NEW PERSPECTIVES ON CONTINUOUS OPTIMIZATION: THEORY AND METHODOLOGY

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Shipu Zhao

May 2023

NEW PERSPECTIVES ON CONTINUOUS OPTIMIZATION: THEORY AND METHODOLOGY

Shipu Zhao, Ph.D.

Cornell University 2023

Large-scale continuous optimization arises in many practical problems such as machine learning, signal processing, and imaging. It is usually challenging to analyze the theoretical properties of optimization algorithms and design scalable algorithms that work well in practice. This dissertation provides new perspectives on continuous optimization in both theory and methodology. From the theoretical side, we present a framework for reasoning about equivalence between a broad class of iterative convex continuous optimization algorithms. The notion of algorithm equivalence can make it easier to understand the connections between optimization algorithms, relate many analytical properties of interest such as convergence or robustness, and further give insights to design new algorithms. From the methodological side, we first present NysADMM, a scalable algorithm for faster composite convex optimization by exploiting the low-rank structure. NysADMM accelerates the inexact linearized Alternating Direction Method of Multipliers (ADMM) by constructing a preconditioner for the ADMM subproblem from a randomized low-rank Nyström approximation. Second, we present GeNI-ADMM, a framework generalized from NysADMM, which encompasses any ADMM algorithm that solves a first- or (generalized) second-order approximation to the ADMM subproblem and allows inexact subproblem solves. It facilitates the theoretical analysis of various approximate ADMM schemes for large-scale composite convex optimization.

BIOGRAPHICAL SKETCH

Shipu Zhao was born and raised in Jilin, China. He attended Tsinghua University for his undergraduate study, where he majored in transportation engineering and economics. During undergraduate time, both his majors needed significant amount of knowledge on mathematics and he was fascinated by the power of mathematics to model and solve real-word problems. He then spent six months at Georgia Tech as a visiting student. At Georgia Tech, he was first exposed to the field of systems engineering and optimization and got a sense of doing academic research. This helps him made up his mind to purse a Ph.D. degree, where he could use mathematics to solve real-world problems.

After undergraduate studies, Shipu joined the systems engineering Ph.D. program at Cornell University. He spent the first two years studying necessary knowledge and wondering his research interest. He then met Prof. Madeleine Udell and was fortunate to have her as his Ph.D. advisor. He was also grateful to have Prof. H. Oliver Gao and Prof. Eilyan Bitar on the committee. He developed a broad interest in optimization from theoretical side to methodological side and worked on continuous optimization for his Ph.D. thesis. He appreciated all the professional knowledge and skills learned and developed during Ph.D. research, and the help offered by professors, family, and friends during this time period.

Dedicated to my family and friends.

ACKNOWLEDGEMENTS

Finally, I reach this finish line for my Ph.D. journey. This reminds me the time when I started my Ph.D. and until now I can finally say I make it. It has been a long and challenging life but abundant with the joy of doing research. I'm grateful to everyone who helped me during my Ph.D. life. Without your firm support, I could not complete this.

First, I would like to thank my advisor, Madeleine Udell. She is an excellent person to work with for her enthusiasm about research, strong knowledge background, and ability to relate complex abstract concepts into practical applications. Further, I'm impressed that she pursues making useful methodologies for practical applications in research, not only about beautiful theory. She helps me with constant encouragement, support, and guidance in research. The most important, she offered me a big favor during my darkest time and helped me to overcome the difficulty. I'm really appreciated and I would never forget that.

Second, I would like to thank my committee members, Eilyan Bitar and H. Oliver Gao. They gave me valuable research suggestions and were always willing to offer help. They gave me strong encouragement during the time when I was about to change my research direction. They also offered strong support to my choices and provided much help for me to finally succeed to make the Ph.D.

Third, I would like to thank my other collaborators. Zachary Frangella: We worked together on several papers and the collaboration is amazing to me. He is an expert in numerical linear algebra and optimization, and I'm constantly learning new knowledge and getting cool ideas through our discussions. Laurent Lessard: We worked together on the algorithm equivalence project. He was the person that brought me to the field of control theory. And he was very patient to me for my trivial questions when I was a beginner in research. Pratik

Rathore: We worked together on the SkethcySGD project. Theo Diamandis and Bartolomeo Stellato: We worked together on the GeNI-ADMM project. They are all brilliant people and essential for me to make solid work.

Last, I would like to thank my family, friends, and research group members. Without your accompany, I could not make it. You are all very valuable to me and I'm so fortunate to meet you during the life journey.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . iv
Acknowledgements . . . v
Table of Contents . . . vii
List of Tables . . . x
List of Figures . . . xi

1 Introduction **1**

2 An automatic system to detect equivalence between iterative algorithms **6**
2.1 Introduction . . . 6
2.2 Related work . . . 8
2.3 Motivating examples . . . 10
2.4 Preliminaries . . . 11
2.4.1 Optimization . . . 12
2.4.2 Algorithms . . . 13
2.4.3 Control theory . . . 16
2.5 Algorithm equivalence . . . 24
2.5.1 Assumptions . . . 24
2.5.2 Oracle equivalence . . . 25
2.5.3 Shift equivalence . . . 27
2.5.4 Discussion . . . 28
2.6 A characterization of oracle equivalence . . . 29
2.6.1 Motivating examples: proof of equivalence . . . 32
2.7 A characterization of shift equivalence . . . 34
2.7.1 Reordering oracle calls . . . 37
2.7.2 Characterization of cyclic permutation . . . 40
2.7.3 Applications: proof of shift equivalence . . . 43
2.8 Algorithm repetition . . . 46
2.9 Algorithm conjugation . . . 50
2.10 Linnaeus . . . 58
2.10.1 Illustrative examples . . . 59
2.10.2 Implementation . . . 67
2.10.3 Black-box vs functional oracles . . . 70
2.11 Conclusion and future work . . . 73

3 NysADMM: faster composite convex optimization via low-rank approximation **76**
3.1 Introduction . . . 76
3.1.1 Contributions . . . 78
3.1.2 Related work . . . 78

3.1.3 Notation and preliminaries . . . 80
3.2 Algorithm . . . 81
3.2.1 Inexact linearized ADMM . . . 81
3.2.2 Randomized Nyström approximation and PCG . . . 83
3.2.3 NysADMM . . . 85
3.2.4 AdaNysADMM . . . 86
3.3 Applications . . . 87
3.3.1 Elastic net . . . 87
3.3.2 Regularized logistic regression . . . 88
3.3.3 Support vector machine . . . 88
3.4 Convergence analysis . . . 89
3.5 Numerical experiments . . . 93
3.5.1 Lasso . . . 94
3.5.2 ℓ_1-regularized logistic regression . . . 97
3.5.3 Support vector machine . . . 98
3.6 Conclusion . . . 99

4 On the (linear) convergence of Generalized Newton Inexact ADMM 100
4.1 Introduction . . . 100
4.1.1 Contributions . . . 102
4.1.2 Notation and preliminaries . . . 103
4.2 Problem statement and ADMM . . . 103
4.3 Generalized Newton Inexact ADMM . . . 104
4.3.1 Related work . . . 109
4.3.2 Algorithms recovered from GeNI-ADMM . . . 109
4.4 Technical preliminaries and assumptions . . . 112
4.4.1 Assumptions . . . 114
4.5 Sublinear convergence of GeNI-ADMM . . . 116
4.5.1 Our approach . . . 119
4.5.2 Controlling the gap function . . . 122
4.5.3 Proof of theorem 4.5.1 . . . 125
4.6 Linear convergence of GeNI-ADMM . . . 130
4.6.1 Sufficient descent . . . 132
4.6.2 Proof of theorem 4.6.1 . . . 134
4.7 Applications . . . 135
4.7.1 Convergence of NysADMM . . . 136
4.7.2 Convergence of sketch-and-solve ADMM . . . 137
4.8 Numerical experiments . . . 140
4.8.1 Lasso regression . . . 143
4.8.2 Logistic Regression . . . 144
4.9 Conclusion and future work . . . 146

A Appendix of chapter 2 **148**
A.1 Proof of (2.14) . . . 148
A.2 Proof of proposition 2.6.1 . . . 151
A.3 Proof of proposition 2.7.1 . . . 152
A.4 Proof of proposition 2.7.3 . . . 155
A.5 Discussions on permutation and its generalization . . . 156
A.6 Proof of proposition 2.7.4 . . . 159
A.7 Proof of shift-equivalence of DR and ADMM continued . . . 160
A.8 Proof of proposition 2.8.1 . . . 161
A.9 Proof of proposition 2.8.3 . . . 162
A.10 Proof of proposition 2.9.3 . . . 164
A.11 Commutativity between conjugation and cyclic permutation . . . 165
A.12 Proof of (2.36) and (2.37) . . . 167

B Appendix of chapter 3 **169**
B.1 Proofs of main results . . . 169
B.1.1 Preliminaries . . . 169
B.1.2 Proofs of Theorem 3.4.1 and Corollary 3.4.1 . . . 172
B.1.3 Proof of Theorem 3.4.2 . . . 174
B.1.4 Proof of Theorem 3.4.3 . . . 177
B.2 Randomized Nyström approximation and Nyström PCG . . . 177
B.3 AdaNysADMM . . . 178

C Appendix of chapter 4 **181**
C.1 Proofs for section 4.4 . . . 181
C.2 Proofs for section 4.5 . . . 182
C.3 Proofs for section 4.6 . . . 188

LIST OF TABLES

3.1 Complexity comparison, for a quadratic loss with Hessian H. Here T_{mv} is the time to compute a matrix vector product with H, κ_2 is the condition number of H, and ε^k is the precision of the kth subproblem solve (3.4). 86
3.2 Statistics of experiment datasets. 93
3.3 Results for low precision lasso experiment. 95
3.4 Results for high precision lasso experiment. 95
3.5 Results for ℓ_1-regularized logistic regression experiment. 98
3.6 Results of SVM experiment. 99

4.1 A structured comparison of related work on the convergence of ADMM and its variants. 110
4.2 Convergence rate comparison of GeNI-ADMM with $\alpha = 0$ when initialized at 0 for quadratic f. 118

LIST OF FIGURES

2.1 Block-diagram representation of an algorithm. This is equivalent to the pair of equations $\mathbf{y} = \mathbf{H}\mathbf{u}$ and $\mathbf{u} = \Phi\mathbf{y}$. 17
2.2 Unrolled-in-time block-diagram representation of an algorithm. . 19
2.3 Unrolled block-diagram representation of oracle equivalence. . . 26
2.4 Unrolled block-diagram representation of shift equivalence. . . . 27
2.5 Directed graph representing dependency of oracle calls in algorithm 2.7.3. 38
2.6 Unrolled block-diagram representation of algorithm conjugation. 52
2.7 Connections between DR, ADMM, and Chambolle-Pock method. 58

3.1 Solution times for varying tolerance ϵ on STL-10. 96

4.1 Linear convergence of NysADMM applied to lasso. 141
4.2 Linear convergence of NysADMM applied to logistic regression. 141
4.3 Convergence (in objective function value) of lasso regression for NysADMM, sketch-and-solve ADMM, and gradient descent ADMM. 142
4.4 Convergence (in objective function value) of elastic net regression for NysADMM, sketch-and-solve ADMM, and gradient descent ADMM. 144
4.5 Convergence (in objective function value) of logistic regression for NysADMM, sketch-and-solve ADMM, and gradient descent ADMM. 145

CHAPTER 1

INTRODUCTION

The practice of solving problems through optimization has become widely prevalent in numerous applied fields. Within the domains of machine learning and data analysis, large-scale continuous optimization is routinely encountered with the trend of Big Data. Nonetheless, the interest in utilizing such optimization techniques for large-scale problems presents several significant challenges that need to be addressed.

The first challenge is to analyze the theoretical properties of optimization algorithms and further distinguish them. During the past 10 years or so, lots of progresses have been made in the field of optimization. New optimization algorithms are regularly proposed in order to capture more complicated models, reduce computational burdens, or obtain stronger performance and convergence guarantees. However, the *novelty* of an algorithm can be difficult to establish because algorithms can be written in different equivalent forms.

For example, algorithm 1.1 was originally proposed by Popov [85] in the context of solving saddle point problems. This method was later generalized by Chiang et al. [22, §4.1] in the context of online optimization. Algorithm 1.2 is a reformulation of algorithm 1.1 adapted for use in generative adversarial networks (GANs) [45]. Algorithm 1.3 is an adaptation of *Optimistic Mirror Descent* [88] used by Daskalakis et al. [25] and also used to train GANs. Finally, algorithm 1.4 was proposed by Malitsky [67] for solving monotone variational inequality problems. (Some of these algorithms were originally proposed in conjunction with projections or other operations that make them more distinct.) In all four algorithms, the vectors x_1^k and x_2^k are algorithm states, η is a tun-

able parameter, and $F^k(\cdot)$ is the gradient of the loss function at time step k.

Algorithm 1.1
(Modified Arrow–Hurwicz)

for $k = 1, 2, \ldots$ **do**
$x_1^{k+1} = x_1^k - \eta F^k(x_2^k)$
$x_2^{k+1} = x_1^{k+1} - \eta F^k(x_2^k)$
end for

Algorithm 1.2
(Extrapolation from the past)

for $k = 1, 2, \ldots$ **do**
$x_2^k = x_1^k - \eta F^{k-1}(x_2^{k-1})$
$x_1^{k+1} = x_1^k - \eta F^k(x_2^k)$
end for

Algorithm 1.3
(Optimistic Mirror Descent)

for $k = 1, 2, \ldots$ **do**
$x_2^{k+1} = x_2^k - 2\eta F^k(x_2^k) + \eta F^{k-1}(x_2^{k-1})$
end for

Algorithm 1.4
(Reflected Gradient Method)

for $k = 1, 2, \ldots$ **do**
$x_1^{k+1} = x_1^k - \eta F^k(2x_1^k - x_1^{k-1})$
end for

Algorithms 1.1 to 1.4 are equivalent in the sense that when suitably initialized, the sequences $(x_1^k)_{k\geq 0}$ and $(x_2^k)_{k\geq 0}$ are identical for all four algorithms.[1] Although these particular equivalences are not difficult to verify and many have been explicitly pointed out in the literature, for example in [45], algorithm equivalence is not always immediately apparent.

The second challenge is to design scalable algorithms that work well in practice. Take the famous Alternating Directions Method of Multipliers (ADMM) framework as an example [12]. Although it is one of the most popular methods for solving composite optimization problems thanks to its ability to provide a general template for a wide swath of problems, it scales poorly for many large-scale problem instances raised in the era of Big Data. The scaling issue arises as ADMM requires solving two subproblems at each iteration, whose cost can increase superlinearly with the problem size. As a concrete example, in the

[1] In their original formulations, algorithms 1.1, 1.2 and 1.4 included projections onto convex constraint sets. We assume an unconstrained setting here for illustrative purposes. Some of the equivalences no longer hold in the constrained case.

case of ℓ_1-logistic regression with an $n \times d$ data matrix, ADMM requires solving an ℓ_2-regularized logistic regression problem at each iteration [12]. If we apply a fast-gradient method, the total complexity of solving the subproblem is $\tilde{O}(nd\sqrt{\kappa})$ [15], where κ is the condition number of the problem. When n and d are in the tens of thousands or larger—a moderate problem size by contemporary machine machine standards—and κ is large, such a high per-iteration cost becomes unacceptable. The latter point on conditioning is especially relevant, as ill-conditioning is ubiquitous in machine learning problems; often $\kappa = \Omega(n)$, in which case the cost of the subproblem solve becomes superlinear in the problem size.

This dissertation provides new perspectives on these two challenges:

1. We present a framework for reasoning about equivalence between a broad class of iterative algorithms, with a focus on algorithms designed for convex optimization. We propose several notions of what it means for two algorithms to be equivalent, and provide computationally tractable means to detect equivalence. The main definition, oracle equivalence, states that two algorithms are equivalent if they result in the same sequence of calls to the function oracles (for suitable initialization). Borrowing from control theory, state-space realizations are used to represent algorithms and characterize algorithm equivalence via transfer functions. The proposed framework can also identify and characterize some algorithm transformations including permutations of the update equations, repetition of the iteration, and conjugation of some of the function oracles in the algorithm. A software package named Linnaeus have also been developed that implements the framework to identify other iterative algorithms that

are equivalent to an input algorithm. More broadly, this framework and software advances the goal of making mathematics searchable.

2. We present a scalable new algorithm, called NysADMM, for composite convex optimization to minimize a smooth convex loss function with a convex regularizer. NysADMM accelerates the inexact ADMM by constructing a preconditioner for the ADMM subproblem from a randomized low-rank Nyström approximation. NysADMM comes with strong theoretical guarantees: it solves the ADMM subproblem in a constant number of iterations when the rank of the Nystrom approximation is the effective dimension of the subproblem regularized Gram matrix. In practice, ranks much smaller than the effective dimension can succeed, so NysADMM uses an adaptive strategy to choose the rank that enjoys analogous guarantees. Numerical experiments on real-world datasets demonstrate that NysADMM can solve important applications, such as the lasso, logistic regression, and support vector machines, in half the time (or less) required by standard solvers.

   The idea of NysADMM is further extended to a generalized framework, GeNI-ADMM. It facilitates theoretical analysis of both existing and new approximate ADMM schemes. GeNI-ADMM encompasses any ADMM algorithm that solves a first- or second-order approximation to the ADMM subproblem inexactly. GeNI-ADMM exhibits the usual $O(1/t)$-convergence rate under standard hypotheses and converges linearly under additional hypotheses such as strong convexity. Moreover, the GeNI-ADMM framework provides explicit convergence rates for ADMM variants accelerated with randomized linear algebra, such as NysADMM and sketch-and-solve ADMM, resolving an important open question on the

convergence of these methods. This analysis quantifies the benefit of improved approximations and can inspire the design of new ADMM variants with faster convergence.

This dissertation is based on three (working) papers. Generally, chapter 2 comes from [113], and chapter 3 comes from [112]. Chapter 4 comes from [42] and is our most recent work. Some contents are reorganized for a clear and concise presentation.

CHAPTER 2

AN AUTOMATIC SYSTEM TO DETECT EQUIVALENCE BETWEEN ITERATIVE ALGORITHMS

This chapter introduces the framework we proposed for reasoning about algorithm equivalence. The contents are mainly based on [113]. In section 2.2, we briefly summarize existing literature related to this chapter. In section 2.3, we introduce three examples of equivalent algorithms that motivate our framework. In section 2.4, we briefly review important background on linear systems and optimization used throughout this chapter. We formally define two notions of algorithm equivalence, *oracle equivalence* and *shift equivalence*, in section 2.5 and discuss how to characterize them via transfer functions in sections 2.6 and 2.7. Certain transformations can also be identified and characterized with our framework including *algorithm repetition*, repeating an algorithm multiple times, and *conjugation*, a transformation using conjugate function oracles. These are discussed in sections 2.8 and 2.9 respectively. In section 2.10, we briefly introduce our software package LINNAEUS for the classification of iterative algorithms.

2.1 Introduction

As discussed in chapter 1, algorithm equivalence is important and is not always immediately apparent. One famous example concerns the relations between the Chambolle-Pock method, Douglas-Rachford splitting, and the alternating directions method of multipliers (ADMM): indeed, showing the connection between Chambolle-Pock and Douglas-Rachford requires a full page of mathematics in

[20]. In contrast, our analysis supports a single coherent view of these algorithms that can be summarized in a commutative diagram (fig. 2.7).

In this chapter, we present a framework for reasoning about algorithm equivalence, with the ultimate goal of making the analysis and design of algorithms more principled and streamlined. The main contributions include:

- A universal way of representing algorithms, inspired by methods from control theory.
- Several definitions of what it means for algorithms to be equivalent.
- A computationally efficient way to verify whether two algorithms are equivalent.

Briefly, our method is to parse each algorithm to a standard form as a linear system in feedback with a nonlinearity; to compute the transfer function of each linear system; and to check, using a computer algebra system, if there are parameter values that make the transfer functions equal.

We also present a software package implementing this framework named LINNAEUS[1], for the classification and taxonomy of iterative algorithms. The software is a search engine, where the input is an algorithm described using natural syntax, and the output is a canonical form for the algorithm along with any known names and pointers to relevant literature. The approach described in this chapter allows LINNAEUS to search over first-order optimization algorithms such as gradient descent with acceleration, ADMM, and the extragradient method. As the database in LINNAEUS grows, it will help algorithm researchers understand and efficiently discover connections between algorithms.

[1]Named after Carl Linnaeus, a botanist and zoologist who invented the modern system of naming organisms.

More generally, LINNAEUS advances the goal of making mathematics searchable.

We must point out a tension in our terminology: the notion of algorithm equivalence we define below is rather broad, which is in order to discover interesting connections between algorithms. As a consequence, equivalent algorithms (in our terminology) can nevertheless be extremely useful for different tasks: for example, writing one algorithm in different ways can yield different generalizations, different interpretations, different computational complexity, and different numerical stability. On the other hand, equivalent algorithms will share many properties, such as convergence, stability, and fixed points.

2.2 Related work

A variety of existing work advances the goal of making mathematics searchable. This work is too diverse to survey here. As an example, consider the On-Line Encyclopedia of Integer Sequences: given a sub-sequence or a keyword, the encyclopedia will find a matching sequence and return useful information such as mathematical motivation for the sequence and links to other literature [96]. As a very different example, recent work in deep learning has led to new language models, such as GPT3, that can generate code snippets, including machine learning models, javascript applications, and SQL queries [1, 2, 14, 94]. As these models are trained from large corpuses of data, we might view such models as implementing a generalized search.

Within the optimization literature, several standard forms have been proposed to represent problems and algorithms. For example, the CVX* modeling

languages represent (disciplined) convex optimization problems in a standard conic form, building up the representations of complex problems from a few basic functions and a small set of composition rules [30, 48, 49, 95, 103]. This chapter builds on a foundation developed by Lessard et al. [61] that represents first-order algorithms as linear systems in feedback with a nonlinearity. Lessard et al. use this representation to analyze convergence properties of an algorithm with integral quadratic constraints. Our work extends theirs with the insight that such representations can be computed automatically by a computer.

There are rich connections between many first-order methods for convex optimization. These algorithms are surveyed in a recent textbook by Ryu and Yin, which summarizes and unifies several operator splitting methods for convex optimization [91]. Many of these connections are well known to experts, but the connections have traditionally been complex to explain, communicate, or even remember. For example, Boyd et al. [12] write, "There are also a number of other algorithms distinct from but inspired by ADMM. For instance, Fukushima [44] applies ADMM to a dual problem formulation, yielding a 'dual ADMM' algorithm, which is shown in [36] to be equivalent to the 'primal Douglas-Rachford' method discussed in [34, §3.5.6]." As another example, Chambolle and Pock in [20] propose a new primal-dual splitting algorithm and demonstrate that transformations of their algorithm can yield Douglas-Rachford splitting and ADMM, using a full page of mathematics to sketch the connection. Using our framework, the (many!) relations between the Chambolle-Pock method, Douglas-Rachford splitting, and ADMM can be established precisely and conveyed efficiently in a commutative diagram; see section 2.9 and fig. 2.7 in particular.

2.3 Motivating examples

To explain what we mean by algorithm equivalence, we introduce three motivating examples in this section. Each provides a different view of how two algorithms might be equivalent.

Algorithm 2.3.1

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = 2x_1^k - x_2^k - \frac{1}{10}\nabla f(2x_1^k - x_2^k)$
$x_2^{k+1} = x_1^k$
end for

Algorithm 2.3.2

for $k = 0, 1, 2, \ldots$ **do**
$\xi_1^{k+1} = \xi_1^k - \xi_2^k - \frac{1}{5}\nabla f(\xi_1^k)$
$\xi_2^{k+1} = \xi_2^k + \frac{1}{10}\nabla f(\xi_1^k)$
end for

The first example consists of algorithms 2.3.1 and 2.3.2. These algorithms are equivalent in a strong sense: when suitably initialized, we may transform the iterates of algorithm 2.3.1 by the invertible linear map $\xi_1^k = 2x_1^k - x_2^k, \xi_2^k = -x_1^k + x_2^k$ to yield the iterates of algorithm 2.3.2. We say that the sequences $(x_1^k)_{k\geq 0}$ and $(x_2^k)_{k\geq 0}$ are *equivalent* to sequences $(\xi_1^k)_{k\geq 0}$ and $(\xi_2^k)_{k\geq 0}$ *up to an invertible linear transformation*.

Algorithm 2.3.3

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = 3x_1^k - 2x_2^k + \frac{1}{5}\nabla f(-x_1^k + 2x_2^k)$
$x_2^{k+1} = x_1^k$
end for

Algorithm 2.3.4

for $k = 0, 1, 2, \ldots$ **do**
$\xi^{k+1} = \xi^k - \frac{1}{5}\nabla f(\xi^k)$
end for

The second example consists of algorithms 2.3.3 and 2.3.4. These algorithms do not even have the same number of state variables, so these algorithms are *not* equivalent up to an invertible linear transformation. But when suitably initialized, we may transform the iterates of algorithm 2.3.3 by the linear map $\xi^k = -x_1^k + 2x_2^k$ to yield the iterates of algorithm 2.3.4. This transformation is

linear but not invertible. Instead, notice that the sequence of calls to the gradient oracle are identical: the algorithms satisfy *oracle equivalence*, a notion we will define formally later in this chapter.

Algorithm 2.3.5

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = \text{prox}_f(x_3^k)$
$x_2^{k+1} = \text{prox}_g(2x_1^{k+1} - x_3^k)$
$x_3^{k+1} = x_3^k + x_2^{k+1} - x_1^{k+1}$
end for

Algorithm 2.3.6

for $k = 0, 1, 2, \ldots$ **do**
$\xi_1^{k+1} = \text{prox}_g(-\xi_1^k + 2\xi_2^k) + \xi_1^k - \xi_2^k$
$\xi_2^{k+1} = \text{prox}_f(\xi_1^{k+1})$
end for

The third example consists of algorithms 2.3.5 and 2.3.6. With suitable initialization, they will generate the same sequence of calls to the proximal operator, ignoring the very first call to one of the oracles. Specifically, algorithm 2.3.6 is initialized as $\xi_1^0 = x_3^0$, $\xi_2^0 = x_1^1$ and the first call to prox_f in algorithm 2.3.5 is ignored. We will say they are equivalent up to a prefix or shift: they satisfy *shift equivalence*.

Generalizing from these motivating examples, we will call algorithms equivalent when they generate an identical sequence (e.g., of states or oracle calls) up to some transformations, with suitable initialization. To make our ideas formal, we need a few definitions and some ideas from control theory. We will then revisit those motivating examples and define algorithm equivalence.

2.4 Preliminaries

We let $\mathbb{R}^n$ denote the standard Euclidean space of n-dimensional vectors, and use boldface lowercase symbols denote semi-infinite sequences of vectors, which

we index using superscripts. For example, we may write $\mathbf{x} := (x^0, x^1, \dots)$, where $x^k \in \mathbb{R}^n$ for each $k \geq 0$. Subscripts index components or subvectors: for example, we may write $x = \left[\begin{smallmatrix} x_1 \\ x_2 \end{smallmatrix}\right] \in \mathbb{R}^n$, where $x_1 \in \mathbb{R}^{n_1}$ and $x_2 \in \mathbb{R}^{n-n_1}$.

2.4.1 Optimization

Optimization problem, objective, and constraints An optimization problem is identified by an objective function and a constraint set. The objective may be written as the sum of several functions, and the constraint set may be the intersection of several sets. As an example, in the optimization problem eq. (2.1) [12]

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Ax + Bz = c, \end{array} \tag{2.1}$$

the objective function is $f(x) + g(z)$ and the constraint set is $\{(x, z) : Ax + Bz = c\}$.

Oracles We assume an oracle model of optimization: we can only access an optimization problem by querying oracles at discrete query points [13, §4; 15, §1; 76, §1]. Oracles might include the gradient or proximal operator of a function, or projection onto a constraint set [8, §6; 39, §2; 82, §1]. Each query to the oracle returns an output such as the function value, gradient, or proximal operator. For example, the oracles for problem eq. (2.1) might include the gradients or proximal operators of f and g, and projection onto the hyperplane $\{(x, z) : Ax + Bz = c\}$.

