution, and in Section 2.5.3 we provide a linear algebraic interpretation that may be useful in both theory and practice.

2.5.1 The Data Structure Problem

In each iteration i of `SimpleSolver` we pick a random $(a, b) \in E \setminus T$ and for feasible $\vec{f} \in \mathbb{R}^E$ compute

$$\alpha^* = \frac{\Delta_{c_{(a,b)}}(\vec{f})}{R_{(a,b)}} = \frac{\vec{f}(a,b)r_{a,b} - (\vec{v}(a) - \vec{v}(b))}{R_{(a,b)}} \quad \text{where} \quad \vec{v}(a) = \sum_{e \in P_{(a,s)}} \vec{f}(e)r_e$$

and s is an arbitrary fixed vertex in V which we refer to as the *root*. Then α^* is added to the flow on every edge in the tree cycle $C_{(a,b)}$. By the antisymmetry of flows and the uniqueness of tree paths, this update is equivalent to (1) adding α^* to the flow on edge (a, b), (2) adding $-\alpha^*$ to the flow on every edge in $P_{(s,b)}$ and then (3) adding α^* to the flow on every edge in $P_{(s,a)}$.

Therefore, to implement cycle updates it suffices to store the flow on off-tree edges in an array and implement a data structure that supports the following operations.

- `init`$(T, s \in V)$: initialize a data structure D given tree T and root $s \in V$.
- `query`$_D(a \in V)$: return $\vec{v}(a) = \sum_{e \in P_{(s,a)}} \vec{f}(e)r_e$.
- `update`$_D(a \in V, \alpha \in \mathbb{R})$: set $\vec{f}(e) := \vec{f}(e) + \alpha$ for all $e \in P_{(s,a)}$.

2.5.2 Recursive Solution

While one could solve this data structure problem with a slight modification of link-cut trees [47], that dynamic data structure is overqualified for our purpose. In contrast to the problems for which link-cut trees were originally designed, in our setting the tree is static, so that much of the sophistication of link-cut trees may be unnecessary. Here we develop a very simple separator decomposition tree based structure that provides worst-case $O(\log n)$ operations that we believe sheds more light on the electric flow problem and may be useful in practice.

Our solution is based on the well known fact that every tree has a good vertex separator, tracing back to Jordan in 1869 [23].

Lemma 2.5.1 (Tree Vertex Separator). *For a spanning tree T rooted at s with $n \geq 2$ vertices, in $O(n)$ time we can compute*

$$(d, T_0, \ldots, T_k) = \textbf{\textit{tree-decompose}}(T) \ ,$$

such that the removal of $d \in V$ (which might equal to s) disconnects T into subtrees $T_0, \ldots, T_k$, where T_0 is rooted at s and contains d as a leaf, while other T_i's are rooted at d. Furthermore, each T_i has at most $n/2 + 1$ vertices.

Proof. We start at root s, and at each step follow an edge to a child whose corresponding subtree is the largest among all children. We continue this procedure until at some vertex d (which might be s itself) the sizes of all its subtrees have no more than $\frac{n}{2}$ vertices. This vertex d is the desired vertex separator. Now if $s \neq d$ we let T_0 be the subtree above d rooted at s with d being the leaf, and let each T_i be a sub-tree below d and rooted at d; or when $s = d$ we let $T_0, \ldots, T_k$ each denote a subtree below d and rooted at d. [5] By pre-computing subtree sizes `tree-decompose`(T) runs in $O(n)$ time. □

[5] For technical purposes, when V has only two vertices with s being the root, we always define $d \in V$ such that $d \neq s$ and define `tree-decompose`(T) to return no sub-trees, i.e., $k = -1$. By orienting the tree away from s and precomputing the number of descendents of each node we can easily implement this in $O(n)$ time.

Applying this lemma recursively induces a separator decomposition tree from which we build our data structure. To execute $\texttt{init}(T, s)$ we compute a vertex separator $(d, T_0, \ldots, T_k)$ and recurse separately on each subtree until we are left with a tree of only two vertices. In addition, we precompute the total weight of the intersection of the root to d path $P_{(s,d)}$ and the root to a path $P_{(s,a)}$, denoted by $\texttt{height}(a)$ for every $a \in V$, by taking a walk over the tree. SeeAlgorithm 2 for details.

With this decomposition we can support `query` and `update` by maintaining just 2 values about the paths $P_{(s,d)}$. For each subtree we maintain the following variables:

- `d_drop`, the total potential drop induced on the the path $P_{(s,d)}$, and
- `d_ext`, the total amount of flow that has been updated using the entire $P_{(s,d)}$ path, i.e. the contribution to $P_{(s,d)}$ from vertices beyond d.

It is easy to see that these invariants can be maintained recursively and that they suffice to answer queries and updates efficiently (see Algorithm 3 and Algorithm 4). Furthermore, because the sizes of trees at least half at each recursion, `init` takes $O(n \log n)$ while `query` and `update` each takes $O(\log n)$ time.

2.5.3 Linear Algebra Solution

Here we unpack the recursion in the previous section to provide a linear algebraic view of our data structure that may be useful in both theory and practice. We show that the `init` procedure in the previous section computes for all $a \in V$ a query vector $\vec{q}(a)$ and an update vector $\vec{u}(a)$ each of support size $O(\log |V|)$, such that the entire state of the data structure is an $O(|V|)$ dimensional vector $\vec{x}$ initialized to $\vec{0}$ allowing `query` and `update` to be as simple as the following,

$$\begin{aligned} \texttt{query}(a) &: \texttt{return } \vec{q}(a) \cdot \vec{x} \\ \texttt{update}(a, \alpha) &: \vec{x} := \vec{x} + \alpha \vec{u}(a) \ . \end{aligned} \tag{2.3}$$

To see this, first note that `init` recursively creates a total of $N = O(|V|)$ subtrees $T_1, \ldots, T_N$. Letting $\texttt{d\_ext}_i$, $\texttt{d\_drop}_i$, $\texttt{height}_i$ denote `d_ext`, `d_drop`, and `height` as-

Algorithm 2: Recursive $\texttt{init}(T, s \in V)$

$\texttt{d\_ext} := 0$, $\texttt{d\_drop} := 0$;
$(d, T_0, \ldots, T_k) := \texttt{tree-decompose}(T)$;
$\forall a \in V : \texttt{height}(a) := \sum_{e \in P_{(s,a)} \cap P_{(s,d)}} r_e$;
if $|V| > 2$ **then**
  $\forall i \in \{0, 1, \ldots, k\} : D_i := \texttt{init}(T_i, d)$;
end

Algorithm 3: Recursive $\texttt{query}(a \in V)$

if $a = d$ **then return** $\texttt{d\_drop}$;
else if $|V| = 2$ **then return** 0;
else if $a \in T_0$ **then**
  return $\texttt{d\_ext} \cdot \texttt{height}(a) + \texttt{query}_{D_0}(a)$;
else
  let T_i be unique tree containing a;
  return $\texttt{d\_drop} + \texttt{query}_{D_i}(a)$;
end

Algorithm 4: Recursive $\texttt{update}(a \in V, \alpha \in \mathbb{R})$

$\texttt{d\_drop} := \texttt{d\_drop} + \alpha \cdot \texttt{height}(a)$;
if $|V| = 2$ **then return**;
if $a \notin T_0$ **then** $\texttt{d\_ext} := \texttt{d\_ext} + \alpha$;
if $a \neq d$ **then**
  let T_i be unique tree containing a;
  $\texttt{update}_{D_i}(a, \alpha)$;
end

sociated with tree T_i, the state of the data structure is completely represented by a vector $\vec{x} \in \mathbb{R}^{\{e,d\} \times N}$ defined as follows

$$\vec{x}_{c,i} \stackrel{\text{def}}{=} \begin{cases} \texttt{d\_ext}_i & \text{if } c = e; \\ \texttt{d\_drop}_i & \text{if } c = d. \end{cases}$$

It is easy to see that given vertex $a \in V$, in order for $\texttt{query}(a)$ or $\texttt{update}(a, \alpha)$ to affect tree T_i in the decomposition, it is necessary that a is a vertex in T_i and a is not the root s_i of T_i. Accordingly, we let $T(a) \stackrel{\text{def}}{=} \{i \in [N] \mid a \in T_i \text{ and } a \neq s_i\}$ be the indices of such trees that a may affect.

Next, in order to fully specify the query vector $\vec{q}(a)$ and the update vector $\vec{u}(a)$, we further refine $T(a)$ by defining $T_0(a) \stackrel{\text{def}}{=} \{i \in T(a) \mid a \text{ is in } T_0 \text{ of } \texttt{tree-decompose}(T_i)\}$ and $T_+(a) \stackrel{\text{def}}{=} T(a) \setminus T_0(a)$. Then, one can carefully check that the following definitions of $\vec{q}(a), \vec{u}(a) \in \mathbb{R}^{\{e,d\} \times N}$ along with their `query` and `update` in (2.3), exactly coincide with our recursive definitions in Algorithm 3 and Algorithm 4 respectively.

$$\vec{q}(a)_{(c,i)} \stackrel{\text{def}}{=} \begin{cases} \texttt{height}_i(a) & c = e \text{ and } i \in T_0(a) \\ 1 & c = d \text{ and } i \in T_+(a) \\ 0 & \text{otherwise} \end{cases}$$

and

$$\vec{u}(a)_{(c,i)} \stackrel{\text{def}}{=} \begin{cases} 1 & c = e \text{ and } i \in T_+(a) \\ \texttt{height}_i(a) & c = d \text{ and } i \in T(a) \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, the properties of `init` and `tree-decompose` immediately implies that each vertex a can appear in at most $O(\log n)$ trees where it is not the root, so $|T(a)| = O(\log n)$. As a consequence, $\vec{q}(a)$ and $\vec{u}(a)$ each has at most $O(\log n)$ non-zero entries. It is also not hard to see that we can pre-compute these vectors in $O(n \log n)$ time using our recursive `init`.

Applying these insights to the definition of a cycle update we see that this data structure allows each cycle update of `SimpleSolver` to be implemented as a single dot product and a single vector addition where one vector in each operation has $O(\log n)$ non-zero entries. Since these vectors can be precomputed each cycle update can be implemented by iterating over a fixed array of $O(\log n)$ pointers and performing simple arithmetic operations. We hope this may be fast in practice.

2.6 Asymptotic Running-Time

Here we give the remaining analysis needed to prove the correctness of `SimpleSolver` and prove Theorem 2.3.2. First we bound the energy of $\vec{f_0}$.

Lemma 2.6.1 (Initial Energy). [6] $\xi(\vec{f}_0) \leq \mathrm{st}(T) \cdot \xi(\vec{f}_{\mathrm{opt}})$.

Proof. Recall that $\vec{f}_0 \in \mathbb{R}^E$ is the *unique* vector that meets the demands and is nonzero only on T. Now, if for every $e \in E$ we sent $\vec{f}_{\mathrm{opt}}(e)$ units of flow along P_e we achieve a vector with the same properties. Therefore, $\vec{f}_0 = \sum_{e \in E} \vec{f}_{\mathrm{opt}}(e)\vec{p}_e$ and by applying the Cauchy-Schwarz inequality we get

$$\xi(\vec{f}_0) = \sum_{e \in T} r_e \Big(\sum_{e' \in E | e \in P_{e'}} \vec{f}_{\mathrm{opt}}(e') \Big)^2 \leq \sum_{e \in T} \left[\Big(\sum_{e' \in E | e \in P_{e'}} \frac{r_e}{r_{e'}} \Big) \Big(\sum_{e' \in E | e \in P_{e'}} r_{e'} \vec{f}_{\mathrm{opt}}(e')^2 \Big) \right] .$$

Applying the crude bound that

$$\forall e \in E \ : \ \sum_{e' \in E | e \in P_{e'}} r_{e'} \vec{f}_{\mathrm{opt}}(e')^2 \leq \xi(\vec{f}_{\mathrm{opt}})$$

and noting that by the definition of stretch

$$\sum_{e \in T} \sum_{e' \in E | e \in T_{e'}} \frac{r_e}{r_{e'}} = \sum_{e' \in E} \sum_{e \in T_e} \frac{r_e}{r_{e'}} = \mathrm{st}(T)$$

the result follows immediately. □

Next, we show how approximate optimality is preserved within polynomial factors when rounding from an ε-approximate electric flow to tree induced voltages.

Lemma 2.6.2 (Tree Voltage Rounding). *Let $\vec{f} \in \mathbb{R}^E$ be a primal feasible flow with $\xi(\vec{f}) \leq (1+\varepsilon) \cdot \xi(\vec{f}_{\mathrm{opt}})$ for $\varepsilon > 0$ and let $\vec{v} \in \mathbb{R}^V$ be the tree induced voltages. Then the following holds*

$$\left\| \vec{v} - \mathbf{L}^\dagger \vec{\chi} \right\|_{\mathbf{L}} \leq \sqrt{\varepsilon \cdot \tau} \left\| \mathbf{L}^\dagger \vec{\chi} \right\|_{\mathbf{L}} .$$

Proof. By Lemma 2.4.5 we know that one random cycle update from `SimpleSolver` is expected to decrease $\xi(\vec{f})$ by $\mathrm{gap}(\vec{f}, \vec{v})/\tau$. Therefore by the optimality of $\vec{f}_{\mathrm{opt}}$ we have

$$\xi(\vec{f}) - \frac{\mathrm{gap}(\vec{f}, \vec{v})}{\tau} \geq \xi(\vec{f}_{\mathrm{opt}})$$

[6] This lemma follows immediately from the well known fact that T is a $\mathrm{st}(T)$-spectral sparsifier of G, cf. [53, Lemma 9.2], but a direct proof is included here for completeness.

and since $\xi(\vec{f}) \leq (1+\varepsilon)\cdot\xi(\vec{f}_{\text{opt}})$ and $\xi(\vec{f}) \leq \xi(\vec{f}_{\text{opt}})$ we have $\text{gap}(\vec{f}_{\text{opt}}, \vec{v}) \leq \varepsilon \cdot \tau \cdot \xi(\vec{f}_{\text{opt}})$. Finally, using that $\vec{v}_{\text{opt}} = \mathbf{L}^{\dagger}\vec{\chi}$, $\vec{f}_{\text{opt}} = \mathbf{R}^{-1}\mathbf{B}\vec{v}_{\text{opt}}$, and $\mathbf{L} = \mathbf{B}^T\mathbf{R}^{-1}\mathbf{B}$ it is straightforward to check

$$\text{gap}(\vec{f}_{\text{opt}}, \vec{v}) = \left\|\vec{v} - \mathbf{L}^{\dagger}\vec{\chi}\right\|_{\mathbf{L}}^2 \quad \text{and} \quad \xi(\vec{f}_{\text{opt}}) = \left\|\mathbf{L}^{\dagger}\vec{\chi}\right\|_{\mathbf{L}}^2 .$$

□

Finally, we have everything we need to prove the correctness of `SimpleSolver`.

Proof of Theorem 2.3.2. Applying Theorem 2.4.1,Lemma 2.6.1 , and the definition of K immediately yields the following.

$$\mathbb{E}[\xi(\vec{f}_K)] - \xi(\vec{f}_{\text{opt}}) \leq \left(1 - \frac{1}{\tau}\right)^{\tau \log(\varepsilon^{-1}\text{st}(T)\tau)} \left(\text{st}\,(T)\,\xi(\vec{f}_{\text{opt}}) - \xi(\vec{f}_{\text{opt}})\right) .$$

Using several crude bounds and applying Lemma 2.6.2 we get

$$\mathbb{E}[\xi(\vec{f}_K)] \leq \left(1 + \frac{\varepsilon}{\tau}\right)\xi(\vec{f}_{\text{opt}}) \quad \text{and} \quad \mathbb{E}\left\|\vec{v}_K - \mathbf{L}^{\dagger}\vec{\chi}\right\|_{\mathbf{L}} \leq \sqrt{\varepsilon}\cdot\left\|\mathbf{L}^{\dagger}\vec{\chi}\right\|_{\mathbf{L}}$$

which is stronger than what we need to prove for the theorem.

All that remains is to bound the running time. To do this, we implement the algorithm `SimpleSolver` using the latest low-stretch spanning tree construction (Theorem 2.2.13) and construct a spanning tree T of stretch $\text{st}(T) = O(m \log n \log\log n)$ in time $O(m \log n \log\log n)$ yielding $\tau = O(m \log n \log\log n)$.

Next, to compute $\vec{f}_0$ we note that given any demand vector $\vec{\chi} \in \mathbb{R}^V$ with $\sum_i \vec{\chi}_i = 0$, the quantity $\vec{f}_0(e)$ on a tree edge $e \in T$ is uniquely determined by the summation of $\vec{\chi}_v$ where v is over the vertices on one side of e. Therefore, $\vec{f}_0$ can be computed via a DFS in $O(n)$ time.

To compute R_e for each off-tree edge $e \in E \setminus T$ we could either use Tarjan's off-line LCA algorithm [56] that runs in a total of $O(m)$ time, or simply use our own data structure in $O(m \log n)$ time. In fact, one can initiate a different instance of our data structure on T, and for each off-tree edge $(a, b) \in E \setminus T$, one can call `update`$(b, 1)$

and $\texttt{update}(a, -1)$, so that $R_e = \texttt{query}(b) - \texttt{query}(a) + r_e$.

Finally, we can initialize our data structure in $O(n \log n)$ time, set the initial flow values in $O(n \log n)$ time, perform each cycle update in $O(\log n)$ time, and compute all the tree voltages in $O(n \log n)$ time using the work in section Section 2.5. Since the total number of iterations of our algorithm is easily seen to be $O(m \log n \log(n\varepsilon^{-1}) \log \log n)$ we get the desired total running time. □

2.7 Numerical Stability

Up until this point our analysis has assumed that all arithmetic operations are exact. In this section, we show that `SimpleSolver` is numerically stable and achieves the same convergence guarantees when implemented with finite-precision arithmetic.