2.4.2 Algorithms

Detecting equivalence between *any* pair of algorithms is beyond the scope of this work. Instead, we restrict our attention to equivalence between iterative linear time invariant optimization algorithms. In the following section, we provide some intuition and define each of these terms. Further formalism of these terms will be provided in the next subsection on control theory.

Iterative algorithms Given an optimization problem and an initial point $x^0 \in \mathcal{X}$, an *iterative algorithm* $\mathcal{A}$ generates a sequence of points $\mathbf{x} := (x^k)_{k\geq 0}$ by repeated application of the map $\mathcal{A} : \mathcal{X} \to \mathcal{X}$. (We do not distinguish the algorithm from its associated map.) Hence, $x^{k+1} = \mathcal{A}(x^k)$ for $k \geq 0$. We call x^k the *state* of the algorithm at *time* k. We make two important simplifying assumptions when treating algorithms.

First, suppose the operator $\mathcal{A}$ calls each different oracle *exactly once*. (We will see how to extend our ideas to more complex algorithms later.) This assumption forbids trivial repetition, such as $\mathcal{A}' := \mathcal{A} \circ \mathcal{A}$. Second, we consider algorithms that are *time-invariant*. In general, one could envision an algorithm $\mathcal{A}^k$ that changes at each timestep. Such time-varying algorithms are common in practice: for example, gradient-based methods with diminishing stepsizes. We view time-varying algorithms as a scheme for switching between different time-invariant algorithms. Since our aim is to reason about algorithm equivalence, we restrict our attention to time-invariant algorithms. A nice benefit of this restriction is that we can define algorithm equivalence independently of the choice of initial point.

The formulation $x^{k+1} = \mathcal{A}(x^k)$ is general enough to include algorithms with multiple timesteps. For example consider algorithm 1.4: $x_1^{k+1} = x_1^k - \eta F(2x_1^k - x_1^{k-1})$. If we define the new state $x_2^k := x_1^{k-1}$ and let $x^k := \begin{bmatrix} x_1^k \\ x_2^k \end{bmatrix}$, then we may rewrite the algorithm as

$$x^{k+1} = \begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \end{bmatrix} = \begin{bmatrix} x_1^k - \eta F(2x_1^k - x_2^k) \\ x_1^k \end{bmatrix} = \mathcal{A}\left(\begin{bmatrix} x_1^k \\ x_2^k \end{bmatrix}\right) = \mathcal{A}(x^k). \tag{2.2}$$

The algorithm $\mathcal{A}$ contains a combination of oracle calls and state updates. Define y^k and u^k to be the input and output of the oracles called at time k, respectively. Now, write three separate equations for the state update, oracle input, and oracle output. Applying this to eq. (2.2), we obtain:

$$\begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1^k \\ x_2^k \end{bmatrix} + \begin{bmatrix} -\eta \\ 0 \end{bmatrix} u^k \qquad \text{(state update)}, \tag{2.3a}$$

$$y^k = \begin{bmatrix} 2 & -1 \end{bmatrix} \begin{bmatrix} x_1^k \\ x_2^k \end{bmatrix} \qquad \text{(oracle input)}, \tag{2.3b}$$

$$u^k = F(y^k) \qquad \text{(oracle output)}. \tag{2.3c}$$

Oracle sequence We have defined an algorithm $\mathcal{A}$ as a map $\mathcal{X} \to \mathcal{X}$. In optimization, it is also conventional to write an algorithm as a sequence of update equations, that are executed sequentially on a computer to implement the map. When this sequence of updates is executed, we may record the sequence of states or the sequence of oracle calls (oracle and its input pairs), which we call the oracle sequence. There may be several ways of writing the algorithm as a sequence of updates, which may produce different state sequences or oracle sequences. We are not aware of any practical algorithm for optimization that may be written to produce two different oracle sequences. Hence we will assume for

now that the oracle sequence produced by an algorithm is unique. [2] We will revisit this assumption later in section 2.7 to see how our ideas extend to more complex (not-yet-discovered) algorithms.

Linear algorithms The equations eq. (2.3) have the general *linear* form

$$x^{k+1} = Ax^k + Bu^k, \tag{2.4a}$$

$$y^k = Cx^k + Du^k, \tag{2.4b}$$

$$u^k = \phi(y^k). \tag{2.4c}$$

We say that a time-invariant algorithm is *linear* if it can be written in the form of eq. (2.4), where x^k is the algorithm state and ϕ is the set of oracles. Here ϕ can be any nonlinear map, including a map with internal state. For example, the oracle ϕ corresponding to the subgradient ∂f of a nondifferentiable function f might make a choice to ensure the output is unique and consistent, for example, by selecting the subgradient of minimum norm; the oracle ϕ corresponding to a stochastic gradient might include an internal random seed that ensures the output is unique and deterministic, given the seed.

In the rest of this chapter, unless specifically noted, our discussion is limited

[2] This assumption eliminates the possibility that some oracles may be permuted without changing the state sequence: e.g.,

Algorithm 2.4.1

for $k = 1, 2, \ldots$ **do**
  $x_1^{k+1} = \mathcal{A}_1(x_1^k)$
  $x_2^{k+1} = \mathcal{A}_2(x_2^k)$
  $x_3^{k+1} = \mathcal{A}_3(x_1^{k+1}, x_2^{k+1})$
end for

Algorithm 2.4.2

for $k = 1, 2, \ldots$ **do**
  $x_2^{k+1} = \mathcal{A}_2(x_2^k)$
  $x_1^{k+1} = \mathcal{A}_1(x_1^k)$
  $x_3^{k+1} = \mathcal{A}_3(x_1^{k+1}, x_2^{k+1})$
end for

Here, the algorithm may be equally well written with the oracle sequence $(\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ as with the oracle sequence $(\mathcal{A}_2, \mathcal{A}_1, \mathcal{A}_3)$. But again, we are not aware of any concrete examples of optimization algorithms with this structure.

to linear algorithms. We will see that the class of linear algorithms includes commonly used algorithms, such as accelerated methods, proximal methods, operator splitting methods, and more [54, 61].

The general form eq. (2.4) represents a convenient parameterization of linear algorithms in terms of matrices (A, B, C, D), but it is only a starting point. For example, algorithms 1.1 to 1.4 have different (A, B, C, D) parameters despite being equivalent algorithms. In the next section, we show how tools from control theory can be brought to bear on these sorts of representations.

Remark For an arbitrary state-space realization (A, B, C, D), the corresponding algorithmic sequence may not exist or may not be unique. However, any implementable practical algorithm, written as a sequence of update equations has a corresponding algorithmic sequence that exists and is unique: it is obtained by performing the steps indicated in the update equations and recording the values of x, u, and y.

2.4.3 Control theory

This subsection provides a brief overview of relevant methods and terminology from control theory. More detail can be found in standard references such as [5, Ch. 1–3] and [108, Ch. 1,2,5].

Algorithms as linear systems Let $\mathbf{u}$ denote the entire sequence of u^k and $\mathbf{y}$ denote the entire sequence of y^k. The equations in eq. (2.4) can be separated into two parts. Equations eq. (2.4a) and eq. (2.4b) define a map $\mathbf{H}$ from $\mathbf{u}$ to $\mathbf{y}$

compactly as $\mathbf{y} = \mathbf{H}\mathbf{u}$, while eq. (2.4c) defines a map $\mathbf{\Phi}$ from $\mathbf{y}$ to $\mathbf{u}$ as $\mathbf{u} = \mathbf{\Phi}\mathbf{y}$, where $\mathbf{\Phi} = \text{diag}\{\phi, \phi, \dots\}$. We can represent these algebraic relations visually via the *block-diagram* shown in fig. 2.1.

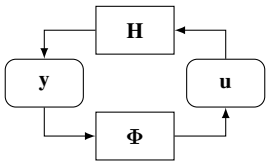

Figure 2.1: Block-diagram representation of an algorithm. This is equivalent to the pair of equations $\mathbf{y} = \mathbf{H}\mathbf{u}$ and $\mathbf{u} = \mathbf{\Phi}\mathbf{y}$.

Consider map $\mathbf{H}$ defined by eq. (2.4a) and eq. (2.4b). For simplicity, we assume that $x^0 = 0$. As we eliminate $\{x^1, \dots, x^k\}$ from eq. (2.4a) and eq. (2.4b), map $\mathbf{H}$ can be represented as a semi-infinite matrix,

$$\begin{bmatrix} y^0 \\ y^1 \\ y^2 \\ y^3 \\ \vdots \end{bmatrix} = \underbrace{\begin{bmatrix} D & 0 & 0 & 0 & \cdots \\ CB & D & 0 & 0 & \cdots \\ CAB & CB & D & 0 & \cdots \\ C(A)^2B & CAB & CB & D & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}}_{\mathbf{H}} \begin{bmatrix} u^0 \\ u^1 \\ u^2 \\ u^3 \\ \vdots \end{bmatrix}. \tag{2.5}$$

In control theory, map $\mathbf{H}$ is considered as a (discrete-time) *system* that maps a sequence of *inputs* $\mathbf{u}$ to a sequence of *outputs* $\mathbf{y}$. Map $\mathbf{H}$ is linear since it can be represented as a semi-infinite matrix. The matrix representation is lower-triangular and it indicates $\mathbf{H}$ is *causal*. Further, $\mathbf{H}$ is time-invariant because the matrix representation is (block) *Toeplitz*, which means that $\mathbf{H}$ is (block) constant along diagonals from top-left to bottom right. Thus, $\mathbf{H}$ is a *causal linear time-invariant system*. For the rest of this chapter, we will work with such systems and we will refer to such systems as *linear systems*.

Further, to combine maps $\mathbf{H}$ and Φ together, a linear algorithm in the form of eq. (2.4) can be regarded as a linear system connected in feedback with a nonlinearity shown by fig. 2.1. At time k, u^k is the input and y^k is the output of the system. Nonlinear feedback ϕ represents the set of oracles such as the gradient or subgradient of a convex function and it maps the output y^k to the input u^k.

State-space realization Reconsider equations eq. (2.4a) and eq. (2.4b). They correspond to the *state-space realization* of system $\mathbf{H}$. In control theory, a state-space realization is characterized by an internal sequence of *states* $\mathbf{x}$ that evolves according to a difference equation with parameters (A, B, C, D):

$$\begin{aligned} x^{k+1} &= Ax^k + Bu^k, \\ y^k &= Cx^k + Du^k, \end{aligned} \quad \text{or equivalently, } \begin{bmatrix} x^{k+1} \\ y^k \end{bmatrix} = L \begin{bmatrix} x^k \\ u^k \end{bmatrix}, \text{ where } L = \begin{bmatrix} A & B \\ C & D \end{bmatrix}. \quad (2.6)$$

Here, $u^k \in \mathbb{R}^m$, $y^k \in \mathbb{R}^p$, and $x^k \in \mathbb{R}^n$. The parameters (A, B, C, D) are matrices of compatible dimensions, so $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $C \in \mathbb{R}^{p\times n}$, and $D \in \mathbb{R}^{p\times m}$. The state-space realization corresponding to the system $\mathbf{H}$ can also be characterized by omitting all vectors and writing the block matrix L shown in eq. (2.6) (right), which is the map from (x^k, u^k) to (x^{k+1}, y^k).

In this work, we rely on such formalism that represents algorithms as linear systems using a state-space realization as eq. (2.6) for each algorithm, following [54, 61]. The state-space realization L represents the linear part of an algorithm and map ϕ represents the nonlinear part. Moreover, we have $\mathcal{A} = (L, \phi)$. In this way, we can unroll fig. 2.1 in time to obtain the block-diagram shown in fig. 2.2. Each dashed box in fig. 2.2 represents map $\mathcal{A}$ for each iteration.

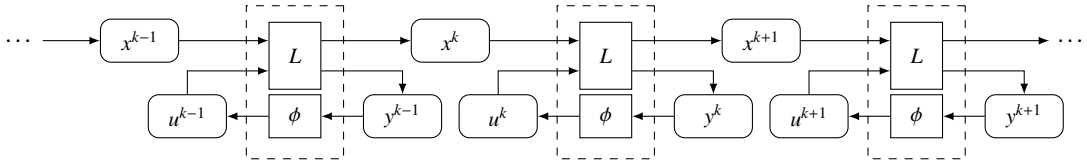

Figure 2.2: Unrolled-in-time block-diagram representation of an algorithm.

Impulse response and transfer function From eq. (2.5), without the assumption that $x^0 = 0$, we can obtain

$$y^k = C(A)^k x^0 + \sum_{j=0}^{k-1} C(A)^{k-(j+1)} B u^j + D u^k. \tag{2.7}$$

The output y^k is the sum of $C(A)^k x^0$, which is due to the initial condition x^0, and $\sum_{j=0}^{k-1} C(A)^{k-(j+1)} B u^j + D u^k$, which is due to the inputs $\{u^0, \ldots, u^k\}$. The compact form $\mathbf{y} = \mathbf{H}\mathbf{u}$ and its matrix representation eq. (2.5) omit the first term that depends on x^0. These representations are formally equivalent to the state-space model only when the state is initialized at $x^0 = 0$. However, linearity of $\mathbf{H}$ allows the two contributions to be studied separately:

$$\text{(total response)} = \underbrace{\text{(zero input response)}}_{\text{set } u^k = 0 \text{ for } k \geq 0} + \underbrace{\text{(zero state response)}}_{\text{set } x^0 = 0}.$$

This decomposition is analogous to writing the general solution to a linear differential (or difference) equation as the sum of a homogeneous solution (due to initial conditions only) and a particular solution (due to the non-homogeneous terms only). We will characterize linear systems by their input-output map. The input-output map depends only on the zero state response, which allows us to avoid details about initialization. For simplicity, we denote the entries in the matrix representation of $\mathbf{H}$ in eq. (2.5) as

$$H^k = \begin{cases} D & k = 0 \\ C(A)^{k-1} B & k \geq 1 \end{cases}. \tag{2.8}$$

To study the zero state response, recall from eq. (2.5) that

$$y^k = H^k u^0 + H^{k-1} u^1 + \cdots + H^1 u^{k-1} + H^0 u^k. \tag{2.9}$$

The sequence $(H^k)_{k\geq 0}$ is called the *impulse response* of $\mathbf{H}$, because it corresponds to the impulsive input $u^0 = 1$ and $u^j = 0$ for $j \geq 1$.

A convenient way to represent $\mathbf{H}$ is via the use of a *transfer function*. To this end, we can represent $\mathbf{y}$ and $\mathbf{u}$ as generating functions in the variable z^{-1}. Equating powers of z^{-1}, we have:

$$\underbrace{\left(y^0 + y^1 z^{-1} + y^2 z^{-2} + \cdots\right)}_{\hat{y}(z)} = \underbrace{\left(H^0 + H^1 z^{-1} + H^2 z^{-2} + \cdots\right)}_{\hat{H}(z)} \underbrace{\left(u^0 + u^1 z^{-1} + u^2 z^{-2} + \cdots\right)}_{\hat{u}(z)}. \tag{2.10}$$

We can recover eq. (2.9) by expanding the multiplication in eq. (2.10) and grouping terms with the same power of z^{-1}. So when written as generating functions, the output is related to the input via multiplication. The functions $\hat{y}$ and $\hat{u}$ are the z-transforms of the sequences $\mathbf{y}$ and $\mathbf{u}$, respectively, and $\hat{H}$ is called the *transfer function*. If $p \geq 2$ or $m \geq 2$ (the H^k are matrices), then $\hat{H}$ is called the *transfer matrix*.

Substituting eq. (2.8) into the definition of the transfer function, we can write a compact form for the formal power series $\hat{H}$, which converges on some appropriate set:

$$\hat{H}(z) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = D + \sum_{k=1}^{\infty} C(A)^{k-1} B z^{-k} = C(zI - A)^{-1} B + D. \tag{2.11}$$

The transfer function $\hat{H}(z) = C(zI - A)^{-1}B + D$ can be directly computed from the state-space matrices (A, B, C, D). Moreover, $\hat{H}(z)$ is a matrix whose entries are rational functions of z. Hence the transfer function provides a computationally efficient way to uniquely characterize the input-output map of a system. We

will use the block notation with solid lines to indicate transfer function as in eq. (2.11).

Linear transformations of state-space realizations Consider a linear transformation of the states x^k in eq. (2.6). Specifically, suppose $Q \in \mathbb{R}^{n\times n}$ is invertible, and define $\tilde{x}^k = Qx^k$ for each k. The new state-space realization in terms of the new variables $\tilde{x}^k$ is

$$\begin{aligned} \tilde{x}^{k+1} &= QAQ^{-1}\tilde{x}^k + QBu^k \\ y^k &= CQ^{-1}\tilde{x}^k + Du^k \end{aligned}, \qquad \tilde{L} = \begin{bmatrix} QAQ^{-1} & QB \\ CQ^{-1} & D \end{bmatrix}. \tag{2.12}$$

It is straightforward to check that $\mathbf{H}$ and $\tilde{\mathbf{H}}$ have the same transfer function. Therefore, whether we apply the linear system $\mathbf{H}$ or $\tilde{\mathbf{H}}$, the same input sequence $\mathbf{u}$ will produce the same output sequence $\mathbf{y}$, although the respective states x^k and $\tilde{x}^k$ will generally be different. So although the state-space realization (A, B, C, D) depend on the coordinates used to represent states x^k, the transfer function is invariant under linear transformations.

This invariance is the key to understanding when two optimization algorithms are the same, even if they look different as written. For example, this idea alone suffices to show that algorithms 2.3.1 and 2.3.2 are equivalent.

Minimal realizations Every set of appropriately-sized state-space parameters (A, B, C, D) produces a transfer matrix whose entries are rational functions of z. Closer inspection of the formula $\hat{H}(z) = C(zI - A)^{-1}B + D$ reveals that $\hat{H}(z) \to D$ as $z \to \infty$. Therefore, the rational entries of $\hat{H}(z)$ must be *proper*: the degree of the numerator cannot exceed the degree of the denominator. Moreover, the

degree of the common denominator of all entries of $\hat{H}(z)$ cannot exceed n (the size of the matrix A). Further, given any transfer matrix $\hat{H}(z)$ whose entries are proper, there exists at least one realization (A, B, C, D) whose transfer function is $\hat{H}(z)$. Any realization of $\hat{H}(z)$ for which the size of A is as small as possible is called *minimal*. All minimal realizations of $\hat{H}(z)$ are related by an invertible state transformation via a suitably chosen invertible matrix Q, as in eq. (2.12).

Realizations can be non-minimal when the transfer function has factors that cancel from both the numerator and denominator. For example, the following pair of state-space equations both have the same transfer function:

$$\hat{H}(z) = \left[\begin{array}{c|c} 1 & 1 \\ \hline 1 & 0 \end{array}\right] = 1 \cdot (z-1)^{-1} \cdot 1 = \frac{1}{z-1},$$

$$\hat{H}(z) = \left[\begin{array}{cc|c} 1 & 2 & 1 \\ 0 & 3 & 0 \\ \hline 1 & 6 & 0 \end{array}\right] = \begin{bmatrix} 1 & 6 \end{bmatrix} \begin{bmatrix} z-1 & -2 \\ 0 & z-3 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{z-3}{z^2-4z+3} = \frac{1}{z-1}.$$

We can detect when two optimization algorithms are equivalent, even when one has additional (redundant) state variables, by computing their minimal realizations. This strategy shows that algorithms 2.3.3 and 2.3.4 are equivalent.

Inverse of state-space realization Consider a state-space system $\mathbf{H}$ with realization eq. (2.6) and for which $m = p$ (input and output dimension are the same). Is it possible to find a state-space system $\mathbf{H}^{-1}$ that maps $\mathbf{y}$ back to $\mathbf{u}$? It turns out this is possible if and only if D is invertible. In this case, the transfer function of $\mathbf{H}^{-1}$ is $\hat{H}^{-1}(z)$, a matrix whose entries are rational functions of z. One possible

state-space realization of the inverse system $\mathbf{H}^{-1}$ is

$$\hat{H}^{-1}(z) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]^{-1} = \left[\begin{array}{c|c} A - BD^{-1}C & BD^{-1} \\ \hline -D^{-1}C & D^{-1} \end{array}\right].$$

This explicit realization can be obtained by applying the matrix inversion lemma to eq. (2.11). We can extend this idea to partial inverses of linear systems. Suppose the input sequence $\mathbf{u}$ is partitioned as

$$\mathbf{u} := (u^0, u^1, \dots) = \left(\begin{bmatrix} u_1^0 \\ u_2^0 \end{bmatrix}, \begin{bmatrix} u_1^1 \\ u_2^1 \end{bmatrix}, \dots\right) = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}, \quad \text{where } u_1^k \in \mathbb{R}^{m_1}, u_2^k \in \mathbb{R}^{m_2} \text{ for all } k \geq 0$$

and similarly for $\mathbf{y}$. The matrix D and transfer matrix $\hat{H}(z)$ can also be partitioned conformally as

$$D = \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} \quad \text{and} \quad \hat{H}(z) = \begin{bmatrix} \hat{H}_{11}(z) & \hat{H}_{12}(z) \\ \hat{H}_{21}(z) & \hat{H}_{22}(z) \end{bmatrix}, \quad \text{where } D_{ij} \in \mathbb{R}^{p_i \times m_j} \text{ and similarly for } \hat{H}(z). \tag{2.13}$$

If D_{11} is invertible, we can partially invert $\mathbf{H}$ with respect to $\mathbf{u}_1$ and $\mathbf{y}_1$ to form a new system $\mathbf{H}'$ that maps $(\mathbf{y}_1, \mathbf{u}_2) \mapsto (\mathbf{u}_1, \mathbf{y}_2)$. The transfer function $\hat{H}'(z)$ of the new system $\mathbf{H}'$ satisfies

$$\hat{H}'(z) = \begin{bmatrix} \hat{H}_{11}^{-1}(z) & -\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \\ \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z) & \hat{H}_{22}(z) - \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \end{bmatrix}. \tag{2.14}$$

A detailed proof of eq. (2.14) is presented in appendix A.1. Note that if D_{22} is invertible, we can perform a similar partial inverse with respect to the second component. When an optimization algorithm is related to another by conjugation of one of the function oracles, their transfer functions are related by (possibly partial) inversion.

2.5 Algorithm equivalence

We are now ready to revisit the motivating examples and formally define algorithm equivalence.

2.5.1 Assumptions

We now formally state the assumptions that we have discussed informally in section 2.4. We assume all algorithms throughout this chapter satisfy these assumptions unless specifically noted.

Assumption 2.5.1. The algorithm is causal, time-invariant, and linear.

Any algorithm satisfying assumption 2.5.1 can be implemented as a sequence of update equations (because it is causal) and can be written in form eq. (2.4) (because it is linear and time-invariant).

Assumption 2.5.2. Given an oracle ϕ, the oracle sequence produced by the algorithm is unique.

Assumption 2.5.2 follows if the output of ϕ is deterministic. It also follows if ϕ has internal state but is deterministic given the sequence of inputs to ϕ so far.

Two algorithms can only produce the same sequences if called on the same set of oracles (or on compatible oracles, for example, related by convex conjugacy). We say that two algorithms are *comparable* if they use the same or compatible oracles.

Assumption 2.5.3. When we compare two algorithms to detect equivalence or other relations, we assume that they are comparable.

We will discuss several kinds of compatible oracles in the sequel.

2.5.2 Oracle equivalence

In the first motivating example, the algorithms have the same number of states, and the state sequences are equivalent up to an invertible linear transformation. We call these algorithms *state-equivalent*.

In the second motivating example, the state sequence of algorithm 2.3.3 can be transformed into the state sequence of algorithm 2.3.4 with a linear transformation. However, unlike the first motivating example, the linear transformation is not invertible; indeed, algorithm 2.3.4 uses fewer state variables than algorithm 2.3.3. Instead, recall that the sequence of calls to the gradient oracle are identical for algorithms 2.3.3 and 2.3.4. Hence these algorithms are *oracle-equivalent*.

Definition 2.5.1. Two algorithms are oracle-equivalent on a set of optimization problems if, for any problem in the set and for all possible oracles, there exist initializations for both algorithms such that the two algorithms generate the same oracle sequence.

Oracle-equivalent algorithms generate identical sequence regardless of oracles. For example, if two oracle-equivalent algorithms both call oracle ∇f and

generate identical oracle sequence, they will still produce identical oracle sequence if we replace oracle ∇f to ∇g or every other possible oracle. Further, oracle equivalence is a symmetric relation. Notice that if the oracle sequences (that is, the oracles and their arguments y^k) are the same, then the oracles produce the same inputs u^k for the linear systems of each algorithm. Hence, as shown in fig. 2.3, oracle-equivalent algorithms have matching input $\mathbf{u}$ and output $\mathbf{y}$ sequences. The solid double-sided arrow indicates the sequences y^k and $\tilde{y}^k$ are identical, and the sequences u^k and $\tilde{u}^k$ are identical.

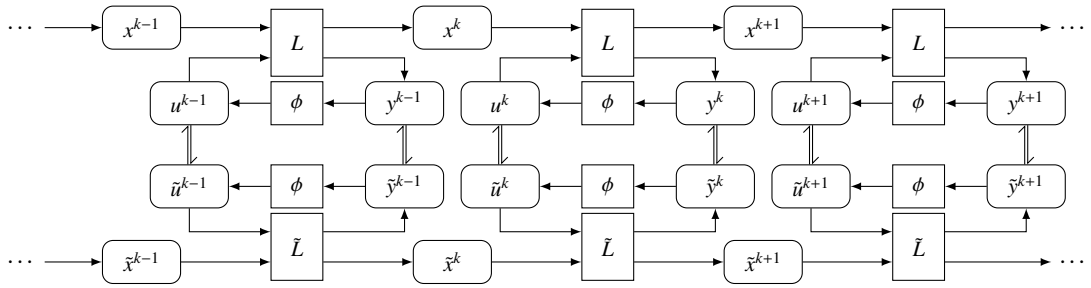

Figure 2.3: Unrolled block-diagram representation of oracle equivalence.

Further, since oracle-equivalent algorithms have identical input and output sequences, many analytical properties of interest, particularly those pertaining to algorithm convergence or robustness, are preserved. For example, suppose the target problem is to minimize $f(x)$ with $x \in R^n$, with solution $x^\star$ and corresponding objective value $f(x^\star)$. Further suppose f is convex and differentiable with oracle ∇f. If two algorithms are oracle-equivalent, the sequence of gradients $\|\nabla f(x)\|$, distance to the solution $\left\|x - x^\star\right\|$, and objective function values $\|f(x) - f(x^\star)\|$ evolve identically, so they have the same worst-case convergence, etc: the gradient sequence and objective value are controlled by the oracle sequence. Moreover, even if the oracle is noisy (e.g., suffers from additive or multiplicative noise, or even adversarial noise), from the point of view of the oracle, the algorithms are indistinguishable and any analytical property that involves

only the oracle sequence will be the same.

2.5.3 Shift equivalence

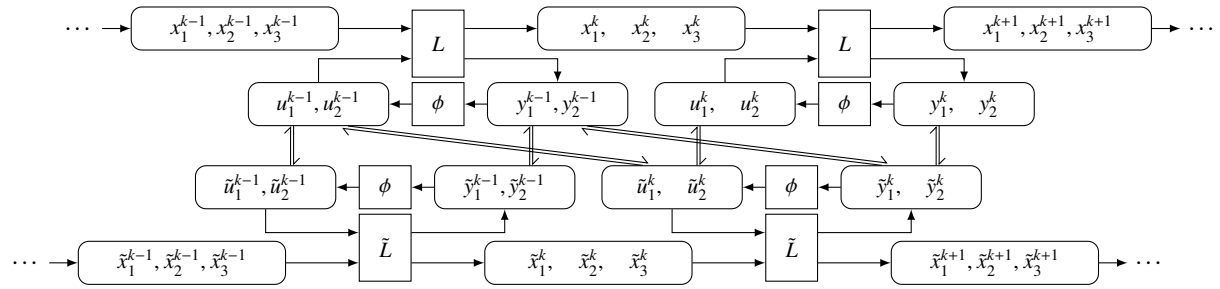

Figure 2.4: Unrolled block-diagram representation of shift equivalence.

Now consider algorithms 2.3.5 and 2.3.6 from the third motivating example. They are not oracle-equivalent. However, their input and output sequences become identical after shifting algorithm 2.3.5 one step backward: these algorithms are *shift-equivalent*.

Definition 2.5.2. Two algorithms are shift-equivalent on a set of problems if, for any problem in the set and for all possible oracles, there exist initializations for both algorithms such that the oracle sequences match up to a prefix.

Shift equivalence can also be interpreted as oracle equivalence up to a shift. We depict shift equivalence graphically in fig. 2.4. Conversely, oracle equivalence can be regarded as a special case of shift equivalence, where the oracle sequences match without any shift. Besides, similar as oracle equivalence, shift equivalence is also symmetric.

2.5.4 Discussion

One algorithm, many interpretations Is it useful to have many different forms of an algorithm, if all the forms are (oracle- or shift-)equivalent? Yes: different rewritings of one algorithm often yield different ("physical") intuition. For example, algorithm 1.1 uses the current loss function for extrapolation [106]; while algorithm 1.2 seems to extrapolate from the previous loss function [19]. Equivalent algorithms can differ in memory usage, computational efficiency, or numerical stability. For example, implementations of algorithms 1.3 and 1.4 lead to different memory usage [25, 67]. In each time step k, algorithm 1.3 needs to store x_2^k, x_2^{k+1} and $F^k(\cdot)$, but algorithm 1.4 only needs to store x_1^k and x_1^{k+1} in memory. These different rewritings also naturally yield different generalizations, for example, by projecting different state variables.

Limitations Do these formal notions of equivalence capture everything an optimization expert might mean by "equivalent algorithms"? No: an example is shown in algorithm 2.5.1. Algorithms 2.5.1 and 2.3.4 are related by a nonlinear state transformation, $x^k = \exp(\xi^k)$. However, none of the equivalences we have discussed capture this example. The difficulty is that algorithm 2.5.1 is a nonlinear algorithm, while all of our machinery for detecting algorithm equivalence requires linearity. While notions of nonlinear equivalence are certainly interesting, in this chapter we will define only those types of equivalence that our framework can detect.

Algorithm 2.5.1

for $k = 1, 2, \ldots$ **do**
$x^{k+1} = x^k \exp(-\frac{1}{5}\nabla f(\log x^k))$
end for

2.6 A characterization of oracle equivalence

In this section, we will discuss how to characterize oracle equivalence via transfer functions. Recall that oracle equivalence, introduced in section 2.5, characterizes an algorithm by its oracle sequence. This sequence is uniquely determined by the initialization of the algorithm (which we ignore) and the input-output map of the linear system representing the algorithm. While the state-space realization of two equivalent algorithms may differ, from section 2.4.3, recall that the transfer function of a linear system uniquely characterizes the system as an input-output map. Fortunately, using eq. (2.11), we can directly calculate the transfer function from the state-space realization of an algorithm; and we can use equality of transfer functions to check if two algorithms are equivalent. This machinery allows us to avoid the issue of initialization (or of the optimization problem!) entirely, as we can check algorithm equivalence without ever producing a sequence of iterates.

More formally, consider two oracle-equivalent algorithms with the same number of oracle calls in each iteration. From section 2.5.2, we know that for every optimization problem, and for all possible oracles, there exist initializations for both algorithms so that the oracle sequence of the two algorithms is the same. Concretely, by picking the initializations of both algorithms appropriately, we can ensure that the first output of the linear systems match. Hence

(since the oracles are the same), the first input of the linear systems match, and so the second output of the linear systems match, etc. By induction, for each possible sequence of input $\mathbf{u}$, they produce identical sequences of output $\mathbf{y}$. Then from section 2.4.3, the algorithms must have identical impulse responses and consequently identical transfer functions. In light of the previous discussion, we have proved the following proposition, since each step in the reasoning above is necessary and sufficient. We defer a detailed mathematical proof to appendix A.2.

Proposition 2.6.1. Algorithms with the same oracle calls in each iteration are oracle-equivalent if and only if they have identical transfer functions.

Importantly, oracle-equivalent algorithms have the same transfer function, even if they have a different number of state variables. But any realization of the algorithm must have at least as many state variables as the minimal realization of the linear system.

Remark It is meaningless to compare algorithms with different oracle calls, as two algorithms are oracle equivalent if there exist initializations for both algorithms such that they generate the same oracle sequence. Hence throughout this section, we make assumption 2.5.3: when we compare two algorithms, we assume both algorithms use the same set of oracles. In this case, by section 2.4.3, we can always initialize both algorithms at zero to satisfy the requirement of oracle equivalence. For any algorithm that involves constant terms in its state-space realization, we can affinely transform it into an equivalent state-space realization without constant terms. Under this affine transformation, zero still satisfies the requirements of initialization for oracle-equivalence. This justifies our

approach to characterize oracle equivalence with transfer functions and ignore the initializations. Further, from eq. (2.7), the effect of initialization diminishes as time step goes to infinity, thus, asymptotically initialization does not affect the behavior of an algorithm such as convergence properties.

Oracle-equivalent algorithms have identical oracle sequences and hence converge to the same fixed point (if they converge). Suppose algorithm $\mathcal{A}_1 : \mathcal{X} \to \mathcal{X}$ with (nonlinear) oracle $\phi : \mathcal{X} \to \mathcal{X}$ and state-space realization (A_1, B_1, C_1, D_1), converges to a fixed point $(y^\star, u^\star, x^\star)$ that satisfies

$$\begin{aligned} x^\star &= A_1 x^\star + B_1 u^\star \\ y^\star &= C_1 x^\star + D_1 u^\star \\ u^\star &= \phi(y^\star). \end{aligned} \tag{2.15}$$

If algorithm $\mathcal{A}_2$ is oracle-equivalent to $\mathcal{A}_1$, $\mathcal{A}_2$ converges to a fixed point $(y^\star, u^\star, \tilde{x}^\star)$ that has the same output and input as the fixed point of $\mathcal{A}_1$; however, the state $\tilde{x}^\star$ may not be the same, or even have the same dimension.

Further, if there is an invertible linear map Q between the states of $\mathcal{A}_1$ and $\mathcal{A}_2$ and $(y^\star, u^\star, x^\star)$ is a fixed point of $\mathcal{A}_1$, then $(y^\star, u^\star, Qx^\star)$ is a fixed point of $\mathcal{A}_2$. We can use this fact to derive a relation between the state-space realizations of the two algorithms: the fixed point equation for $\mathcal{A}_2$ can be written as

$$\begin{aligned} Qx^\star &= QA_1Q^{-1}Qx^\star + QB_1u^\star \\ y^\star &= C_1Q^{-1}Qx^\star + D_1u^\star \\ u^\star &= \phi(y^\star), \end{aligned} \tag{2.16}$$

which shows that the state-space realization of $\mathcal{A}_2$ is

$$\begin{bmatrix} QA_1Q^{-1} & QB_1 \\ C_1Q^{-1} & D_1 \end{bmatrix}, \tag{2.17}$$

which can be obtained by eq. (2.12).

2.6.1 Motivating examples: proof of equivalence

Now, we will revisit the first and second motivating examples and apply proposition 2.6.1 to show equivalence. We perform the computation using the gradient oracle (∇f) as the oracle to compute the state-space realizations and transfer functions.

Algorithms 2.3.1 and 2.3.2 The state-space realization and transfer function of algorithm 2.3.1 are shown as

$$\hat{H}_1(z) = \left[\begin{array}{cc|c} 2 & -1 & -\frac{1}{10} \\ 1 & 0 & 0 \\ \hline 2 & -1 & 0 \end{array}\right] = \begin{bmatrix} 2 & -1 \end{bmatrix} \left(zI - \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} -\frac{1}{10} \\ 0 \end{bmatrix} = \frac{-2z+1}{10(z-1)^2}.$$

The state-space realization and the transfer function of algorithm 2.3.2 are

$$\hat{H}_2(z) = \left[\begin{array}{cc|c} 1 & -1 & -\frac{1}{5} \\ 0 & 1 & \frac{1}{10} \\ \hline 1 & 0 & 0 \end{array}\right] = \begin{bmatrix} 1 & 0 \end{bmatrix} \left(zI - \begin{bmatrix} 1 & -1 \\ 0 & 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} -\frac{1}{5} \\ \frac{1}{10} \end{bmatrix} = \frac{-2z+1}{10(z-1)^2}.$$

Hence we see algorithms 2.3.1 and 2.3.2 have the same transfer function, so by proposition 2.6.1 they are oracle-equivalent. In fact, since the algorithms have the same number of state variables, there exists an invertible linear transformation

$$Q = \begin{bmatrix} 2 & -1 \\ -1 & 1 \end{bmatrix}$$

to convert the state-space realization of algorithm 2.3.1 to the state-space realization of algorithm 2.3.2 following eq. (2.12).

Algorithms 2.3.3 and 2.3.4 The state-space realization and transfer function of algorithm 2.3.3 are

$$\hat{H}_3(z) = \left[\begin{array}{cc|c} 3 & -2 & \frac{1}{5} \\ 1 & 0 & 0 \\ \hline -1 & 2 & 0 \end{array}\right] = \begin{bmatrix} -1 & 2 \end{bmatrix} \left(zI - \begin{bmatrix} 3 & -2 \\ 1 & 0 \end{bmatrix} \right)^{-1} \begin{bmatrix} \frac{1}{5} \\ 0 \end{bmatrix} = -\frac{1}{5(z-1)}.$$

The state-space realization and transfer function of algorithm 2.3.4 are

$$\hat{H}_4(z) = \left[\begin{array}{c|c} 1 & -\frac{1}{5} \\ \hline 1 & 0 \end{array}\right] = \begin{bmatrix} 1 \end{bmatrix} \left(zI - \begin{bmatrix} 1 \end{bmatrix} \right)^{-1} \begin{bmatrix} -\frac{1}{5} \end{bmatrix} = -\frac{1}{5(z-1)}.$$

Algorithms 2.3.3 and 2.3.4 have the same transfer function, so by proposition 2.6.1 they are oracle-equivalent. On the other hand, they have different numbers of states. Consider the invertible linear transformation

$$Q = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix}.$$

Applying Q to the state-space realization of algorithm 2.3.3 leads to

$$\left[\begin{array}{cc:c} 1 & 0 & -\frac{1}{5} \\ -1 & 2 & 0 \\ \hdashline 1 & 0 & 0 \end{array}\right],$$

where we have used dashed lines to demarcate the blocks in the state-space realization. This has the same minimal realization as algorithm 2.3.4 by section 2.4.3.

$$\left[\begin{array}{c:c} 1 & -\frac{1}{5} \\ \hdashline 1 & 0 \end{array}\right].$$

Note that the state-space realization of algorithm 2.3.4 is a minimal realization. This shows the reason why algorithms 2.3.3 and 2.3.4 are equivalent even if they have different numbers of states.

Now we show how the sausage was made. Algorithm 2.3.3 was designed by starting with the more complex *Triple momentum algorithm* algorithm 2.6.1 [61, 104] and choosing parameters of the algorithm so its transfer function matched algorithm 2.3.4.

Algorithm 2.6.1 Triple momentum algorithm

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = (1+\beta)x_1^k - \beta x_2^k - \alpha \nabla f((1+\eta)x_1^k - \eta x_2^k)$
$x_2^{k+1} = x_1^k$
end for

The state-space realization and transfer function of algorithm 2.6.1 are

$$\hat{H}_7(z) = \left[\begin{array}{cc|c} 1+\beta & -\beta & -\alpha \\ 1 & 0 & 0 \\ \hline 1+\eta & -\eta & 0 \end{array}\right] = -\frac{\alpha((\eta+1)z - \eta)}{(z-1)(z-\beta)}. \tag{2.18}$$

We now demand that eq. (2.18)(right) must equal the transfer function of algorithm 2.3.4 for all values of z, resulting in the equations

$$\begin{aligned} 5\alpha(\eta+1) &= 1 \\ 5\alpha\eta &= \beta. \end{aligned} \tag{2.19}$$

We solve for the parameters α, η and β to find a solution $\alpha = -\frac{1}{5}$, $\beta = 2$ and $\eta = -2$ to eq. (2.19) that corresponds to algorithm 2.3.3. Other solutions exist: for example, $\alpha = 1$, $\beta = -4$ and $\eta = -\frac{4}{5}$ solves eq. (2.19) and yields another (different!) algorithm equivalent to algorithm 2.3.4.

2.7 A characterization of shift equivalence

We can also characterize shift equivalence using transfer functions. Suppose an algorithm uses more than one oracle, and the call to the second oracle depends

on the value of the first. Take algorithm 2.3.5 as example: at iteration k, the first update equation calls the oracle prox_f to compute $x_1^{k+1} = \text{prox}_f(x_3^k)$, and the second update equation calls the oracle prox_g to compute $x_2^{k+1} = \text{prox}_g(2x_1^{k+1} - x_3^k)$. This second update relies on the value of x_1^{k+1}. Imagine now that we reorder the update equations by some permutation. Generally this change produces an entirely different algorithm. But if the permutation is a *cyclic* permutation, the order of the oracle calls is preserved. In the example of algorithm 2.3.5, we could start with the update equation $x_2^{k+1} = \text{prox}_g(2x_1^{k+1} - x_3^k)$ and produce exactly the same sequence of oracle calls (after the first) by initializing x_1^{k+1} and x_3^k appropriately. This new algorithm is shift-equivalent to algorithm 2.3.5 by definition 2.5.2.

Algorithm 2.3.5 has three update equations, and so there are two other algorithms that may be produced by cyclic permutations of algorithm 2.3.5, shown below as algorithms 2.7.1 and 2.7.2.

Algorithm 2.7.1

for $k = 0, 1, 2, \ldots$ **do**
$x_2^{k+1} = \text{prox}_g(2x_1^k - x_3^k)$
$x_3^{k+1} = x_3^k + x_2^{k+1} - x_1^k$
$x_1^{k+1} = \text{prox}_f(x_3^{k+1})$
end for

Algorithm 2.7.2

for $k = 0, 1, 2, \ldots$ **do**
$x_3^{k+1} = x_3^k + x_2^k - x_1^k$
$x_1^{k+1} = \text{prox}_f(x_3^{k+1})$
$x_2^{k+1} = \text{prox}_g(2x_1^{k+1} - x_3^{k+1})$
end for

Both are shift-equivalent to algorithm 2.3.5, but algorithm 2.7.2 is also oracle-equivalent to algorithm 2.3.5. (We will revisit and formally prove this result later.) It is easy to see why: the oracles prox_f and prox_g are called in the same order in algorithms 2.7.2 and 2.3.5, but in the opposite order in algorithm 2.7.1.

We introduce notation to generalize this idea to more complex algorithms.

Consider an algorithm $\mathcal{A}$ that consists of m update equations and makes n sequential oracle calls in each iteration. We insist that no update equation may contain more than one oracle call, so $m \geq n$. At iteration k, the algorithm generates states $x_1^k, \ldots, x_m^k$, outputs $y_1^k, \ldots, y_n^k$, and inputs $u_1^k, \ldots, u_n^k$, respectively. Consider any permutation $\tilde{\pi}$ of the sequence $(m) = (1, \ldots, m)$. We call algorithm $\mathcal{B} = P_{\tilde{\pi}}\mathcal{A}$ a *permutation* of algorithm $\mathcal{A}$ if $\mathcal{B}$ performs the update equations of $\mathcal{A}$ in the order $\tilde{\pi}$ at each iteration. The algorithms $\mathcal{A}$ and $\mathcal{B}$ are shift-equivalent if and only if $\tilde{\pi}$ is a cyclic permutation of (m).

Proposition 2.7.1. An algorithm and any of its cyclic permutations are shift-equivalent. Any two shift-equivalent algorithms are equivalent to cyclic permutations of each other.

Proof. We provide a proof sketch here, and defer a detailed proof to appendix A.3. Let us name the oracle calls of the original algorithm $\mathcal{A}$ so that the oracles are called in order (n).

Cyclic permutation implies shift equivalence. Suppose $\mathcal{B} = P_{\tilde{\pi}}\mathcal{A}$ where $\tilde{\pi}$ is a cyclic permutation of (m). The permutation of update equations may reorder the oracle calls within one iteration, so that the oracle calls in algorithm $\mathcal{B}$ follow a cyclic permutation π of (n) (possibly, the identity). Hence $\mathcal{A}$ and $\mathcal{B}$ are shift-equivalent. (If the permutation is the identity, then the algorithms are also oracle-equivalent.)

Shift equivalence implies cyclic permutation. Suppose algorithms $\mathcal{A}$ and $\mathcal{B}$ are shift-equivalent. If they are also oracle-equivalent, then they can be written using the same set of update equations. If they are not oracle-equivalent, we can always find a cyclic permutation of the update equations of $\mathcal{A}$ that produces the

same oracle sequence as $\mathcal{B}$. Therefore $\mathcal{A}$ and $\mathcal{B}$ are equivalent to cyclic permutations of each other. (In the first case, the permutation is the identity.) □

2.7.1 Reordering oracle calls

Most optimization algorithms proceed by sequential updates, each of which depends on the previous update. However, for completeness, we consider a more general class of equivalences that arises for algorithms whose oracle updates have a more complex dependency structure. We may express the order of oracle calls at each iteration using a directed graph, where the graph has edge from oracle i to oracle j if oracle call j depends on the result of oracle call i (within the same iteration). In other words, within the iteration we must call oracle i before oracle j. We call this directed graph the *oracle dependence graph* (ODG) of the algorithm.

An example is provided below as algorithm 2.7.3. Note that we are not aware of any practical algorithm for optimization with this ODG. It is constructed only for illustration.

Algorithm 2.7.3

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = x_4^k - t\nabla f(x_4^k)$
$x_2^{k+1} = x_1^{k+1} - t\nabla g(x_1^{k+1})$
$x_3^{k+1} = x_1^{k+1} - t\nabla h(x_1^{k+1})$
$x_4^{k+1} = \mathrm{prox}_{tf}(\frac{1}{2}x_2^{k+1} + \frac{1}{2}x_3^{k+1})$
end for

Algorithm 2.7.4

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = x_4^k - t\nabla f(x_4^k)$
$x_3^{k+1} = x_1^{k+1} - t\nabla h(x_1^{k+1})$
$x_2^{k+1} = x_1^{k+1} - t\nabla g(x_1^{k+1})$
$x_4^{k+1} = \mathrm{prox}_{tf}(\frac{1}{2}x_2^{k+1} + \frac{1}{2}x_3^{k+1})$
end for

Figure 2.5 expresses the dependency of oracle calls within each iteration of

algorithm 2.7.3. At each iteration, oracle calls 2 (∇g) and 3 (∇h) depends on the result of oracle call 1 (∇f); oracle call 4 (prox_{tf}) depends on the results of oracle calls 1, 2, and 3.

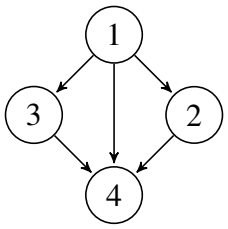

Figure 2.5: Directed graph representing dependency of oracle calls in algorithm 2.7.3.

An algorithm is always written as a sequence of update equations. But some algorithms might have a directed graph that may be written as a sequence (with all edges pointing forward) in more than one way, and so can be implemented as a sequence of oracle calls in more than one way. For illustration, consider algorithms 2.7.3 and 2.7.4. At each iteration, the oracle calls of algorithms 2.7.3 and 2.7.4 are identical: that is, calls to oracles ∇f, ∇g, ∇h, and prox_{tf} are identical. The only difference is that the oracle calls ∇g and ∇h are swapped in the oracle sequence at each iteration. Notice that the state-space realizations of these algorithms *still* have the same transfer function (after swapping the second and third columns and rows), consistent with the fact that algorithms 2.7.3 and 2.7.4 share the same directed graph of oracle calls (fig. 2.5).

We know of no practical optimization algorithm like this. However, were one to be discovered, we would suggest an expanded definition of oracle equivalence: two algorithms are oracle-equivalent if there exists a way of writing each algorithm as a sequence of updates so that both algorithms have the same sequence of oracle calls. The transfer function still identifies algorithms that are oracle-equivalent in this expanded sense.

The oracle calls in an algorithm at each iteration are always written in sequential form. This sequential form is lost in the state-space realization of the algorithm. However, the order (dependency) of oracle calls is encoded in the D matrix of the state-space realization. In this sense, the D matrix encodes the adjacency matrix of the directed graph. We have $D_{ij} \neq 0$ if and only if oracle call i depends on the results of oracle call j at each iteration. For example, in the state-space realization of algorithm 2.7.3, the D matrix is

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ -t & 0 & 0 & 0 \\ -t & 0 & 0 & 0 \\ -t & -\frac{1}{2}t & -\frac{1}{2}t & 0 \end{bmatrix}.$$

In light of this discussion, we can strengthen proposition 2.7.1 to proposition 2.7.2.

Proposition 2.7.2. An algorithm and any of its cyclic permutations are shift-equivalent; further, if they share the same D matrix in their state-space realizations, they are also oracle-equivalent. Any two shift-equivalent algorithms are equivalent to cyclic permutations of each other.

If an algorithm contains m update equations and n oracle calls at each iteration ($m \geq n$), there are m possible cyclic permutations on the update equations. According to the D matrix in the state-space realization, we can group the m cyclic permutations into n distinct equivalent classes. Algorithms within each equivalence class are oracle-equivalent and shift-equivalent, while algorithms in different equivalent classes are only shift-equivalent. The n distinct equivalence classes correspond to the n cyclic permutations of the original order of oracle calls (n).

2.7.2 Characterization of cyclic permutation

In the remainder of this chapter, let us restrict our attention to algorithms for which a (cyclic) permutation of the algorithm changes the update order of oracle calls within one iteration, or in other words, changes the D matrix in the state-space realization. In this way, we call algorithm $\mathcal{B} = P_\pi \mathcal{A}$ a permutation of algorithm $\mathcal{A}$ if $\mathcal{B}$ performs the update equations of $\mathcal{A}$ in a different order such that the update order of oracle calls of $\mathcal{B}$ is π at each iteration.

Suppose $\mathcal{A}$ has state-space realization (A, B, C, D), and $\mathcal{B} = P_\pi \mathcal{A}$ where $\pi = (j+1, \ldots, n, 1, \ldots, j)$ for $1 < j < n$ is a cyclic permutation of (n). We will show how to recognize this relationship between the algorithms using their transfer functions. Partition the oracle calls into two parts, $(1, \ldots, j)$ and $(j+1, \ldots, n)$, and partition the input and output sequences in the same way: $\bar{\mathbf{u}}_1$, $\bar{\mathbf{u}}_2$ for inputs and $\bar{\mathbf{y}}_1$, $\bar{\mathbf{y}}_2$ for outputs. The state-space realization $L_\mathcal{A}$ and transfer function $\hat{H}_\mathcal{A}(z)$ can also be partitioned accordingly as

$$L_\mathcal{A} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right], \tag{2.20}$$

$$\hat{H}_\mathcal{A}(z) = \begin{bmatrix} C_1(zI - A)^{-1}B_1 + D_{11} & C_1(zI - A)^{-1}B_2 + D_{12} \\ C_2(zI - A)^{-1}B_1 + D_{21} & C_2(zI - A)^{-1}B_2 + D_{22} \end{bmatrix} = \begin{bmatrix} \hat{H}_{11}(z) & \hat{H}_{12}(z) \\ \hat{H}_{21}(z) & \hat{H}_{22}(z) \end{bmatrix}.$$

Now we can say how the transfer function of an algorithm is related to that of its cyclic permutation. Recall that by assumption 2.5.3, when we compare transfer functions to detect shift equivalence (or cyclic permutations), both algorithms call the same set of oracles in each iteration.

Proposition 2.7.3. Instate notation as in eq. (2.20) and assume $D_{12} = 0$. Then $\mathcal{B}$ is equivalent to $P_\pi \mathcal{A}$ if and only if the transfer function of $\mathcal{B}$ satisfies

$$\hat{H}_{\mathcal{B}}(z) = \begin{bmatrix} \hat{H}_{11}(z) & z\hat{H}_{12}(z) \\ \hat{H}_{21}(z)/z & \hat{H}_{22}(z) \end{bmatrix}. \tag{2.21}$$

Proof. We provide a proof sketch here and defer a detailed proof to appendix A.4. The state-space realization of $P_\pi \mathcal{A}$ is

$$\left[\begin{array}{cc:cc} A & B_1 & 0 & B_2 \\ 0 & 0 & I & 0 \\ \hdashline C_1A & C_1B_1 & D_{11} & C_1B_2 \\ C_2 & D_{21} & 0 & D_{22} \end{array}\right]. \tag{2.22}$$

From the state-space realization, we may compute the transfer function as

$$\hat{H}_{\mathcal{B}}(z) = \begin{bmatrix} C_1(zI - A)^{-1}B_1 + D_{11} & zC_1(zI - A)^{-1}B_2 \\ C_2(zI - A)^{-1}B_1/z + D_{21}/z & C_2(zI - A)^{-1}B_2 + D_{22} \end{bmatrix} = \begin{bmatrix} \hat{H}_{11}(z) & z\hat{H}_{12}(z) \\ \hat{H}_{21}(z)/z & \hat{H}_{22}(z) \end{bmatrix}.$$

Finally, note two algorithms are equivalent if and only if they have identical transfer functions by proposition 2.6.1. □

We have assumed that $D_{12} = 0$ for algorithm $\mathcal{A}$. This assumption is quite weak. In fact, D_{12} must be 0 for any algorithm $\mathcal{A}$ that can be represented as a causal linear time-invariant system. Here, causal means that we can implement the algorithm by calling state update equations sequentially. To see this, suppose the state update equations have been arranged in this order, and use eq. (2.5) to write down the matrix representation of the infinite dimensional map

H that maps input **u** to output **y** corresponding to $\mathcal{A}$ as eq. (2.23):

$$\mathbf{H} = \begin{bmatrix} D_{11} & D_{12} & 0 & 0 & 0 & 0 & \cdots \\ D_{21} & D_{22} & 0 & 0 & 0 & 0 & \cdots \\ C_1B_1 & C_1B_2 & D_{11} & D_{12} & 0 & 0 & \cdots \\ C_2B_1 & C_2B_2 & D_{21} & D_{22} & 0 & 0 & \cdots \\ C_1AB_1 & C_1AB_2 & C_1B_1 & C_1B_2 & D_{11} & D_{12} & \cdots \\ C_2AB_1 & C_2AB_2 & C_2B_1 & C_2B_2 & D_{21} & D_{22} & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots \end{bmatrix}. \tag{2.23}$$

We can see that map **H** is (block) Toeplitz. Further, if algorithm $\mathcal{A}$ is causal, map **H** must be lower-triangular, and so D_{12} must be 0.

By causality, at each iteration the former oracle calls must be independent with the latter oracle calls while the latter calls can depend on the former calls. This indicates that there are no directed cycles in the directed graph representing oracle calls at each iteration for any causal algorithm. In other words, the graph is a directed acyclic graph (DAG). This is consistent with the fact that any causal algorithm has a lower-triangular D matrix (lower-triangular adjacency matrix of the directed graph).

Note that algorithms are not always written with state update equations ordered causally: for example, the state-space realization eq. (2.22) has a non-zero D_{12} block. However, we may reorder these equations so that each equation depends only on previously-computed quantities to reveal that the iteration is causal; after this rearrangement, the new D_{12} block is 0. We discuss permutations further in appendix A.5.

The fixed points of an algorithm and its cyclic permutations are the same up to a permutation, as stated by proposition 2.7.4.

Proposition 2.7.4. If algorithm $\mathcal{A}$ converges to a fixed point $(\bar{y}_1^\star, \bar{y}_2^\star, \bar{u}_1^\star, \bar{u}_2^\star, x^\star)$, then its cyclic permutation $P_\pi \mathcal{A}$ converges to fixed point $(\bar{y}_2^\star, \bar{y}_1^\star, \bar{u}_2^\star, \bar{u}_1^\star, x^\star)$.

A detailed proof is provided in appendix A.6.