For simplicity of exposition, we assume that the resistances r_e and the coordinates of the demand vector $\vec{\chi}$ are all represented as b-bit integers with absolute values bounded by $N = 2^b$. We show that `SimpleSolver` works when arithmetic operations are implemented with $O(\max(b, \log n, \log 1/\varepsilon))$ bits of precision (which is necessary simply to guarantee we can represent an answer meeting the required error bound). In particular, if the entries of the input and ε can be stored in $\log n$-bit words, our running time guarantees hold in the standard unit cost RAM model, which only allows arithmetic operations on $O(\log n)$-bit numbers to be done in constant time.

We start our algorithm by multiplying the demand vector $\vec{\chi}$ by $\lceil 4mN^2/\varepsilon \rceil$, to ensure that in $\vec{f}_{\text{opt}}$ there exist at least $\lceil 4mN^2/\varepsilon \rceil$ total units of flow on the edges. Since the resistances are at least 1, this guarantees that $\xi(\vec{f}_{\text{opt}}) \geq \lceil 4mN^2/\varepsilon \rceil$. Next, we ensure that throughout our algorithm all flow vectors $\vec{f}_i$ are integer vectors. At each iteration of `SimpleSolver`, when we are given a primal feasible flow $\vec{f}$ and want to compute the optimal cycle update $\alpha^* \stackrel{\text{def}}{=} \frac{\vec{f}^T \mathbf{R} \vec{c}_e}{\vec{c}_e^T \mathbf{R} \vec{c}_e}$, we round this fraction to the nearest integer and suppose we pick some $\tilde{\alpha} \in \mathbb{Z}$ such that $|\alpha^* - \tilde{\alpha}| \leq \frac{1}{2}$.

Based on the modifications above, it is easy to see that our algorithm can be run on a RAM machine with word size $O(\max(b, \log n, \log 1/\varepsilon))$, since all flow values are integers bounded above by $O(\text{poly}(N, n, 1/\varepsilon))$. All we have left to show is the

convergence analysis for such integral updates.

We do so by strengthening our cycle update lemma, Lemma 2.4.3. For each cycle update, if we add $\tilde{\alpha} = \alpha^*(1+\delta)$ units of flow on the cycle instead of α^*, the corresponding energy decrease is

$$\begin{aligned}\xi(\vec{f} - \alpha^*(1+\delta)\vec{c}_e) - \xi(\vec{f}) &= (\vec{f} - \alpha^*(1+\delta)\vec{c}_e)^T\mathbf{R}(\vec{f} - \alpha^*(1+\delta)\vec{c}_e) - \vec{f}^T\mathbf{R}\vec{f} \\ &= -2\alpha^*(1+\delta)\vec{f}^T\mathbf{R}\vec{c}_e + (\alpha^*(1+\delta))^2\,\vec{c}_e^T\mathbf{R}\vec{c}_e \\ &= -\frac{\Delta_{c_e}(\vec{f})^2}{R_e}(1-\delta^2)\ ,\end{aligned}$$

where the last equality has used the fact that $R_e = \vec{c}_e^T\mathbf{R}\vec{c}_e$. We have $|\delta| \leq \frac{1}{2\alpha^*}$, so, as long as $\alpha^* \geq 1$, the decrease in the energy decrease is at least $\frac{3}{4}$ of what it would be for $\delta = 0$. We call an off-tree edge "good" if its $\alpha^* \geq 1$, and "bad" otherwise. We can rewrite the duality gap as

$$\begin{aligned}\text{gap}(\vec{f},\vec{v}) &= \sum_{e\in E\setminus T}\frac{\Delta_{c_e}(\vec{f})^2}{r_e} = \sum_{e\in E\setminus T}\frac{\Delta_{c_e}(\vec{f})^2}{R_e}\frac{R_e}{r_e} \\ &= \sum_{\substack{e\in E\setminus T\\ e \text{ is bad}}}\frac{\Delta_{c_e}(\vec{f})^2}{R_e^2}\frac{R_e^2}{r_e} + \sum_{\substack{e\in E\setminus T\\ e \text{ is good}}}\frac{\Delta_{c_e}(\vec{f})^2}{R_e}\frac{R_e}{r_e} \\ &\leq mN^2 + \sum_{\substack{e\in E\setminus T\\ e \text{ is good}}}\frac{\Delta_{c_e}(\vec{f})^2}{R_e}\frac{R_e}{r_e}\ .\end{aligned}$$

As a consequence, if one samples[7] each tree cycle $\vec{c}_e$ with probability to $\frac{R_e}{r_e\tau}$, then the expected energy decrease is at least:

$$\begin{aligned}\mathbb{E}\left[\xi(\vec{f}_i) - \xi(\vec{f}_{i-1})\middle|\text{gap}(\vec{f}_{i-1},\vec{v}_{i-1})\right] &\leq \sum_{\substack{e\in E\setminus T\\ e \text{ is good}}}\left(\frac{R_e}{r_e\tau}\right)\left(-\frac{\Delta_{c_e}(\vec{f})^2}{R_e}\frac{3}{4}\right) \\ \leq -\frac{\left(\text{gap}(\vec{f}_{i-1},\vec{v}_{i-1}) - mN^2\right)}{4\tau/3} &\leq -\frac{\left(\xi(\vec{f}_{i-1}) - \xi(\vec{f}_{\text{opt}}) - mN^2\right)}{4\tau/3}\ .\end{aligned}$$

[7]Recall that exact sampling takes expected $O(1)$ time on a machine with $O(\log n)$-sized words.

If one defines a random variable $D_i \stackrel{\text{def}}{=} \xi(\vec{f}_i) - \xi(\vec{f}_{\text{opt}}) - mN^2$, then

$$\mathbb{E}[D_i | D_{i-1}] \leq \left(1 - \frac{1}{4\tau/3}\right) D_{i-1} \ .$$

Using the same analysis as before, after $K = \frac{4\tau}{3} \log \frac{2\text{st}(T)}{\varepsilon \cdot p}$, we have with probability at least $1 - p$ that $\xi(\vec{f}_K) \leq (1 + \frac{\varepsilon}{2}) \left(\xi(\vec{f}_{\text{opt}}) + mN^2\right) \leq (1 + \varepsilon)\xi(\vec{f}_{\text{opt}})$ as desired.

Chapter 3

Efficient Accelerated Coordinate Descent

In the previous chapter we saw how how repeated application of a simple computational efficient update rule can be used to obtain a nearly linear time SDD solver. This idea of using simple sublinear-time iterative steps to solve a convex optimization problem is in fact an old one [24, 62, 4]. It is an algorithmic design principle that has seen great practical success [39, 21, 4] but has been notoriously difficult to analyze. In the past few years great strides have been taken towards developing a theoretical understanding of randomized variants of these approaches. Beyond the results in the previous chapter, in 2006 Strohmer and Vershynin [55] showed that a particular sublinear update algorithm for solving overconstrained linear systems called *randomized Kaczmarz* converges exponentially and in 2010 Nesterov [43] analyzed randomized analog of gradient descent that updates only a single coordinate in each iteration, called *coordinate gradient descent method* and provided a computationally inefficient but theoretically interesting accelerated variant, called *accelerated coordinate gradient descent method (ACDM).*

In this chapter we provide a framework that both strengthens and unifies these results. We present a more general version of Nesterov's ACDM and show how to implement it so that each iteration has the same asymptotic runtime as its non-accelerated variants. We show that this method is numerically stable and optimal

under certain assumptions. Then we show how to use this method to outperform conjugate gradient in solving a general class of symmetric positive definite systems of equations. Furthermore, we show how to cast both randomized Kaczmarz and `SimpleSolver` in this framework and achieve faster running times using ACDM.

The rest of this chapter is organized as follows. In Section 3.1, we introduce the problem and function properties that we will use for optimization. In Section 3.2, we briefly review the mathematics behind gradient descent, accelerated gradient descent, and coordinate descent. In Section 3.3, we present our general ACDM implementation, prove correctness, and show how to implement the update steps efficiently. In Section 3.4, we show how to apply ACDM to achieve faster runtimes for various linear system solving scenarios and in Appendix A we provide missing details of the proof of ACDM convergence and provide numerical stability proofs.

3.1 Preliminaries

For the remainder of this chapter we change notation slightly and focus on the general unconstrained minimization problem [1]

$$\min_{\vec{x} \in \mathbb{R}^n} f(\vec{x})$$

where the *objective function* $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and convex, meaning that

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \quad : \quad f(\vec{y}) \geq f(\vec{x}) + \langle \nabla f(x), \vec{y} - \vec{x} \rangle \ .$$

We let $f^* \stackrel{\text{def}}{=} \min_{\vec{x} \in \mathbb{R}^n} f(\vec{x})$ denote the minimum value of this optimization problem and we let $\vec{x}^* \stackrel{\text{def}}{=} \arg\min_{\vec{x} \in \mathbb{R}^n} f(\vec{x})$ denote an arbitrary point that achieves this value.

To minimize f, we restrict our attention to *first-order* iterative methods, that is algorithms that generate a sequence of points $\vec{x}_1, \ldots, \vec{x}_k$ such that $\lim_{k \to \infty} f(\vec{x}_k) = f^*$,

[1] Many of the results in this chapter can be generalized to constrained minimization problems [43], problems where f is strongly convex with respect to different norms [43], and problems where each coordinate is a higher dimension variable (i.e. the block setting) [45]. However, for simplicity we focus on the basic coordinate unconstrained problem in the Euclidian norm.

while only evaluating the objective function, f, and its gradient ∇f, at points. In other words, other than the global function parameters related to f that we define in this section, we assume that the algorithms under consideration have no further knowledge regarding the structure of f. To compare such algorithms, we say that *an iterative method has convergence rate* r if $f(\vec{x}_k) - f^* \leq O((1-r)^k)$ for this method.

Now, we say that f *has convexity parameter* σ *with respect to some norm* $\|\cdot\|$ if the following holds

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \ : \ f(\vec{y}) \geq f(\vec{x}) + \langle \nabla f(\vec{x}), \vec{y} - \vec{x} \rangle + \frac{\sigma}{2} \|\vec{y} - \vec{x}\|^2 \tag{3.1}$$

and we say f *strongly convex* if $\sigma > 0$. We refer to the right hand side of (3.1) as the *lower envelope of* f *at* $\vec{x}$ and for notational convenience when the norm $\|\cdot\|$ is not specified explicitly we assume it to be the standard Euclidian norm $\|x\| \stackrel{\text{def}}{=} \sqrt{\sum_i \vec{x}_i^2}$.

Furthermore, we say f has L-Lipschitz gradient if

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \ : \ \|\nabla f(\vec{y}) - \nabla f(\vec{x})\| \leq L \|\vec{y} - \vec{x}\|$$

The definition is related to an upper bound on f as follows:

Lemma 3.1.1. *[41, Thm 2.1.5] For continuously differentiable* $f : \mathbb{R}^n \to \mathbb{R}$ *and* $L > 0$, *it has* L*-Lipschitz gradient if and only if*

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \ : \ f(\vec{y}) \leq f(\vec{x}) + \langle \nabla f(\vec{x}), \vec{y} - \vec{x} \rangle + \frac{L}{2} \|\vec{x} - \vec{y}\|^2 \ . \tag{3.2}$$

We call the right hand side of (3.2) the *upper envelope of* f *at* $\vec{x}$.

The convexity parameter μ and the Lipschitz constant of the gradient L provide lower and upper bounds on f. They serve as the essential characterization of f for first-order methods and they are typically the only information about f provided to both gradient and accelerated gradient descent methods (besides the oracle access to f and ∇f). For twice differentiable f, these values can also be computed by properties of the Hessian of f by the following well known lemma:

Lemma 3.1.2 ([43]). *Twice differentiable $f : \mathbb{R}^n \to \mathbb{R}$ has convexity parameter μ and L-Lipschitz gradient with respect to norm $\|\cdot\|$ if and only if $\forall \vec{x} \in \mathbb{R}^n$ the Hessian of f at $\vec{x}$, $\nabla^2 f(\vec{x}) \in \mathbb{R}^{n \times n}$ satisfies*

$$\forall \vec{y} \in \mathbb{R}^n \ : \ \mu \|\vec{y}\|^2 \leq \vec{y}^T \left(\nabla^2 f(\vec{x})\right) \vec{y} \leq L \|\vec{y}\|^2$$

To analyze *coordinate-based iterative methods*, that is iterative methods that only consider one component of the current point or current gradient in each iteration, we need to define several additional parameters characterizing f. For all $i \in [n]$, let $\vec{e}_i \in \mathbb{R}^n$ denote the standard basis vector for coordinate i, let $f_i(\vec{x}) \in \mathbb{R}^n$ denote the partial derivative of f at $\vec{x}$ along $\vec{e}_i$, i.e. $f_i(\vec{x}) \stackrel{\text{def}}{=} \vec{e}_i^T \nabla f(\vec{x})$, and let $\vec{f}_i(\vec{x})$ denote the corresponding vector, i.e $\vec{f}_i(\vec{x}) \stackrel{\text{def}}{=} f_i \cdot \vec{e}_i$. We say that f *has component-wise Lipschitz continuous gradient with Lipschitz constants* $\{L_i\}$ if

$$\forall \vec{x} \in \mathbb{R}^n, \ \forall t \in \mathbb{R}, \ \forall i \in [n] \ : \ |f_i(\vec{x} + t \cdot \vec{e}_i) - f_i(\vec{x})| \leq L_i \cdot |t|$$

and for all $\alpha \geq 0$ we let $S_\alpha \stackrel{\text{def}}{=} \sum_{i=1}^n L_i^\alpha$ denote the total component-wise Lipschitz constant of ∇f. Later we will see that S_α has a similar role for coordinate descent as L has for gradient descent.

We give two examples for convex functions induced by linear systems and calculate their parameters. Note that even though one example can be deduced from the other, we provide both as the analysis allows us to introduce more notation.

Example 3.1.3. *Let $f(\vec{x}) \stackrel{\text{def}}{=} \frac{1}{2}\langle \mathbf{A}\vec{x}, \vec{x} \rangle - \langle \vec{x}, \vec{b} \rangle$ for symmetric positive definite matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. Since $\mathbf{A} = \mathbf{A}^T$ clearly $\nabla f(\vec{x}) = \mathbf{A}\vec{x} - \vec{b}$ and $\nabla^2 f(\vec{x}) = \mathbf{A}$. Therefore, by Lemma 3.1.2, L and σ satisfy*

$$\sigma \|\vec{x}\|^2 \leq \vec{x}^T \mathbf{A} \vec{x} \leq L \|\vec{x}\|^2 \ .$$

Consequently, σ is the the smallest eigenvalue $\lambda_{\min}$ of $\mathbf{A}$ and L is the largest eigenvalue $\lambda_{\max}$ of $\mathbf{A}$. Furthermore, $\forall i \in [n]$ we see that $f_i(\vec{x}) = \vec{e}_i^T \left(\mathbf{A}\vec{x} - \vec{b}\right)$ and therefore L_i

satisfies

$$\forall t \in \mathbb{R} \ : \ |t| \cdot |\mathbf{A}_{ii}| = |\vec{e}_i^T \mathbf{A}(te_i)| \leq L_i |t| \ .$$

Since the positive definiteness of $\mathbf{A}$ *implies that* $\mathbf{A}$ *is positive on diagonal, we have* $L_i = \mathbf{A}_{ii}$*, and consequently* $S_1 = \mathrm{tr}(\mathbf{A}) = \sum_{i=1}^n \mathbf{A}_{ii} = \sum_{i=1}^n \lambda_i$ *where* λ_i *are eigenvalues of* $\mathbf{A}$*.*

Example 3.1.4. *Let* $f(\vec{x}) = \frac{1}{2} \|\mathbf{A}\vec{x} - b\|^2$ *for any matrix* $\mathbf{A}$*. Then the gradient* $\nabla f(\vec{x}) = \mathbf{A}^T \left(\mathbf{A}\vec{x} - \vec{b}\right)$ *and the hessian* $\nabla^2 f(\vec{x}) = \mathbf{A}^T\mathbf{A}$*. Hence,* σ *and* L *satisfy*

$$\sigma \|\vec{x}\|^2 \leq \vec{x}^T \mathbf{A}^T \mathbf{A} \vec{x} \leq L \|\vec{x}\|^2$$

and we see that σ *is the the smallest eigenvalue* $\lambda_{\min}$ *of* $\mathbf{A}^T\mathbf{A}$ *and* L *is the largest eigenvalue* $\lambda_{\max}$ *of* $\mathbf{A}^T\mathbf{A}$*. As in the previous example, we therefore have* $L_i = \|a_i\|^2$ *where* a_i *is the* i*-th column of* $\mathbf{A}$ *and* $S_1 = \sum \|a_i\|^2 = \|\mathbf{A}\|_F^2$*, the Frobenius norm of* $\mathbf{A}$*.*

3.2 Review of Previous Iterative Methods

In this section, we briefly review several standard iterative first-order method for smooth convex minimization. This overview is by no means all-inclusive, our goal is simply to familiarize the reader with numerical techniques we will make heavy use of later and motivate our presentation of the accelerated coordinate descent method. For a more comprehensive review, there are multiple good references, e.g. [41], [14].

In Section 3.2.1, we briefly review the standard gradient descent method. In Section 3.2.2, we show how to improve gradient descent by motivating and reviewing Nesterov's *accelerated gradient descent method* [40] through a more recent presentation of his via *estimate sequences* [41]. In Section 3.2.3, we review Nesterov's *coordinate gradient descent method* [43]. In the next section we combine these concepts to present a general and efficient accelerated coordinate descent scheme.

3.2.1 Gradient Descent

Given an initial point $\vec{x}_0 \in \mathbb{R}^n$ and step sizes $h_k \in \mathbb{R}$, the *gradient descent method* applies the following simple iterative update rule:

$$\forall k \geq 0 \ : \ \vec{x}_{k+1} := \vec{x}_k - h_k \nabla f(\vec{x}_k).$$

For $h_k = \frac{1}{L}$, this method simply chooses the minimum point of the upper envelope of f at $\vec{x}_k$:

$$\vec{x}_{k+1} = \arg\min_{\vec{y}} \left\{ f(\vec{x}_k) + \langle \nabla f(\vec{x}_k), \vec{y} - \vec{x}_k \rangle + \frac{L}{2} \|\vec{y} - \vec{x}_k\|^2 \right\}.$$

Thus, we see that the gradient descent method is a greedy method that chooses the minimum point based on the worst case estimate of the function based on the value of $f(x_k)$ and $\nabla f(x_k)$. It is well known that it provides the following guarantee [41, Cor 2.1.2, Thm 2.1.15]

$$f(\vec{x}_k) - f^* \leq \frac{L}{2} \cdot \min \left\{ \left(1 - \frac{\sigma}{L}\right)^k, \frac{4}{k+4} \right\} \|\vec{x}_0 - \vec{x}^*\|^2 \ . \tag{3.3}$$

3.2.2 Accelerated Gradient Descent

To speed up the greedy and memory-less gradient descent method, one could make better use of the history and pick the next step to be the smallest point in upper envelope of all points computed. Formally, one could try the following update rule

$$\vec{x}_{k+1} := \arg\min_{\vec{y} \in \mathbb{R}^n} \min_{\sum_{k=1}^n t_k \vec{z}_k = \vec{y}} \left\{ \sum_{k=1}^n t_k \left(f(\vec{x}_k) + \langle \nabla f(\vec{x}_k), \vec{z}_k - \vec{x}_k \rangle + \frac{L}{2} \|\vec{z}_k - \vec{x}_k\|^2 \right) \right\}.$$

However, this problem is difficult to solve efficiently and requires storing all previous points. To overcome this problem, Nesterov [40, 41] suggested to use a quadratic function to estimate the function. Formally, we define an *estimate sequence* as follows: [2]

[2]Note that our definition deviates slightly from Nesterov's [41, Def 2.2.1] in that we include condition 3.5.

Definition 3.2.1 (Estimate Sequence). *A triple of sequences* $\{\phi_k(x), \eta_k, \vec{x}_k\}_{k=0}^{\infty}$ *is called an* estimate sequence *of* f *if* $\lim_{k\to\infty} \eta_k = 0$ *and for any* $\vec{x} \in \mathbb{R}^n$ *and* $k \geq 0$ *we have*

$$\phi_k(\vec{x}) \leq (1 - \eta_k) f(\vec{x}) + \eta_k \phi_0(\vec{x}) \tag{3.4}$$

and

$$f(\vec{x}_k) \leq \min_{\vec{x}\in\mathbb{R}^n} \phi_k(\vec{x}). \tag{3.5}$$

An estimate sequence of f is an approximate lower bound of f which is slightly above f^*. This relaxed definition allows us to find a better computable approximation of f instead of relying on the worst case upper envelope at each step.

A good estimate sequence gives an efficient algorithm [41, Lem 2.2.1] by the following

$$\lim_{k\to\infty} f(\vec{x}_k) - f^* \leq \lim_{k\to\infty} \eta_k \left(\phi_0(\vec{x}^*) - f^*\right) = 0.$$

Since an estimate sequence is an approximate lower bound, a natural computable candidate is to use the convex combination of lower envelopes of f at some points.

Since it can be shown that any convex combinations of lower envelopes at evaluation points $\{y_k\}$ satisfies (3.4) under some mild condition, additional points $\{y_k\}$ other than $\{x_k\}$ can be used to tune the algorithm. Nesterov's *accelerated gradient descent method* can be obtained by tuning the the free parameters $\{y_k\}$ and $\{\eta_k\}$ to satisfy (3.5). Among all first order methods, this method is optimal up to constants in terms of number of queries made to f and ∇f. The performance of the accelerated gradient descent method can be characterized as follows: [41]

$$f(\vec{x}_k) - f^* \leq L \cdot \min\left\{\left(1 - \sqrt{\frac{\sigma}{L}}\right)^k, \frac{4}{(k+2)^2}\right\} \|\vec{x}_0 - \vec{x}^*\|^2 \ . \tag{3.6}$$

3.2.3 Coordinate Descent

The *coordinate descent method* of Nesterov [43] is a variant of gradient descent in which only one coordinate of the current iterate is updated at a time. For a fixed $\alpha \in \mathbb{R}$, each iteration k of coordinate descent consists of picking a random a random

coordinate $i_k \in [n]$ where

$$\Pr[i_k = j] = P_\alpha(j) \quad \text{where} \quad P_\alpha(j) \stackrel{\text{def}}{=} \frac{L_{i_k}^\alpha}{S_\alpha}$$

and then performing a gradient descent step on that coordinate:

$$\vec{x}_{k+1} := \vec{x}_k - \frac{1}{L_{i_k}} \vec{f}_{i_k}(\vec{x}_k).$$

To analyze this algorithm's convergence rate, we define the norm $\|\cdot\|_{1-\alpha}$, its dual $\|\cdot\|_{1-\alpha}^*$, and the inner product $\langle \cdot, \cdot \rangle_{1-\alpha}$ which induces this norm as follows:

$$\|\vec{x}\|_{1-\alpha} \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^n L_i^{1-\alpha} \vec{x}_i^2} \quad \text{and} \quad \|\vec{x}\|_{1-\alpha}^* \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^n L_i^{-(1-\alpha)} \vec{x}_i^2}$$

and

$$\langle \vec{x}, \vec{y} \rangle_{1-\alpha} \stackrel{\text{def}}{=} \sum_{i=1}^n L_i^{1-\alpha} \vec{x}_i \vec{y}_i$$

and we let $\sigma_{1-\alpha}$ denote the convexity parameter of f with respect to $\|\cdot\|_{1-\alpha}$.

Using the definition of coordinate-wise Lipschitz constant, each step can be shown to have the following guarantee on expected improvement [43]

$$f(\vec{x}_k) - \mathbb{E}[f(\vec{x}_{k+1})] \geq \frac{1}{2S_\alpha} \left(\|\nabla f(\vec{x}_k)\|_{1-\alpha}^* \right)^2.$$

and further analysis shows the following convergence guarantee coordinate descent [43]

$$\mathbb{E}[f(\vec{x}_k)] - f^* \leq \min \left\{ \frac{2S_\alpha}{k+4} \left(\max_{f(\vec{y}) \leq f(\vec{x}_0)} \|\vec{y} - \vec{x}^*\|_{1-\alpha}^2 \right), \left(1 - \frac{\sigma_{1-\alpha}}{S_\alpha}\right)^k (f(\vec{x}_0) - f^*) \right\}.$$

3.3 General Accelerated Coordinate Descent

In this section, we present our general and iteration-efficient *accelerated coordinate descent method (ACDM)*. In particular, we show how to improve the asymptotic con-

vergence rate of any coordinate descent based algorithm without paying asymptotic cost in the running time due to increased cost in querying and updating a coordinate. We remark that the bulk of the credit for conceiving of such a method belongs to Nesterov [43] who provided a different proof of convergence for such a method for the $\alpha = 0$ case, however we note that changes to the algorithm were necessary to deal with the $\alpha = 1$ case used in all of our applications.

The rest of this section is structured as follows. In Section 3.3.1, we introduce and prove the correctness of general ACDM through what we call *(probabilistic) estimation sequences*, in Section 3.3.2, we present the numerical stability results we achieve for this method, and in Section 3.3.3, we show how to implement this method efficiently. In Appendix A.2, we include some additional details for the correctness proof and in Appendix A.3, we provide the details of the numerical stability proof.

3.3.1 ACDM by Probabilistic Estimate Sequences

Following the spirit of the estimate sequence proof of accelerated gradient descent [41], here we present a proof of ACDM convergence through what we call a *(probabilistic) estimation sequence.*

Definition 3.3.1 ((Probabilistic) Estimate Sequence). *A triple of sequences denoted,* $\{\phi_k(\vec{x}), \eta_k, \vec{x}_k\}_{k=0}^{\infty}$, *where* $\phi_k : \mathbb{R}^n \to \mathbb{R}$ *and* $\vec{x}_k \in \mathbb{R}^n$ *are chosen according to some probability distribution is called a* (probabilistic) estimate sequence of f *if* $\lim_{k\to 0} \eta_k = 0$ *and for all* $k \geq 0$ *we have*

$$\forall \vec{x} \in \mathbb{R}^n \quad : \quad \mathbb{E}\left[\phi_k(\vec{x})\right] \leq (1 - \eta_k) f(\vec{x}) + \eta_k \mathbb{E}\left[\phi_0(\vec{x})\right] \tag{3.7}$$

and

$$\mathbb{E}\left[f(\vec{x}_k)\right] \leq \min_{\vec{x} \in \mathbb{R}^n} \mathbb{E}\left[\phi_k(\vec{x})\right] \tag{3.8}$$

A probabilistic estimation sequence gives a randomized minimization method due to the following

$$\lim_{k\to\infty} \mathbb{E}\left[f(\vec{x}_k)\right] - f^* \leq \lim_{k\to\infty} \eta_k \left(\mathbb{E}\phi_0(x^*) - f^*\right) = 0 \ .$$

Since a probabilistic estimation sequence can be constructed using random partial derivatives, rather than a full gradient computations, there is hope that in certain cases probabilistic estimation sequences require less information for fast convergence and therefore outperform their deterministic counterparts.

Similar to the accelerated gradient descent method, in the following lemma we first show how to combine a sequence of lower envelopes to satisfy condition (3.7) and prove that it preserves a particular structure on the current lower bound.

Lemma 3.3.2 ((Probabilistic) Estimate Sequence Construction). *Let* $\phi_0(\vec{x}) \in \mathbb{R}^n \to \mathbb{R}$ *and* $\{\vec{y}_k, \theta_k, i_k\}_{k=0}^{\infty}$ *be such that*

- *Each* $\vec{y}_k \in \mathbb{R}^n$
- *Each* $\theta_k \in (0,1)$ *and* $\sum_{k=0}^{\infty} \theta_k = \infty$
- *Each* $i_k \in [n]$ *is chosen randomly such that* $\forall i \in [n]$ *we have* $\Pr[i_k = i] = \frac{L_i^{\alpha}}{S_{\alpha}}$.

Then the pair of sequences $\{\phi_k(\vec{x}), \eta_k\}_{k=0}^{\infty}$ *defined by*

- $\eta_0 = 1$ *and* $\eta_{k+1} = (1-\theta_k)\eta_k$
- $\phi_{k+1}(\vec{x}) = (1-\theta_k)\phi_k(\vec{x}) + \theta_k \left[f(\vec{y}_k) + \frac{S_{\alpha}}{L_{i_k}} \langle \vec{f}_{i_k}(\vec{y}_k), \vec{x} - \vec{y}_k \rangle_{1-\alpha} + \frac{\sigma_{1-\alpha}}{2} \|\vec{x} - \vec{y}_k\|_{1-\alpha}^2 \right]$

satisfies condition (3.7). Furthermore, if $\phi_0(\vec{x}) = \phi_0^* + \frac{\zeta_0}{2} \|\vec{x} - \vec{v}_0\|_{1-\alpha}^2$, *then this process produces a sequence of quadratic functions of the form* $\phi_k(\vec{x}) = \phi_k^* + \frac{\zeta_k}{2} \|\vec{x} - \vec{v}_k\|_{1-\alpha}^2$ *where*

$$\begin{aligned}
\zeta_{k+1} &= (1-\theta_k)\zeta_k + \theta_k \sigma_{1-\alpha} && (3.9)\\
v_{k+1} &= \frac{1}{\zeta_{k+1}} \left[(1-\theta_k)\zeta_k \vec{v}_k + \theta_k \sigma_{1-\alpha} \vec{y}_k - \frac{S_{\alpha}\theta_k}{L_{i_k}} \vec{f}_{i_k}(\vec{y}_k) \right] && (3.10)\\
\phi_{k+1}^* &= (1-\theta_k)\phi_k^* + \theta_k f(\vec{y}_k) - \frac{\theta_k^2 S_{\alpha}^2}{2\zeta_{k+1}} \frac{(f_{i_k}(\vec{y}_k))^2}{L_{i_k}^{1+\alpha}} \\
&\quad + \frac{\theta_k(1-\theta_k)\zeta_k}{\zeta_{k+1}} \left(\frac{\sigma_{1-\alpha}}{2} \|\vec{y}_k - \vec{v}_k\|_{1-\alpha}^2 + \frac{S_{\alpha}}{L_{i_k}} \langle \vec{f}_{i_k}(\vec{y}_k), \vec{v}_k - \vec{y}_k \rangle_{1-\alpha} \right) .
\end{aligned}$$

Proof. The proof follows from direct calculations however for completeness and for later use we provide a proof of an even more general statement in Appendix A.1. □

In the following theorem, we show how to choose $\vec{x}$, $\vec{y}$ and θ to satisfy the condition (3.8) and thereby derive a simple form of the general accelerated coordinate descent method. We also show that the number of iterations required for ACDM is $\tilde{O}\left(\sqrt{\frac{S_\alpha n}{\sigma_{1-\alpha}}}\right)$ which is strictly better than the number of iterations required for coordinate descent method, $\tilde{O}\left(\frac{S_\alpha}{\sigma_{1-\alpha}}\right)$. Note that while several of the definitions specifications in the following theorem statement may at first glance seem unnatural our proof will show that they are nearly forced in order to achieve certain algorithm design goals.

Theorem 3.3.3 (Simple ACDM). *For all $i \in [n]$ let $\tilde{L}_i^\alpha \stackrel{\text{def}}{=} \max(L_i^\alpha, S_\alpha/n)$* [3] *and let $\tilde{S}_\alpha \stackrel{\text{def}}{=} \sum_{i=1}^n \tilde{L}_i^\alpha$. Furthermore, for any $\vec{x}_0 \in \mathbb{R}^n$ and for all $k \geq 0$, let*

$$\vec{y}_0 = \vec{x}_0 \qquad \phi_0^* = f(\vec{x}_0) \qquad \phi_0(\vec{x}) = \phi_0^* + \frac{\sigma_{1-\alpha}}{2}\|\vec{x} - \vec{x}_0\|_{1-\alpha}^2 \qquad \theta_k = \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$$

Then applying Lemma 3.3.2 with these parameters, $\tilde{L}_i$ as the coordinate-wise gradient Lipshitz constants, and choosing $\vec{y}_k$ and $\vec{x}_k$ such that

$$\forall k \geq 1 \;:\; \frac{\theta_k \zeta_k}{\zeta_{k+1}}(\vec{v}_k - \vec{y}_k) + \vec{x}_k - \vec{y}_k = 0 \quad \text{and} \quad \vec{x}_k = \vec{y}_{k-1} - \frac{1}{\tilde{L}_{i_{k-1}}}\vec{f}_{i_k}(\vec{y}_{k-1})$$

yields a probabilistic estimate sequence. This accelerated coordinate descent method *satisfies*

$$\forall k \geq 0 \;:\; \mathbb{E}[f(\vec{x}_k)] - f^* \leq \left(1 - \frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{S_\alpha n}}\right)^k \left(f(\vec{x}_0) - f^* + \sigma_{1-\alpha}\|\vec{x}^* - \vec{x}_0\|_{1-\alpha}^2\right) . \tag{3.11}$$

Proof. By construction we know that condition (3.7) of probabilistic estimation sequences holds. It remains to show that $\vec{x}_k$ satisfies condition (3.8) and analyze the convergence rate. To prove (3.8), we proceed by induction to prove the following

[3] We introduce this to avoid small sampling probabilities as they cause an issue in achieving the optimal $\tilde{O}\left(\sqrt{\frac{S_\alpha n}{\sigma_{1-\alpha}}}\right)$ convergence rate. This modification can be avoided by choosing a different distribution over coordinate updates in compute x_{k+1} which makes the algorithm more complicated and potentially more expensive.

statement:

$$\forall k \geq 0 \ : \ \mathbb{E}_k\left[f(\vec{x}_k)\right] \leq \mathbb{E}_k\left[\min_{\vec{x}\in\mathbb{R}^n} \phi_k(\vec{x})\right] = \mathbb{E}_k\left[\phi_k^*\right] \ .$$

where $\mathbb{E}_k$ indicates the expectation up to iteration k. The base case $f(\vec{x}_0) \leq \phi_0(\vec{x}_0)$ is trivial and we proceed by induction assuming that $\mathbb{E}_k\left[f(\vec{x}_k)\right] \leq \mathbb{E}_k\left[\phi_k^*\right]$. By Lemma (3.3.2) and the inductive hypothesis we get

$$\begin{aligned}\mathbb{E}_k\left[\phi_{k+1}^*\right] \ \geq \ \mathbb{E}_k\Bigg[&(1-\theta_k)f(\vec{x}_k) + \theta_k f(\vec{y}_k) - \frac{\theta_k^2\tilde{S}_\alpha^2}{2\zeta_{k+1}}\frac{\left(f_{i_k}(\vec{y}_k)\right)^2}{\tilde{L}_{i_k}^{1+\alpha}} \\ &+\frac{\theta_k(1-\theta_k)\zeta_k}{\zeta_{k+1}}\left(\frac{\sigma_{1-\alpha}}{2}\left\|\vec{y}_k-\vec{v}_k\right\|_{1-\alpha}^2 + \frac{\tilde{S}_\alpha}{\tilde{L}_{i_k}}\langle\vec{f}_{i_k}(\vec{y}_k), \vec{v}_k-\vec{y}_k\rangle_{1-\alpha}\right)\Bigg]\end{aligned}$$

where for notational convenience we drop the expectation in each of the variables. By convexity $f(\vec{x}_k) \geq f(\vec{y}_k) + \langle\nabla f(\vec{y}_k), \vec{x}_k - \vec{y}_k\rangle$ so applying this and the definitions of $\|\cdot\|_{1-\alpha}$ and $\langle\cdot,\cdot\rangle_{1-\alpha}$ we get that ϕ_{k+1}^* is lower bounded by

$$f(\vec{y}_k) - \frac{\theta_k^2\tilde{S}_\alpha^2 f_{i_k}(\vec{y}_k)^2}{2\zeta_{k+1}\tilde{L}_{i_k}^{1+\alpha}} + (1-\theta_k)\left(\frac{\tilde{S}_\alpha}{\tilde{L}_{i_k}^{\alpha}}\left\langle\vec{f}_{i_k}(\vec{y}_k), \frac{\theta_k\zeta_k}{\zeta_{k+1}}(\vec{v}_k-\vec{y}_k)\right\rangle + \langle\nabla f(\vec{y}_k), \vec{x}_k-\vec{y}_k\rangle\right)$$

Using that $\forall i \in [n]$, $\Pr[i_k = i] = \frac{\tilde{L}_i^\alpha}{\tilde{S}_\alpha}$ we get that $\mathbb{E}_{k+1}\left[\phi_{k+1}^*\right]$ is lower bounded by

$$\mathbb{E}_{k+1}\left[f(\vec{y}_k) - \frac{\theta_k^2\tilde{S}_\alpha^2}{2\zeta_{k+1}}\frac{\left(f_{i_k}(\vec{y}_k)\right)^2}{\tilde{L}_{i_k}^{1+\alpha}}\right] + (1-\theta_k)\left\langle\nabla f(\vec{y}_k), \frac{\theta_k\zeta_k}{\zeta_{k+1}}(\vec{v}_k-\vec{y}_k) + \vec{x}_k - \vec{y}_k\right\rangle \ .$$

From this formula we see that $\vec{y}_k$ was chosen specifically to cancel the second term so

$$\mathbb{E}_{k+1}\left[\phi_{k+1}^*\right] \geq \mathbb{E}_{k+1}\left[f(\vec{y}_k) - \frac{\theta_k^2\tilde{S}_\alpha^2}{2\zeta_{k+1}}\frac{\left(f_{i_k}(\vec{y}_k)\right)^2}{\tilde{L}_{i_k}^{1+\alpha}}\right] = \sum_{i=1}^n\left[f(\vec{y}_k) - \frac{\theta_k^2\tilde{S}_\alpha}{2\zeta_{k+1}}\frac{f_i(\vec{y}_k)^2}{\tilde{L}_i}\right]$$

and it simply remains to choose θ_k and $\vec{x}_{k+1}$ so that $\mathbb{E}_{k+1}\left[f(\vec{x}_{k+1}]\right)$ is smaller than this quantity.