2.7.3 Applications: proof of shift equivalence

Algorithms 2.3.5 and 2.3.6 Now, we can revisit algorithms 2.3.5 and 2.3.6 in the third motivating example and show that they are shift-equivalent. Here the oracles of algorithms 2.3.5 and 2.3.6 are prox_f and prox_g. The transfer function of algorithm 2.3.5 is

$$\hat{H}_5(z) = \left[\begin{array}{ccc|cc} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 2 & 0 \end{array}\right] = \left[\begin{array}{cc} -\frac{1}{z-1} & \frac{1}{z-1} \\ \frac{2z-1}{z-1} & -\frac{1}{z-1} \end{array}\right].$$

The transfer functions of algorithm 2.3.6 is

$$\hat{H}_6(z) = \left[\begin{array}{cc|cc} 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ \hline 1 & -1 & 0 & 1 \\ -1 & 2 & 0 & 0 \end{array}\right] = \left[\begin{array}{cc} -\frac{1}{z-1} & \frac{z}{z-1} \\ \frac{2z-1}{z(z-1)} & -\frac{1}{z-1} \end{array}\right].$$

From propositions 2.7.1 and 2.7.3, we know that they are shift-equivalent and equivalent up to a cyclic permutation.

Algorithms 2.7.1 and 2.7.2 Here we revisit algorithms 2.7.1 and 2.7.2 at the beginning of this chapter and show their relations with algorithm 2.3.5. The

oracles are prox_f and prox_g. The transfer function of algorithm 2.7.1 is

$$\hat{H}_8(z) = \left[\begin{array}{ccc|cc} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 1 \\ \hline -1 & 0 & 1 & 0 & 1 \\ 2 & 0 & -1 & 0 & 0 \end{array}\right] = \left[\begin{array}{cc} -\frac{1}{z-1} & \frac{z}{z-1} \\ \frac{2z-1}{z(z-1)} & -\frac{1}{z-1} \end{array}\right].$$

The transfer function of algorithm 2.7.2 is

$$\hat{H}_9(z) = \left[\begin{array}{ccc|cc} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ -1 & 1 & 1 & 0 & 0 \\ \hline -1 & 1 & 1 & 0 & 0 \\ 1 & -1 & -1 & 2 & 0 \end{array}\right] = \left[\begin{array}{cc} -\frac{1}{z-1} & \frac{1}{z-1} \\ \frac{2z-1}{z-1} & -\frac{1}{z-1} \end{array}\right].$$

From propositions 2.7.1 and 2.7.3, we know that algorithms 2.7.1 and 2.3.5 are shift-equivalent and equivalent up to a cyclic permutation. From proposition 2.6.1, we know algorithms 2.7.2 and 2.3.5 are oracle-equivalent, thus they are also shift-equivalent.

Algorithm 2.7.5
Douglas-Rachford splitting

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = \text{prox}_{tf}(x_3^k)$
$x_2^{k+1} = \text{prox}_{t(g \circ L)}(2x_1^{k+1} - x_3^k)$
$x_3^{k+1} = x_3^k + x_2^{k+1} - x_1^{k+1}$
end for

Algorithm 2.7.6 ADMM

for $k = 0, 1, 2, \ldots$ **do**
$\xi_1^{k+1} = \text{argmin}_\xi \{g(\xi) + \frac{\rho}{2} \left\| A\xi + B\xi_2^k - c + \xi_3^k \right\|^2\}$
$\xi_2^{k+1} = \text{argmin}_\xi \{f(\xi) + \frac{\rho}{2} \left\| A\xi_1^{k+1} + B\xi - c + \xi_3^k \right\|^2\}$
$\xi_3^{k+1} = \xi_3^k + A\xi_1^{k+1} + B\xi_2^{k+1} - c$
end for

Douglas-Rachford splitting and ADMM Consider a last example of algorithm permutation: Douglas-Rachford splitting (DR) (algorithm 2.7.5 [31,

35]) and the alternating direction method of multipliers (ADMM) (algorithm 2.7.6 [91, §8]). Suppose that linear operator L is invertible, $A = L^{-1}$, $B = -I$, and $c = 0$ in eq. (2.1). Then both DR and ADMM solve problem eq. (2.1) [12, 63, 107], and the update equations of ADMM can be simplified as algorithm 2.7.7. Further, we assume $\rho = 1/t$ in ADMM. We will compute the

Algorithm 2.7.7 Simplified ADMM

for $k = 0, 1, 2, \ldots$ **do**
$\xi_1^{k+1} = L\mathrm{prox}_{\frac{1}{\rho}(g\circ L)}(\xi_2^k - \xi_3^k)$
$\xi_2^{k+1} = \mathrm{prox}_{\frac{1}{\rho}f}(L^{-1}\xi_1^{k+1} + \xi_3^k)$
$\xi_3^{k+1} = \xi_3^k + L^{-1}\xi_1^{k+1} - \xi_2^{k+1}$
end for

transfer function of both algorithms using prox_{tf} and $\mathrm{prox}_{t(g\circ L)}$ as the oracles. The transfer function of DR is

$$\hat{H}_{10}(z) = \left[\begin{array}{ccc|cc} 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \\ 0 & 0 & I & -I & I \\ \hline 0 & 0 & I & 0 & 0 \\ 0 & 0 & -I & 2I & 0 \end{array}\right] = \left[\begin{array}{cc} -\frac{1}{z-1}I & \frac{1}{z-1}I \\ \frac{2z-1}{z-1}I & -\frac{1}{z-1}I \end{array}\right] \tag{2.24}$$

and the transfer function of ADMM is

$$\hat{H}_{11}(z) = \left[\begin{array}{ccc|cc} 0 & 0 & 0 & 0 & L \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & I & -I & I \\ \hline 0 & 0 & I & 0 & I \\ 0 & I & -I & 0 & 0 \end{array}\right] = \left[\begin{array}{cc} -\frac{1}{z-1}I & \frac{z}{z-1}I \\ \frac{2z-1}{z(z-1)}I & -\frac{1}{z-1}I \end{array}\right]. \tag{2.25}$$

From propositions 2.7.1 and 2.7.3, we know that DR and ADMM (with $\rho = 1/t$) are shift-equivalent and that DR is equivalent to a cyclic permutation of ADMM. In fact, it is also possible to write the state-space realization for each algorithm

using the gradient (or subgradient) of f and g as the oracle. The transfer functions depend on the choice of oracle, but in either case, we obtain the same results: the algorithms are shift-equivalent. We discuss the details further in appendix A.7. We can write the state-space realizations of DR and ADMM using the (sub)gradients as oracles in appendix A.7: the corresponding D_{12} blocks are still zero and thus still satisfy causality.

2.8 Algorithm repetition

In previous sections, we have defined equivalence between algorithms with the same number of oracle calls in each iteration. This section considers how to identify relations between two algorithms when the number of oracles in each iteration differs. For example, we would like to detect when one algorithm consists of another, simpler algorithm, repeated twice or more, possibly with changes to variables or shifts that obscure the relation.

Consider an algorithm $\mathcal{A}$. Given a problem and an initialization, the algorithm will generate state sequence $(x^k_{\mathcal{A}})_{k\geq 0}$, input sequence $(u^k_{\mathcal{A}})_{k\geq 0}$, and output sequence $(y^k_{\mathcal{A}})_{k\geq 0}$, respectively. Specifically, the update at time step k can be written as $x^{k+1}_{\mathcal{A}} = \mathcal{A}(x^k_{\mathcal{A}})$. Suppose we have another algorithm $\mathcal{B}$ such that $\mathcal{B} = \mathcal{A}^2$: repeating $\mathcal{A}$ twice gives the same result as $\mathcal{B}$. We call $\mathcal{B}$ a repetition of $\mathcal{A}$.

Just as in the previous sections, algorithm repetition can be characterized by the transfer function. Here, assumption 2.5.3 ensures the algorithms compared call the same set of oracles, although the number of times each oracle is called may be different.

Proposition 2.8.1. Suppose $\mathcal{A}$ has state-space realization (A, B, C, D). Then $\mathcal{B}$ is

equivalent to $\mathcal{A}^2$ if and only if its transfer function has the form

$$\begin{bmatrix} C(zI - A^2)^{-1}AB + D & C(zI - A^2)^{-1}B \\ CA(zI - A^2)^{-1}AB + CB & CA(zI - A^2)^{-1}B + D \end{bmatrix}. \tag{2.26}$$

Detailed proof of proposition 2.8.1 is provided in appendix A.8.

Algorithm 2.8.1 Gradient method

for $k = 0, 1, 2, \ldots$ **do**
$x^{k+1} = x^k - t\nabla f(x^k)$
end for

Algorithm 2.8.2
Repetition of gradient method

for $k = 0, 1, 2, \ldots$ **do**
$\xi_2^{k+1} = \xi_1^k - t\nabla f(\xi_1^k)$
$\xi_1^{k+1} = \xi_2^{k+1} - t\nabla f(\xi_2^{k+1})$
end for

One example of repetition consists the gradient method algorithm 2.8.1 and its repetition algorithm 2.8.2. Both call the same set of oracles (∇f). The transfer functions of each algorithm are computed as $\hat{H}_{12}(z)$ and $\hat{H}_{13}(z)$ respectively:

$$\hat{H}_{12}(z) = \left[\begin{array}{c|c} 1 & -t \\ \hline 1 & 0 \end{array}\right] = -\frac{t}{z-1}, \qquad \hat{H}_{13}(z) = \left[\begin{array}{c|cc} 1 & -t & -t \\ \hline 1 & 0 & 0 \\ 1 & -t & 0 \end{array}\right] = \begin{bmatrix} -\frac{t}{z-1} & -\frac{t}{z-1} \\ -\frac{tz}{z-1} & -\frac{t}{z-1} \end{bmatrix}.$$

Proposition 2.8.1 reveals how the transfer function changes when an algorithm is repeated twice. In fact, we can identify an algorithm that has been repeated arbitrarily many times. Suppose algorithm C is $\mathcal{A}$ repeated $n \geq 1$ times: $C = \mathcal{A}^n$.

Proposition 2.8.2. Suppose $\mathcal{A}$ has state-space realization (A, B, C, D). Then C is equivalent to $\mathcal{A}^n$ for $n \geq 1$ if and only if C has a transfer function given by eq. (2.28).

Proof. Sufficiency. We can represent C with state-space realization

$$\left[\begin{array}{c|cccccc} A^n & A^{n-1}B & \dots & \dots & AB & B \\ \hline C & D & 0 & 0 & \dots & 0 \\ CA & CB & D & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ CA^{n-1} & CA^{n-2}B & \dots & \dots & CB & D \end{array}\right]. \tag{2.27}$$

Note that $(zI - A^n)^{-1}A^l = A^l(zI - A^n)^{-1}$ for any n and l. Let $\tilde{C} = C(zI - A^n)^{-1}$, and compute the transfer function of C:

$$\left[\begin{array}{cccccc} \tilde{C}A^{n-1}B + D & \tilde{C}A^{n-2}B & \dots & \dots & \tilde{C}AB & \tilde{C}B \\ \tilde{C}A^nB + CB & \tilde{C}A^{n-1}B + D & \dots & \dots & \tilde{C}A^2B & \tilde{C}AB \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ \tilde{C}A^{2n-2}B + CA^{n-2}B & \tilde{C}A^{2n-3}B + CA^{n-3}B & \dots & \dots & \tilde{C}A^nB + CB & \tilde{C}A^{n-1}B + D \end{array}\right]. \tag{2.28}$$

Necessity is provided by proposition 2.6.1 since the transfer function uniquely characterizes an equivalence class of algorithms. □

Remark Proposition 2.8.1 is a special case of proposition 2.8.2 when $n = 2$. The dimension of transfer function of C is n times the dimension of transfer function of $\mathcal{A}$. Similarly, the dimension of input and output of C is n times the dimension of the input and output of $\mathcal{A}$. At time step k, we have $y_C^k = (y_{\mathcal{A}}^{nk}, \dots, y_{\mathcal{A}}^{(n+1)k-1})$ and $u_C^k = (u_{\mathcal{A}}^{nk}, \dots, u_{\mathcal{A}}^{(n+1)k-1})$.

Just as for oracle equivalence and cyclic permutations, the fixed points of an algorithm and its repetitions are related, as shown in proposition 2.8.3.

Proposition 2.8.3. If algorithm $\mathcal{A}$ converges to a fixed point $(y^\star, u^\star, x^\star)$, then its repetition $\mathcal{A}^n$ for $n \geq 1$ converges to fixed point $(y', u', x^\star)$, with $y' = y^\star \bigotimes \mathbb{1}^n$

and $u' = u^\star \bigotimes \mathbb{1}^n$. Here $\bigotimes$ is the Kronecker product and $\mathbb{1}^n$ is an n dimensional vector whose entries are all ones.

Detailed proof is provided in appendix A.9. Since $\mathcal{A}^n$ repeats $\mathcal{A}$ n times, the input and output of the fixed point of $\mathcal{A}^n$ are obtained by repeating the input and output on the corresponding fixed point of $\mathcal{A}$ n times.

Repetition gives us many more ways to combine algorithms into complex and unwieldly (but convergent) new methods. We can repeat a sequence of iterations from different algorithms and regard them together as a new algorithm. Suppose we choose n algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_n$ with state-space realizations $(A_1, B_1, C_1, D_1), \ldots, (A_n, B_n, C_n, D_n)$ and run one iteration of each as a single iteration of our new monster algorithm. For simplicity, suppose the state-space realization matrices A_i, B_i, C_i, D_i for each algorithm $\mathcal{A}_i$ have the same dimensions as all others $i = 1, \ldots, n$. (Otherwise the result is harder to write down, but still straightforward to compute.) Then we can represent the resulting monster algorithm with transfer function

$$\left[\begin{array}{c|cccccc} \prod_{i=n}^{1} A_i & \prod_{i=n}^{2} A_i B_1 & \ldots & \ldots & A_n B_{n-1} & B_n \\ \hline C_1 & D_1 & 0 & 0 & \ldots & 0 \\ C_2 A_1 & C_2 B_1 & D_2 & 0 & \ldots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ C_n \prod_{i=n-1}^{1} A_i & C_n \prod_{i=n-1}^{2} A_i B_1 & \ldots & \ldots & C_n B_{n-1} & D_n \end{array}\right]. \tag{2.29}$$

Hence one way to develop a new optimization algorithm would be to combine existing algorithms into a new monster algorithm with similar convergence properties but (perhaps) new exciting interpretations. For example, we could combine gradient descent with the proximal point method to derive a proximal

gradient method for minimizing $f(x)$: $\mathrm{prox}_f(x - \nabla f(x))$. (We are not aware of any published optimization algorithms that have been constructed in this way.)

Using our software, it would be easy to detect such algorithm surgery by searching over all pairs (or trios, etc) of known algorithms. This combinatorial search is still not too expensive, since the list of known algorithms is still rather small, and the number of algorithms that makes up a monster algorithm is limited by the number of oracle calls at each iteration of the monster algorithm.

2.9 Algorithm conjugation

In this section, we introduce one last algorithm transformation, conjugation, which alters the oracle calls but results in algorithms that still bear a family resemblance.

In convex optimization, algorithm conjugation naturally relates some oracles to others [90; 91, §2]: for example, when $f^*(y) = \sup_x\{x^T y - f(x)\}$ is the Fenchel conjugate of f [39, §3],

- $(\partial f)^{-1} = \partial f^*$, and
- *Moreau's identity.* $I - \mathrm{prox}_f = \mathrm{prox}_{f^*}$.

We can rewrite any algorithm in terms of different, also easily computable, oracles using these identities. Consider a simple example: we will obfuscate the proximal gradient method (algorithm 2.9.1 [8, §10; 9]) by rewriting it in terms of the conjugate of the original oracle prox_g, using Moreau's identity, as algorithm 2.9.2 [74].

Algorithm 2.9.1
Proximal gradient method

for $k = 0, 1, 2, \ldots$ **do**
$x^{k+1} = \text{prox}_{tg}(x^k - t\nabla f(x^k))$
end for

Algorithm 2.9.2
Conjugate of proximal gradient method

for $k = 0, 1, 2, \ldots$ **do**
$\xi^{k+1} = \xi^k - t\nabla f(\xi^k) - t\text{prox}_{\frac{1}{t}g^*}(\frac{1}{t}(\xi^k - t\nabla f(\xi^k)))$
end for

The transfer function of the algorithm changes when we rewrite the algorithm to call a different oracle, such as calling prox_{f^*} instead of prox_f. Yet the sequence of states is preserved! Similarly, when we rewrite an algorithm to call ∂f^* instead of ∂f, the resulting algorithm is related to the original algorithm by swapping the input and output sequences. We say that algorithm $\mathcal{B} = C_\kappa \mathcal{A}$ is a conjugate of algorithm $\mathcal{A}$ if algorithm $\mathcal{B}$ results from rewriting algorithm $\mathcal{A}$ to use the conjugates of the oracles in set $\kappa \subseteq [n]$, where $[n] = \{1, \ldots, n\}$ is the set of oracle indices for algorithm $\mathcal{A}$. Interestingly, conjugation preserves the state sequence but not the oracle sequence. We will also call two algorithms conjugates if they are oracle-equivalent to a conjugate pair. Our goal in this section is to describe how to identify conjugate algorithms.

For simplicity in the remainder of this section, we suppose that all oracles are (sub)gradients. To detect equivalence of algorithms involving prox using methods presented here, we may write the state-space realization of the algorithm in terms of (sub)gradients:

$$u = \text{prox}_f(y) \qquad \Longleftrightarrow \qquad y \in u + \partial f(u).$$

In fact, our software uses this method to check algorithm conjugation.

Restricting to (sub)gradients, we see from the identity $(\partial f)^{-1} = \partial f^*$ that algorithm conjugation swaps the input and output of an algorithm: the algorithm

after conjugation takes the output of the original algorithm as input and produces the input of the original one as output. As shown in fig. 2.6, the input sequence of the algorithm after conjugation is the original output sequence and the output sequence in the algorithm after conjugation is the original input sequence.

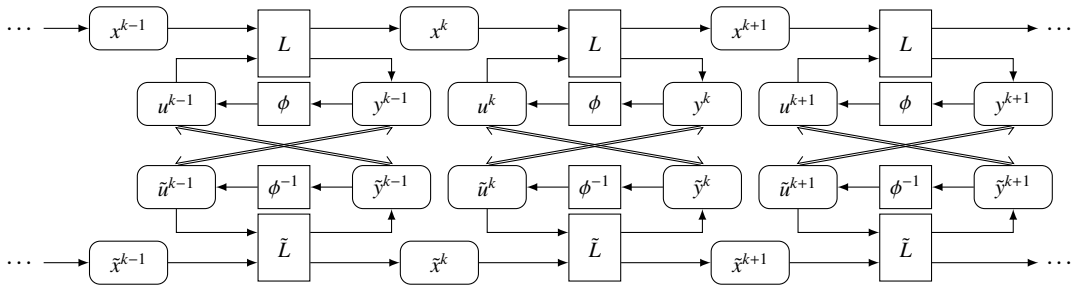

Figure 2.6: Unrolled block-diagram representation of algorithm conjugation.

First, let's introduce a bit of standard notation. Suppose an algorithm $\mathcal{A}$ contains n oracle calls in each iteration. The cardinality of a subset $\kappa \subseteq [n]$ is $|\kappa|$ and the complement is $\bar{\kappa} = [n] \setminus \kappa$. For any matrix $M \in \mathbb{R}^{n\times n}$, $M[\kappa, \nu]$ is the submatrix of M whose rows and columns are indexed by κ and $\nu \subseteq [n]$, respectively. We write $M[\kappa, \kappa]$ as $M[\kappa]$ for simplicity. For $i \in [n]$, the conjugation operator C_i conjugates oracle i: it replaces the ith oracle by its inverse. The operator C_κ conjugates all oracles in the set $\kappa \subseteq [n]$ to produce the conjugate algorithm $C_\kappa\mathcal{A}$.

Proposition 2.9.1. Suppose $\mathcal{A}$ has state-space realization (A, B, C, D) and transfer function $\hat{H}(z)$, and $D[\kappa]$ is invertible. Then $\mathcal{B}$ is equivalent to $C_\kappa\mathcal{A}$ if and only if the transfer function $\hat{H}'(z)$ of $\mathcal{B}$ satisfies

$$P\hat{H}'(z)P^T = \begin{bmatrix} \hat{H}[\kappa]^{-1}(z) & -\hat{H}[\kappa]^{-1}(z)\hat{H}[\kappa, \bar{\kappa}](z) \\ \hat{H}[\bar{\kappa}, \kappa](z)\hat{H}[\kappa]^{-1}(z) & \hat{H}[\bar{\kappa}](z) - \hat{H}[\bar{\kappa}, \kappa](z)\hat{H}[\kappa]^{-1}(z)\hat{H}[\kappa, \bar{\kappa}](z) \end{bmatrix}. \quad (2.30)$$

Here P is a permutation matrix that swaps rows and columns so indices in κ

come first:

$$P\hat{H}(z)P^T = \begin{bmatrix} \hat{H}[\kappa](z) & \hat{H}[\kappa,\bar{\kappa}](z) \\ \hat{H}[\bar{\kappa},\kappa](z) & \hat{H}[\bar{\kappa}](z) \end{bmatrix}. \tag{2.31}$$

Proof. Sufficiency. Without loss of generality, suppose the oracles $\kappa = \{1, \ldots, |\kappa|\}$ appear first,

$$\hat{H}(z) = \begin{bmatrix} \hat{H}[\kappa](z) & \hat{H}[\kappa,\bar{\kappa}](z) \\ \hat{H}[\bar{\kappa},\kappa](z) & \hat{H}[\bar{\kappa}](z) \end{bmatrix}, \qquad D = \begin{bmatrix} D[\kappa] & D[\kappa,\bar{\kappa}] \\ D[\bar{\kappa},\kappa] & D[\bar{\kappa}] \end{bmatrix},$$

and consequently the permutation matrix P is the identity. We obtain the desired results from eq. (2.14) by setting $D_{11} = D[\kappa]$, $\hat{H}_{11}(z) = \hat{H}[\kappa](z)$, $\hat{H}_{12}(z) = \hat{H}[\kappa,\bar{\kappa}](z)$, $\hat{H}_{21}(z) = \hat{H}[\bar{\kappa},\kappa](z)$, and $\hat{H}_{22}(z) = \hat{H}[\bar{\kappa}](z)$.

Necessity is provided by proposition 2.6.1 as the transfer function uniquely characterizes an equivalence class of algorithms.

□

From proposition 2.9.1, the transfer function $\hat{H}(z)$ of algorithm $\mathcal{A}$ is partially inverted when the algorithm is conjugated by C_κ. The new transfer function $\hat{H}'(z)$ results from applying the Sweep operator with indices κ to $\hat{H}(z)$ [46, 102]. If we consider the input and output sequences for each oracle separately, for any oracle in κ, the input sequence corresponding to $C_\kappa\mathcal{A}$ is the original output sequence in $\mathcal{A}$ and the output sequence corresponding to $C_\kappa\mathcal{A}$ is the original input sequence in $\mathcal{A}$. The input and output sequences of oracles in $[n] \setminus \kappa$ remain unchanged in the new algorithm $C_\kappa\mathcal{A}$. Here, assumption 2.5.3 ensures the algorithms compared call either same oracles or their corresponding conjugate oracles and in each iteration the number of oracle calls are the same.

Proposition 2.9.1 assumes that $D[\kappa]$ is invertible. In fact, $C_\kappa \mathcal{A}$ is a causal algorithm if and only if $D[\kappa]$ is invertible. We need not condition on causality in the proposition, since any algorithm that can be written down as a set of update equations is necessarily causal.

Now we consider two special cases: conjugating 1) a single oracle, or 2) all of the oracles.

Corollary 2.9.1. Consider algorithm $\mathcal{A}$ with state-space realization (A, B, C, D) and transfer function $\hat{H}(z) \in \mathbb{R}^{n \times n}$.

(a) Suppose $D_{kk} \neq 0$ for any $k \in [n]$. Then the new transfer function $\hat{H}'(z)$ of $C_k \mathcal{A}$ can be expressed entrywise as

$$h'_{ij}(z) = \begin{cases} 1/h_{kk}(z) & i = k,\ j = k \\ -h_{kj}(z)/h_{kk}(z) & i = k,\ j \neq k \\ h_{ik}(z)/h_{kk}(z) & i \neq k,\ j = k \\ h_{ij}(z) - h_{ik}(z)h_{kj}(z)/h_{kk}(z) & i \neq k,\ j \neq k, \end{cases} \tag{2.32}$$

as $h_{ij}(z)$ and $h'_{ij}(z)$ $1 \leq i, j \leq n$ denote the entries of $\hat{H}(z)$ and $\hat{H}'(z)$ respectively.

(b) Suppose D is invertible. Then the transfer function $\hat{H}'(z)$ of $C_{[n]} \mathcal{A}$ satisfies $\hat{H}'(z) = \hat{H}^{-1}(z)$.

Proximal gradient Now we can revisit algorithms 2.9.1 and 2.9.2 and show that they are conjugate. The transfer functions of algorithms 2.9.1 and 2.9.2 are computed as $\hat{H}_{14}(z)$ and $\hat{H}_{15}(z)$ below. Note that the state-space realizations are written in terms of (sub)gradients. From corollary 2.9.1, they are conjugate with respect to the second oracle.

$$\hat{H}_{14}(z) = \begin{bmatrix} -\frac{t}{z-1} & -\frac{t}{z-1} \\ -\frac{tz}{z-1} & -\frac{tz}{z-1} \end{bmatrix}, \qquad \hat{H}_{15}(z) = \begin{bmatrix} 0 & \frac{1}{z} \\ -1 & -\frac{z-1}{tz} \end{bmatrix}$$

Algorithm 2.9.3 Chambolle-Pock method

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} = \text{prox}_{\tau f}(x_1^k - \tau M^T x_2^k)$
$x_2^{k+1} = \text{prox}_{\sigma g^*}(x_2^k + \sigma M(2x_1^{k+1} - x_1^k))$
end for

DR and Chambolle-Pock Another important example is the relation between DR (algorithm 2.7.5) and the primal-dual optimization method proposed by Chambolle and Pock (algorithm 2.9.3 [20; 80]). Note that algorithm 2.7.5 has parameter t and linear operator L, and algorithm 2.9.3 has parameters τ and σ and linear operator M. Let $M = L$ so that algorithms 2.9.3 and 2.7.5 solve the same problem. Further suppose that M is invertible and $MM^T = \delta I$ for any $\delta > 0$. By corollary 2.9.1, we know that they are conjugate with respect to the second oracle if $\tau = t$ and $\sigma = 1/(\delta t)$. So DR and the Chambolle-Pock method (when the parameter value $\tau = t$ and $\sigma = 1/(\delta t)$) are conjugate. The transfer functions of algorithms 2.9.3 and 2.7.5 are provided below as $\hat{H}_{10}(z)$ and $\hat{H}_{16}(z)$ respectively. We will say more about how to discover the correct parameter restriction in **??**.

$$\hat{H}_{10}(z) = \begin{bmatrix} -\frac{tz}{z-1} I & -\frac{t}{z-1} L^T \\ \frac{t(1-2z)}{z-1} L & -\frac{tz}{z-1} LL^T \end{bmatrix}, \quad \hat{H}_{16}(z) \xrightarrow[\sigma=\frac{1}{\delta t}, \tau=t]{M=L, LL^T=\delta I} \begin{bmatrix} \frac{t(1-z)}{z} I & \frac{1}{z} L^T (LL^T)^{-1} \\ \frac{1-2z}{z} (LL^T)^{-1} L & \frac{1-z}{tz} (LL^T)^{-1} \end{bmatrix}$$

In order to test equivalence, all algorithms must use the same set of oracles. This requirement becomes tricky when algorithms are written in terms of an argmin: what is the oracle? To resolve this issue, we compute the state-space

realization of every algorithm in this section using the subgradient as the oracle. All these subgradient oracles are associated with proximal operators, and so they are unique-valued, even though subgradients are generally set-valued: the input-output pairs match those returned by the proximal operator. (These subgradient oracles are used for the analysis but need not be computed explicitly.)

The fixed points of an algorithm and its conjugate are related as stated in proposition 2.9.2.

Proposition 2.9.2. If an algorithm $\mathcal{A}$ converges to a fixed point $(y[\kappa]^\star, y[\bar{\kappa}]^\star, u[\kappa]^\star, u[\bar{\kappa}]^\star, x^\star)$, then its conjugate $C_\kappa \mathcal{A}$ converges to fixed point $(u[\kappa]^\star, y[\bar{\kappa}]^\star, y[\kappa]^\star, u[\bar{\kappa}]^\star, x^\star)$.