To meet condition (3.8), we simply need to choose $\{\theta_k\}_{k=0}^\infty$ so $\frac{\theta_k^2\tilde{S}_\alpha}{\zeta_{k+1}} = \frac{1}{2n}$. Using

$\tilde{L}_i^{\alpha} \geq \frac{S_\alpha}{n} \geq \frac{\tilde{S}_\alpha}{2n}$, we have

$$\mathbb{E}_{k+1}\left[\phi_{k+1}^*\right] \geq \sum_{i=1}^{n}\left[f(\vec{y}_k) - \frac{1}{2\tilde{S}_\alpha}\frac{f_i(\vec{y}_k)^2}{\tilde{L}_i^{1-\alpha}}\right]$$

To compute $\vec{x}_{k+1}$, we use the fact that applying Lemma 3.1.1 to the formula $f(\vec{y}_k - t\vec{f}_i(\vec{y}_k))$ yields

$$f\left(\vec{y}_k - \frac{1}{\tilde{L}_i}\vec{f}_i(\vec{y}_k)\right) \leq f(\vec{x}) + \left\langle \vec{f}_i(\vec{y}_k), -\frac{1}{\tilde{L}_i}\vec{f}_i(\vec{y}_k)\right\rangle + \frac{\tilde{L}_i}{2}\left(\frac{1}{\tilde{L}_i}\vec{f}_i(\vec{y}_k)\right)^2 \leq f(\vec{y}_k) - \frac{f_i(\vec{y}_k)^2}{2\tilde{L}_i}$$

and therefore for $\vec{x}_{k+1}$ as defined we have

$$\mathbb{E}_{k+1}\left[f(\vec{x}_{k+1})\right] \leq \sum_{i=1}^{n}\left[f(\vec{y}_k) - \frac{f_i(\vec{x})^2}{2\tilde{S}_\alpha \tilde{L}_i^{1-\alpha}}\right] \leq \mathbb{E}_{k+1}\left[\phi_{k+1}^*\right] \ .$$

Recalling that $\zeta_{k+1} = (1-\theta_k)\zeta_k + \theta_k\sigma_{1-\alpha}$ we see that choosing $\zeta_0 = \sigma_{1-\alpha}$ implies $\zeta_k = \sigma_{1-\alpha}$ for all k and therefore choosing $\theta_k = \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$ completes the proof that the chosen parameters produce a probabilistic estimate sequence. Furthermore, we see that this choice implies that $\eta_k = \left(1 - \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^k$. Therefore, by the definition of a probabilistic estimate sequence and the fact that $\tilde{S}_\alpha \geq 2S_\alpha$, equation (3.11) follows. □

3.3.2 Numerical Stability

In the previous section, we provided a simple proof how to achieve an ACDM with a convergence rate of $\tilde{O}(\sqrt{\frac{S_\alpha n}{\sigma_{1-\alpha}}})$. While sufficient for many purposes, the algorithm does not achieve the ideal dependence on initial error for certain regimes. For consistency with [43] we perform the change of variables

$$\forall k \geq 0 \ : \ \alpha_k \stackrel{\text{def}}{=} \frac{\theta_k\zeta_k}{\zeta_k + \theta_k\sigma_{1-\alpha}} \qquad \beta_k \stackrel{\text{def}}{=} \frac{(1-\theta_k)\zeta_k}{\zeta_{k+1}} \qquad \gamma_k \stackrel{\text{def}}{=} \frac{\tilde{S}_\alpha\theta_k}{\zeta_{k+1}}$$

and by better tuning θ_k we derive the following algorithm.

Accelerated Coordinate Descent Method
1. Define $\tilde{L}_i = \max(L_i, (S_\alpha/n)^{1/\alpha})$ and $\tilde{S}_\alpha = \sum \tilde{L}_i^\alpha$
2. Define $\vec{v}_0 = \vec{x}_0, a_0 = \frac{1}{2n}, b_0 = 2$
3. For $k \geq 0$ iterate:
3a. Find $\alpha_k, \beta_k, \gamma_k \geq \frac{1}{2n}$ such that $\gamma_k^2 - \frac{\gamma_k}{2n} = \left(1 - \frac{\gamma_k \sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)\frac{a_k^2}{b_k^2} = \beta_k \frac{a_k^2}{b_k^2} = \frac{\beta_k \gamma_k}{2n}\frac{1-\alpha_k}{\alpha_k}$.
3b. $\vec{y}_k = \alpha_k \vec{v}_k + (1-\alpha_k)\vec{x}_k$.
3c. Choose i_k according to $P_\alpha(i) = \tilde{L}_i^\alpha / \tilde{S}_\alpha$.
3d. $\vec{x}_{k+1} = \vec{y}_k - \frac{1}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k), v_{k+1} = \beta_k \vec{v}_k + (1-\beta_k)\vec{y}_k - \frac{\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k)$.
3e. $b_{k+1} = \frac{b_k}{\sqrt{\beta_k}}$ and $a_{k+1} = \gamma_k b_{k+1}$.

In Appendix A.2 and A.3, we give a proof of convergence of this method in the unit-cost RAM model by studying the following potential function [43] for suitable constants $a_k, b_k \in \mathbb{R}$,

$$a_k \left(\mathbb{E}\left[f(\vec{x}_k)\right] - f^*\right) + b_k \mathbb{E}\left[\|\vec{v}_k - x_*\|_{1-\alpha}^2\right] \quad .$$

The following theorem states that error in vector updates does not grow too fast and Lemma A.2.1 further states that errors in the coefficients α, β, and γ also does not grow too fast. Hence, $O(\log n)$ bits of precision is sufficient to implement ACDM with the following convergence guarantees. Below we state this result formally and we refer the reader to Appendix A.3 for the proof.

Theorem 3.3.4 (Numerical Stability of ACDM). *Suppose that in each iteration of ACDM step 3d has additive error ε, i.e. there exists $\vec{\varepsilon}_{1,k}, \vec{\varepsilon}_{2,k} \in \mathbb{R}^n$ with $\|\vec{\varepsilon}_{1,k}\|_{1-\alpha} \leq \varepsilon$ and $\|\vec{\varepsilon}_{2,k}\|_{1-\alpha} \leq \varepsilon$ such that step 3 is*

$$\vec{x}_{k+1} := \vec{y}_k - \frac{1}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k) + \vec{\varepsilon}_{1,k} \quad \text{and} \quad \vec{v}_{k+1} := \beta_k \vec{v}_k + (1-\beta_k)\vec{y}_k - \frac{\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k) + \vec{\varepsilon}_{2,k}.$$

If $\varepsilon < \frac{\sigma_{1-\alpha}^2}{8\tilde{S}_\alpha^2 n}$ and $k \geq \sqrt{\frac{2\tilde{S}_\alpha n}{\sigma_{1-\alpha}}}$, then we have the convergence guarantee

$$\sigma_{1-\alpha}\mathbb{E}\left[\|\vec{v}_{k+1} - \vec{x}^*\|_{1-\alpha}^2\right] + \left(\mathbb{E}\left[f(\vec{x}_{k+1})\right] - f^*\right) \leq \delta_k \tag{3.12}$$

where

$$\delta_k \stackrel{\text{def}}{=} 24kS_\alpha\varepsilon^2 + 32\sigma_{1-\alpha}\left(1 - \frac{1}{5}\sqrt{\frac{\sigma_{1-\alpha}}{S_\alpha n}}\right)^k \left(||x_0 - x^*||_{1-\alpha}^2 + \frac{1}{S_\alpha^2}\left(f(x_0) - f^*\right)\right)$$

and the additional convergence guarantee that $\frac{1}{k}\sum_{j=k}^{2k-1} \|\nabla f(\vec{y}_k)\|_{1-\alpha}^2 \leq 2000\frac{\tilde{S}_\alpha}{n}\delta_k$.

This theorem provides useful estimates for the error $f(\vec{x}_{k+1}) - f^*$, the residual $\|\vec{v}_{k+1} - \vec{x}^*\|_{1-\alpha}^2$, and the norm of gradient $\|\nabla f(\vec{y}_k)\|_{1-\alpha}^*$. Note how the estimate depends on the initial error $f(\vec{x}_0) - f^*$ mildly as compared to Theorem 3.3.3. We use this fact in creating an efficient SDD solver in Section 3.4.3.

3.3.3 Efficient Iteration

In both Nesterov's paper [43] and later work [45] the original ACDM proposed by Nesterov was not recommend since a naive implementation takes $O(n)$ time to update the vector $\vec{v}_k$, and therefore is likely slower than accelerated gradient descent. This implementation issue is a potential serious problem under the assumption that the oracle to compute gradient can only accept a single vector as input. However, if we make the mild assumption that we can compute $\nabla f(t\vec{x} + s\vec{y})$ for $s, t \in \mathbb{R}$ and $\vec{y}, \vec{y} \in \mathbb{R}$ in the same asymptotic runtime as it takes to compute $\nabla f(\vec{x})$ (i.e. we do not need to compute the sum explicitly), then we can implement ACDM without additional asymptotic computational costs per iteration as compared to the cost of the coordinate descent method performing an update on the given coordinate. In the following lemma, we prove this fact and show that the technique can be executed in the unit-cost RAM model without additional asymptotic costs. Note that the method we propose does introduce potential numerical problems which we argue are minimal.

Lemma 3.3.5 (Efficient Iterations). *For* $S_\alpha = O(\mathsf{poly}(n))$ *and* $\sigma_{1-\alpha} = \Omega(\mathsf{poly}(\frac{1}{n}))$ *each iteration of ACDM can be implemented in* $O(1)$ *time plus the time to make one oracle call of the form* $\vec{f}_{i_k}(t\vec{x} + s\vec{y})$ *for* $s, t \in \mathbb{R}$ *and* $\vec{x}, \vec{y} \in \mathbb{R}^n$, *using at most an additional* $O(\log n)$ *bits of precision so long as the number of iterations of ACDM is* $O\left(\sqrt{\frac{S_\alpha n}{\sigma_{1-\alpha}}}\log(n)\right)$.

Proof. By the specification of the ACDM algorithm, we have

$$\begin{aligned}\vec{y}_{k+1} &= \alpha_{k+1}\vec{v}_{k+1} + (1-\alpha_{k+1})\vec{x}_{k+1} \\ &= \alpha_{k+1}\left(\beta_k\vec{v}_k + (1-\beta_k)\vec{y}_k + \frac{\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y})\right) + (1-\alpha_{k+1})\left(\vec{y}_k - \frac{1}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k)\right) \\ &= \alpha_{k+1}\beta_k\vec{v}_k + (\alpha_{k+1}(1-\beta_k) + (1-\alpha_{k+1}))\,\vec{y}_k - \frac{1-\alpha_{k+1}-\alpha_{k+1}\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y})\end{aligned}$$

we see that each update step of ACDM can be rewritten as

$$\begin{pmatrix}\vec{v}_{k+1}^T \\ \vec{y}_{k+1}^T\end{pmatrix} := \mathbf{A}_k\begin{pmatrix}\vec{v}_k^T \\ \vec{y}_k^T\end{pmatrix} - \vec{s}_k$$

where

$$\mathbf{A}_k = \begin{pmatrix}\beta_k & 1-\beta_k \\ \alpha_{k+1}\beta_k & 1-\alpha_{k+1}\beta_k\end{pmatrix} \quad \text{and} \quad \vec{s}_k = \begin{pmatrix}\frac{\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k)^T \\ \frac{1-\alpha_{k+1}-\alpha_{k+1}\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k)^T\end{pmatrix} .$$

Therefore to implement ACDM in each iteration we can just maintain vectors $\vec{v}, \vec{y} \in \mathbb{R}^n$ and matrix $\mathbf{B}_k \in \mathbb{R}^{2\times 2}$ such that $\begin{pmatrix}\vec{v}_k^T \\ \vec{y}_k^T\end{pmatrix} = \mathbf{B}_k\begin{pmatrix}\vec{v}^T \\ \vec{y}^T\end{pmatrix}$. With this representation each update step is $\mathbf{B}_{k+1}\begin{pmatrix}\vec{v}^T \\ \vec{y}^T\end{pmatrix} := \mathbf{A}_k\mathbf{B}_k\begin{pmatrix}\vec{v}^T \\ \vec{y}^T\end{pmatrix} - \vec{s}_k$ and by our oracle assumption, we can implement the following equivalent update step

$$\mathbf{B}_{k+1} := \mathbf{A}_k\mathbf{B}_k \quad \text{and} \quad \begin{pmatrix}\vec{v}^T \\ \vec{y}^T\end{pmatrix} := \begin{pmatrix}\vec{v}^T \\ \vec{y}^T\end{pmatrix} - \mathbf{B}_{k+1}^{-1}\vec{s}_k$$

in time $O(1)$ plus the time needed for one oracle call.

To complete the lemma, we simply need to show that this scheme is numerically stable. Note that

$$\det(\mathbf{A}_k) = \beta_k(1-\alpha_{k+1}\beta_k) - \alpha_{k+1}\beta_k(1-\beta_k) = \beta_k(1-\alpha_{k+1}) .$$

Now by Lemma A.2.1, we know that $\alpha_k \leq 32 \max\left\{\frac{1}{k}, \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right\}$ for $k > 1$ and we know that $\beta_k \geq 1 - \frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$ for all k. Therefore, letting $\kappa \stackrel{\text{def}}{=} \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$ and recalling that we assumed that the total number of iterations is $O(\kappa^{-1}\log n)$ we get

$$\begin{aligned}\det(\mathbf{B}_k) &\geq (1-\kappa)^{O(\kappa^{-1}\log n)}(1-32\kappa)^{O(\kappa^{-1}\log n)}\prod_{k=2}^{\kappa^{-1}}\left(1-\frac{32}{k}\right) \\ &= \Omega\left(\mathsf{poly}\left(\frac{1}{n}\right)e^{-32\log\kappa}\right) = \Omega\left(\mathsf{poly}\left(\frac{1}{n}\right)\right).\end{aligned}$$

Hence, $O(\log n)$ bits of precision suffice to implement this method. □

3.4 Faster Linear System Solvers

In this section, we show how ACDM can be used to achieve asymptotic runtimes that outperform various state-of-the-art methods for solving linear systems in a variety of settings. In Section 3.4.1, we show how to use coordinate descent to outperform conjugate gradient under minor assumptions regarding the eigenvalues of the matrix under consideration, in Section 3.4.2, we show how to improve the convergence rate of randomized Kaczmarz type methods, and in Section 3.4.3, we show how to use the ideas to achieve the current fastest known numerically stable solver for symmetric diagonally dominant (SDD) systems of equations.

3.4.1 Comparison to Conjugate Gradient Method

Here we compare the performance of ACDM to conjugate gradient (CG) and show that under mild assumptions about the linear system being solved ACDM achieves a better asymptotic running time.

For symmetric positive definite (SPD) matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and vector $\vec{b} \in \mathbb{R}^n$, we solve the linear system of equations $\mathbf{A}\vec{x} = \vec{b}$ via the following equivalence unconstrained quadratic minimization problem:

$$\min_{\vec{x}\in\mathbb{R}^n} f(\vec{x}) \stackrel{\text{def}}{=} \frac{1}{2}\langle \mathbf{A}\vec{x}, \vec{x}\rangle - \langle \vec{b}, \vec{x}\rangle \ . \tag{3.13}$$

Let m denote the number of nonzero entries in $\mathbf{A}$, let nnz_i denote the number of nonzero entries in the ith row of $\mathbf{A}$. To make the analysis simpler, we assume that the nonzero entries is somewhat uniform, namely $\forall i \in [n]$ we have $\text{nnz}_i = O(\frac{m}{n})$. This assumption is trivially met for dense matrices, finite difference matrices, etc. Letting $0 \leq \lambda_1 \leq \ldots \leq \lambda_n$ denote the eigenvalues of $\mathbf{A}$, we get the following running time guarantee for ACDM.