For simplicity, detailed proof is provided in appendix A.10. Intuitively, as we invert the input-output map of $u[\kappa]$ and $y[\kappa]$, the corresponding parts in the fixed point are also inverted.

Proposition 2.9.3. Suppose algorithm $\mathcal{A}$ has state-space realization (A, B, C, D), where $D_{ii} \neq 0$ and $D_{jj} \neq 0$. Then $C_i C_j \mathcal{A} = C_j C_i \mathcal{A} = C_{\{ij\}} \mathcal{A}$.

Proof. By corollary 2.9.1, if $D_{ii} \neq 0$ and $D_{jj} \neq 0$, then $C_i \mathcal{A}$ and $C_j \mathcal{A}$ are causal. Note that entries above diagonal of D are all zero because $\mathcal{A}$ is causal. Thus, $\det(D[\{ij\}]) = D_{ii} D_{jj} \neq 0$ and $C_{\{ij\}} \mathcal{A}$ is causal. The commutative property of the Sweep operator gives the result $C_i C_j \mathcal{A} = C_j C_i \mathcal{A} = C_{\{ij\}} \mathcal{A}$ [46, 102]. □

Proposition 2.9.3 states that conjugation of different oracles commutes. This justifies our notation C_κ for set κ, as the order of the oracles in κ is irrelevant. Further, conjugation and cyclic permutation also commute; see proposition A.11.1 and proof in appendix A.11.

DR and ADMM We showed in section 2.7.3 that the DR (algorithm 2.7.5) and ADMM (algorithm 2.7.6) are related by permutation with a certain choice of parameters. Here, we show that they are related by permutation and conjugation (in either order, as they commute), with a different choice of parameters: $A = L^T, B = I, c = 0, \rho = t$ for ADMM. Further suppose that linear operator L is invertible. The transfer function of this special parameterization of ADMM is shown as $\hat{H}_{17}(z)$. Relations between DR and ADMM can be illustrated as follows. Recall $\hat{H}_{10}(z)$ is the transfer function of DR. Here we can observe that different choices of parameters of algorithms can lead to different relations between algorithms.

$$\hat{H}_{17}(z) = \begin{bmatrix} -\frac{z}{t(z-1)}I & \frac{z}{t(z-1)}L^{-1} \\ \frac{2z-1}{tz(z-1)}L^{-T} & -\frac{z}{t(z-1)}(LL^T)^{-1} \end{bmatrix} \xrightarrow{C_{12}} \begin{bmatrix} -\frac{tz}{z-1}I & -\frac{tz}{z-1}L^T \\ \frac{t(1-2z)}{z(z-1)}L & -\frac{tz}{z-1}LL^T \end{bmatrix} \xrightarrow{P_{21}} \begin{bmatrix} -\frac{tz}{z-1}I & -\frac{t}{z-1}L^T \\ \frac{t(1-2z)}{z-1}L & -\frac{tz}{z-1}LL^T \end{bmatrix} = \hat{H}_{10}(z)$$

The commutative property is important to identify relations between algorithms efficiently. For example, suppose we would like to identify the relations between algorithms 2.7.5 and 2.7.6, with transfer functions $\hat{H}_{10}(z)$ and $\hat{H}_{17}(z)$. We can first perform conjugation and next permutation on algorithm 2.7.5, and then test equivalence between the resulting algorithm and algorithm 2.7.6. We need not try permutation followed by conjugation; as these commute, both orders lead to the same transfer function.

We have already shown several relations between DR (algorithm 2.7.5), ADMM (algorithm 2.7.6), and the Chambolle-Pock method (algorithm 2.9.3) using conjugation and permutation. We represent these relations in fig. 2.7. The figure relates 8 different algorithms: Starting from DR, since it contains 2 oracles, there are 2 possible different algorithms by permutation. From the state-space

realization, we can conjugate both oracles, which yields 4 different algorithms by conjugation of different oracles. Therefore, in total there are $2 \times 4 = 8$ possible different algorithms, including both ADMM and Chambolle-Pock. In the figure, C_1 and C_2 denote conjugation with respect to the first and second oracles respectively, P denotes permutation, and we can move between algorithms by applying the transformation on each edge, in either direction, as each transformation is an involution.

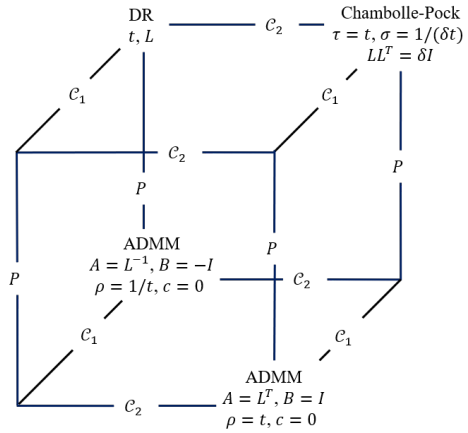

Figure 2.7: Connections between DR, ADMM, and Chambolle-Pock method.

2.10 Linnaeus

In this section, we introduce our software package called LINNAEUS that implements these ideas in detail. This package can be used by researchers (or peer reviewers) who wish to understand the novelty of new algorithmic ideas and connections to existing algorithms. The input is an algorithm described in user-friendly syntax with variables, parameters, functions, oracles, and update equations. The system will automatically translate the input algorithm into a canonical form (the transfer function) and use the canonical form to identify

whether the algorithm is equivalent to any reference algorithm, possibly after transformations such as permutation, conjugation, or repetition. Further, the software can also serve as a search engine, which will identify connections from the input algorithm to existing algorithms in the literature that appear in LINNAEUS's algorithm library.

2.10.1 Illustrative examples

We use LINNAEUS to identify the relations between algorithms presented previously in this chapter. These examples demonstrate the power and simplicity of LINNAEUS. Code for these examples can be found at `https://github.com/udellgroup/Linnaeus_software`.

Algorithms 2.3.1 and 2.3.2 The following code identifies that algorithms 2.3.1 and 2.3.2 are oracle-equivalent. We input algorithms 2.3.1 and 2.3.2 with variables, oracles, and update equations, and parse them into state-space realizations. Then we check oracle equivalence using the function `is_equivalent`. The system returns `True`, consistent with our analytical results in sections 2.3 and 2.6.

```
# define Algorithm 2.3.1
algo1 = Algorithm("Algorithm 2.3.1")


# add oracle gradient of f to Algorithm 2.3.1
gradf = algo1.add_oracle("gradf")
```

```
# add variables x1, x2, and x3 to Algorithm 2.3.1
x1, x2, x3 = algo1.add_var("x1", "x2", "x3")

# add update equations
# x3 <- 2x1 - x2
algo1.add_update(x3, 2*x1 - x2)
# x2 <- x1
algo1.add_update(x2, x1)
# x1 <- x3 - 1/10*gradf(x3)
algo1.add_update(x1, x3 - 1/10*gradf(x3))

# parse Algorithm 2.3.1, translate it into␣
 ↪canonical form
algo1.parse()
```

```
Parse Algorithm 2.3.1:
```

$$x_3 \leftarrow 2x_1 - x2$$

$$x_2 \leftarrow x_1$$

$$x_1 \leftarrow x_3 - 0.1\text{gradf}(x_3)$$

```
algo2 = Algorithm("Algorithm 2.3.2")
xi1, xi2, xi3 = algo2.add_var("xi1", "xi2",␣
 ↪"xi3")
gradf = algo2.add_oracle("gradf")


# xi3 <- xi1
algo2.add_update(xi3, xi1)
# xi1 <- xi1 - xi2 - 1/5*gradf(xi1)
algo2.add_update(xi1, xi1 - xi2 - 1/5*gradf(xi3))
# xi2 <- xi2 + 1/10*gradf(xi3)
algo2.add_update(xi2, xi2 + 1/10*gradf(xi3))


algo2.parse()
```

```
Parse Algorithm 2.3.2:
```

$\xi_3 \leftarrow \xi_1$

$\xi_1 \leftarrow \xi_1 - \xi_2 - 0.2\text{gradf}(\xi_3)$

$\xi_2 \leftarrow \xi_2 + 0.1\text{gradf}(\xi_3)$

```
# check oracle equivalence
lin.is_equivalent(algo1, algo2, verbose = True)
```

```
------------------------------------------------------------
Algorithm 2.3.1 is equivalent to Algorithm 2.3.2.
------------------------------------------------------------
True
```

Algorithms 2.3.5 and 2.3.6 The second example identifies that algorithms 2.3.5 and 2.3.6 are shift-equivalent. We input and parse the algorithms into state-space realizations and then check shift equivalence (cyclic permutation) using the function `is_permutation`. The system returns `True`, consistent with results in sections 2.3 and 2.7.

```
algo5 = Algorithm("Algorithm 2.3.5")
x1, x2, x3 = algo5.add_var("x1", "x2", "x3")
proxf, proxg = algo5.add_oracle("proxf", "proxg")

# x1 <- proxf(x3)
algo5.add_update(x1, proxf(x3))
# x2 <- proxg(2x1 - x3)
algo5.add_update(x2, proxg(2*x1 - x3))
# x3 <- x3 + x2 - x1
algo5.add_update(x3, x3 + x2 - x1)

algo5.parse()
```

```
------------------------------------------------------------
Parse Algorithm 2.3.5:
```

$x_1 \leftarrow \text{proxf}(x_3)$

$x_2 \leftarrow \text{proxg}(2x_1 - x_3)$

$x_3 \leftarrow x_3 + x_2 - x_1$

```
algo4 = Algorithm("Algorithm 2.3.6")
xi1, xi2 = algo4.add_var("xi1", "xi2")
proxf, proxg = algo4.add_oracle("proxf", "proxg")

# xi1 <- proxg(-xi1 + 2xi2) + xi1 - xi2
algo4.add_update(xi1, proxg(-xi1 + 2*xi2) + xi1
 ↪- xi2)
# xi2 <- proxf(xi1)
algo4.add_update(xi2, proxf(xi1))

algo4.parse()
```

```
Parse Algorithm 2.3.6:
```

$\xi_1 \leftarrow \text{proxg}(-\xi_1 + 2\xi_2) + \xi_1 - \xi_2$

$\xi_2 \leftarrow \text{proxf}(\xi_1)$

```
# check cyclic permutation (shift equivalence)
lin.is_permutation(algo5, algo6, verbose = True)
```

```
------------------------------------------------------------
Algorithm 2.3.5 is a permutation of Algorithm 2.3.6.
------------------------------------------------------------
True
```

DR and ADMM The third illustrative example shows that DR and ADMM are related by permutation and conjugation, as we saw in section 2.9. Further, LINNAEUS can even reveal the specific parameter choice required for the relation to hold. Just as in section 2.9, suppose both DR and ADMM solve problem eq. (2.1) with $A = L^T$, $B = I$, and $c = 0$. We input and parse DR and ADMM. To detect the relations, we use function `test_conjugate_permutation` to check conjugation and permutation between DR and ADMM. The results are the same as section 2.9.

```
DR = Algorithm("Douglas-Rachford splitting")
x1, x2, x3 = DR.add_var("x1", "x2", "x3")
t = DR.add_parameter("t")
L = DR.add_parameter("L", commutative = False)

# x1 <- prox_tf(x3)
DR.add_update(x1, lin.prox(f, t)(x3))
# x2 <- prox_tgL(2x1 - x3)
DR.add_update(x2, lin.prox(g, t, L)(2*x1 - x3))
```

```
# x3 <- x3 + x2 - x1
DR.add_update(x3, x3 + x2 - x1)


DR.parse()
```

Parse Douglas-Rachford splitting:

$$x_1 \leftarrow \text{prox}_{tf}(x_3)$$

$$x_2 \leftarrow \text{prox}_{t(g \circ L)}(2x_1 - x_3)$$

$$x_3 \leftarrow x_3 + x_2 - x_1$$

```
ADMM = Algorithm("ADMM")
f, g = ADMM.add_function("f", "g")
rho = ADMM.add_parameter("rho")
L = ADMM.add_parameter("L", commutative = False)
xi1, xi2, xi3 = ADMM.add_var("xi1", "xi2", "xi3")


# xi1 <- argmin(x1, g^*(xi1) + 1/2*rho*||T(L)xi1␣
 ↪+ xi2 + xi3||^2)
ADMM.add_update(xi1, lin.argmin(xi1, g(xi1) + 1/
 ↪2*rho*lin.norm_square(T(L)*xi1 + xi2 + xi3)))
# xi2 <- argmin(x2, f^*(xi2) + 1/2*rho*||T(L)xi1␣
 ↪+ xi2 + xi3||^2)
```

```
ADMM.add_update(xi2, lin.argmin(xi2, f(xi2) + 1/
 ↪2*rho*lin.norm_square(T(L)*xi1 + xi2 + xi3)))
# xi3 <- xi3 + T(L)xi1 + xi2
ADMM.add_update(xi3, xi3 + T(L)*xi1 + xi2)


ADMM.parse()
```

Parse ADMM:

$$\xi_1 \leftarrow \text{argmin}_{\xi_1}\{g(\xi_1) + 0.5\rho\text{norm}_{\text{square}}(T(L)\xi_1 + \xi_2 + \xi_3)\}$$

$$\xi_2 \leftarrow \text{argmin}_{\xi_2}\{f(\xi_2) + 0.5\rho\text{norm}_{\text{square}}(T(L)\xi_1 + \xi_2 + \xi_3)\}$$

$$\xi_3 \leftarrow T(L)\xi_1 + \xi_2 + \xi_3$$

```
# check permutation and conjugation
# between DR and ADMM
lin.test_conjugate_permutation(DR, ADMM)
```

==========

Parameters of Douglas-Rachford splitting:

$$t, L$$

```
Parameters of ADMM:
```

$$\rho, L$$

```
Douglas-Rachford splitting is a conjugate permutation of
↪ADMM, if the parameters satisfy:
```

$$\rho = t$$

$$L = L$$

```
============================================================
------------------------------------------------------------
```

2.10.2 Implementation

In this subsection, we briefly describe the implementation of LINNAEUS. All expressions in LINNAEUS are defined symbolically, using the python package for symbolic mathematics *sympy*. In LINNAEUS, an algorithm is specified by defining variables, parameters, functions, oracles, and update equations. All variables and parameters are symbolic, so there is no need to specialize problem dimensions or parameter choices. The system automatically translates an input algorithm into its state-space realization and computes the transfer function. The transfer functions can be compared and manipulated as needed to establish various kinds of equivalences or other relations between algorithms.

Parameter declaration Parameters of the algorithm can be declared as scalar (commutative) or vector or matrix (noncommutative). The following code shows how to add scalar `t` and matrix `L` to `algo1`.

```
# add a scalar parameter t
t = algo1.add_parameter("t")
# add a matrix parameter L
L = algo1.add_parameter("L", commutative = False)
```

Parameter specification Given two input algorithms, LINNAEUS computes the transfer functions and can compare them to detect equivalence and other relations. Some algorithms are equivalent or related only when the parameters satisfy a certain condition: for example, DR and ADMM. If the transfer functions of each algorithm use different parameters, LINNAEUS form symbolic equations and solve the equations to determine conditions that, if satisfied by the algorithm parameters, yield the desired relation between the algorithms; see eq. (2.19) in section 2.6.

Oracles and function Oracles play the starring role in our framework: oracle equivalence is possible only if two algorithms share the same oracles. In LINNAEUS, we provide two approaches to declare and add oracles to an algorithm. The black-box approach is to define oracles as black boxes. When parsing the algorithm, the system treats each oracle as a distinct entity unrelated to any other oracle. An oracle declared using syntax `add_oracle` uses the black-box approach. For example, we may add oracles ∇f and prox_g to algorithm `algo1`:

```
# add oracle gradient of f in the first approach
gradf = algo1.add_oracle("gradf")
# add oracle prox of g in the first approach
proxg = algo1.add_oracle("proxg")
```

The functional approach is to define oracles in terms of the (sub)gradient of a function. When parsing an algorithm, all the oracles will be decomposed into (sub)gradients and the state-space realization given in terms of (sub)gradients. We say that two algorithms are oracle-equivalent in terms of functional oracles if they are oracle-equivalent after rewriting the algorithm to use only (sub)gradient oracles. This approach is critical to allow us to identify algorithm conjugation, since conjugate algorithms use different (conjugate) oracles. If every algorithm is represented in terms of (sub)gradients, algorithm conjugation can be detected using proposition 2.9.1. Fortunately, common oracles such as prox and argmin can be easily written in terms of (sub)gradients: for example, $\text{prox}_f(x) = (I - \partial f)^{-1}(x)$ and argmin as eq. (2.33).

To use the functional approach, users must define and add functions to the algorithm first using `add_function` and then declare and add oracles. The following code shows how to use the functional approach to declare and add oracles ∇f and prox_f.

```
# add function f
f = algo1.add_function("f")
# gradient of f with repect to x1
lin.grad(f)(x1)
```

```
# prox of f with repect to x2 and parameter t
lin.prox(f,t)(x2)
```

2.10.3 Black-box vs functional oracles

Are two algorithms equivalent with respect to black-box oracles if and only if they are equivalent with respect to functional oracles? Intuitively, when oracles are defined in terms of (sub)gradients, it might be possible to identify more relations with other algorithms. However, as stated in proposition 2.10.1, for algorithms that use only proximal operators, argmins, and (sub)gradients as oracles, equivalence is preserved under both black-box and functional definitions of oracles.

Proposition 2.10.1. Suppose two algorithms use only proximal operators, argmins, and (sub)gradients as oracles. Then the two algorithms are equivalent with respect to black-box oracles if and only if they are also equivalent with respect to functional oracles.

Proof. Since for any function g and any t, $\text{prox}_{tg}(x) = \text{argmin}_y\{tg(y) + \frac{1}{2}\|x - y\|^2\}$, we can treat proximal operator as a special case of argmin. Without loss of generality, any argmin oracle in a linear algorithm has the form

$$z = \text{argmin}_x \left\{ \lambda g(x) + \frac{1}{2} \begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \right\}.$$

Here z is the value of the oracle and y can be regarded as the argument, which means from the perspective of a linear system, z is the input and y is the output.

The parameter λ can be a scalar or matrix, g is a function, and Q_{11}, Q_{12}, Q_{21}, Q_{22} are parameter matrices. Specifically,

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix}$$

is a symmetric matrix and

$$\frac{1}{2}\begin{bmatrix} x \\ y \end{bmatrix}^T \begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

is a quadratic term with respect to x and y. The matrix Q_{11} must be invertible if the argmin oracle is single-valued. To recover the proximal operator, choose a scalar λ and set

$$\begin{bmatrix} Q_{11} & Q_{12} \\ Q_{21} & Q_{22} \end{bmatrix} = \begin{bmatrix} I & -I \\ -I & I \end{bmatrix}.$$

If g is a convex function, the argmin oracle can be written in terms of the subgradient oracle ∂g as follows,

$$z \in -Q_{11}^{-1}\lambda\partial g(z) - Q_{11}^{-1}Q_{12}y. \tag{2.33}$$

Suppose we have an algorithm with $n + m$ oracles in total, consisting of n argmins and m (sub)gradients. We can group the argmins and the (sub)gradients together respectively and partition the state-space realization accordingly as

$$\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right], \tag{2.34}$$

where C_1, B_1 correspond to the argmins, C_2, B_2 correspond to the (sub)gradients, and D is partitioned accordingly into D_{11}, D_{12}, D_{21}, and D_{22}. The transfer func-

tion can be represented accordingly as

$$\hat{H}(z) = \begin{bmatrix} \hat{H}_{11}(z) & \hat{H}_{12}(z) \\ \hat{H}_{21}(z) & \hat{H}_{22}(z) \end{bmatrix} = \begin{bmatrix} C_1(zI - A)^{-1}B_1 + D_{11} & C_1(zI - A)^{-1}B_2 + D_{12} \\ C_2(zI - A)^{-1}B_1 + D_{21} & C_2(zI - A)^{-1}B_2 + D_{22} \end{bmatrix}.$$

The input and output are partitioned as $(\bar{u}_1, \bar{u}_2)$ and $(\bar{y}_1, \bar{y}_2)$, where $\bar{y}_1 = (y_1, \ldots, y_n)$, $\bar{y}_2 = (y_{n+1}, \ldots, y_{n+m})$, $\bar{u}_1 = (z_1, \ldots, z_n)$, and $\bar{u}_2 = (\nabla f_{n+1}(y_{n+1}), \ldots, \nabla f_{n+m}(y_{n+m}))$. For each $i \in \{1, \ldots, n\}$ we have

$$z_i = \operatorname{argmin}_x \left\{ \lambda_i f_i(x) + \frac{1}{2} \begin{bmatrix} x \\ y_i \end{bmatrix}^T \begin{bmatrix} Q_{11}^i & Q_{12}^i \\ Q_{21}^i & Q_{22}^i \end{bmatrix} \begin{bmatrix} x \\ y_i \end{bmatrix} \right\} \tag{2.35}$$

where Q_{11}^i is invertible for any $i \in \{1, \ldots, n\}$.

Now we rewrite the linear system so that the nonlinearities corresponding to the argmins for the new linear system are (sub)gradients. Let $\lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, $Q_1 = \operatorname{diag}(Q_{11}^1, \ldots, Q_{11}^n)$, $Q_2 = \operatorname{diag}(Q_{12}^1, \ldots, Q_{12}^n)$, and $M_1 = Q_1^{-1}Q_2$, and $M_2 = Q_1^{-1}\lambda$. The new state-space realization in terms of the (sub)gradient oracles is

$$\left[\begin{array}{c|cc} A - B_1(I + M_1D_{11})^{-1}M_1C_1 & -B_1(I + M_1D_{11})^{-1}M_2 & B_2 - B_1(I + M_1D_{11})^{-1}M_1D_{12} \\ \hline -(I + M_1D_{11})^{-1}M_1C_1 & -(I + M_1D_{11})^{-1}M_2 & -(I + M_1D_{11})^{-1}M_1D_{12} \\ C_2 - D_{21}(I + M_1D_{11})^{-1}M_1C_1 & -D_{21}(I + M_1D_{11})^{-1}M_2 & D_{22} - D_{21}(I + M_1D_{11})^{-1}M_1D_{12} \end{array}\right]. \tag{2.36}$$

We can compute the transfer function as

$$\hat{H}'(z) = \begin{bmatrix} \hat{H}'_{11}(z) & \hat{H}'_{12}(z) \\ \hat{H}'_{21}(z) & \hat{H}'_{22}(z) \end{bmatrix} = \begin{bmatrix} -(I + M_1\hat{H}_{11}(z))^{-1}M_2 & -(I + M_1\hat{H}_{11}(z))^{-1}M_1\hat{H}_{12}(z) \\ -\hat{H}_{21}(z)(I + M_1\hat{H}_{11}(z))^{-1}M_2 & \hat{H}_{22}(z) - \hat{H}_{21}(z)(I + M_1\hat{H}_{11}(z))^{-1}M_1\hat{H}_{12}(z) \end{bmatrix}. \tag{2.37}$$

Note that $I + M_1D_{11}$ is invertible (otherwise the algorithm is not causal) and consequently $I + M_1\hat{H}_{11}(z)$ is invertible. The matrix Q_1 is also invertible, since Q_{11}^i is invertible for any $i \in \{1, \ldots, n\}$. A detailed proof of eq. (2.36) and eq. (2.37) is provided in appendix A.12. Therefore, we know that if $\hat{H}(z)$ is fixed then $\hat{H}'(z)$ is also fixed. □

2.11 Conclusion and future work

In this chapter, we have presented a framework for reasoning about equivalence between a broad class of iterative algorithms by using ideas from control theory to represent optimization algorithms. The main insight is that by representing an algorithm as a linear dynamical system in feedback with a static nonlinearity, we can recognize equivalent algorithms by detecting algebraic relations between the transfer functions of the associated linear systems. This framework can identify algorithms that result in the same sequence of oracle calls, or algorithms that are the same up to shifts of the update equations, repetition of the updates with the same unit block, and conjugation of the function oracles. These ideas are implemented in the software package LINNAEUS, which allows researchers to search for algorithms that are related to a given input and identify parameter settings that make the algorithms equivalent. Our goal is to allow researchers add new algorithms to LINNAEUS as they are developed, so that LINNAEUS can remain a valuable resource for algorithm designers seeking to understand connections (if any) to previous methods.

Our framework requires that the algorithm is linear in the state and oracle outputs, but not necessarily in the parameters. This constraint still allows us to handle a surprisingly large class of algorithms. There are several interesting directions for future work.

Can we detect equivalence between stochastic or randomized algorithms? Our framework applies to such algorithms with almost no modifations, simply by allowing random oracles. For example, we can accept oracles like random search $\operatorname{argmin}\{f(x + \omega_i) : i = 1, \ldots, k\}$, stochastic gradient $\nabla f(x) + \omega$, or noisy

gradient $\nabla f(x + \omega)$. The definition of oracle equivalence would need a slight modification: for algorithms that use (pseudo-)randomized oracles, two algorithms are oracle-equivalent if they generate identical sequences of oracle calls given the same random seed.

Can we detect equivalence between parallel or distributed algorithms? Surprisingly, our framework still works for parallel or distributed algorithms. Notice that in a parallel algorithm, many oracle calls may be independently executed on different processors at about the same time. The precise ordering of these calls is not determined by the algorithm, and so different runs of the algorithm can generate different oracle sequences. However, all the possible oracle sequences generated by the same algorithm share the same dependence graph. Using the formalism defined in section 2.7.1, we can see that our framework can identify equivalence between parallel or distributed algorithms using the expanded definition of oracle equivalence: two algorithms are oracle-equivalent if there exists a way of writing each algorithm as a sequence of updates so that they generate identical sequences of oracle calls.

Can we detect equivalence between adaptive or nonlinear algorithms? Transfer functions are only defined for linear time-invariant (LTI) systems, so the LTI assumption in our framework is critical. Nevertheless, many of the other concepts from section 2.4.3 do extend to systems that are *almost* LTI. For example, an algorithm with parameters that change on a fixed schedule but is otherwise linear, such as gradient descent with a diminishing stepsize, can be regarded as a linear time-varying (LTV) system [5], and the notion of a transfer function has been generalized to LTV systems [55]. If, instead, the parameters change adaptively based on the other state variables, the system can be regarded

as a linear parameter varying (LPV) system [72] or a switched system [98]. Examples of such algorithms include nonlinear conjugate gradient methods and quasi-Newton methods.

For these more complicated cases, it is still reasonable to ask whether two algorithms invoke the same sequence of oracle calls. Discovering representations for nonlinear or time-varying algorithms that suffice to check equivalence is an interesting direction for future research.

CHAPTER 3

NYSADMM: FASTER COMPOSITE CONVEX OPTIMIZATION VIA LOW-RANK APPROXIMATION

This chapter introduces NysADMM for faster composite convex optimization aimed to address the scalability issue in large-scale optimization. The contents are mainly based on [112]. Section 3.2 introduces the NysADMM algorithm and necessary background from RandNLA. Section 3.3 lists a variety of applied problems that can be solved by NysADMM. Section 3.4 states the theoretical guarantees for NysADMM. Section 3.5 compares NysADMM and standard optimization solvers numerically on several applied problems.

3.1 Introduction

Consider the composite convex optimization problem

$$\text{minimize}_{x\in\mathbb{R}^d}\ \ell(Ax;b) + r(x). \tag{3.1}$$

We assume that ℓ and r are convex and ℓ is smooth. In machine learning, generally ℓ is a loss function, r is a regularizer, $A \in \mathbb{R}^{n\times d}$ is a feature matrix, and $b \in \mathbb{R}^n$ is the label or response. Throughout this chapter we assume that a solution to (3.1) exists. A canonical example of (3.1) is the lasso problem,

$$\text{minimize}\ \frac{1}{2}\|Ax-b\|_2^2 + \gamma\|x\|_1, \tag{3.2}$$

where $\ell(Ax;b) = \frac{1}{2}\|Ax-b\|_2^2$ and $r(x) = \gamma\|x\|_1$. We discuss more applications of (3.1) in section 3.3.

The alternating directions method of multipliers (ADMM) is a popular algorithm to solve optimization problems of the form (3.1). However, as we dis-

cussed in chapter 1, when the matrix A is large, ADMM suffers from scalability issue, that is each iteration of ADMM requires solving a large subproblem. For example, consider the lasso where the loss ℓ is quadratic. At each iteration, ADMM solves a regularized least-squares problem at a cost of $O(nd^2)$ flops. On the other hand, it is not necessary to solve each subproblem exactly to ensure convergence: ADMM strategies that solve the subproblems inexactly are called inexact ADMM, and can be shown to converge when the sequence of errors is summable [35]. Unfortunately, it can be challenging even to satisfy this relaxed criterion. Consider again the lasso problem. At each iteration, inexact ADMM solves the regularized least-squares subproblem (3.4) approximately, for example, using the iterative method of conjugate gradients (CG). We call this method inexact ADMM with CG. The number of CG iterations required to achieve accuracy ϵ increases with the square root of the condition number κ_2 of the regularized Hessian, $O\left(\sqrt{\kappa_2}\log(\frac{\kappa_2}{\epsilon})\right)$. Alas, the condition number of large-scale data matrices is generally high, and later iterations of inexact ADMM require high accuracy, so inexact ADMM with CG still converges too slowly to be practical.

In this work we show how to speed up inexact ADMM using preconditioned conjugate gradients (PCG) as a subproblem solver. We precondition with randomized Nyström preconditioning [41], a technique inspired by recent developments in randomized numerical linear algebra (RandNLA). We call the resulting algorithm NysADMM ("nice ADMM"): inexact ADMM with PCG using randomized Nyström preconditioning. The Nyström preconditioner reduces the number of iterations required to solve the subproblem to ϵ-accuracy to $O\left(\log(\frac{1}{\epsilon})\right)$, independent of the condition number. For non-quadratic loss functions, NysADMM uses linearized inexact ADMM and accelerates the linear subproblem solve similarly.

3.1.1 Contributions

1. We provide a general algorithmic framework for solving large scale lasso, ℓ_1-regularized logistic regression, and SVM problems.
2. Our theory shows that at each iteration only a constant number of matrix vector products (matvecs) are required to solve the ADMM subproblem, provided we have constructed the preconditioner appropriately. If the loss function is quadratic, only a constant number of matvecs are required to achieve convergence.
3. We develop a practical adaptive algorithm that increases the rank until the conditions of our theory are met, which ensures the theoretical benefits of the method can be realized in practice.
4. Even a preconditioner with lower rank often succeeds in speeding up inexact ADMM with PCG. Our analysis is also able to explain this phenomenon.
5. Our algorithm beats standard solvers such as glmnet, SAGA, and LIBSVM on large dense problems like lasso, logistic regression, and kernalized SVMs: it yields equally accurate solutions and often runs 2–4 times faster.

3.1.2 Related work

Our work relies on recent advancements in RandNLA for solving regularized least squares problems $(A^TA + \mu I)x = A^Tb$ for x, given a design matrix $A \in \mathbb{R}^{n\times d}$, righthand side $b \in \mathbb{R}^n$, and regularization $\mu \in \mathbb{R}$, using a *sketch* of the design matrix A [57]. NysADMM adapts the randomized Nyström preconditioner of [41].

These algorithms begin by forming a *sketch* $Y = A\Omega$ of A (or A^T) with a random dimension reduction map $\Omega \in \mathbb{R}^{d \times s}$ [68, 109]. For example, Ω may be chosen to have iid Gaussian entries. These algorithms obtain significant computational speedups by using a sketch size $s \ll \min\{n, d\}$ and working with the sketch in place of the original matrix to construct a preconditioner for the linear system. [41] and [57] show that these randomized preconditioners work well when the sketch size grows with the *effective dimension* (eq. (3.7)) of the Gram matrix (assuming, for [57] that we have access to a matrix square root). As the effective dimension is never larger than d and often significantly smaller, these results substantially improve on prior work in randomized preconditioning [70, 89] that requires a sketch size $s \gtrsim d$. Many applications require even smaller sketch sizes: for example, for NysADMM, a fixed sketch size $s = 50$ suffices even for extremely large problems.

We are not the first to use RandNLA to accelerate iterative optimization. [47, 83] both use iterative sketching to accelerate Newton's method, while [23] use randomized preconditioning to accelerate interior point methods for linear programmming. The approach taken here is closest in spirit to [23], as we also use randomized preconditioning. However, the preconditioner used in [23] requires the data matrix to have many more columns than rows, while ours can handle any (sufficiently large) dimensions.

NysADMM can solve many traditional machine learning problems, such as lasso, regularized logistic regression, and support vector machines (SVMs). In contrast, standard solvers for these problems use a wider variety of convex optimization techniques. For example, one popular lasso solver, glmnet [43], relies on coordinate descent (CD), while solvers for SVMs, such as LIBSVM [21], more

often use sequential minimal optimization [84], a kind of pairwise CD on the dual problem. For regularized logistic regression, especially for ℓ_1 regularization, stochastic gradient algorithms are most commonly used [26, 92]. Other authors propose to solve lasso with ADMM [12, 111]. Our work, motivated by the ADMM quadratic programming framework of [97], is the first to accelerate ADMM with randomized preconditioning, thereby improving on the performance of standard CD or stochastic gradient solvers for each of these important classes of machine learning problems on large-scale dense data. Unlike [97], our work relies on inexact ADMM and can handle non-quadratic loss functions, which allows NysADMM to solve problems such as regularized logistic regression.

3.1.3 Notation and preliminaries

We call a matrix psd if it is positive semidefinite. The notation $a \gtrsim b$ means that $a \geq Cb$ for some absolute constant C. Given a matrix H, we denote its spectral norm by $\|H\|$. We denote the Moore-Penrose pseudoinverse of a matrix M by $M^\dagger$. For $\rho > 0$ and a symmetric psd matrix H, we define $H_\rho = H + \rho I$. We say a positive sequence $\{\varepsilon^k\}_{k=1}^\infty$ is summable if $\sum_{k=1}^\infty \varepsilon^k < \infty$. We denote the Loewner ordering on the cone of symmetric psd matrices by $\preceq$, that is $A \preceq B$ if and only if $B - A$ is psd.

3.2 Algorithm

3.2.1 Inexact linearized ADMM

To solve problem (3.1), we apply the ADMM framework. Algorithm 3.2.1 shows the standard ADMM updates, where the regularizer $r = g + h$ is split into a smooth part g and a nonsmooth part h.

Algorithm 3.2.1 ADMM

input: feature matrix A, response b, loss function ℓ, regularization g and h, step-size ρ

repeat

$x^{k+1} = \operatorname{argmin}_x\{\ell(Ax; b) + g(x) + \frac{\rho}{2}\|x - z^k + u^k\|_2^2\}$

$z^{k+1} = \operatorname{argmin}_z\{h(z) + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2\}$

$u^{k+1} = u^k + x^{k+1} - z^{k+1}$

until convergence

output: solution $x_\star$ of problem (3.1)

In each iteration, two subproblems are solved sequentially to update variables x and z. The z-subproblem often has a closed-form solution. For example, if $h(x) = \|x\|_1$, the z-subproblem is the soft thresholding, and if h is the indicator function of a convex set C, the z-subproblem is projection onto the set C.

There is usually no closed-form solution for the x-subproblem. Instead, it is usually solved inaccurately by an iterative scheme, especially for large-scale applications. To simplify the subproblem, inspired by linearized ADMM, we assume ℓ and g are twice differentiable and notice that the x update is close to the minimum of a quadratic function given by the Taylor expansion of ℓ and g

at the current iterate:

$$\tilde{x}^{k+1} = \text{argmin}_x\{\ell(A\tilde{x}^k; b) + (x - \tilde{x}^k)^T A^T \nabla \ell(A\tilde{x}^k; b) + \frac{1}{2}(x - \tilde{x}^k)^T A^T H^\ell(A\tilde{x}^k; b)A(x - \tilde{x}^k) + g(\tilde{x}^k)$$
$$+ (x - \tilde{x}^k)^T \nabla g(\tilde{x}^k) + \frac{1}{2}(x - \tilde{x}^k)^T H^g(\tilde{x}^k)(x - \tilde{x}^k) + \frac{\rho}{2}\|x - \tilde{z}^k + \tilde{u}^k\|_2^2\}. \tag{3.3}$$

Here H^ℓ and H^g are the Hessian of ℓ and g respectively. We assume throughout the chapter that H^ℓ and H^g are psd matrices, this is a very minor assumption, and is satisfied by all the applications we consider. The solution to this quadratic minimization may be obtained by solving the linear system

$$(A^T H^\ell(A\tilde{x}^k; b)A + H^g(\tilde{x}^k) + \rho I)x = r^k \tag{3.4}$$

where $r^k = \rho\tilde{z}^k - \rho\tilde{u}^k + A^T H^\ell(A\tilde{x}^k; b)A\tilde{x}^k + H^g(\tilde{x}^k)\tilde{x}^k - A^T \nabla \ell(A\tilde{x}^k; b) - \nabla g(\tilde{x}^k)$.

The inexact ADMM algorithm we propose solves (3.4) approximately at each iteration.

Algorithm 3.2.2 Inexact ADMM

input: feature matrix A, response b, loss function ℓ, regularization g and h, step-size ρ, positive summable sequence $\{\varepsilon^k\}_{k=0}^\infty$
  repeat
    find $\tilde{x}^{k+1}$ that solves (3.4) within tolerance ε^k
    $\tilde{z}^{k+1} = \text{argmin}_z\{h(z) + \frac{\rho}{2}\|\tilde{x}^{k+1} - z + \tilde{u}^k\|_2^2\}$
    $\tilde{u}^{k+1} = \tilde{u}^k + \tilde{x}^{k+1} - \tilde{z}^{k+1}$
  until convergence
output: solution $x_\star$ of problem (3.1)

For a quadratic loss ℓ, when $\sum_{k=0}^\infty \varepsilon^k < \infty$ and under various other conditions, if optimization problem (3.1) has an optimal solution, the $\{\tilde{x}^k\}_{k=0}^\infty$ sequence generated by algorithm 3.2.2 converges to the optimal solution of (3.1) [35, 37].

From [12], quantity $r_d^{k+1} = \rho(\tilde{z}^k - \tilde{z}^{k+1})$ can be regarded as the dual residual and $r_p^{k+1} = \tilde{x}^{k+1} - \tilde{z}^{k+1}$ can be viewed as the primal residual at iteration $k + 1$. This suggests that we can terminate the ADMM iterations when the primal and dual

residuals become very small. The primal and dual tolerances can be chosen based on an absolute and relative criterion, such as

$$\|r_p^k\|_2 \leq \epsilon^{\text{abs}} + \epsilon^{\text{rel}} \max\{\|\tilde{x}^k\|_2, \|\tilde{z}^k\|_2\}$$
$$\|r_d^k\|_2 \leq \epsilon^{\text{abs}} + \epsilon^{\text{rel}} \|\rho \tilde{u}^k\|_2.$$

The relative criteria ϵ^{rel} might be 10^{-3} or 10^{-4} in practice. The choice of absolute criteria ϵ^{abs} depends on the scale of the variable values. More details can be found in [12].

3.2.2 Randomized Nyström approximation and PCG

Nyström approximation constructs a low-rank approximation of a symmetric psd matrix H. Let $\Omega \in \mathbb{R}^{d \times s}$ be a test matrix (often, random Gaussian [41, 101]) with sketch size $s \geq 1$. The Nyström approximation with respect to Ω is given by

$$H\langle\Omega\rangle = (H\Omega)(\Omega^T H\Omega)^{\dagger}(H\Omega)^T. \tag{3.5}$$

The Nyström approximation $H\langle\Omega\rangle$ is symmetric, psd, and has rank at most s (lemma B.1.1). Naive implementation of the Nyström approximation based on (3.5) is numerically unstable. Algorithm B.2.1 in appendix B.2 states a stable procedure to compute a randomized Nyström approximation from [101].

Algorithm B.2.1 returns the randomized Nyström approximation of matrix H in the form of an eigendecomposition: $H_{\text{nys}} = U\hat{\Lambda}U^T$. Let $\hat{\lambda}_s$ be the sth eigenvalue. The randomized Nyström preconditioner and its inverse take the form

$$\begin{aligned} P &= \frac{1}{\hat{\lambda}_s + \rho} U(\hat{\Lambda} + \rho I)U^T + (I - UU^T), \\ P^{-1} &= (\hat{\lambda}_s + \rho)U(\hat{\Lambda} + \rho I)^{-1}U^T + (I - UU^T) \end{aligned} \tag{3.6}$$

[41]. In a slight abuse of terminology, we sometimes refer to the sketch size s as the rank of the Nyström preconditioner. We will use the the term sketch size and rank interchangeably throughout this chapter. The Nyström preconditioner may be applied to vectors in $O(ds)$ time and only requires $O(ds)$ floating point numbers to store. The details of how to implement PCG with (3.6) are provided in appendix B.2 in algorithm B.2.2. We now provide some background on Nyström PCG and motivation for why we have paired it with ADMM.

Nyström PCG improves on standard CG both in theory and in practice for matrices with a small *effective dimension* [41], which we now define. Given a symmetric psd matrix $H \in \mathbb{R}^{d\times d}$ and regularization $\rho > 0$, the *effective dimension* of H is

$$d_{\text{eff}}(\rho) = \text{tr}(H(H + \rho I)^{-1}). \tag{3.7}$$

The effective dimension may be viewed as smoothed count of the eigenvalues of H greater than or equal to ρ. We always have $d_{\text{eff}}(\rho) \leq d$, and we expect $d_{\text{eff}}(\rho) \ll d$ whenever H exhibits spectral decay.

In machine learning, most feature matrices naturally exhibit polynomial or exponential spectral decay [29], thus we expect that $d_{\text{eff}} \ll d$. The randomized Nyström preconditioner in [41] exploits the smallness of $d_{\text{eff}}(\rho)$ to build an highly effective preconditioner. [41] show that if (3.6) is constructed with a sketch size $s \gtrsim d_{\text{eff}}(\rho)$, then the condition number of the preconditioned system is constant with high probability. An immediate consequence is that PCG solves the preconditioned system to ϵ-accuracy in $O\left(\log(\frac{1}{\epsilon})\right)$ iterations, independent of the condition number of H.

Observe the Hessian $A^T H^\ell A + H^g$ in the inexact ADMM subproblem (3.4) is formed from the feature matrix A. Based on the preceding discussion, we expect

the Hessian to exhibit spectral decay and for the effective dimension to be small to moderate in size. Hence we should expect Nyström PCG to accelerate the solution of (3.4) significantly.

3.2.3 NysADMM

Integrating Nyström PCG with inexact ADMM, we obtain NysADMM, presented in algorithm 3.2.3.

Algorithm 3.2.3 NysADMM

input: feature matrix A, response b, loss function ℓ, regularization g and h, step-size ρ, positive summable sequence $\{\varepsilon^k\}_{k=0}^{\infty}$
$[U, \hat{\Lambda}]$ = RandNyströmApprox($A^T H^{\ell} A + H^g, s$) ▷ use algorithm B.2.1 in appendix B.2
repeat
use Nyström PCG (algorithm B.2.2 in appendix B.2) to find $\tilde{x}^{k+1}$ that solves (3.4) within tolerance ε^k
$\tilde{z}^{k+1} = \operatorname{argmin}_z \{h(z) + \frac{\rho}{2}\|\tilde{x}^{k+1} - z + \tilde{u}^k\|_2^2\}$
$\tilde{u}^{k+1} = \tilde{u}^k + \tilde{x}^{k+1} - \tilde{z}^{k+1}$
until convergence
output: solution $x_\star$ of problem (3.1)

Our theory for algorithm 3.2.3, shows that if the sketch size $s \gtrsim d_{\text{eff}}(\rho)$, then with high probability subproblem (3.4) will be solved to ϵ-accuracy in $O\left(\log(\frac{1}{\epsilon})\right)$ iterations (corollary 3.4.1). When the loss ℓ is quadratic and the sequence of tolerances $\{\varepsilon^k\}_{k=0}^{\infty}$ is decreasing with $\sum_{k=0}^{\infty} \varepsilon^k < \infty$, NysADMM is guaranteed to converge as $k \to \infty$ with only a constant number of matvecs per iteration (theorem 3.4.2). Table 3.1 compares the complexity of inexact ADMM with CG vs. NysADMM for K iterations under the hypotheses of theorem 3.4.2. NysADMM achieves a significant decrease in runtime over inexact ADMM with CG, as the iteration complexity no longer depends on the condition number κ_2.

Table 3.1: Complexity comparison, for a quadratic loss with Hessian H. Here T_{mv} is the time to compute a matrix vector product with H, κ_2 is the condition number of H, and ε^k is the precision of the kth subproblem solve (3.4).

Method	Complexity
Inexact ADMM with CG	$O\left(\sum_{k=1}^{K} T_{\text{mv}} \sqrt{\kappa_2} \log\left(\frac{\kappa_2}{\varepsilon^k}\right)\right)$
NysADMM	$O\left(T_{\text{mv}} d_{\text{eff}}(\rho)\right) + \sum_{k=1}^{K} T_{\text{mv}} \left(4 + \left\lceil 2 \log\left(\frac{R}{\varepsilon^k \rho}\right) \right\rceil\right)$

3.2.4 AdaNysADMM

Two practical problems remain in realizing the success predicted by the theoretical analysis of table 3.1. These bounds are achieved by selecting the sketch size to be $d_{\text{eff}}(\rho)$, but the effective dimension is 1) seldom known in practice, and 2) often larger than required to achieve good convergence of NysADMM. Fortunately, a simple adaptive strategy for choosing the sketch size, inspired by [41], can achieve the same guarantees as in table 3.1. This strategy chooses a tolerance ϵ and doubles the sketch size s until the empirical condition number $\frac{\hat{\lambda}_s + \rho}{\rho}$ satisfies

$$\frac{\hat{\lambda}_s + \rho}{\rho} \leq 1 + \epsilon. \tag{3.8}$$

Theorem 3.4.3 guarantees that (3.8) holds when $s \geq d_{\text{eff}}(\rho)$ and that when (3.8) holds, the true condition number is on the order of $1 + \epsilon$ with high probability. We refer to (3.8) as the empirical condition number as it provides an estimate of the true condition number of the preconditoned system (theorem 3.4.3).

Thus, to enjoy the guarantees of theorem 3.4.3 in practice, we may employ the adaptive version of NysADMM, which we call AdaNysADMM. We provide the pseudocode for AdaNysADMM in algorithm B.3.2 in appendix B.2. Furthermore, as we use a Gaussian test matrix, it is possible to construct a larger sketch from a smaller one. Hence the total computational work needed by the

adaptive strategy is not much larger than if the effective dimension were known in advance. Indeed, AdaNysADMM differs from NysADMM only in the construction of the preconditioner. The dominant cost in forming the precondition is computing the sketch is $H\Omega$, which costs $O(T_{\text{mv}} d_{\text{eff}}(\rho))$. As AdaNysADMM reuses computation, the dominant complexity for constructing the Nyström preconditioner remains $O(T_{\text{mv}} d_{\text{eff}}(\rho))$. Consequently, the overall complexity of AdaNysADMM is the same as NysADMM in table 3.1.

3.3 Applications

Here we discuss various applications that can be reformulated as instances of (3.1) and solved by algorithm 3.2.3.

3.3.1 Elastic net

Elastic net generalizes lasso and ridge regression by adding both the ℓ_1 and ℓ_2 penalty to the least squares problem:

$$\text{minimize} \quad \frac{1}{2}\|Ax - b\|_2^2 + \frac{1}{2}(1-\gamma)\|x\|_2^2 + \gamma\|x\|_1 \tag{3.9}$$

Parameter $\gamma > 0$ interpolates between the ℓ_1 and ℓ_2 penalties. NysADMM applies with $\ell(Ax; b) = \frac{1}{2}\|Ax - b\|_2^2$, $g(x) = \frac{1}{2}(1-\gamma)\|x\|_2^2$, and $h(x) = \gamma\|x\|_1$. The Hessian matrices for ℓ and g are $A^T A$ and $(1-\gamma)I$ respectively.

3.3.2 Regularized logistic regression

Regularized logistic regression minimizes a logistic loss function together with an ℓ_1 regularizer:

$$\text{minimize } -\sum_i (b_i(Ax)_i - \log(1 + \exp((Ax)_i))) + \gamma\|x\|_1 \tag{3.10}$$

NysADMM applies with $\ell(Ax; b) = -\sum_i (b_i(Ax)_i - \log(1 + \exp((Ax)_i)))$ and $h(x) = \gamma\|x\|_1$. The inexact ADMM update chooses x^{k+1} to minimize a quadratic approximation of the log-likelihood,

$$\text{minimize } \frac{1}{2}\sum_i w_i^k(q_i^k - (Ax)_i)^2 + \frac{\rho}{2}\|x - \tilde{z}^k + \tilde{u}^k\|_2^2,$$

where w_i^k and q_i^k depend on the current estimate $\tilde{x}^k$ as

$$\begin{aligned} w_i^k &= \frac{1}{2 + \exp(-(A\tilde{x}^k)_i) + \exp((A\tilde{x}^k)_i)} \\ q_i^k &= (A\tilde{x}^k)_i + \frac{b_i - \frac{1}{1+\exp(-(A\tilde{x}^k)_i)}}{w_i^k}. \end{aligned}$$

Therefore, the solution of the x-subproblem can be approximated by solving the linear system

$$(A^T \text{diag}(w^k)A + \rho I)x = \rho\tilde{z}^k - \rho\tilde{u}^k + A^T \text{diag}(w^k)q^k.$$

Here w^k and q^k are the vectors for w_i^k and q_i^k. The Hessian matrix of ℓ is given by $A^T \text{diag}(w^k)A$.

3.3.3 Support vector machine

To reformulate the SVM problem for solution with NysADMM, consider the dual SVM problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}x^T \text{diag}(b)K\text{diag}(b)x - \mathbf{1}^T x \\ \text{subject to} \quad & x^T b = 0 \\ & 0 \le x \le C. \end{aligned} \tag{3.11}$$

Variable x is the dual variable, b is the label or response, and C is the penalty parameter for misclassification. For linear SVM, $K = A^T A$ where A is a feature matrix; and for nonlinear SVM, K is the corresponding kernel matrix. The SVM problem can be reformulated as (3.1) by setting $\ell(Ax; b) = \frac{1}{2} x^T \mathrm{diag}(b) K \mathrm{diag}(b) x$, $g(x) = -\mathbf{1}^T x$, and h is the indicator function for convex constraint set $x^T b = 0,\ 0 \leq x \leq C$. The Hessian matrix for ℓ is $\mathrm{diag}(b) K \mathrm{diag}(b)$.

3.4 Convergence analysis

This section provides a convergence analysis for NysADMM. All proofs for the results in this section may be found in appendix B.1. First we show Nyström PCG can solve any quadratic problem in a constant number of iterations.

Theorem 3.4.1. Let H be a symmetric positive semidefinite matrix, $\rho > 0$ and set $H_\rho = H + \rho I$. Suppose we construct the randomized Nyström preconditioner with sketch size $s \geq 8\left(\sqrt{d_{\mathrm{eff}}(\rho)} + \sqrt{8 \log(\frac{16}{\delta})}\right)^2$. Then

$$\kappa_2(P^{-1/2} H_\rho P^{-1/2}) \leq 8 \tag{3.12}$$

with probability at least $1 - \delta$.

Theorem 3.4.1 strengthens results in [41], which provides sharp expectation bounds on the condition number of the preconditioned system, but gives loose high probability bounds based on Markov's inequality. Our result tightens these bounds, showing that Nyström PCG enjoys an exponentially small failure probability.

As an immediate corollary, we can solve (3.4) with a few iterations of PCG using the Nyström preconditioner.

Corollary 3.4.1. Instate the hypotheses of theorem 3.4.1 and let $\tilde{x}_\star$ denote the solution of (3.4). Then with probability at least $1-\delta$, the iterates $\{x_t\}_{t\geq 1}$ produced by Nyström PCG on problem (3.4) satisfy

$$\frac{\|x_t - \tilde{x}_\star\|_2}{\|\tilde{x}_\star\|_2} \leq \left(\frac{1}{2}\right)^{t-4}. \tag{3.13}$$

Thus, after $t \geq \left\lceil \frac{\log\left(\frac{16\|\tilde{x}_\star\|_2}{\epsilon}\right)}{\log(2)} \right\rceil$ iterations,

$$\|x_t - \tilde{x}_\star\|_2 \leq \epsilon. \tag{3.14}$$

Corollary 3.4.1 ensures that we can efficiently solve the sub-problem to the necessary accuracy at each iteration. This result allows us to prove convergence of NysADMM.

Theorem 3.4.2. Consider the problem in (3.1) with quadratic loss $\ell(Ax;b) = \frac{1}{2}\|Ax-b\|_2^2$ and the smooth part g of regularizer r has constant Hessian. Define initial iterates $\tilde{x}^0$, $\tilde{z}^0$ and $\tilde{u}^0 \in \mathbb{R}^d$, stepsize $\rho > 0$, and summable tolerance sequence $\{\varepsilon^k\}_{k=0}^\infty \subset \mathbb{R}_+$. Assume at kth ADMM iteration, the norm of the right-hand side of the linear system r^k is bounded by constant R for all k. Construct the Nyström preconditioner with sketch size

$$s \geq 8\left(\sqrt{d_{\text{eff}}(\rho)} + \sqrt{8\log\left(\frac{16}{\delta}\right)}\right)^2$$

and solve problem (3.1) with NysADMM, using $T^k = 4 + \left\lceil 2\log\left(\frac{R}{\varepsilon^k\rho}\right)\right\rceil$ iterations for PCG at the kth ADMM iteration. Then with probability at least $1-\delta$,

1. For all $k \geq 0$, each iterate $\tilde{x}^{k+1}$ satisfies

$$\|\tilde{x}^{k+1} - x^{k+1}\|_2 \leq \varepsilon^k, \tag{3.15}$$

where x^{k+1} is the exact solution of (3.4).

2. As $k \to \infty$, $\{\tilde{x}^k\}_{k=0}^{\infty}$ converges to a solution of the primal (3.1) and $\{\rho\tilde{u}^k\}_{k=0}^{\infty}$ converges to a solution of the dual problem of (3.1).

Theorem 3.4.2 establishes convergence of NysADMM for a quadratic loss. The quadratic loss already covers many applications of interest including the lasso, elastic-net, and SVMs. We conjecture that a modification of our argument can show that NysADMM converges linearly for any strongly convex loss, but we leave this extension to future work.

The next result makes rigorous the claims made in section 3.2.4: it shows we can determine whether or not we have reached the effective dimension by monitoring the empirical condition number $(\hat{\lambda}_s + \rho)/\rho$.

Theorem 3.4.3. Suppose, for some user defined tolerance $\epsilon > 0$, the sketch size satisfies

$$s \geq 8\left(\sqrt{d_{\text{eff}}\left(\frac{\epsilon\rho}{6}\right)} + \sqrt{8\log\left(\frac{16}{\delta}\right)}\right)^2.$$

Then the empirical condition number of the Nyström preconditioned system $P^{-1/2}H_rP^{-1/2}$ satisfies

$$\frac{\hat{\lambda}_s + \rho}{\rho} \leq 1 + \frac{\epsilon}{42}. \tag{3.16}$$

Furthermore, with probability at least $1 - \delta$,

$$\left|\kappa_2(P^{-1/2}H_\rho P^{-1/2}) - \frac{\hat{\lambda}_s + \rho}{\rho}\right| \leq \epsilon. \tag{3.17}$$

Theorem 3.4.3 shows that once the empirical condition number is sufficiently close to 1, so too is the condition number of the preconditioned system. Hence it is possible to reach the effective dimension by doubling the sketch size of the Nyström approximation until the empirical condition number falls below the

desired tolerance. Theorem 3.4.3 ensures the true condition number is close to this empirical estimate with high probability.

Theorem 3.4.3 also helps explain why sketch sizes much smaller than the effective dimension can succeed in practice. The point is best illustrated by instantiating an explicit parameter selection in theorem 3.4.3, which yields the following corollary.