Theorem 3.4.1 (ACDM on SPD Systems). *Assume* $\mathbf{A}$ *is a SPD matrix with the property that* $\text{nnz}_i = O(\frac{m}{n})$ *for all* $i \in [n]$. *Let the numerical rank* $r(\mathbf{A}) = \sum_{i=1}^{n} \lambda_i/\lambda_n$. *ACDM applied to* (3.13) *with* $\alpha = 1$[4] *produces an approximate solution in*

$$\tilde{O}\left(m\sqrt{\frac{r(\mathbf{A})}{n}}\sqrt{\frac{\lambda_n}{\lambda_1}}\log\frac{1}{\varepsilon}\right)$$

time with ε *error in* $\mathbf{A}$ *norm in expectation.*

Proof. The running time of ACDM with $\alpha = 1$ depends on σ_0 and S_1. From Example 3.1.3, the total component-wise Lipschitz constant S_1 is the trace of A, which is $\sum \lambda_i$ and the convexity parameter σ_0 is λ_1, therefore by Theorem 3.3.4 the convergence rate of ACDM is $\sqrt{\frac{\lambda_1}{n\sum_{i=1}^{n}\lambda_i}}$ as desired. Furthermore, the running time of each step depends on the running time of the oracle, i.e. computing $f_i(x) = (Ax)_i$, which by our assumption on nnz_i takes time $O\left(\frac{m}{n}\right)$. □

To compare with conjugate gradient, we know that one crude bound for the rate of convergence of conjugate gradient is $O\left(\sqrt{\frac{\lambda_1}{\lambda_n}}\right)$. Hence the total running time of CG to produce an epsilon approximate solution is $\tilde{O}\left(m\sqrt{\frac{\lambda_n}{\lambda_1}}\log\frac{1}{\varepsilon}\right)$. Therefore, with this bound ACDM is always faster or matches the running time since the numerical rank of $\mathbf{A}$ is always less than or equals to n. Thus, we see that when the numerical rank of $\mathbf{A}$ is $o(n)$, ACDM will likely [5] have a faster asymptotic running time.

To be more fair in our comparison to conjugate gradient, we note that in [54] tighter bound on the performance of CG was derived and they showed that in fact

[4] Here we only consider the $\alpha = 1$ case for easier comparison with conjugate gradient.

[5] The running time of CG may be asymptotic faster when the eigenvalues form clusters.

CG has a running time of $\tilde{O}\left(\sum \text{nnz}_i \left(\frac{\sum_{i=1}^n \lambda_i}{\lambda_1}\right)^{1/3}\right)$ implying that ACDM is faster than CG when $\sum_{i=1}^n \lambda_i \leq n^3 \lambda_1$ and it is usually satisfied. In the extreme cases that the condition is false, CG will need to run for $O(n)$ iterations at which point an exact answer could be computed.

3.4.2 Accelerating Randomized Kaczmarz

The Kaczmarz method [24] is an iterative algorithm to solve $\mathbf{A}\vec{x} = b$ for any full row rank matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. Letting $\vec{a}_i \in \mathbb{R}^n$ denote the i-th row of the matrix $\mathbf{A}$, we know that the solution of $\mathbf{A}\vec{x} = \vec{b}$ is the intersection of the hyperplanes $H_i \stackrel{\text{def}}{=} \{x : \langle \vec{a}_i, \vec{x}_k \rangle = \vec{b}_i\}$. The Kaczmarz method simply iteratively picks one of these hyperplanes and projects onto it by the following formula:

$$x_{k+1} := \text{proj}_{H_{i_k}}(\vec{x}_k) \quad \text{where} \quad \text{proj}_{H_{i_k}}(\vec{x}_k) + \frac{\vec{b}_{i_k} - \langle a_{i_k}, \vec{x}_k \rangle}{\|a_{i_k}\|^2} a_{i_k} \ .$$

There are many schemes that can be chosen to pick the hyperplane i_k, many of which are difficult to analyze and compare, but in a breakthrough result, Strohmer and Vershynin in 2008 analyzed the randomized schemes which sample the hyperplane with probability proportional to $\|a_i\|_2^2$. They proved the following

Theorem 3.4.2 (Strohmer and Vershynin [55]). *The Kaczmarz method samples row i with probability proportionally to $\|a_i\|_2^2$ at each iteration and yields the following*

$$\forall k \geq 0 \ : \ \mathbb{E}\left[\|\vec{x}_k - \vec{x}^*\|_2^2\right] \leq (1 - \kappa(\mathbf{A})^{-2})^k \|\vec{x}_0 - \vec{x}^*\|^2$$

where $\vec{x}^* \in \mathbb{R}^n$ is such that $\mathbf{A}\vec{x}^* = \vec{b}$, $\kappa(\mathbf{A}) \stackrel{\text{def}}{=} \|\mathbf{A}^{-1}\|_2 \cdot \|\mathbf{A}\|_F$ is the relative condition number *of $\mathbf{A}$, $\mathbf{A}^{-1}$ is the left inverse of $\mathbf{A}$, $\|\mathbf{A}^{-1}\|_2$ is the smallest non-zero spectral value of $\mathbf{A}$ and $\|\mathbf{A}\|_F^2 \stackrel{\text{def}}{=} \sum a_{ij}^2$ is the Frobenius norm of $\mathbf{A}$*

Here we show show to cast this algorithm as an instance of coordinate descent and obtain an improved convergence rate by applying ACDM. We remark that accelerated Kaczmarz will sample rows with a slightly different probability distribution. As long

as this does not increase the expected computational cost of an iteration, it will yield an algorithm with a faster asymptotic running time.

Theorem 3.4.3 (Accelerated Kaczmarz). *The ACDM method samples row i with probability proportionally to* $\max\left\{\|a_i\|_2^2, \frac{\|\mathbf{A}\|_F^2}{m}\right\}$ *and performs extra $O(1)$ work at each iteration. It yields the following*

$$\forall k \geq 0 \ : \ \mathbb{E}\|\vec{x}_k - \vec{x}^*\|_2^2 \leq 3\left(1 - \frac{\kappa(\mathbf{A})^{-1}}{2\sqrt{m}}\right)^k \|\vec{x}_0 - \vec{x}^*\|^2 .$$

Proof. To cast Strohmer and Vershynin's randomized Kaczmarz algorithm in the framework of coordinate descent, we consider minimizing the objective function of theorem theorem directly, i.e. $\min_{\vec{x}\in\mathbb{R}^n} \frac{1}{2}\|\vec{x} - \vec{x}^*\|_2^2$. Since $\mathbf{A}$ has full row rank, we write $\vec{x} = \mathbf{A}^T\vec{y}$ and consider the equivalent problem $\min_{\vec{y}\in\mathbb{R}^m} \frac{1}{2}\left\|\mathbf{A}^T\vec{y} - \vec{x}^*\right\|_2^2$. Expanding the objective function and using $\mathbf{A}\vec{x}^* = \vec{b}$, we get

$$\left\|\mathbf{A}^T\vec{y} - \vec{x}^*\right\|_2^2 = \left\|\mathbf{A}^T\vec{y}\right\|_2^2 - 2\langle\vec{b}, \vec{y}\rangle + \|\vec{x}^*\|_2^2 \ .$$

Therefore, we attempt solve the following equivalent problem using accelerated coordinate descent.

$$\min_{\vec{y}\in\mathbb{R}^m} f(\vec{y}) \quad \text{where} \quad f(\vec{y}) \stackrel{\text{def}}{=} \frac{1}{2}\left\|\mathbf{A}^T\vec{y}\right\|_2^2 - \langle\vec{b}, \vec{y}\rangle \ .$$

From Example 3.1.4, we know that the i-th direction component-wise Lipschitz constant is $L_i = \|a_i\|^2$ where a_i is the i-th row of $\mathbf{A}$ and we know that $\nabla f(\vec{y}) = \mathbf{A}\mathbf{A}^T\vec{y} - b$. Therefore, each step of the coordinate descent method consists of the following [6]

$$\vec{y}_{k+1} \ := \ \vec{y}_k - \frac{1}{L_{i_k}}\vec{f}_{i_k}(\vec{y}_k) \ = \ \vec{y}_k - \frac{1}{\|\vec{a}_{i_k}\|^2}\left(\mathbf{A}\mathbf{A}^T\vec{y}_k - \vec{b}\right)_{i_k} \ .$$

Recalling that we had performed the transformation $\vec{x} = \mathbf{A}^T\vec{y}$ we see that the corre-

[6] We ignore the thresholding here for illustration purpose.

sponding step in $\vec{x}$ is

$$\vec{x}_{k+1} = \vec{x}_k - \frac{1}{\|a_i\|^2}\left(\mathbf{A}\vec{x}_k - \vec{b}\right)_{i_k} = \vec{x}_k + \frac{\vec{b}_{i_k} - \langle \vec{a}_{i_k}, \vec{x}_k \rangle}{\|a_{i_k}\|_2^2}\vec{a}_{i_k}.$$

Therefore, the randomized coordinate descent method applied this way yields precisely the randomized Kaczmarz method of Strohmer and Vershynin.

However, to apply the ACDM method and provide complete theoretical guarantees we need to address the problem that f as we have constructed it is not strongly convex. This is clear by the fact that the null space of $\mathbf{A}^T$ may be non trivial.

To remedy this problem we let $Z \subseteq \mathbb{R}^m$ denote the null space of $\mathbf{A}^T$, i.e. $Z \stackrel{\text{def}}{=} \{\vec{x} \in \mathbb{R}^m \mid \mathbf{A}^T\vec{x} = 0\}$, and we define the semi-norm $\|\cdot\|_{Z^\perp}$ on $\mathbb{R}^m$ by $\|\vec{y}\|_{Z^\perp} \stackrel{\text{def}}{=} \inf_{\vec{z}\in Z} \|y + z\|$. Now it is not hard to see that f is strongly convex under this seminorm with convexity parameter $\sigma_{Z^\perp} = \|\mathbf{A}^{-1}\|_2^{-2}$. Furthermore, one can prove similarly to the proof of Lemma 3.3.2 and Theorem 3.3.3 that the algorithm in this theorem achieves the desired convergence rate.

To see this, we let $\mathbf{P} \in \mathbb{R}^{m\times m}$ denote the orthogonal projection onto the image of $\mathbf{A}$ we note that $\|\vec{y}\|_{Z^\perp} = \sqrt{\vec{y}^T\mathbf{P}\vec{y}}$. Therefore we can apply the general form of Lemma 3.3.2 proved in Appendix A.1 to derive a similar estimate sequence result for this norm. Unfortunately, the resulting estimate sequence requires working with $\mathbf{P}^\dagger f_{i_k}(\vec{y}_k)$. However, the proof of Theorem 3.3.3 still works using the same algorithm because of the facts that $\mathbf{P}^\dagger\nabla f(\vec{x}) = \nabla f(\vec{x})$,

$$\forall \vec{y} \in \mathbb{R}^n, \forall \vec{z} \in Z \ : \ f(\vec{y}+\vec{z}) = f(\vec{y}) \quad \text{and} \quad \nabla f(\vec{y}+\vec{z}) = \nabla(\vec{y}) \ ,$$

and $\|\vec{x}\|_2^2 \geq \|\vec{x}\|_{Z^\perp}^2$. This gives the same convergence guarantee with $\sigma_{1-\alpha}$ replaced with $\|\mathbf{A}^{-1}\|_2^{-2}$, $S_\alpha = \|\mathbf{A}\|_F^2$ and $\|\cdot\|_{1-\alpha} = \|\cdot\|_{Z^\perp}$. The desired convergence rate follows from the strong convexity of f.

To justify the cost of each iteration, note that although the iterations work on the $\vec{y}$ variable, we can apply $\mathbf{A}^T$ on all iterations and just do the operations on $\vec{x}$ variables which is more efficient. □

3.4.3 Faster SDD Solvers in the Unit-cost RAM Model

Here we show how to cast the `SimpleSolver` algorithm of Chapter 2 as an instance of coordinate descent, and by applying ACDM we obtain an algorithm for solving SDD systems that has an asymptotic running time of $O\left(m \log^{3/2} n \sqrt{\log \log n} \log(\varepsilon^{-1} \log n)\right)$ in the unit-cost RAM model. More precisely we prove the following.

Theorem 3.4.4. *By applying ACDM to* `SimpleSolver`*, we can produce* $\vec{v} \in \mathbb{R}^V$ *such that* $\left\|\vec{v} - \mathbf{L}^\dagger \vec{\chi}\right\|_{\mathbf{L}} \leq \varepsilon \left\|\mathbf{L}^\dagger \vec{\chi}\right\|_{\mathbf{L}}$ *in time* $O\left(m \log^{3/2} n \sqrt{\log \log n} \log(\varepsilon^{-1} \log n)\right)$.[7]

Proof. Recall that instead of computing $\vec{v}$ directly, we can instead focus on the following electric flow problem [8]

$$\min_{\mathbf{B}^T \vec{f} = \vec{\chi}} \frac{1}{2} \xi(\vec{f}) \quad \text{where} \quad \xi(\vec{f}) \stackrel{\text{def}}{=} \left\|\vec{f}\right\|_{\mathbf{R}}^2 \stackrel{\text{def}}{=} \sqrt{\sum_{e \in E} r_e \vec{f}(e)^2}$$

Now to turn this into an unconstrained minimization problem we first let $\vec{f}_0$ be feasible flow, i.e. a vector such that $\mathbf{B}^T \vec{f}_0 = \vec{\chi}$. With this, the problem can be simplified to

$$\min_{\mathbf{B}^T \vec{c} = 0} \frac{1}{2} \left\|\vec{f}_0 + \vec{c}\right\|_{\mathbf{R}}^2.$$

However, from Section 2.2 we know that $\{\vec{c} \in \mathbb{R}^E \mid \mathbf{B}^T \vec{c} = 0\}$ is simply the set of circulations of the graph, i.e. *cycle space*, and for spanning tree T the set of tree cycles for the off tree edges, i.e. $\{\vec{c}_e \mid e \in E \setminus T\}$ form a basis. Therefore, using this basis we see that we can simplify the problem further to

$$\min_{\vec{y} \in \mathbb{R}^{E \setminus T}} \frac{1}{2} \left\|\vec{f}_0 + \mathbf{C}\vec{y}\right\|_{\mathbf{R}}^2 \quad \text{where } \mathbf{C} \stackrel{\text{def}}{=} [c_{e_1} c_{e_2} \cdots] \in \mathbb{R}^{E \times E \setminus T}.$$

Now, for every *off-tree edge*, i.e. $e \in E \setminus T$, there is only one cycle c_e that passes through it. So, if we let $\mathbf{R}_{E \setminus T} \in \mathbb{R}^{E \setminus T \times E \setminus T}$ be the diagonal matrix for the resistances

[7] Although not the focus of this section, we remark that this procedure produces an ε-approximate electric flow in time $O\left(m \log^{3/2} n \sqrt{\log \log n} \log(\varepsilon^{-1})\right)$ in the unit-cost RAM model. Furthermore, by the definition of ACDM, we see that the operators produced by this algorithm are linear and by similar analysis as in [30] the algorithm can be used to produce a linear approximation to $\mathbf{L}^\dagger$.

[8] To make the analysis cleaner we scale the objective function by $\frac{1}{2}$. This has no substantial effect on our ability to compute approximate voltages from approximate solutions.

of the off tree edges we have that for any $\vec{y} \in E \setminus T$

$$\vec{y}^T \mathbf{C}^T \mathbf{R} \mathbf{C} \vec{y} \geq \vec{y}^T \mathbf{C}^T \mathbf{R}_{E \setminus T} \mathbf{C} \vec{y} = \sum_{e \in E \setminus T} r_e \vec{y}(e)^2 \geq \left(\min_{e \in E \setminus T} r_e \right) \|\vec{y}\|_2^2 .$$

Therefore, the convexity parameter of $\|\vec{z}_0 + \mathbf{C}\vec{y}\|_{\mathbf{R}}^2$ is at least $\min_{e \in E \setminus T} r_e$. However, this could be wasteful if the resistances vary, so rescalethe space, $\tilde{y} = \mathbf{R}^{1/2}\vec{y}$, to get the following problem:[9]

$$\min_{\tilde{y} \in \mathbb{R}^{E \setminus T}} g(\tilde{y}) \quad \text{where} \quad g(\tilde{y}) = \frac{1}{2} \left\| \vec{f}_0 + \mathbf{C}\mathbf{R}^{-1/2}\tilde{y} \right\|_{\mathbf{R}}^2 .$$

By the reasoning above, the convexity parameter of g with respect to the Euclidian norm is 1 and we bound the e-th direction component-wise Lipschitz constant by

$$\forall e \in E \setminus T \ : \ L_e = \mathbb{1}_e^T \mathbf{R}^{-1/2} \mathbf{C}^T \mathbf{R} \mathbf{C} \mathbf{R}^{-1/2} \mathbb{1}_e = \frac{\vec{c}_e^T \mathbf{R} \vec{c}_e}{r_e} = \frac{R_e}{r_e} = \mathrm{st}\,(e) + 1$$

Therefore we see that

$$\frac{S_1}{\sigma_0} = S_1 = \sum_{e \in E \setminus T} \frac{R_e}{r_e} = \tau(T) = O(m + \mathrm{st}\,(T))$$

Thereby justifying our naming of τ as "tree condition number."

Finally, applying the coordinate descent method to g and letting $e_k \in E \setminus T$ denote the off-tree edge i.e., coordinate, picked in iteration k, we get

$$\begin{aligned}
\tilde{y}_{k+1} &:= \tilde{y}_k - \frac{1}{L_{e_k}} \vec{g}_{e_k}(\tilde{y}_k) && \text{(Definition of Update Step)} \\
&= \tilde{y}_k - \frac{1}{L_{e_k}} \left(\mathbf{R}^{-1/2} \mathbf{C}^T \mathbf{R} (\vec{f}_0 + \mathbf{C}\mathbf{R}^{-1/2}\tilde{y}) \right)_{e_k} \cdot \mathbb{1}_{e_k} && \text{(Computation of } \nabla g(\tilde{y})) \\
&= \tilde{y}_k - \frac{1}{L_{e_k} r_{e_k}^{1/2}} \sum_e \vec{c}_{e_k}(e) r_{e_k} (\vec{f}_0 + \mathbf{C}\mathbf{R}^{-1/2}\tilde{y})(e) \cdot \mathbb{1}_{e_k} && \text{(Definition of } \mathbf{C} \text{ and } \mathbf{R}) \\
&= \tilde{y}_k - \frac{1}{L_{e_k} r_{e_k}^{1/2}} \sum_{e \in \vec{c}_{e_k}} r_e (\vec{f}_0 + \mathbf{C}\mathbf{R}^{-1/2}\tilde{y})(e) \cdot \mathbb{1}_{e_k} && \text{(Definition of } C_{e_k})
\end{aligned}$$

[9]To avoid confusion between a flow vector and the objective function in just this section we use g instead of f to denote the objective function.