Corollary 3.4.2. Instate the hypotheses of theorem 3.4.3 with $\epsilon = 100$. Then with a sketch size of $s \gtrsim d_{\text{eff}}(16\rho)$ the following holds

1. $(\hat{\lambda}_s + \rho)/\rho \leq 1 + \frac{100}{42}$.
2. With probability at least $1 - \delta$,

$$\left| \kappa_2(P^{-1/2} H_\rho P^{-1/2}) - 1 - \frac{100}{42} \right| \leq 100.$$

Corollary 3.4.2 shows that for a coarse tolerance of $\epsilon = 100$, a sketch size of $s \gtrsim d_{\text{eff}}(16\rho)$ suffices to ensure that the condition number of $P^{-1/2} H_\rho P^{-1/2}$ is no more than around 100. Two practical observations cement the importance of this corollary. First, $d_{\text{eff}}(16\rho)$ is often significantly smaller than $d_{\text{eff}}(\rho)$, possibly by an order of magnitude or more. Second, with a condition number around 100, PCG is likely to converge very quickly. In fact, for modest condition numbers, PCG is known to converge much faster in practice than the theory would suggest [100]. It is only when the condition number reaches around 10^3, that convergence starts to slow. Thus, corollary 3.4.2 helps explain why it is not necessary for the sketch size to equal the effective dimension in order for NysADMM to obtain significant accelerations.

3.5 Numerical experiments

Table 3.2: Statistics of experiment datasets.

Name	instances n	features d	nonzero %
STL-10	13000	27648	96.3
CIFAR-10	60000	3073	99.7
CIFAR-10-rf	60000	60000	100.0
smallNorb-rf	24300	30000	100.0
E2006.train	16087	150348	0.8
sector	6412	55197	0.3
p53-rf	16592	20000	100.0
connect-4-rf	16087	30000	100.0
realsim-rf	72309	50000	100.0
rcv1-rf	20242	30000	100.0
cod-rna-rf	59535	60000	100.0

In this section, we evaluate the performance of NysADMM on different large-scale applications: lasso, ℓ_1-regularized logistic regression, and SVM. For each type of problems, we compare NysADMM with popular standard solvers. We run all experiments on a server with 128 Intel Xeon E7-4850 v4 2.10GHz CPU cores and 1056GB. We repeat every numerical experiment ten times and report the mean solution time. We highlight the best-performing method in bold. The tolerance of NysADMM at each iteration is chosen as the geometric mean $\varepsilon^{k+1} = \sqrt{r_p^k r_d^k}$ of the ADMM primal residual r_p and dual residual r_d at the previous iteration, as in [97]. See [12] for more motivation and details. An alternative is to choose the tolerance sequence as any decaying sequence with respect to the righthand side norm as the number of NysADMM iteration increases, e.g., $\varepsilon^k = \|r^k\|_2/k^\beta$, where β is a predefined factor. These two strategies perform similarly; our experiments use the first strategy.

We choose a sketch size $s = 50$ to compute the Nyström approximation throughout our experiments. Inspired by theorem 3.4.3 and corollary 3.4.2, even

if the sketch size is much smaller than the effective dimension, NysADMM can still achieve significant acceleration in practice.

To support experiments with standard solvers, for each problem class we use the same stopping criterion and other parameter settings as the standard solver. These experiments use datasets with $n > 10,000$ or $d > 10,000$ from LIBSVM [21], UCI [32], and OpenML [105], with statistics summarized in table 3.2. We use a random feature map [86, 87] to generate features for the data sets CIFAR-10, smallnorb, realsim, rcv1, and cod-rna, which increases both predictive performance and problem dimension.

3.5.1 Lasso

This subsection demonstrates the performance of NysADMM to solve the standard lasso problem (3.2). Here we compare NysADMM with three standard lasso solvers, SSNAL [62], mfIPM [40], and glmnet [43]. SSNAL is a Newton method based solver; mfIPM is an interior point method based solver and glmnet is a coordinate descent based solver. In practice, these three solvers and NysADMM rely on different stopping criteria. In order to make a fair comparison, in our experiments, the accuracy of a solution x for (3.2) is measured by the following relative Karush–Kuhn–Tucker (KKT) residual [62]:

$$\eta = \frac{\|x - \mathrm{prox}_{\gamma\|\cdot\|_1}(x - A^T(Ax - b))\|}{1 + \|x\| + \|Ax - b\|}. \tag{3.18}$$

For a given tolerance ϵ, we stop the tested algorithms when $\eta < \epsilon$. Note that stopping criterion (3.18) is rather strong: if $\eta \leq 10^{-2}$ for NysADMM, then the primal and dual gaps for ADMM are $\lesssim 10^{-4}$, which suffices for most applications. Indeed, for many machine learning problems, lower bounds on the statistical

performance of the estimator [65] imply an unavoidable level of statistical error that is greater than this optimization error for most applications. Optimizing the objective beyond the level of statistical error [3, 66] does not improve generalization. For standard lasso experiments, we fix the regularization parameter at $\gamma = 1$.

Table 3.3: Results for low precision lasso experiment.

Task	Time for $\epsilon = 10^{-1}$ (s)			
	NysADMM	mfIPM	SSNAL	glmnet
STL-10	**165**	573	467	278
CIFAR-10-rf	**251**	655	692	391
smallNorb-rf	**219**	552	515	293
E2006.train	**313**	875	903	554
sector	**235**	678	608	396
realsim-rf	**193**	–	765	292
rcv1-rf	**226**	563	595	273
cod-rna-rf	**208**	976	865	324

Table 3.4: Results for high precision lasso experiment.

Task	Time for $\epsilon = 10^{-2}$ (s)			
	NysADMM	mfIPM	SSNAL	glmnet
STL-10	**406**	812	656	831
CIFAR-10-rf	**715**	1317	1126	1169
smallNorb-rf	**596**	896	768	732
E2006.train	1657	1965	**1446**	2135
sector	957	1066	**875**	1124
realsim-rf	**732**	–	1035	922
rcv1-rf	**593**	853	715	736
cod-rna-rf	**715**	1409	1167	997

Table 3.3 and Table 3.4 show results for lasso experiments. The average solution time for NysADMM, mfIPM, SSNAL, and glmnet with $\epsilon = 10^{-1}, 10^{-2}$ on different tasks are provided. Here mfIPM fails to solve the realsim-rf instance

since it requires $n < d$. For precision of $\epsilon = 10^{-1}$, NysADMM is faster than all other solvers and at least 3 times faster than both mfIPM and SSNAL. For precision of $\epsilon = 10^{-2}$, NysADMM is still faster than all other solvers for all instances except E2006.train and sector. The results are fair since both SSNAL and mfIPM are second-order solvers and can reach high precision. NysADMM and glmnet are first-order solvers; they reach low precision quickly, but improve accuracy more slowly than a second order method. In practice, for large-scale machine learning problems, a low precision solution usually suffices, as decreasing optimization error beyond the statistical noise in the problem does not improve generalization. Further, our algorithm achieves bigger improvements on dense datasets compared with sparse datasets, as the factors of the Nyström approximation are dense even for sparse problems. To further illustrate the results, we

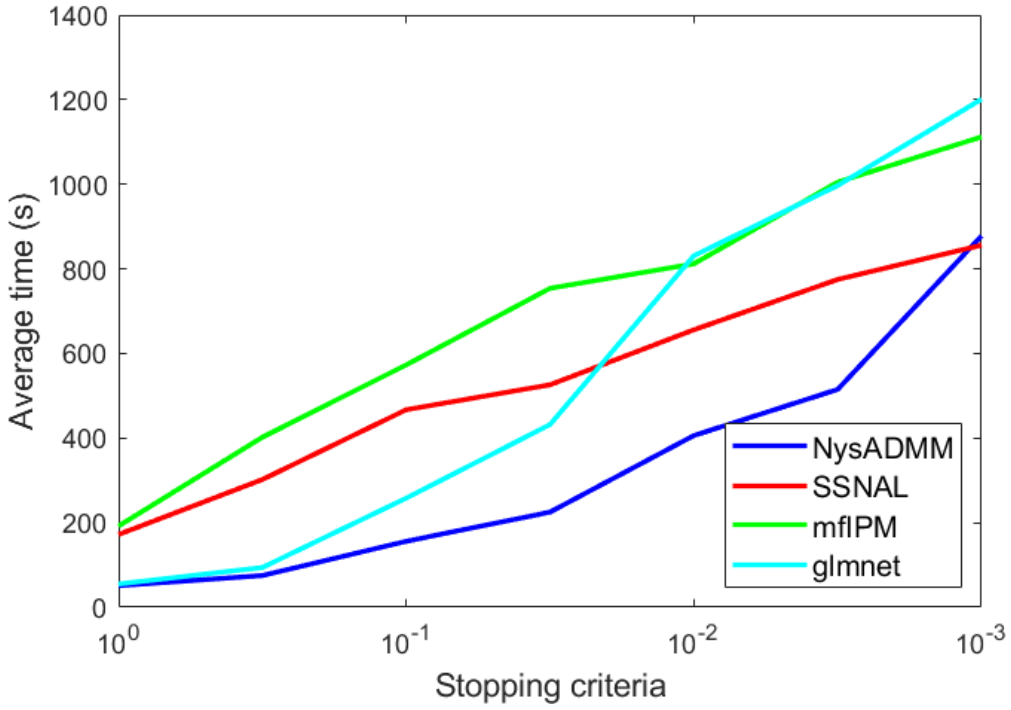

Figure 3.1: Solution times for varying tolerance ϵ on STL-10.

vary the value of ϵ from 1.0 to 10^{-3} on STL-10 task and plot the average solution

time for four methods in fig. 3.1. We can see NysADMM is as least as fast as other solvers when $\epsilon > 10^{-3}$, and often twice as fast for many practical values of ϵ.

3.5.2 ℓ_1-regularized logistic regression

This subsection demonstrates the performance of NysADMM on ℓ_1-regularized logistic regression, (3.10) from section 3.3.2. We test the method on binary classification problems using the same random feature map as in section 3.5.1.

The ℓ_1-regularized logistic regression experiments compare NysADMM with the SAGA algorithm, a stochastic average gradient like algorithm [26] implemented in sklearn, and the accelerated proximal gradient (APG) algorithm [9, 75, 81]. For the purpose of fair comparison, all the algorithms are stopped when the maximum relative change in the problem variable (that is, the regression coefficients) $\frac{\|x_k - x_{k+1}\|_\infty}{\|x_k\|_\infty}$ is less than the tolerance. The tolerance is set to 10^{-3}; other settings match the default settings of the sklearn logistic regression solver.

An overview of ℓ_1-regularized logistic regression experiment results are provided in table 3.5. NysADMM uniformly out performs SAGA, solving each problem at least twice as fast. Similarly, NysADMM is at least twice as fast as APG on all datasets except STL-10, where it performs comparably. In the cases of p53-rf and connect-4-rf, NysADMM runs significantly faster than its competitors, being four times faster than SAGA and three times faster than APG. These large performance gains are due to the size of the problem instances and their conditioning. From [26], the convergence speed of SAGA depends on the problem instance size and condition number. Our test cases have large instance

sizes and condition numbers, which lead to slow convergence of SAGA. The situation with APG is similar. Indeed, although ADMM and proximal gradient methods generally have the same $O(1/t)$-convergence rate [9, 50], NysADMM is less sensitive ill-conditioning than APG.

Table 3.5: Results for ℓ_1-regularized logistic regression experiment.

Task	NysADMM (s)	SAGA (s)	APG (s)
STL-10	3012	6083	**2635**
CIFAR-10-rf	**7884**	21256	17292
p53-rf	**528**	2116	1880
connect-4-rf	**866**	4781	7365
smallnorb-rf	**1808**	6381	4408
rcv1-rf	**1237**	3988	2759
con-rna-rf	**7528**	21513	16361

3.5.3 Support vector machine

This subsection demonstrates the performance of NysADMM on kernel SVM problem for binary classification, (3.11) from section 3.3.3. The SVM experiments compare NysADMM with the LIBSVM solver [21]. LIBSVM uses sequential minimal optimization (SMO) to solve the dual SVM problem. We use the same stopping criteria as the LIBSVM solver, which stops the NysADMM method when the ADMM dual gap reaches 10^{-4} level. All SVM experiments use the RBF kernel. Table 3.6 shows the results of SVM experiments. On these problems, NysADMM is at least 3 times faster (and up to 58 times faster) than the LIBSVM solver. Consider problem formulation (3.11), with the RBF kernel. The Gram matrix $\mathrm{diag}(b)K\mathrm{diag}(b)$ is dense and approximately low rank: exactly the setting in which NysADMM should be expected to perform well. In constrast, the SMO-type decomposition in LIBSVM solver works better for sparse

Table 3.6: Results of SVM experiment.

Task	NysADMM time (s)	LIBSVM time (s)
STL-10	**208**	11573
CIFAR-10	**1636**	8563
p53-rf	**291**	919
connect-4-rf	**7073**	42762
realsim-rf	**17045**	52397
rcv1-rf	**564**	32848
cod-rna-rf	**4942**	36791

problems, as it updates only two variables at each iteration.

3.6 Conclusion

In this chapter, we have developed a scalable new algorithm, NysADMM, that combines inexact ADMM and the randomized low-rank Nyström approximation to accelerate composite convex optimization. We show that NysADMM exhibits strong benefits both in theory and in practice. Our theory shows that when the Nyström preconditioner is constructed with an appropriate rank, NysADMM requires only a constant number of matvecs to solve the ADMM subproblem. We have also provided an adaptive strategy for selecting the rank that possesses a similar computational profile to the non-adaptive algorithm, and allows us to realize the theoretical benefits in practice. Further, numerical results demonstrate that NysADMM is as least twice as fast as standard methods on large dense lasso, regularized logistic regression, and kernalized SVM problems. More broadly, this chapter shows the promise of recent advances in RandNLA to provide practical accelerations for important large-scale optimization algorithms.

CHAPTER 4

ON THE (LINEAR) CONVERGENCE OF GENERALIZED NEWTON INEXACT ADMM

This chapter introduces GeNI-ADMM, a generalized framework extended from NysADMM for efficient theoretical analysis of approximate ADMM schemes. This chapter comes from [42] and is our most recent work. In section 4.2, we formally state the optimization problem that we focus on in this paper and briefly introduce ADMM. In section 4.3 we introduce the Generalized Newton Inexact ADMM framework and give a review of ADMM and its variants. Section 4.4 gives various techinical background and assumputions needed for our analysis. Section 4.5 establishes that GeNI-ADMM converges at an $O(1/t)$-rate in the convex setting. Section 4.6 then shows that, when the objective is strongly convex, GeNI-ADMM converges linearly. In Section 4.7, we apply our theory to establish convergence rates for two methods that naturally fall out of our framework, and we illustrate these results numerically in Section 4.8.

4.1 Introduction

Recall our discussion in chapters 1 and 3. ADMM provides a unified way to solve various convex machine learning problems. However, it suffers from scalability issue and it is hard to scale ADMM to large problem sizes. Randomized numerical linear algebra (randNLA) offers a promising set of tools to address this issue. Recently [112] proposed the algorithm NysADMM, which uses a randomized fast linear system solver to scale ADMM up to with tens of thousands of samples with hundreds of thousands of features. The results

in [112] show that ADMM combined with the randNLA primitive runs 3 to 15 times faster than state-of-the-art solvers on machine learning problems from LASSO to SVM to logistic regression. Unfortunately, the convergence of randomized or approximate ADMM solvers like NysADMM is not well understood. NysADMM approximates the x-subproblem using a linearization based on a second-order Taylor expansion, which transforms the x-subproblem into a Newton-step, *i.e.*, a linear system solve. It then solves this system approximately (and quickly) using a randomized linear system solver. The convergence of this scheme, which combines linearization and inexactness, is not covered by prior theory for approximate ADMM; prior theory covers either linearization [79] or inexact solves [35] but not both.

In this work, we bridge the gap between theory and practice to explain the excellent performance of approximate linearized ADMM schemes like NysADMM [112]. We introduce a framework called Generalized Newton Inexact ADMM, which we refer to as GeNI-ADMM (pronounced *genie-ADMM*). GeNI-ADMM includes NysADMM and many other prior approximate ADMM schemes as special cases. The name is inspired by the fact the linearized x-subproblem in GeNI-ADMM may be viewed as a generalized Newton-step. GeNI-ADMM allows for inexactness in both the x-subproblem and the z-subproblem. We show GeNI-ADMM exhibits the usual $O(1/t)$-convergence rate under standard hypotheses, with linear convergence under additional hypotheses. Our analysis also clarifies the value of using curvature in the generalized Newton step: approximate ADMM schemes that take advantage of curvature converge faster than those that do not at a rate that depends on the conditioning of the subproblem. As the GeNI-ADMM framework covers any approximate ADMM scheme that replaces the x-subproblem by a linear system solve,

our convergence theory covers any ADMM scheme that uses fast linear system solvers. Given the recent flurry of activity on fast linear system solvers within the (randomized) numerical linear algebra community [41, 57, 69], these results will help realize these benefits for optimization problems as well. To demonstrate the power of the GeNI-ADMM framework, we establish convergence of NysADMM and another randNLA-inspired scheme, sketch-and-solve ADMM, whose convergence was left as an open problem in [16].

4.1.1 Contributions

Our contributions maybe summarized concisely as follows:

1. We provide a general ADMM framework GeNI-ADMM, that encompasses prior approximate ADMM schemes as well as new ones. It can take advantage of second-order information and allows for inexact subproblem-solves.
2. We show that GeNI-ADMM converges at the usual $O(1/t)$-rate, despite all the approximations it makes. Further, it enjoys a linear convergence rate under additional hypotheses.
3. We apply our framework to show some randNLA-based approximate ADMM schemes converge at the same rate as vanilla ADMM, answering some open questions in the literature.

4.1.2 Notation and preliminaries

We call a matrix psd if it is positive semidefinite. We denote the convex cone of $n \times n$ real symmetric psd matrices by $\mathbf{S}_+^n$. We denote the Loewner ordering on $\mathbf{S}_+^n$ by $\preceq$, that is $A \preceq B$ if and only if $B - A$ is psd. Given a matrix H, we denote its spectral norm by $\|H\|$. If f is a smooth function we denote its smoothness constant by L_f. We say a positive sequence $\{\varepsilon^k\}_{k\geq 1}$ is summable if $\sum_{k=1}^{\infty} \varepsilon^k < \infty$.

4.2 Problem statement and ADMM

Let $\mathcal{X}$ and $\mathcal{Z}$ be finite-dimensional inner-product spaces with inner-product $\langle \cdot, \cdot \rangle$ and norm $\| \cdot \|$. We wish to solve the convex constrained optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) + g(z) \\ \text{subject to} & Mx = z \\ & x \in X,\ z \in Z, \end{array} \tag{4.1}$$

with variables x and z, where $X \subset \mathcal{X}$ and $Z \subset \mathcal{Z}$ are closed convex sets, f is a smooth convex function, g is a convex proper lower-semicontinuous (lsc) function, and $M : \mathcal{X} \mapsto \mathcal{Z}$ is a bounded linear operator. We can write problem (4.1) as the saddle point problem

$$\underset{y \in Y}{\text{maximize}}\ \underset{x \in X, z \in Z}{\text{minimize}}\ f(x) + g(z) + \langle y, Mx - z \rangle, \tag{4.2}$$

where $Y \subset \mathcal{Z}$ is a closed convex set. The saddle-point formulation will play an important role in our analysis. Perform the change of variables $u = y/\rho$ and define the Lagrangian

$$L_\rho(x, z, u) := f(x) + g(z) + \langle \rho u, Mx - z \rangle.$$

Then (4.2) may be written concisely as

$$\underset{u\in U}{\text{maximize}}\ \underset{x\in X, z\in Z}{\text{minimize}}\ L_\rho(x,z,u). \tag{4.3}$$

One important algorithm to solve (4.1) is the ADMM algorithm (algorithm 4.2.1). Our presentation uses the scaled form of ADMM, using the change of variables $u = y/\rho$, and we maintain this convention throughout this chapter. In fact, ADMM (algorithm 4.2.1) does not require f to be smooth, but just a convex proper lsc function, like g.

Algorithm 4.2.1 ADMM

input: convex proper lsc functions f and g, constraint matrix M, stepsize ρ

repeat

$x^{k+1} = \underset{x\in X}{\text{argmin}}\{f(x) + \frac{\rho}{2}\|Mx - z^k + u^k\|^2\}$

$z^{k+1} = \underset{z\in Z}{\text{argmin}}\{g(z) + \frac{\rho}{2}\|Mx^{k+1} - z + u^k\|^2\}$

$u^{k+1} = u^k + Mx^{k+1} - z^{k+1}$

until convergence

output: solution $(x^\star, z^\star)$ of problem (4.1)

4.3 Generalized Newton Inexact ADMM

As shown in algorithm 4.2.1, at each iteration of ADMM, two subproblems are solved sequentially to update variables x and z. ADMM is often the method of choice when the z-subproblem has a closed-form solution. For example, if $g(x) = \|x\|_1$, the z-subproblem is the soft thresholding, and if g is the indicator function of a convex set $\mathcal{S}$, the z-subproblem is projection onto set $\mathcal{S}$ [82, Chapter 6]. However, even when there is a closed-form solution it may be expensive to compute. For example, the solution of the z-subproblem in ADMM may require projecting onto a convex set lacking a computationally efficient projection oracle, like the psd cone.

Let us consider the x-subproblem

$$x^{k+1} = \operatorname*{argmin}_{x \in X} \left\{ f(x) + \frac{\rho}{2} \|Mx - z^k + u^k\|^2 \right\}. \tag{4.4}$$

In contrast to the z-subproblem, there is usually no closed-form solution for the x-subproblem. Instead, it is often solved inaccurately by an iterative scheme, especially for large-scale applications. This solve can be very expensive when the problem is large. To reduce computational effort, many authors have suggested to replace this problem with a simplified subproblem that is easier to solve. We highlight several strategies to do so below.

Augmented Lagrangian linearization. One strategy is to linearize the augmented Lagrangian term $\frac{\rho}{2}\|Mx - z^k + u^k\|^2$ in the ADMM subproblem and replace it by the quadratic penalty $\frac{1}{2}\|x - x^k\|_P^2$ for some (carefully chosen) positive definite matrix P. More formally, the strategy adds a quadratic term to form a new subproblem

$$x^{k+1} = \operatorname*{argmin}_{x \in X} \left\{ f(x) + \frac{\rho}{2} \|Mx - z^k + u^k\|^2 + \frac{1}{2}\|x - x^k\|_P^2 \right\},$$

which is substantially easier to solve for an appropriate choice of P. One canonical choice is $P = \eta I - \rho M^T M$, where $\eta > 0$ is a constant. For this choice, the quadratic terms involving M cancel, and we may omit constants with respect to x, resulting in the subproblem

$$x^{k+1} = \operatorname*{argmin}_{x \in X} \left\{ f(x) + \rho \langle Mx, Mx^k - z^k + u^k \rangle + \frac{\eta}{2}\|x - x^k\|^2 \right\}.$$

We see that the augmented Lagrangian term in (4.4) is replaced by an isotropic quadratic penalty. This strategy allows the subproblem solve to be replaced by a proximal operator with a (possibly) closed-form solution. This strategy has been variously called *preconditioned ADMM*, proximal ADMM, and (confusingly) linearized ADMM [27, 50, 79].

Function approximation. The second strategy to simplify the x-subproblem is to approximate the function f by a first- or second-order approximation [52, 79, 91, 112], forming the new subproblem

$$x^{k+1} = \underset{x \in X}{\operatorname{argmin}} \left\{ f(x^k) + \langle \nabla f(x^k), x - x^k \rangle + \frac{\eta}{2} \|x - x^k\|_H^2 + \frac{\rho}{2} \|Mx - z^k + u^k\|^2 \right\} \quad (4.5)$$

where H is the Hessian of f at x^k. The resulting subproblem is quadratic and may be solved by solving a linear system or (for $M = I$) performing a linear update, as detailed below in section 4.3.2.

Inexact subproblem solve. The third strategy is to solve the ADMM subproblems inexactly to achieve some target accuracy, either in absolute error or relative error. An absolute-error criterion chooses the subproblem error a priori [35], while a relative-error criterion requires the subproblem error to decrease as the algorithm nears convergence, for example, by setting the error target at each iteration proportional to $\|u^{k+1} - u^k - \rho(z^{k+1} - z^k)\|$ [38].

Approximations used by GeNI-ADMM. The GeNI-ADMM framework allows for any combination of the three strategies: augmented Lagrangian linearization, function approximation, and inexact subproblem solve. Consider the generalized second-order approximation to f

$$f(x) \approx f(\tilde{x}^k) + \langle x - \tilde{x}^k, \nabla f(\tilde{x}^k) \rangle + \frac{\eta^k}{2} \|x - \tilde{x}^k\|_{\Theta^k}^2, \quad (4.6)$$

where $\{\Theta^k\}_{k \geq 1}$ is a sequence of psd matrices that approximate the Hessian of f. GeNI-ADMM uses this approximation in the x-subproblem, resulting in the new subproblem

$$\begin{aligned} \tilde{x}^{k+1} = \underset{x \in X}{\operatorname{argmin}} \{ & f(\tilde{x}^k) + \langle x - \tilde{x}^k, \nabla f(\tilde{x}^k) \rangle + \frac{\eta^k}{2} \|x - \tilde{x}^k\|_{\Theta^k}^2 \\ & + \alpha\rho \langle Mx, M\tilde{x}^k - \tilde{z}^k + \tilde{u}^k \rangle + \frac{(1-\alpha)\rho}{2} \|Mx - \tilde{z}^k + \tilde{u}^k\|^2 \}, \end{aligned} \quad (4.7)$$

where $\alpha \in \{0, 1\}$ controls whether the augmented Lagrangian term $\frac{\rho}{2}\|Mx - \tilde{z}^k + \tilde{u}^k\|^2$ is linearized.

We refer to (4.7) as a generalized Newton step. The intuition for the name is made plain when $X = \mathbb{R}^d$, in which case the update becomes

$$\left(\eta^k \Theta^k + (1-\alpha)\rho M^T M\right) \tilde{x}^{k+1} = \eta^k \Theta^k \tilde{x}^k - \nabla f(\tilde{x}^k) - \rho M^T(\alpha M \tilde{x}^k - \tilde{z}^k + \tilde{u}^k). \tag{4.8}$$

Equation (4.8) shows that the x-subproblem reduces to a linear system solve, just like the Newton update. We can also interpret GeNI-ADMM as a linearized proximal augmented Langrangian (P-ALM) method [51, 52]. From this point of view, GeNI-ADMM replaces f in the P-ALM step by its linearization and adds a specialized penalty defined by the Θ^k-norm.

The z-subproblem in GeNI-ADMM remains unchanged, however we allow it to be solved inexactly to account for when exact solves are prohibitively expensive. Similarly, it is unreasonable to assume that (4.7) is solved exactly at each iteration. Indeed, the hallmark of the NysADMM scheme from [112] is that it solves (4.7) inexactly in a highly efficient manner by using a randomized linear system solver. Thus we allow for inexactness in both the x and z-subproblems in our generalized ADMM template, which we present in algorithm 4.3.1.

Algorithm 4.3.1 Generalized Newton Inexact ADMM (GeNI-ADMM)

input: stepsize ρ, $\alpha \in \{0, 1\}$, sequence of psd matrices $\{\Theta^k\}_{k\geq 1}$, step-size sequence $\{\eta^k\}_{k\geq 1}$, forcing sequences $\{\varepsilon_x^k\}_{k\geq 1}, \{\varepsilon_z^k\}_{k\geq 1}$,

repeat

$$\tilde{x}^{k+1} \overset{\varepsilon_x^k}{\approx} \operatorname*{argmin}_{x\in X}\{f(\tilde{x}^k) + \langle x - \tilde{x}^k, \nabla f(\tilde{x}^k)\rangle + \tfrac{\eta^k}{2}\|x - \tilde{x}^k\|^2_{\Theta^k} + \alpha\rho\langle Mx, M\tilde{x}^k - \tilde{z}^k + \tilde{u}^k\rangle + \tfrac{(1-\alpha)\rho}{2}\|Mx - \tilde{z}^k + \tilde{u}^k\|^2\}$$

$$\tilde{z}^{k+1} \overset{\varepsilon_z^k}{\approx} \operatorname*{argmin}_{z\in Z}\{g(z) + \tfrac{\rho}{2}\|M\tilde{x}^{k+1} - z + \tilde{u}^k\|^2\}$$

$$\tilde{u}^{k+1} = \tilde{u}^k + M\tilde{x}^{k+1} - \tilde{z}^{k+1}$$

until convergence

output: solution $(x^\star, z^\star)$ of problem (4.1)

The primary features that differentiates Algorithm 4.3.1 from ADMM (algorithm 4.2.1), is that the x-subproblem is now a generalized-Newton step, and both subproblems may be solved inexactly. Given the inexactness and the use of the generalized Newton step in place of the original x-subproblem, we refer to algorithm 4.3.1 as Generalized Newton Inexact ADMM (GeNI-ADMM). The inexactness schedule is controlled by the *forcing sequences* $\{\varepsilon_x^k\}_{k\geq 1}, \{\varepsilon_z^k\}_{k\geq 1}$, which specify how accurately the x and z-subproblems are solved at each iteration. We note the inexactness used for the x and z-subproblems are not the same; to easily distinguish between them we make the following definition.