Recalling $\vec{y} = \mathbf{R}^{-1/2}\tilde{y}$ and the derivation of L_e, we can write this equivalently as

$$\vec{y}_{k+1} := \vec{y}_k - \left[\frac{1}{R_e}\sum_{e\in\vec{c}_{e_k}} r_e(\vec{f}_0 + \mathbf{C}\vec{y})(e)\right]\cdot \mathbb{1}_{e_k}.$$

Noting that the $\vec{f}$ with $\mathbf{B}^T\vec{f} = \vec{\chi}$ corresponding to $\vec{y}$ is $\vec{f} = \vec{f}_0 + \mathbf{C}\vec{y}$, we write the update equivalently as

$$\vec{f}_{k+1} := \vec{f}_k - \left[\frac{1}{R_e}\sum_{e\in\vec{c}_{e_k}} r_e\vec{f}_k(e)\right]\cdot \vec{c}_{e_k}$$

which is precisely the cycle update of `SimpleSolver`. Thus we see that `SimpleSolver` is an instantiation of coordinate descent where the data structure is used to make sure that calls to $\vec{g}_{e_k}$ and updates to $\tilde{y}_{e_k}$ can be implemented in $O(\log n)$. Therefore, by applying ACDM we can obtain a faster algorithm. Note that to apply ACDM efficiently computations of $\vec{g}_{e_k}$ need to be performed on the sum of two vectors without explicitly summing. However, since here ∇g is linear, we can just call the oracle on the two vectors separately and use two separate data structure for updating coordinates.

While the above insight suffices to compute *electric flows* a little more work needs to be done to compute the desired voltages. However, by Lemma 2.6.2 we know that it suffices to show that $\|\nabla g(\tilde{y})\|_2^2 \leq \varepsilon f^*$. Note that

$$\|\nabla g(\tilde{y})\|_2^2 = \left\|\mathbf{R}^{-1/2}\mathbf{C}^T\mathbf{R}(\vec{f}_0 + \mathbf{C}\mathbf{R}^{-1/2}\tilde{y})\right\|_2^2 = \sum_{e\in E\setminus T}\frac{1}{r_e}\left(\sum_{e'\in\vec{c}_e}\vec{z}(e')r_{e'}\right)^2$$

and therefore if we choose $\tilde{y}_0 = \vec{0}$. Then, we see that

$$\|\tilde{y}_0 - \tilde{y}^*\|_2^2 = \|\tilde{y}^*\|_2^2 = \left\|\mathbf{R}_{E\setminus T}^{1/2}\vec{y}^*\right\|_2^2 \leq 2f^*,$$

and using Lemma 2.6.1 we have that

$$\frac{1}{S_\alpha^2}(g(\tilde{y}_0) - g^*) \leq \frac{g^*}{S_\alpha} \leq g^*$$

Therefore so long as we choose our spanning tree using Theorem 2.2.13 we have $\frac{S_1}{|E\setminus T|} = O(\log n \log\log n)$ and after $k = O(m\sqrt{\log n \log\log n}\log\frac{\log n}{\varepsilon})$ iterations of ACDM, by Theorem 3.3.4, we have that

$$\frac{1}{k}\sum_{j=k}^{2k-1} \|\nabla g(\vec{y}_k)\|_2^2 \leq \varepsilon f^*.$$

Therefore, if we stop ACDM at random iteration between k and $2k-1$, we have that $\mathbb{E}\,\|\nabla g(\vec{y}_k)\|_2^2 \leq \varepsilon g^*$ as desired. Therefore, in time $O\big(m\log^{3/2} n\sqrt{\log\log n}\log(\varepsilon^{-1}\log n)\big)$ we can compute the desired $\vec{v}$. □

Appendix A

Additional ACDM Proofs

Here we present additional proofs needed to complete the analysis Chapter 3.

A.1 Probabilistic Estimate Sequence Form

Here we prove a stronger variant of Lemma 3.3.2 that is useful for Section 3.4.2. Throughout this section we let $\mathbf{A} \in \mathbb{R}^{n \times n}$ denote a symmetric positive semidefinite matrix, we let $\|\vec{x}\|_{\mathbf{A}} \stackrel{\text{def}}{=} \sqrt{\vec{x}^T \mathbf{A} \vec{x}}$ be the induced norm, we let $\langle \vec{x}, \vec{y} \rangle_{\mathbf{A}} \stackrel{\text{def}}{=} \vec{x}^T \mathbf{A} \vec{y}$ be the inner product which induces this norm, and we let $\mathbf{A}^\dagger$ denote the Moore-penrose pseudoinverse of $\mathbf{A}$. Taking $\mathbf{A}$ to be the diagonal matrix with $\mathbf{A}_{ii} = L_{1-\alpha}$ and applying the following lemma yields Lemma 3.3.2 as an immediate corollary.

Lemma A.1.1 (General (Probabilistic) Estimate Sequence Construction). *Let $\mathbf{A}$ be a positive semidefine matrix such that f is strongly convex with respect to the norm $\|\cdot\|_{\mathbf{A}}$ with convexity parameter $\sigma_{\mathbf{A}}$. Furthermore let $\phi_0(\vec{x})$, $\{\vec{y}_k, \theta_k, i_k\}_{k=0}^{\infty}$ be such that*

- *$\phi_0 : \mathbb{R}^n \to \mathbb{R}$ is an arbitrary function*
- *Each $\vec{y}_k \in \mathbb{R}^n$,*
- *Each $\theta_k \in (0,1)$ and $\sum_{k=0}^{\infty} \theta_k = \infty$,*
- *Each $i_k \in [n]$ is chosen randomly with probability p_{i_k} .*

Then the pair of sequences $\{\phi_k(\vec{x}), \eta_k\}_{k=0}^{\infty}$ defined by

- *$\eta_0 = 1$ and $\eta_{k+1} = (1 - \theta_k)\eta_k$,*

- $\phi_{k+1}(\vec{x}) := (1-\theta_k)\phi_k(\vec{x}), +\theta_k\left[f(\vec{y}_k) + \frac{1}{p_{i_k}}\langle \vec{f}_{i_k}(\vec{y}_k), \vec{x}-\vec{y}_k\rangle + \frac{\sigma_{\mathbf{A}}}{2}\|\vec{x}-\vec{y}_k\|_{\mathbf{A}}^2\right]$

satisfies condition (3.7). Furthermore, if $\phi_0(\vec{x}) = \phi_0^* + \frac{\zeta_0}{2}\|\vec{x}-\vec{v}_0\|_{\mathbf{A}}^2$, *then this process produces a sequence of quadratic functions of the form* $\phi_k(\vec{x}) = \phi_k^* + \frac{\zeta_k}{2}\|\vec{x}-\vec{v}_k\|_{\mathbf{A}}^2$ *where*

$$\begin{aligned}
\zeta_{k+1} &= (1-\theta_k)\zeta_k + \theta_k\sigma_{\mathbf{A}}, \\
\vec{v}_{k+1} &= \frac{1}{\zeta_{k+1}}\left[(1-\theta_k)\zeta_k\vec{v}_k + \theta_k\sigma_{\mathbf{A}}\vec{y}_k - \frac{\theta_k}{p_{i_k}}\mathbf{A}^\dagger\vec{f}_{i_k}(\vec{y}_k)\right], \\
\phi_{k+1}^* &= (1-\theta_k)\phi_k^* + \theta_k f(\vec{y}_k) - \frac{\theta_k^2}{2\zeta_{k+1}p_{i_k}^2}\left\|\vec{f}_{i_k}(\vec{y}_k)\right\|_{\mathbf{A}^\dagger}^2 \\
&\quad + \frac{\theta_k(1-\theta_k)\zeta_k}{\zeta_{k+1}}\left(\frac{\sigma_{\mathbf{A}}}{2}\|\vec{y}_k-\vec{v}_k\|_{\mathbf{A}}^2 + \frac{1}{p_{i_k}}\langle\mathbf{A}^\dagger\vec{f}_{i_k}(\vec{y}_k), \vec{v}_k-\vec{y}_k\rangle_{\mathbf{A}}\right) .
\end{aligned}$$

Proof. [1] First prove by induction on k that $\mathbb{E}[\phi_k(\vec{x})] \leq (1-\eta_k)f(\vec{x}) + \eta_k\mathbb{E}[\phi_0(\vec{x})]$. The base case, $k=0$, follows trivially from $\eta_0 = 1$. Assuming it holds for some k, explicitly writing out the expectation over i_k yields for all $\vec{x}\in\mathbb{R}^n$, $\mathbb{E}[\phi_{k+1}(\vec{x})]$ is given by the following

$$\mathbb{E}\left[\sum_{i_k} p_{i_k}\left((1-\theta_k)\phi_k(\vec{x}) + \theta_k\left[f(\vec{y}_k) + p_{i_k}^{-1}\langle\vec{f}_{i_k}(\vec{y}_k), \vec{x}-\vec{y}_k\rangle + \frac{\sigma_{\mathbf{A}}}{2}\|\vec{x}-\vec{y}_k\|_{\mathbf{A}}^2\right]\right)\right].$$

Applying the inductive hypothesis, we get that $\mathbb{E}[\phi_{k+1}(\vec{x})]$ is equal to the following.

$$(1-\theta_k)\left[(1-\eta_k)f(\vec{x}) + \eta_k\mathbb{E}\left[\phi_0(\vec{x})\right]\right] + \theta_k\mathbb{E}\left[f(\vec{y}_k) + \langle\nabla f(\vec{y}_k), \vec{x}-\vec{y}_k\rangle + \frac{\sigma_{\mathbf{A}}}{2}\|\vec{x}-\vec{y}_k\|_{\mathbf{A}}^2\right].$$

By strong convexity and the definition of η_{k+1}, we then have that

$$\begin{aligned}
\mathbb{E}[\phi_{k+1}(\vec{x})] &\leq (1-\theta_k)(1-\eta_k)f(\vec{x}) + \theta_k f(\vec{x}) + (1-\theta_k)\eta_k\mathbb{E}\left[\phi_0(\vec{x})\right] \\
&= (1-\eta_{k+1})f(\vec{x}) + \eta_{k+1}\mathbb{E}\left[\phi_0(\vec{x})\right].
\end{aligned}$$

Next, we prove the form of ϕ_k again by induction on k. Suppose that $\phi_k = \phi_k^* + \frac{\zeta_k}{2}\|\vec{x}-\vec{v}_k\|_{\mathbf{A}}^2$. Now, we prove the form of ϕ_{k+1}. By the update rule and the inductive

[1] Note that our proof was heavily influenced by the proof in [41] for estimate sequences.

hypothesis, we see that

$$\forall \vec{x} \in \mathbb{R}^n \ : \ \nabla^2 \phi_{k+1}(\vec{x}) = (1-\theta_k)\nabla^2 \phi_k(\vec{x}) + \theta_k \sigma_{\mathbf{A}} \mathbf{A} = \left[(1-\theta_k)\zeta_k + \theta_k \sigma_{\mathbf{A}}\right] \mathbf{A} \ .$$

Consequently, $\phi_{k+1}(\vec{x}) = \phi^*_{k+1} + \frac{\zeta_{k+1}}{2} \left\| \vec{x} - \vec{v}_{k+1} \right\|^2_{\mathbf{A}}$ for ζ_{k+1} as desired and ϕ^*_{k+1} and $\vec{v}_{k+1}$ yet to be determined.

To compute $\vec{v}_{k+1}$, we note that $\nabla \phi_k(\vec{x}) = \zeta_k \mathbf{A}(\vec{x} - \vec{v}_k)$. Therefore by update rule

$$\begin{aligned} \nabla \phi_{k+1}(\vec{x}) &= (1-\theta_k)\zeta_k \mathbf{A}(\vec{x} - \vec{v}_k) + \frac{\theta_k}{p_{i_k}} \vec{f}_{i_k} + \theta_k \sigma_{\mathbf{A}} \mathbf{A}(\vec{x} - \vec{y}_k) \\ &= \zeta_{k+1} \mathbf{A} \vec{x} - (1-\theta_k)\zeta_k \mathbf{A} \vec{v}_k + \frac{\theta_k}{p_{i_k}} \vec{f}_{i_k} - \theta_k \sigma_{\mathbf{A}} \mathbf{A} \vec{y}_k \end{aligned}$$

Therefore, we have that $\vec{v}_{k+1}$ must satisfy

$$\zeta_{k+1} \mathbf{A} \vec{x} - \zeta_{k+1} \mathbf{A} \vec{v}_{k+1} = \zeta_{k+1} \mathbf{A} \vec{x} - (1-\theta_k)\zeta_k \mathbf{A} \vec{v}_k + \frac{\theta_k}{p_{i_k}} \vec{f}_{i_k}(\vec{y}_k) - \theta \sigma_{\mathbf{A}} \mathbf{A} \vec{y}_k.$$

So, applying $\mathbf{A}^\dagger$ to each side yields the desired formula for $\vec{v}_{k+1}$.

Finally, to compute ϕ^*_{k+1}, we note that, by the form of $\phi_{k+1}(\vec{x})$ and the update rule for creating $\phi_{k+1}(\vec{x})$, looking at $\phi_{k+1}(\vec{y}_k)$ yields

$$\begin{aligned} \phi^*_{k+1} + \frac{\zeta_{k+1}}{2} \left\| \vec{y}_k - \vec{v}_{k+1} \right\|^2_{\mathbf{A}} &= (1-\theta_k)\phi_k(\vec{y}_k) + \theta_k f(\vec{y}_k) \\ &= (1-\theta_k) \left[\phi^*_k + \frac{\zeta_k}{2} \left\| \vec{y}_k - \vec{v}_k \right\|^2_{\mathbf{A}} \right] + \theta_k f(\vec{y}_k). \end{aligned}$$

Now, by value of v_{k+1}, we see that

$$\begin{aligned} \zeta^2_{k+1} \left\| \vec{v}_{k+1} - \vec{y}_k \right\|^2_{\mathbf{A}} &= \left\| (1-\theta_k)\zeta_k \vec{v}_k + (\theta_k \sigma_{\mathbf{A}} - \zeta_{k+1}) \vec{y}_k - \frac{\theta_k}{p_{i_k}} \mathbf{A}^\dagger \vec{f}_{i_k}(\vec{y}_k) \right\|^2_{\mathbf{A}} \\ &= \left\| (1-\theta_k)\zeta_k (\vec{v}_k - \vec{y}_k) - \frac{\theta_k}{p_{i_k}} \mathbf{A}^\dagger \vec{f}_{i_k}(\vec{y}_k) \right\|^2_{\mathbf{A}} \\ &= (1-\theta_k)^2 \zeta^2_k \left\| \vec{v}_k - \vec{y}_k \right\|^2_{\mathbf{A}} - \frac{2(1-\theta_k)\theta_k \zeta_k}{p_{i_k}} \langle \mathbf{A}^\dagger \vec{f}_{i_k}(\vec{y}_k), \vec{v}_k - \vec{y}_k \rangle_{\mathbf{A}} \\ &\quad + \frac{\theta^2_k}{p^2_{i_k}} \left\| \mathbf{A}^\dagger \vec{f}_{i_k}(\vec{y}_k) \right\|^2_{\mathbf{A}}. \end{aligned}$$

Combining these two formulas and noting that

$$(1-\theta_k)\frac{\zeta_k}{2}-\frac{\zeta_{k+1}}{2}\cdot\frac{1}{\zeta_{k+1}^2}(1-\theta_k)^2\zeta_k^2=\frac{(1-\theta_k)\zeta_k}{2\zeta_{k+1}}\left[\zeta_{k+1}-(1-\theta_k)\zeta_k\right]=\frac{\theta_k(1-\theta_k)\zeta_k}{\zeta_{k+1}}\cdot\frac{\sigma_{\mathbf{A}}}{2}$$

yields the desired form of ϕ_{k+1}^* and completes the proof. □

A.2 Bounding ACDM Coefficients

In the following lemma we prove bounds on $\alpha_k, \beta_k, \gamma_k, a_k$ and b_k required for Theorem 3.3.4 and Lemma 3.3.5.

Lemma A.2.1 (ACDM Coefficients). *In ACDM γ_k is increasing to $\sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}$ and β_k is decreasing to $1-\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$. Furthermore, for all $k \geq 1$ we have $\alpha_k \leq 32\max\left(\frac{1}{k}, \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)$ and for all $k \geq 0$ we have*

$$a_k \geq \sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}\left(\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}-\left(1-\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}\right), \tag{A.1}$$

$$b_k \geq \left(\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}+\left(1-\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}\right). \tag{A.2}$$

Proof. We begin by estimating γ_k. Using that $\gamma_k = a_{k+1}/b_{k+1}$ and defining $\gamma_{-1} \stackrel{\text{def}}{=} a_0/b_0 = 1/(4n)$ we know that $\gamma_{k+1}^2 - \frac{\gamma_{k+1}}{2n} = \left(1-\frac{\gamma_{k+1}\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)\gamma_k^2$ and therefore $\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}} - \gamma_{k+1}^2 = \left(1-\frac{\gamma_{k+1}\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)\left(\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}-\gamma_k^2\right)$ for all $k \geq 0$. Consequently, γ_k is increasing to $\sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}$ and since $\beta_k = 1-\frac{\gamma_k\sigma_{1-\alpha}}{\tilde{S}_\alpha}$, we have that β_k is decreasing to $1-\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$. Furthermore since $2\tilde{S}_\alpha n \geq \sigma_{1-\alpha}$ we have that all $\beta_k \in (0,1)$, $b_k \leq b_{k+1}$, and $a_k \leq a_{k+1}$.

Using these insights we can now estimate the growth of coefficients a_k and b_k. As in [43] the growth can be estimated by a recursive relation between a_k and b_k. For b_k, our definitions imply

$$b_k^2 = \beta_k b_{k+1}^2 = \left(1-\frac{\sigma_{1-\alpha}}{\tilde{S}_\alpha}\gamma_k\right)b_{k+1}^2 = \left(1-\frac{\sigma_{1-\alpha}}{\tilde{S}_\alpha}\frac{a_{k+1}}{b_{k+1}}\right)b_{k+1}^2 .$$

Therefore since $b_k \leq b_{k+1}$ we have $\frac{\sigma_{1-\alpha}}{\tilde{S}_\alpha}a_{k+1}b_{k+1} \leq b_{k+1}^2 - b_k^2 \leq 2b_{k+1}(b_{k+1}-b_k)$ and

consequently

$$b_{k+1} \geq b_k + \frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha} a_{k+1} \ . \tag{A.3}$$

To bound a_k, the definitions imply $\frac{a_{k+1}^2}{b_{k+1}^2} - \frac{a_{k+1}}{2nb_{k+1}} = \gamma_k^2 - \frac{\gamma_k}{2n} = \beta_k \frac{a_k^2}{b_k^2} = \frac{a_k^2}{b_{k+1}^2}$. Therefore since $a_k \leq a_{k+1}$ we have $\frac{1}{2n} a_{k+1} b_{k+1} = a_{k+1}^2 - a_k^2 \leq 2a_{k+1}(a_{k+1} - a_k)$ and consequently

$$a_{k+1} \geq a_k + \frac{1}{4n} b_{k+1} \geq a_k + \frac{1}{4n} b_k \ . \tag{A.4}$$

Using (A.4) and (A.3) we can prove the growth rates (A.1) and (A.2) by induction.

All that remains is to prove the upper bound on α_k. Note that $\frac{1-\alpha_k}{\alpha_k} = \frac{2n\gamma_k - 1}{\beta_k}$ and hence $\alpha_k \sim \frac{\beta_k}{2n\gamma_k - 1} \sim \frac{1}{2n\gamma_k}$. Therefore, we wish to find a lower bound of γ_k, which in turn can be found by a upper bound of β_k. Recalling that $b_{k+1} = \frac{b_k}{\sqrt{\beta_k}}$ we see that $b_k^2 = \frac{b_0^2}{\beta_0 \cdots \beta_{k-1}} \leq \frac{b_0^2}{\beta_{k-1}^k}$. Therefore, by our lower bound on b_k and the fact that $b_0 = 2$, we see that when $k \leq 2\sqrt{\frac{2\tilde{S}_\alpha n}{\sigma_{1-\alpha}}}$, we have

$$\begin{aligned}
\beta_k &\leq 2^{2/k} \left(\left(1 + \frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}} \right)^k + \left(1 - \frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}} \right)^k \right)^{-2/k} \\
&\leq 2^{2/k} \left(2 + \frac{\sigma_{1-\alpha}}{8\tilde{S}_\alpha n} k(k-1) \right)^{-2/k} = \left(1 + \frac{\sigma_{1-\alpha}}{16\tilde{S}_\alpha n} k(k-1) \right)^{-2/k} \\
&\leq \exp\left(-\frac{\sigma_{1-\alpha}}{8\tilde{S}_\alpha n}(k-1) \right) \leq 1 - \frac{\sigma_{1-\alpha}}{16\tilde{S}_\alpha n}(k-1).
\end{aligned}$$

Using $\beta_k = 1 - \frac{\gamma_k \sigma_{1-\alpha}}{\tilde{S}_\alpha}$, we get $\gamma_k \geq \frac{1}{16} \min\left(\frac{k-1}{n}, \sqrt{\frac{2\tilde{S}_\alpha}{\sigma_{1-\alpha} n}} \right)$. Therefore since $\frac{1}{\alpha_k} - 1 = \frac{1-\alpha_k}{\alpha_k} = \frac{n\gamma_k - 1}{\beta_k} \geq n\gamma_k - 1$ for all k we have that $\alpha_k \leq \frac{1}{n\gamma_k} \leq 16 \max\left\{ \frac{1}{k-1}, \sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}} \right\}$ and the result follows immediately. □

A.3 Numerical Stability of ACDM

Proof of Theorem 3.3.4. Following the structure in [43] we begin by defining $T_i(\vec{x}) \stackrel{\text{def}}{=} \vec{x} - \frac{1}{L_i}\vec{f}_i(\vec{x})$ for all $\vec{x} \in \mathbb{R}^n$ and $r_k^2 \stackrel{\text{def}}{=} \|\vec{v}_k - \vec{x}^*\|_{1-\alpha}^2$ for all $k \geq 0$. Expanding $\vec{v}_k$ yields

$$\begin{aligned}
r_{k+1}^2 &= \|\vec{\varepsilon}_{2,k} + \beta_k\vec{v}_k + (1-\beta_k)\vec{y}_k - \frac{\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k) - \vec{x}^*\|_{1-\alpha}^2 \\
&\leq \left(1+\frac{1}{t}\right)\|\vec{\varepsilon}_{2,k}\|_{1-\alpha}^2 + (1+t)\|\beta_k\vec{v}_k + (1-\beta_k)\vec{y}_k - \frac{\gamma_k}{\tilde{L}_{i_k}}\vec{f}_{i_k}(\vec{y}_k) - \vec{x}^*\|_{1-\alpha}^2 \\
&\leq \left(1+\frac{1}{t}\right)\varepsilon^2 + (1+t)\|\beta_k\vec{v}_k + (1-\beta_k)\vec{y}_k - \vec{x}^*\|_{1-\alpha}^2 + (1+t)\frac{\gamma_k^2}{\tilde{L}_{i_k}^{1+\alpha}}f_{i_k}(\vec{y}_k)^2 \\
&\quad +2\frac{\gamma_k}{\tilde{L}_{i_k}^{\alpha}}(1+t)\left\langle \vec{f}_{i_k}(\vec{y}_k), \vec{x}^* - \beta_k\vec{v}_k - (1-\beta_k)\vec{y}_k\right\rangle \\
&\leq \left(1+\frac{1}{t}\right)\varepsilon^2 + (1+t)\|\beta_k\vec{v}_k + (1-\beta_k)\vec{y}_k - \vec{x}^*\|_{1-\alpha}^2 \\
&\quad +2\frac{\gamma_k^2}{\tilde{L}_{i_k}^{\alpha}}(1+t)\left(f(\vec{y}_k) - f(T_{i_k}(\vec{y}_k))\right) \\
&\quad +2\frac{\gamma_k}{\tilde{L}_{i_k}^{\alpha}}(1+t)\left\langle \vec{f}_{i_k}(\vec{y}_k), \vec{x}^* - \vec{y}_k + \frac{\beta_k(1-\alpha_k)}{\alpha_k}(\vec{x}_k - \vec{y}_k)\right\rangle, \qquad \text{(A.5)}
\end{aligned}$$

where t is yet to be determined.

Using standard properties of f, we bound the error induced by $\vec{\varepsilon}_{1,k}$ as follows:

$$\begin{aligned}
f(T_{i_k}(\vec{y}_k)) &= f(\vec{x}_{k+1} - \vec{\varepsilon}_{1,k}) && \text{(Assumption of Step 3d)} \\
&\geq f(\vec{x}_{k+1}) - \langle\vec{\varepsilon}_{1,k}, \nabla f(\vec{x}_{k+1})\rangle && \text{(Convexity of } f\text{)} \\
&= f(\vec{x}_{k+1}) - \langle\vec{\varepsilon}_{1,k}, \nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}^*)\rangle && (\nabla f(\vec{x}^*) = \vec{0}) \\
&\geq f(\vec{x}_{k+1}) - \|\vec{\varepsilon}_{1,k}\|_{1-\alpha}\|\nabla f(\vec{x}_{k+1}) - \nabla f(\vec{x}^*)\|_{1-\alpha}^* && \text{(Cauchy Schwarz)} \\
&\geq f(\vec{x}_{k+1}) - \varepsilon\tilde{S}_\alpha\|\vec{x}_{k+1} - \vec{x}^*\|_{1-\alpha} && \text{(Lemma 2 of [43])} \\
&\geq f(\vec{x}_{k+1}) - \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\left(f(\vec{x}_{k+1}) - f^*\right). && \text{(A.6)}
\end{aligned}$$

Taking the expectation of both sides of (A.5) in i_k and using (A.6) and $\tilde{L}_{i_k}^{\alpha} \geq \frac{\tilde{S}_\alpha}{2n}$ yield

$$\begin{aligned}
\mathbb{E}_{i_k}\left[(r_{k+1}^2)\right] &\leq \left(1+\frac{1}{t}\right)\varepsilon^2 + (1+t)\beta_k r_k^2 + (1+t)(1-\beta_k)||\vec{y}_k - \vec{x}^*||_{1-\alpha}^2 \\
&\quad +4\frac{\gamma_k^2 n}{\tilde{S}_\alpha}(1+t)\left(f(\vec{y}_k) - \mathbb{E}_{i_k}\left[f(\vec{x}_{k+1})\right] + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\left(\mathbb{E}_{i_k}\left[f(\vec{x}_{k+1})\right] - f^*\right)\right) \\
&\quad +2(1+t)\frac{\gamma_k}{\tilde{S}_\alpha}\left\langle \nabla f(\vec{y}_k), \vec{x}^* - \vec{y}_k + \frac{\beta_k(1-\alpha_k)}{\alpha_k}(\vec{x}_k - \vec{y}_k)\right\rangle \\
&\leq \left(1+\frac{1}{t}\right)\varepsilon^2 + (1+t)\beta_k r_k^2 + (1+t)(1-\beta_k)||\vec{y}_k - \vec{x}^*||_{1-\alpha}^2 \\
&\quad +4\frac{\gamma_k^2 n}{\tilde{S}_\alpha}(1+t)\left(f(\vec{y}_k) - \mathbb{E}_{i_k}\left[f(\vec{x}_{k+1})\right] + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\left(\mathbb{E}_{i_k}\left[f(\vec{x}_{k+1})\right] - f^*\right)\right) \\
&\quad +2(1+t)\frac{\gamma_k}{\tilde{S}_\alpha}\left(f^* - f(\vec{y}_k) - \frac{1}{2}\sigma_{1-\alpha}||\vec{y}_k - \vec{x}^*||_{1-\alpha}^2\right. \\
&\quad \left. +\frac{\beta_k(1-\alpha_k)}{\alpha_k}\left(f(\vec{x}_k) - f(\vec{y}_k)\right)\right).
\end{aligned}$$

Now since we defined coefficients so the following hold

$$1-\beta_k = \frac{\gamma_k}{\tilde{S}_\alpha}\sigma_{1-\alpha} \quad \text{and} \quad \gamma_k^2 - \frac{\gamma_k}{2n} = \frac{\gamma_k\beta_k(1-\alpha_k)}{2n\alpha_k},$$

we see that the terms for $\|\vec{y}_k - \vec{x}^*\|_{1-\alpha}^2$ and $f(\vec{y}_k)$ cancel and we obtain

$$\begin{aligned}
\mathbb{E}_{i_k}\left[r_{k+1}^2\right] &\leq \left(1+\frac{1}{t}\right)\varepsilon^2 + (1+t)\beta_k r_k^2 \\
&\quad -4(1+t)\frac{\gamma_k^2 n}{\tilde{S}_\alpha}\left(\mathbb{E}_k\left[f(\vec{x}_{k+1})\right] - \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\left(\mathbb{E}_{i_k}\left[f(\vec{x}_{k+1})\right] - f^*\right)\right) \\
&\quad +2(1+t)\frac{\gamma_k}{\tilde{S}_\alpha}\left(f^* + \frac{\beta_k(1-\alpha_k)}{\alpha_k}f(\vec{x}_k)\right) .
\end{aligned}$$

Multiplying both sides by $\frac{\tilde{S}_\alpha}{n}b_{k+1}^2$ and recalling the following

$$b_{k+1}^2 = \frac{1}{\beta_k}b_k^2, \quad a_{k+1}^2 = \gamma_k^2 b_{k+1}^2, \quad \gamma_k^2 - \frac{\gamma_k}{2n} = \beta_k\frac{a_k^2}{b_k^2} = \frac{\gamma_k\beta_k(1-\alpha_k)}{2n\alpha_k},$$

we get

$$\begin{aligned}\frac{\tilde{S}_\alpha}{n}b_{k+1}^2 E_{i_k}(r_{k+1}^2) \;\leq\; & \left(1+\frac{1}{t}\right)\frac{\tilde{S}_\alpha}{n}b_{k+1}^2\varepsilon^2 + (1+t)\frac{\tilde{S}_\alpha}{n}b_k^2 r_k^2 - 4(1+t)\cdot \\ & \left(a_{k+1}^2\left(1-\frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\right)\left(E_{i_k}f(\vec{x}_{k+1})-f^*\right) - a_k^2\left(f(\vec{x}_k)-f^*\right)\right).\end{aligned}$$

Dropping the expectation in each variable we have that in expectation to iteration k

$$\begin{aligned}& \frac{\tilde{S}_\alpha}{n}b_{k+1}^2(r_{k+1}^2) + (1+t)\left(1-\frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\right)4a_{k+1}^2\left(f(\vec{x}_{k+1})-f^*\right) \\ \leq\; & \left(1+\frac{1}{t}\right)\frac{\tilde{S}_\alpha}{n}b_{k+1}^2\varepsilon^2 + (1+t)\left(\frac{\tilde{S}_\alpha}{n}b_k^2 r_k^2 + 4a_k^2\left(f(\vec{x}_k)-f^*\right)\right).\end{aligned}$$

Letting $t \stackrel{\text{def}}{=} \frac{2\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}$. and writing $\tilde{r}_k^2 = \frac{\tilde{S}_\alpha}{n}E_{i_{k-1}}[r_k^2]$, $\phi_k = \mathbb{E}_{i_k}\left[f(\vec{x}_{k+1})\right] - f^*$, we have

$$\begin{aligned}b_{k+1}^2\tilde{r}_{k+1}^2 + 4a_{k+1}^2\phi_{k+1} \;\leq\; & \frac{\tilde{S}_\alpha}{n}\left(1+\frac{1}{t}\right)b_{k+1}^2\varepsilon^2 + (1+t)\left(b_k^2\tilde{r}_k^2 + 4a_k^2\phi_k\right) \\ \leq\; & \frac{\tilde{S}_\alpha}{n}\varepsilon^2\left(1+\frac{1}{t}\right)\sum_{j=1}^{k+1}(1+t)^{k+1-j}b_j^2 \\ & +4(1+t)^{k+1}\left(\tilde{r}_0^2 + \frac{1}{2n\tilde{S}_\alpha}\phi_0\right). \qquad \text{(A.7)}\end{aligned}$$

Now, we claim the following inequalities:

$$\begin{cases}(1+t)^{k+1-j}b_j^2 \text{ is increasing} & (1) \\ \sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}} \geq \gamma_k \geq \frac{1}{2n} & (2)\cdot \\ a_k \geq \frac{1}{2}\sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1} & (3)\end{cases}$$

Using the claims above and (A.7), we get

$$\begin{aligned}
& E_{i_k}\left(2\sigma_{1-\alpha}||\vec{v}_{k+1}-\vec{x}^*||_{1-\alpha}^2+4(f(\vec{x}_{k+1})-f^*)\right) \\
= & \frac{2n\sigma_{1-\alpha}}{\tilde{S}_\alpha}\tilde{r}_{k+1}^2+4\phi_{k+1} \\
\leq & \frac{b_{k+1}^2}{a_{k+1}^2}\tilde{r}_{k+1}^2+4\phi_{k+1} \\
\leq & \frac{\tilde{S}_\alpha}{n}\varepsilon^2\left(1+\frac{1}{t}\right)(k+1)\frac{b_{k+1}^2}{a_{k+1}^2}+\frac{4(1+t)^{k+1}}{a_{k+1}^2}\left(\tilde{r}_0^2+\frac{1}{2n\tilde{S}_\alpha}\phi_0\right) \\
\leq & 6\tilde{S}_\alpha\varepsilon^2(k+1)+\frac{32n\sigma_{1-\alpha}}{\tilde{S}_\alpha}\left(1+\frac{2\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\right)^{k+1}\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{-2(k+1)}\left(\tilde{r}_0^2+\frac{\phi_0}{2n\tilde{S}_\alpha}\right)
\end{aligned}$$

By the assumption that $\frac{2\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}<\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$ inequality (3.12) follows from the following

$$\begin{aligned}
& E_{i_k}\left(2\sigma_{1-\alpha}||\vec{v}_{k+1}-\vec{x}^*||_{1-\alpha}^2+4(f(\vec{x}_{k+1})-f^*)\right) \\
\leq & 12k\tilde{S}_\alpha\varepsilon^2+32\sigma_{1-\alpha}\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{-k}\left(\|\vec{x}_0-\vec{x}^*\|_{1-\alpha}^2+\frac{1}{2\tilde{S}_\alpha^2}\left(f(x_0)-f^*\right)\right) \\
\leq & 24kS_\alpha\varepsilon^2+32\sigma_{1-\alpha}\left(1-\frac{1}{5}\sqrt{\frac{\sigma_{1-\alpha}}{S_\alpha n}}\right)^{k}\left(\|\vec{x}_0-\vec{x}^*\|_{1-\alpha}^2+\frac{1}{S_\alpha^2}\left(f(x_0)-f^*\right)\right).
\end{aligned}$$

Now, we prove the claims.

Claim (1): Since $a_{k+1}=\gamma_k b_{k+1}\geq\frac{1}{2n}b_{k+1}$ and (A.3), we have $b_{k+1}\geq(1+\frac{\sigma_{1-\alpha}}{4\tilde{S}_\alpha n})b_k$. By the assumption $t=\frac{2\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}<\frac{\sigma_{1-\alpha}}{4\tilde{S}_\alpha n}$, we see that $(1+t)^{k+1-j}b_j^2$ is increasing.

Claim (2): Follows from Lemma A.2.1.

Claim (3): Using Lemma A.2.1 and $k\geq\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}$, we have

$$\begin{aligned}
a_k & \geq \sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}\left(\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}-\left(1-\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}\right) \\
& \geq \sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}\left(\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}-1\right) \\
& \geq \frac{1}{2}\sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}\left(1+\frac{1}{2}\sqrt{\frac{\sigma_{1-\alpha}}{2\tilde{S}_\alpha n}}\right)^{k+1}.
\end{aligned}$$

To bound the norm of the gradient, we show that if $\|\nabla f(\vec{y}_k)\|_{1-\alpha}^2$ is large for

many steps then $f(\vec{x}_k)$ decreases substantially. For notational simplicity we omit the expectation symbol and using (A.6), we have

$$\begin{aligned} f(\vec{x}_{k+1}) &\leq f(T_{i_k}(\vec{y}_k)) + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\left(f(\vec{x}_{k+1}) - f^*\right) \\ &\leq f(\vec{y}_k) - \frac{1}{\tilde{S}_\alpha}\left(||\nabla f(\vec{y}_k)||^*_{1-\alpha}\right)^2 + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\left(f(\vec{x}_{k+1}) - f^*\right). \end{aligned}$$

To bound $f(\vec{y}_k)$, we consider the lower envelops at $\vec{y}_k$ and obtain

$$\begin{aligned} f(\vec{y}_k) &\leq f(\vec{x}_k) - \langle \nabla f(\vec{y}_k), \alpha_k(\vec{x}_k - \vec{v}_k)\rangle \\ &\leq f(\vec{x}_k) + \alpha_k||\nabla f(\vec{y}_k)||^*_{1-\alpha}||\vec{x}_k - \vec{v}_k||_{1-\alpha} \\ &\leq f(\vec{x}_k) + \alpha_k||\nabla f(\vec{y}_k)||^*_{1-\alpha}\left(||\vec{x}_k - \vec{x}^*||_{1-\alpha} + ||\vec{v}_k - \vec{x}^*||_{1-\alpha}\right) \\ &\leq f(\vec{x}_k) + \alpha_k||\nabla f(\vec{y}_k)||^*_{1-\alpha}\left(\frac{1}{\sqrt{\sigma_{1-\alpha}}}\sqrt{f(\vec{x}_k) - f^*} + ||\vec{v}_k - \vec{x}^*||_{1-\alpha}\right). \end{aligned}$$

Combining with Lemma A.2.1, we have

$$f(\vec{x}_{k+1}) \leq f(\vec{x}_k) - \frac{1}{\tilde{S}_\alpha}\left(\|\nabla f(\vec{y}_k)\|^*_{1-\alpha}\right)^2 + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\delta_{k+1} + 3\sqrt{\frac{\sigma_{1-\alpha}}{n\tilde{S}_\alpha}}\,\|\nabla f(\vec{y}_k)\|^*_{1-\alpha}\sqrt{\frac{\delta_k}{\sigma_{1-\alpha}}}.$$

Summing up from k to $2k$ then yields

$$f(\vec{x}_{2k}) \leq f(\vec{x}_k) - \sum_{j=k}^{2k-1}\frac{\left(||\nabla f(\vec{y}_k)||^*_{1-\alpha}\right)^2}{\tilde{S}_\alpha} + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\sum_{j=k+1}^{2k}\delta_j + \frac{32}{\sqrt{n\tilde{S}_\alpha}}\sum_{j=k}^{2k-1}\sqrt{\delta_j}\,\|\nabla f(\vec{y}_j)\|^*_{1-\alpha}$$

Since $f(\vec{x}_k) - f(\vec{x}_{2k}) \leq f(\vec{x}_k) - f^* \leq \delta_k$, we have

$$\begin{aligned} \frac{1}{\tilde{S}_\alpha}\sum_{j=k}^{2k-1}\left(||\nabla f(\vec{y}_j)||^*_{1-\alpha}\right)^2 &\leq \delta_k + \frac{\varepsilon\tilde{S}_\alpha}{\sigma_{1-\alpha}}\sum_{j=k+1}^{2k}\delta_j + 32\sqrt{\frac{1}{n\tilde{S}_\alpha}}\sum_{j=k}^{2k-1}\|\nabla f(\vec{y}_j)\|^*_{1-\alpha}\sqrt{\delta_j} \\ &\leq \delta_k\left(1 + \frac{\varepsilon k\tilde{S}_\alpha}{\sigma_{1-\alpha}}\right) + 32\sqrt{\frac{1}{n\tilde{S}_\alpha}}\sum_{j=k}^{2k-1}\|\nabla f(\vec{y}_j)\|^*_{1-\alpha}\sqrt{\delta_j} \\ &\leq \delta_k\left(1 + \frac{\varepsilon k\tilde{S}_\alpha}{\sigma_{1-\alpha}}\right) + 32\sqrt{\frac{k\delta_k}{n}}\sqrt{\frac{1}{\tilde{S}_\alpha}\sum_{j=k}^{2k-1}\left(\|\nabla f(\vec{y}_k)\|^*_{1-\alpha}\right)^2}. \end{aligned}$$

Solving this quadratic equation about $\sqrt{\frac{1}{\tilde{S}_\alpha}\sum_{j=k}^{2k-1}\left(\|\nabla f(\vec{y}_k)\|_{1-\alpha}^*\right)^2}$, we get

$$\sqrt{\frac{1}{\tilde{S}_\alpha}\sum_{j=k}^{2k-1}\left(\|\nabla f(\vec{y}_k)\|_{1-\alpha}^*\right)^2} \le 16\sqrt{\frac{k\delta_k}{n}} + \frac{1}{2}\sqrt{\left(32\sqrt{\frac{k\delta_k}{n}}\right)^2 + 4\delta_k\left(1+\frac{\varepsilon k\tilde{S}_\alpha}{\sigma_{1-\alpha}}\right)}$$

Since $1+\frac{\varepsilon k\tilde{S}_\alpha}{\sigma_{1-\alpha}} \le 2$ and $k \ge n$ by assumptions we have

$$\sqrt{\frac{1}{\tilde{S}_\alpha}\sum_{j=k}^{2k-1}\left(\|\nabla f(\vec{y}_k)\|_{1-\alpha}^*\right)^2} \le 16\sqrt{\frac{k\delta_k}{n}} + \frac{1}{2}\sqrt{(32^2+8)\frac{k\delta_k}{n}} \le 33\sqrt{\frac{k\delta_k}{n}}.$$

Therefore $\frac{1}{\tilde{S}_\alpha}\sum_{j=k}^{2k-1}\left(\|\nabla f(\vec{y}_k)\|_{1-\alpha}^*\right)^2 \le 2000\frac{k\delta_k}{n}$. as desired. □

Lemma A.3.1 (Coefficient Stability in ACDM). *Step 3a of ACDM can be computed by the following formulas*

$$\begin{aligned}
\gamma_{k+1} &= \min\left\{\frac{1}{2}\left[\left(\frac{1}{2n}-\frac{\gamma_k^2\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)+\sqrt{\left(\frac{1}{2n}-\frac{\gamma_k^2\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)^2+4\gamma_k^2}\right] \;,\; \sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}\right\},\\
\beta_{k+1} &= 1-\frac{\gamma_{k+1}\sigma_{1-\alpha}}{\tilde{S}_\alpha},\\
\alpha_{k+1} &= \frac{\beta_{k+1}}{\beta_{k+1}+2n\gamma_{k+1}-1} = \frac{\frac{2n}{\gamma_k}-\frac{\sigma_{1-\alpha}}{2n\tilde{S}_\alpha}}{1-\frac{\sigma_{1-\alpha}}{2n\tilde{S}_\alpha}}.
\end{aligned}$$

and can be implemented using $O(\log n)$ bits precision to obtain any poly$(\frac{1}{n})$ additive accuracy.

Proof. Since $\gamma_k = \frac{a_{k+1}}{b_{k+1}}$ we know that $\gamma_{k+1}^2 - \frac{\gamma_{k+1}}{2n} = \left(1-\frac{\gamma_{k+1}\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)\gamma_k^2$, and by re-arranging terms we get that $\gamma_{k+1}^2 - \left(\frac{1}{2n}-\frac{\gamma_k^2\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)\gamma_{k+1} - \gamma_k^2 = 0$. Applying the quadratic formula and Lemma A.2.1 then yields the formula for γ_{k+1} and the other formulas follow by direct computation.

Now, expanding the formula for γ_{k+1} and using $2\tilde{S}_\alpha n \ge \sigma_{1-\alpha}$ implies that $\gamma_k \ge \frac{1}{2n}$ and therefore the denominator of α_{k+1} is larger than β_{k+1}. Consequently the equations of β and α depends continuously on γ. Thus, it is suffices to show that $O(\log n)$ bits precision suffice to compute γ_k. To prove this, we define $f : \mathbb{R}^n \to \mathbb{R}$ denote the

quadratic formula so that with full precision $\gamma_{k+1} = f(\gamma_k)$. Direct calculation yields

$$f'(\gamma) = -\frac{\gamma\sigma_{1-\alpha}}{\tilde{S}_\alpha} + \frac{\left(\frac{\gamma^2\sigma_{1-\alpha}}{\tilde{S}_\alpha} - \frac{1}{2n}\right)\frac{\gamma\sigma_{1-\alpha}}{\tilde{S}_\alpha} + 2\gamma}{\sqrt{\left(\frac{1}{2n} - \frac{\gamma^2\sigma_{1-\alpha}}{\tilde{S}_\alpha}\right)^2 + 4\gamma^2}}$$

Since $0 \leq \gamma \leq \sqrt{\frac{\tilde{S}_\alpha}{2n\sigma_{1-\alpha}}}$ we have that $0 \leq f'(\gamma) \leq 1 - \frac{\gamma\sigma_{1-\alpha}}{\tilde{S}_\alpha} < 1$ and therefore, f is a contraction mapping. Consequently, if the calculation of γ has up to ε additive error, k steps of calculation can accumulate up to $k\varepsilon/(1 - \max f'(\gamma))$ additive error. Hence, $O(\log n)$ bits of precision suffice. □

Bibliography

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *CoRR*, abs/0808.2017, 2008.

[2] Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 395–406, New York, NY, USA, 2012. ACM.

[3] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24:78–100, February 1995.

[4] Heinz H. Bauschke and Jonathan M. Borwein. On projection algorithms for solving convex feasibility problems. *SIAM Rev.*, 38(3):367–426, September 1996.

[5] Amir Beck and Marc Teboulle. A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems. *SIAM Journal on Imaging Sciences*, 2(1):183–202, January 2009.

[6] Stephen Becker, J Bobin, and EJ Candès. NESTA: a fast and accurate first-order method for sparse recovery. *SIAM Journal on Imaging Sciences*, 4(1):1–39, 2011.

[7] M. Bern, J. Gilbert, B. Hendrickson, N. Nguyen, and S. Toledo. Support-graph preconditioners. *SIAM Journal on Matrix Analysis and Applications*, 27(4):930–951, 2006.

[8] D. Bienstock and G. Iyengar. Solving fractional packing problems in oast(1/esp) iterations. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, STOC '04, pages 146–155, New York, NY, USA, 2004. ACM.

[9] Guy E. Blelloch, Ioannis Koutis, Gary L. Miller, and Kanat Tangwongsan. Hierarchical diagonal blocking and precision reduction applied to combinatorial multigrid. In *SC*, pages 1–12, 2010.

[10] Bela Bollobas. *Modern Graph Theory*. Springer, 1998.

[11] E.G. Boman, D. Chen, B. Hendrickson, and S. Toledo. Maximum-weight-basis preconditioners. *Numerical linear algebra with applications*, 11(8-9):695–721, 2004.

[12] E.G. Boman, B. Hendrickson, and S. Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. *SIAM Journal on Numerical Analysis*, 46(6):3264–3284, 2008.

[13] Erik G. Boman and Bruce Hendrickson. Support theory for preconditioning. *SIAM J. Matrix Anal. Appl.*, 25:694–717, March 2003.

[14] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[15] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial: second edition.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[16] Paul Christiano, Jonathan A Kelner, Aleksander Madry, Daniel A Spielman, and Shang-Hua Teng. Electrical flows, laplacian systems, and faster approximation of maximum flow in undirected graphs. In *Proceedings of the 43rd annual ACM symposium on Theory of computing*, pages 273–282. ACM, 2011.

[17] Samuel I. Daitch and Daniel A. Spielman. Faster approximate lossy generalized flow via interior point algorithms. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, pages 451–460, 2008.

[18] Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 494–503, 2005.

[19] T Goldstein, BRENDAN ODonoghue, and Simon Setzer. Fast alternating direction optimization methods. *CAM report*, pages 12–35, 2012.

[20] Keith D. Gremban, Gary L. Miller, and Marco Zagha. Performance evaluation of a new parallel preconditioner. In *IPPS*, pages 65–69, 1995.

[21] Gabor T Herman. *Fundamentals of computerized tomography: image reconstruction from projections.* Springer, 2009.

[22] Luqman Hodgkinson and Richard Karp. Algorithms to detect multiprotein modularity conserved during evolution. In Jianer Chen, Jianxin Wang, and Alexander Zelikovsky, editors, *Bioinformatics Research and Applications*, volume 6674 of *Lecture Notes in Computer Science*, pages 111–122. Springer Berlin / Heidelberg, 2011.

[23] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew Math*, 70:185–190, 1869.

[24] Stefan Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bull. Internat. Acad. Polon.Sci. Lettres A*, page 335357, 1937.

[25] Carl T Kelley. *Iterative methods for optimization*, volume 18. Society for Industrial and Applied Mathematics, 1987.

[26] Jonathan Kelner and Petar Maymounkov. Electric routing and concurrent flow cutting. *Theor. Comput. Sci.*, 412(32):4123–4135, July 2011.

[27] Jonathan A. Kelner and Aleksander Madry. Faster generation of random spanning trees. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, pages 13–21, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Jonathan A. Kelner, Gary L. Miller, and Richard Peng. Faster approximate multicommodity flow using quadratically coupled flows. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1–18, New York, NY, USA, 2012. ACM.

[29] Jonathan A Kelner, Lorenzo Orecchia, Yin Tat Lee, and Aaron Sidford. An almost-linear-time algorithm for approximate max flow in undirected graphs, and its multicommodity generalizations. *arXiv preprint arXiv:1304.2338*, 2013.

[30] Jonathan A. Kelner, Lorenzo Orecchia, Aaron Sidford, and Zeyuan Allen Zhu. A Simple, Combinatorial Algorithm for Solving SDD Systems in Nearly-Linear Time. January 2013.

[31] Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *Proceedings of the 51st Annual Symposium on Foundations of Computer Science*, 2010.

[32] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for sdd linear systems. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 590 –598, oct. 2011.

[33] Ioannis Koutis, Gary L. Miller, and David Tolliver. Combinatorial preconditioners and multilevel solvers for problems in computer vision and image processing. *Computer Vision and Image Understanding*, 115(12):1638–1646, 2011.

[34] Yin Tat Lee, Satish Rao, and Nikhil Srivastava. A New Approach to Computing Maximum Flows using Electrical Flows. *Proceedings of the 45th symposium on Theory of Computing - STOC '13*, 2013.

[35] Yin Tat Lee and Aaron Sidford. Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems. *CoRR*, abs/1305.1922, 2013.

[36] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th international conference on World Wide Web*, WWW '08, pages 695–704, New York, NY, USA, 2008. ACM.

[37] Chung-Shou Liao, Kanghao Lu, Michael Baym, Rohit Singh, and Bonnie Berger. Isorankn: spectral methods for global alignment of multiple protein networks. *Bioinformatics*, 25(12):i253–i258, 2009.

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

[39] Frank Natterer. *The mathematics of computerized tomography.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[40] Yu. Nesterov. A method for solving a convex programming problem with convergence rate 1/k^2. *Doklady AN SSSR*, 269:543–547, 1983.

[41] Yu Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume I. 2003.

[42] Yu. Nesterov. Rounding of convex sets and efficient gradient methods for linear programming problems. *Optimization Methods Software*, 23(1):109–128, February 2008.

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

[44] Lorenzo Orecchia, Sushant Sachdeva, and Nisheeth K. Vishnoi. Approximating the exponential, the lanczos method and an $\tilde{O}(m)$-time spectral algorithm for balanced separator. In *Proceedings of the 44th symposium on Theory of Computing*, STOC '12, pages 1141–1160, New York, NY, USA, 2012. ACM.

[45] Peter Richtárik and Martin Takáč. Iteration Complexity of Randomized Block-Coordinate Descent Methods for Minimizing a Composite Function. page 33, July 2011.

[46] Jonah Sherman. Breaking the multicommodity flow barrier for $O\left(\sqrt{\log n}\right)$-approximations to sparsest cut. In *Proceedings of the 50th Annual Symposium on Foundations of Computer Science*, 2009.

[47] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[48] D.A. Spielman and N. Srivastava. Graph sparsification by effective resistances. *SIAM Journal on Computing*, 40(6):1913–1926, 2011.

[49] Daniel A. Spielman. Algorithms, graph theory, and the solution of laplacian linear equations. In *Proceedings of the 39th international colloquium conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 24–26, Berlin, Heidelberg, 2012. Springer-Verlag.

[50] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, New York, NY, USA, 2004. ACM.

[51] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. *CoRR*, abs/0809.3232, 2008.

[52] Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008.

[53] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105v5, 2012.

[54] Daniel A Spielman and Jaeoh Woo. A Note on Preconditioning by Low-Stretch Spanning Trees. March 2009.

[55] Thomas Strohmer and Roman Vershynin. A randomized kaczmarz algorithm with exponential convergence. *Journal of Fourier Analysis and Applications*, 15:262–278, 2009.

[56] Robert Endre Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26(4):690–715, October 1979.

[57] Shang-Hua Teng. The laplacian paradigm: Emerging algorithms for massive graphs. In *TAMC*, pages 2–14, 2010.

[58] Pravin M. Vaidya. Solving linear equations with symmetric diagonally dominant matrices by constructing good preconditioners. Unpublished manuscript UIUC 1990. A talk based on the manuscript was presented at the IMA Workshop on Graph Theory and Sparse Matrix Computation, October 1991, Minneapolis., 1990.

[59] Nisheeth Vishnoi. $Lx = b$. Monograph, available at http://research.microsoft.com/en-us/um/people/nvishno/site/Lxb-Web.pdf.

[60] Konstantin Voevodski, Shang-Hua Teng, and Yu Xia. Finding local communities in protein networks. *BMC Bioinformatics*, 10(1):297, 2009.

[61] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.

[62] W Zangwill. *Nonlinear Programming: A Unified Approach.* Prentice-Hall, 1969.