Definition 4.3.1 (ε-minimizer and ε-minimum)**.** Let $h : \mathcal{T} \mapsto \mathbb{R}$ be strongly-convex and let $t^\star = \operatorname{argmin}_{t'\in\mathcal{T}} h(t')$.

- (ε-minimizer) Given $t \in \mathcal{T}$, we say t is an *ε-minimizer* of $\operatorname{minimize}_{t\in\mathcal{T}} h(t)$ and write

$$t \overset{\varepsilon}{\approx} \operatorname*{argmin}_{t'\in\mathcal{T}} h(t') \quad \text{if and only if} \quad \|t - t^\star\| \leq \varepsilon.$$

  In words, t is nearly equal to the argmin of $h(t)$ in set $\mathcal{T}$.

- (ε-minimum) Given $t \in \mathcal{T}$, we say t gives an *ε-minimum* of $\operatorname{minimize}_{t\in\mathcal{T}} h(t)$ and write

$$t \overset{\varepsilon}{\cong} \operatorname*{argmin}_{t'\in\mathcal{T}} h(t') \quad \text{if and only if} \quad h(t) - h(t^\star) \leq \varepsilon.$$

  In words, t produces nearly the same objective value as the argmin of $h(t)$ in set $\mathcal{T}$.

Thus, from definition 4.3.1 and algorithm 4.3.1, we see for each iteration k that $\tilde{x}^{k+1}$ is an ε_x^k-minimizer of the x-subproblem, while $\tilde{z}^{k+1}$ gives an ε_z^k-minimum of the z-subproblem.

4.3.1 Related work

The literature on the convergence of ADMM and its variants is vast, so we focus on prior work most relevant to our setting and provide a review in table 4.1. GeNI-ADMM is distinguished from all prior works in table 4.1 that it allows (almost) all these approximations and more, and provides explicit rates of convergence to support choices between algorithms.

The "x" ("z") in table 4.1 denotes that a paper uses the corresponding strategy of the column to simplify the x-subproblem (z-subproblem). "**AL**" is the abbreviation of augmented Lagrangian. In the "**Function approximation**" column, the "f" ("g") indicates that a paper approximates function "f" ("g") in the x-subproblem (z-subproblem). "Stochastic gradient" means a paper uses first-order function approximation but replace the gradient term with a stochastic gradient. "Generalized second-order" means a paper uses second-order function approximation as (4.5), but H is not necessarily the Hessian.

4.3.2 Algorithms recovered from GeNI-ADMM

Various ADMM schemes in the literature can be recovered by selecting the parameters in algorithm 4.3.1 appropriately. Here we mention a few of the most important special cases to show the breadth of GeNI-ADMM and so the reader may develop concrete intuition for the general framework provided by algorithm 4.3.1.

Table 4.1: A structured comparison of related work on the convergence of ADMM and its variants.

References	Convergence rate	Problem	AL linearization	Function approximation	Subproblem inexactness	Year	Remarks
Eckstein et al. [35]	-	Convex	-	-	x, z absolute-error	1992	
He and Yuan [50]	Ergodic $O(1/t)$	Convex	x	-	-	2012	By variational inequalities
Monteiro and Svaiter [73]	Ergodic $O(1/t)$	Convex	-	-	-	2013	By hybrid proximal extragradient method
Deng and Yin [27]	Linear	Convex	x, z	-	-	2016	Assuming strong convexity
Eckstein and Yao [38]	-	Convex	-	-	x relative-error	2018	
Yuan et al. [110]	Locally linear	Convex	x, z	-	-	2020	Without strong convexity, structured f
Ouyang et al. [79]	Ergodic $O(1/t)$	Convex	x	f first-order	-	2015	
Ryu and Yin [91]	-	Convex	x, z	f, g first-order	-	2022	
Ouyang et al. [78]	$O(1/\sqrt{t})$	Stochastic, convex	-	f stochastic gradient	-	2013	
Hermans et al. [51]	-	Quadratic, convex	-	f generalized second-order	x absolute-error	2019	PALM type
Hermans et al. [52]	Linear	Quadratic, nonconvex	-	f generalized second-order	x absolute-error	2022	PALM type
This work	Ergodic $O(1/t)$, linear	Convex	x	f generalized second-order	x, z, absolute-error	-	Linear convergence under strong convexity

NysADMM We start with the recent NysADMM scheme from [112]. The original NysADMM scheme assumes the unconstrained case $M = I$. We shall maintain this for simplicity of presentation, even though it is not strictly necessary. Having noted this, NysADMM is recovered by taking $\alpha = 0$ and $\Theta^k = H^k$, the Hessian of f at the kth iteration. Substituting this information into (4.7) leads to the following update for the x-subproblem.

$$\left(\eta^k H_f^k + \rho I\right) \tilde{x}^{k+1} = \eta^k H_f^k \tilde{x}^k - \nabla f(\tilde{x}^k) + \rho(\tilde{z}^k - \tilde{u}^k). \tag{4.9}$$

Gradient descent ADMM We next consider the two linearization schemes from [79]. The first scheme we call *gradient descent ADMM* (GD-ADMM) and is appropriate to use when it is inexpensive to solve a least-squares problem with M. GD-ADMM is obtained from GeNI-ADMM by setting $\alpha = 0$ and $\Theta^k = I$ for all k. The $\tilde{x}^{k+1}$ update (4.8) for GD-ADMM simplifies to

$$\tilde{x}^{k+1} = \tilde{x}^k - (\rho M^T M + \eta^k I)^{-1} \left(\nabla f(\tilde{x}^k) + \rho M^T (M\tilde{x}^k - \tilde{z}^k + \tilde{u}^k)\right). \tag{4.10}$$

The second scheme, *linearized-gradient descent ADMM* (LGD-ADMM), is useful when M is not simple, so that the update (4.10) no longer inexpensive. It linearizes the augmented Lagrangian term in the x-subproblem by setting $\alpha = 1$ in (4.8) yields the $\tilde{x}^{k+1}$ update

$$\tilde{x}^{k+1} = \tilde{x}^k - \frac{1}{\eta^k} \left(\nabla f(\tilde{x}^k) + \rho M^T (M\tilde{x}^k - \tilde{z}^k + \tilde{u}^k)\right). \tag{4.11}$$

Observe in the unconstrained case, when $M = I$, the updates (4.10) and (4.11) are equivalent and thus they generate the same iterate sequences when initialized at the same point [113]. Indeed, they are both generated by performing a gradient step on the augmented Lagrangian (4.7), for suitable choices of the parameters. Notably, this terminology differs from [79], who refer to (4.10) as "linearized

ADMM" (L-ADMM) and (4.11) as "linearized preconditioned ADMM" (LP-ADMM). We choose our terminology to emphasize that GD-ADMM accesses f via its gradient, as in the literature the term "linearized ADMM" is usually reserved for methods that access f through its prox operator [27, 50, 82].

Sketch-and-solve ADMM If $M = I$, and we select Θ^k to be matrix such that $\Theta^k + \rho I$ is cheap to factor, we call the resulting scheme *sketch-and-solve ADMM*. The name *sketch-and-solve ADMM* is motivated by the fact that such Θ^k can often be obtained via sketching techniques. However, the method works for any Θ^k, not only ones constructed by sketching methods. We provide the full details of sketch-and-solve ADMM in section 4.7.2. To our knowledge sketch-and-solve ADMM has not been formally proposed or previously analyzed.

The main goal of the rest of this chapter is to prove convergence of algorithm 4.3.1 under appropriate hypotheses on the sequences $\{\Theta^k\}_{k\geq 1}, \{\varepsilon_x^k\}_{k\geq 1}, \{\varepsilon_z^k\}_{k\geq 1}$.

4.4 Technical preliminaries and assumptions

We start by introducing some important concepts that will play a central role in our analysis. The first is Θ-*relative smoothness*, which is crucial to establish that GeNI-ADMM benefits from curvature information provided by the Hessian.

Definition 4.4.1 (Θ-relative smoothness)**.** We say $f : \mathcal{D} \to \mathbf{R}$ is Θ-relatively smooth with respect to the bounded function $\Theta : \mathcal{D} \to \mathbf{S}_+^n$ if there exists $\hat{L}_\Theta > 0$ such that for all $x, y \in \mathcal{D}$

$$f(x) \leq f(y) + \langle \nabla f(y), x - y \rangle + \frac{\hat{L}_\Theta}{2} \|x - y\|_{\Theta(y)}^2. \tag{4.12}$$

That is, the function f is smooth with respect to the Θ-norm. Definition 4.4.1 generalizes *relative smoothness*, introduced in [47] to analyze Newton's method. The definition in [47] takes Θ to be the Hessian of f, H_f. When f belongs to the popular family of generalized linear models, then (4.12) holds with a value of $\hat{L}_{H_f}$ that is independent of the conditioning of the problem [47]. For instance, if f is quadratic and $\Theta(y) = H_f$, then (4.12) holds with equality for $\hat{L}_{H_f} = 1$. Conversely, if we take $\Theta = I$, which corresponds to GD-ADMM, then $\hat{L}_\Theta = L_f$. Here, L_f denotes the smoothness constant of f. Our theory will rely on the fact that $\hat{L}_\Theta$ is much smaller than the smoothness constant L_f of f for methods that take advantage of curvature, and will rely on $\hat{L}_\Theta$ to characterize the faster convergence speed of these methods.

The other important idea we shall need is the notion of an *ε-subgradient* [10, 53].

Definition 4.4.2 (ε-subgradient)**.** Let $r : \mathcal{D} \to \mathbf{R}$ be a convex function and $\varepsilon > 0$. We say that $s \in \mathcal{D}^*$ is an ε-subgradient for r at $z \in \mathcal{D}$ if, for every $z' \in \mathcal{D}$, we have

$$r(z') - r(z) \geq \langle s, z' - z \rangle - \varepsilon.$$

We denote the set of ε-subgradients for r at z by $\partial_\varepsilon r(z)$.

Clearly any subgradient is an ε-subgradient, so definition 4.4.2 provides a natural weakening of a subgradient. The ε-subgradient is critical for analyzing z-subproblem inexactness, and our usage in this context is inspired by the convergence analysis of inexact proximal gradient methods [93]. We shall need the following proposition whose proof may be found in [10, Proposition 4.3.1].

Proposition 4.4.1. Let r, r_1, and r_2 be convex functions. Then for any z, the following holds:

1. $0 \in \partial_\varepsilon r(z)$ if and only if $r(z) \overset{\varepsilon}{\cong} \operatorname{argmin}_{z'} r(z')$, that is z gives an ε-minimum of $\operatorname{minimize}_{z'} r(z')$.
2. $\partial_\varepsilon(r_1 + r_2)(z) \subset \partial_\varepsilon r_1(z) + \partial_\varepsilon r_2(z)$.

With proposition 4.4.1 established, we prove the following lemma in appendix C.2, which will play a critical role in establishing the convergence of GeNI-ADMM.

Lemma 4.4.1. Let $\tilde{z}^{k+1}$ give an ε_z^k-minimum of the z-subproblem. Then there exists a $\|\tilde{s}\| \leq \sqrt{\frac{2\varepsilon_z^k}{\rho}}$ such that

$$\rho(M\tilde{x}^{k+1} - \tilde{z}^{k+1} + \tilde{u}^k - \tilde{s}) \in \partial_{\varepsilon_z^k} g(\tilde{z}^{k+1}).$$

4.4.1 Assumptions

In this section we present the main assumptions required by our analysis.

Assumption 4.4.1 (Existence of saddle point)**.** There exists an optimal primal solution $(x^\star, z^\star) \in X \times Z$ for (4.1) and an optimal dual solution $u^\star \in U$ such that $(x^\star, z^\star, u^\star)$ is a saddle point of (4.2). Here $U \subset \mathcal{Z}$ is a closed convex set and $\rho U = Y$. We denote the optimal objective value of (4.1) as $p^\star$.

Assumption 4.4.2. The sequence $\{\Theta^k\}_{k\geq 1}$ satisfies

$$(1 - \zeta^{k-1})\Theta^{k-1} \leq \Theta^k \leq (1 + \zeta^{k-1})\Theta^{k-1}, \ \ \forall k \geq 2, \tag{4.13}$$

where $\{\zeta^k\}_{k\geq 1}$ is a summable sequence, that is, $\sum_{k=1}^\infty \zeta^k < \infty$.

Intuitively, this assumption requires that the Θ^k's does not change too much between iterations. Note this assumption may always be enforced by changing Θ^k only finitely many times.

We also define the following constants which shall be useful in our analysis,

$$\tau_\zeta := \prod_{k\geq 2}(1+\zeta^k),$$

$$\mathcal{E}_\zeta = \tau_\zeta\left(\sum_{k=0}^{\infty}\zeta^k\right) < \infty.$$

Note assumption 4.4.2 implies $\tau_\zeta < \infty$ and for any $v \in \mathbb{R}^p$ that $\|v\|_{\Theta_k} \leq \tau_\zeta \|v\|_{\Theta^1}$ for all $k \geq 1$.

Assumption 4.4.3 (Summability of the forcing sequences)**.** The forcing sequences $\{\varepsilon_x^k\}_{k\geq 1}$ and $\{\varepsilon_z^k\}_{k\geq 1}$ satisfy

$$\mathcal{E}_x = \sum_{k=1}^{\infty}\varepsilon_x^k < \infty, \quad \mathcal{E}_z = \sum_{k=1}^{\infty}\sqrt{\varepsilon_z^k} < \infty.$$

Further we define the constants $K_{\varepsilon_x} := \sup_{k\geq 1}\varepsilon_x^k$, $K_{\varepsilon_z} := \sup_{k\geq 1}\sqrt{\varepsilon_z^k}$. Observe K_{ε_x} and K_{ε_z} are finite owing to the summability hypotheses.

Assumption 4.4.3 seems to require a more accurate solution of the z-subproblem than the x-subproblem. While it is true that the sequence $\{\varepsilon_z^k\}_{k\geq 1}$ must decay faster than $\{\varepsilon_x^k\}_{k\geq 1}$, the z-subproblem needs only a ε_z^k-minimum, which is much weaker than the ε_x^k-minimizer required for the x-subproblem.

Assumption 4.4.4 (x-Subproblem oracle)**.** We assume algorithm 4.3.1 is equipped with an oracle for solving the x-subproblem that each iteration produces an approximate solution $\tilde{x}^{k+1}$ satisfying:

$$\max\{\|\tilde{x}^{k+1} - x^{k+1}\|_{\eta^k\Theta^k+\rho M^T M}, \|\tilde{x}^{k+1} - x^{k+1}\|\} \leq \varepsilon_x^k,$$

where x^{k+1} denotes the exact solution of the x-subproblem at iteration $k+1$.

Assumption 4.4.4 is motivated by the fact in practical applications where $X = \mathbb{R}^d$, the x-subproblem reduces to (4.7), which is a linear system solve. If

we have a fast linear system oracle like in NysADMM, then the condition of assumption 4.4.4 is always met. In addition, if we compute the solution to (4.7) exactly, as in GD-ADMM and LGD-ADMM, then the condition is trivially satisfied.

Assumption 4.4.5 (Regularity of f and g)**.** The function f is β_f-Lipschitz twice-continuously differentiable and is relatively smooth with respect to Θ. The function g is finite-valued, convex, and lower semi-continuous.

Assumption 4.4.6 (Boundedness of the iterates)**.** We make the following boundedness hypotheses on the iterates $\{\tilde{x}^k\}_{k\geq 1}$, $\{\tilde{z}^k\}_{k\geq 1}$, $\{\tilde{u}^k\}_{k\geq 1}$:

$$R_{x^\star,\Theta^1} := \sup_{k\geq 1} \|\tilde{x}^k - x^\star\|_{\Theta^1} < \infty, \quad R_{x^\star,M} := \sup_{k\geq 1} \|\tilde{x}^k - x^\star\|_{M^T M} < \infty$$

$$\sup_{k\geq 1} \|\tilde{x}^k - x^\star\|, \ \sup_{k\geq 1} \|\tilde{u}^k - u^\star\|, \ \sup_{k\geq 1} \|\tilde{z}^k - z^\star\| \leq R_{x^\star,u^\star,z^\star} < \infty.$$

Assumptions 4.4.1, 4.4.5, and 4.4.6 are all standard. Assumption 4.4.5 ensures that it makes sense to talk about the Hessian of f and that f is relatively smooth.

4.5 Sublinear convergence of GeNI-ADMM

This section establishes our main theorem, theorem 4.5.1, which shows that algorithm 4.3.1 enjoys the same $O(1/t)$-convergence rate as standard ADMM.

Theorem 4.5.1 (Ergodic convergence)**.** Define constants $R_\star = \max\{R_{x^\star,\Theta^1}, R_{x^\star,M}, R_{x^\star,u^\star,z^\star}, \|u^\star\|\}$, $d_{x^\star,\Theta^1} = \|\tilde{x}^1 - x^\star\|_{\Theta^1}$, $d_{u^\star} = \|\tilde{u}^1 - u^\star\|$, $d_{z^\star} = \|\tilde{z}^1 - z^\star\|$. Let $p^\star$ denote the optimum of (4.1) and $\eta^k = \eta = \hat{L}_\Theta + \frac{\alpha\rho\|M\|^2}{\theta}$ such that $\Theta^k \succeq \theta I$ for all k. For each $t \geq 1$, denote $\bar{x}^{t+1} = \frac{1}{t}\sum_{k=2}^{t+1} \tilde{x}^k$, and $\bar{z}^{t+1} = \frac{1}{t}\sum_{k=2}^{t+1} \tilde{z}^k$, where $\{\tilde{x}^k\}_{k\geq 1}$ and $\{\tilde{z}^k\}_{k\geq 1}$ are the iterates produced by algorithm 4.3.1 with forcing sequences $\{\varepsilon_x^k\}_{k\geq 1}$ and $\{\varepsilon_z^k\}_{k\geq 1}$ with

$\tilde{u}^1 = 0$ and $\tilde{z}^1 = M\tilde{x}^1$. Instate Assumptions 4.4.1-4.4.6. Then the suboptimality gap satisfies

$$f(\bar{x}^{t+1}) + g(\bar{z}^{t+1}) - p^\star \leq \frac{1}{t}\left(\frac{\eta}{2}d^2_{x^\star,\Theta^1} + \frac{\rho(1-\alpha)}{2}d^2_{z^\star} + C_x\mathcal{E}_x + C_z\mathcal{E}_z + \frac{\eta}{2}\mathcal{E}_\zeta R^2_\star\right) =: \frac{1}{t}\Gamma,$$

where C_x and C_z are constants that depend linearly upon $\beta_f, K_{\varepsilon_x}, K_{\varepsilon_z}$, and $R_\star$. Furthermore, the feasiblity gap satisfies

$$\|M\bar{x}^{t+1} - \bar{z}^{t+1}\| \leq \frac{2}{t}\sqrt{\frac{\Gamma}{\rho} + d^2_{u^\star}},$$

Consequently, after $O(1/\epsilon)$ iterations,

$$f(\bar{x}^{t+1}) + g(\bar{z}^{t+1}) - p^\star \leq \epsilon, \text{ and } \|M\bar{x}^{t+1} - \bar{z}^{t+1}\| \leq \epsilon.$$

Theorem 4.5.1 shows with a constant value of η and appropriate forcing sequences, the suboptimality gap and the feasibility residuals both go to zero at a rate of $O(1/t)$. Hence the overall convergence rate of ADMM is preserved despite all the approximations involved in GeNI-ADMM. However, some schemes yield better constants than others.

To see this difference, consider the case when f is a convex quadratic function and $M = I$. Let H_f be the Hessian of f. Consider two special cases with $\alpha = 0$: (1) NysADMM $\Theta^k = H_f$ (4.9) and (2) GD-ADMM $\Theta^k = I$ (4.10). Further, suppose that NysADMM and GD-ADMM are initialized at 0. The rates of convergence guaranteed by theorem 4.5.1 for algorithm 4.3.1 for both methods are outlined in table 4.2. In the first case, the relative smoothness constant satisfies $\hat{L}_\Theta = 1$, while in the second, $\hat{L}_\Theta = \lambda_1\left(H_f\right)$, which is the largest eigenvalue of H_f. Hence by Cauchy-Schwarz, we always have

$$\lambda_1\left(H_f\right)\|x^\star\|^2 \geq \|x^\star\|^2_{H_f}.$$

Thus, NysADMM always improves over GD-ADMM, and improves significantly when H_f exhibits a decaying spectrum, provided $x^\star$ is not concentrated

Table 4.2: Convergence rate comparison of GeNI-ADMM with $\alpha = 0$ when initialized at 0 for quadratic f.

Method	NysADMM $\left(\Theta^k = H_f\right)$	GD-ADMM $\left(\Theta^k = I\right)$
Feasibility gap	$O\left(\frac{1}{t}\left(\sqrt{\frac{2}{\rho}\|x^\star\|^2_{H_f}}\right)\right)$	$O\left(\frac{1}{t}\left(\sqrt{\frac{2}{\rho}\lambda_1\left(H_f\right)\|x^\star\|^2}\right)\right)$
Suboptimality gap	$O\left(\frac{1}{2t}\|x^\star\|^2_{H_f}\right)$	$O\left(\frac{1}{2t}\lambda_1\left(H_f\right)\|x^\star\|^2\right)$

on the top eigenvector of H_f. We formalize the latter property in the following definition.

Definition 4.5.1 ($\mu_{\mathbf{v}}$-incoherence)**.** Let $(\lambda_1(H_f), \mathbf{v})$ be the largest eigenpair of H_f. We say $x^\star$ is $\mu_{\mathbf{v}}$-incoherent if there exists $0 \leq \mu_{\mathbf{v}} < 1$ such that:

$$|\langle \mathbf{v}, x^\star \rangle|^2 \leq \mu_{\mathbf{v}} \|x^\star\|^2. \tag{4.14}$$

Definition 4.5.1 is a weak form of the incoherence condition from compressed sensing and matrix completion, which plays a key role in signal and low-rank matrix recovery [17, 18]. In words, $x^\star$ is $\mu_{\mathbf{v}}$-incoherent if its energy is not solely concentrated on the top eigenvector of H_f and can be expected to hold generically. The parameter $\mu_{\mathbf{v}}$ controls the allowable concentration. When $\mu_{\mathbf{v}} = 0$, $x^\star$ is orthogonal to $\mathbf{v}$, so its energy is distributed amongst the other eigenvectors. Conversely, the closer $\mu_{\mathbf{v}}$ is to 1, the more concentrated $x^\star$ is on $\mathbf{v}$.

Using $\mu_{\mathbf{v}}$-incoherence, we can say more about how NysADMM improves on GD-ADMM.

Proposition 4.5.1. Suppose $x^\star$ is μ_V-incoherent. Then the following bound holds

$$\frac{\lambda_1(H_f)\|x^\star\|^2}{\|x^\star\|^2_{H_f}} \geq \max\left\{\frac{1}{\frac{\lambda_2(H_f)}{\lambda_1(H_f)} + \mu_{\mathbf{v}}}, 1\right\}.$$

Hence if $\lambda_1(H_f) \gg 1$ and $\mu_{\mathbf{v}} \leq C^{-1}/2$ and $\lambda_1(H_f)/\lambda_2(H_f) \leq C^{-1}/2$, where $C \geq 1$,

then

$$\frac{\lambda_1(H_f)\|x^\star\|^2}{\|x^\star\|_{H_f}^2} \geq C.$$

Proposition 4.5.1 shows when $x^\star$ is $\mu_{\mathbf{v}}$-incoherent and H_f has decaying spectrum, NysADMM yields a significant improvement over GD-ADMM. As a concrete example, consider when $C = 2$, i.e. no more than 25% of the energy in $x^\star$ concentrates on $\mathbf{v}$ and $\lambda_2(H_f) \leq 0.25\lambda_1(H_f)$, then proposition 4.5.1 implies the ergodic convergence of NysADMM is about twice as fast as GD-ADMM. In fact, we observe this performance improvement in practice (see section 3.5 for corroborating numerical evidence). We see that just as Newton's method improves on gradient descent for ill-conditioned problems, NysADMM is less sensitive to ill-conditioning than GD-ADMM.

We now move to proving theorem 4.5.1.

4.5.1 Our approach

To prove theorem 4.5.1, we take the gap function approach proposed in [79]. Let $W = X \times Z \times U$, $\hat{w} = (\hat{x}, \hat{z}, \hat{u})$, and $w = (x, z, u)$, where $w, \hat{w} \in W$. Define the *gap function*

$$Q(\hat{w}, w) \coloneqq [f(x) + g(z) + \langle \rho\hat{u}, Mx - z\rangle] - [f(\hat{x}) + g(\hat{z}) + \langle \rho u, M\hat{x} - \hat{z}\rangle]. \tag{4.15}$$

The gap function is motivated by the saddle-point formulation eq. (4.2) of eq. (4.1), and by construction, it is concave in its first argument and convex in the second. Observe the important inequality

$$Q(w^\star, w) = L_\rho(x, z, u^\star) - L_\rho(x^\star, z^\star, u) \geq 0,$$

where the inequality follows by definition of $(x^\star, z^\star, u^\star)$ being a saddle-point. Hence the gap function may be viewed as measuring the distance to the saddle $w^\star$.

Further, given a closed set $U \subset \mathcal{Z}$ and $v \in \mathcal{Z}$ we define

$$\begin{aligned}\ell_U(v, w) &:= \sup_{\hat{u}\in U}\{Q(\hat{w}, w) + \langle v, \rho\hat{u}\rangle \mid \hat{w} = (\hat{x}, \hat{z}, \hat{u}),\ \hat{x} = x^\star,\ \hat{z} = z^\star\} \\ &= f(x) + g(z) - p^\star + \sup_{\hat{u}\in U}\langle \rho\hat{u}, v - (Mx - z)\rangle.\end{aligned} \tag{4.16}$$

The appropriateness of the name gap function becomes clear when we consider the following result from [79]. As the proof is elementary, we provide it in appendix C.2 for the reader's leisure.

Lemma 4.5.1. For any $U \subseteq \mathcal{Z}$, suppose $\ell_U(Mx - z, w) \le \epsilon < \infty$ and $\|Mx - z\| \le \delta$, where $w = (x, z, u) \in W$. Then

$$f(x) + g(z) - p^\star \le \epsilon. \tag{4.17}$$

In other words, (x, z) is an approximate solution of (4.1) with suboptimality gap ϵ and feasibility gap δ. Further, if $U = \mathcal{Z}$, for any v such that $\ell_U(v, w) \le \epsilon < \infty$ and $\|v\| \le \delta$, we have $v = Mx - z$.

Lemma 4.5.1 shows that if we can find w such that $\ell_U(Mx - z, w) \le \epsilon$ and $\|Mx - z\| \le \delta$, then we have an approximate optimal solution to (4.1) with gaps ϵ and δ, that is, $\ell_U(Mx - z, w)$ controls the suboptimality and feasibility gaps.

Proof sketch for theorem 4.5.1 Lemma 4.5.1 provides a natural path for proving theorem 4.5.1. If we can find points $\bar{w} = (\bar{x}, \bar{z}, \bar{u}) \in W$, $\bar{v} \in \mathcal{Z}$ such that

$$\ell_U(\bar{v}, \bar{w}) \le O(1/t),$$

then lemma 4.5.1 ensures the suboptimality and feasibility gaps are $O(1/t)$, and so theorem 4.5.1 is proved. In order to see how we may find such a $\bar{w}$, let us suppose that algorithm 4.3.1 has been run for $t+1$ iterations, and the following bound has been established

$$\sum_{k=2}^{t+1} Q(x^\star, z^\star, u; \tilde{w}^k) \leq O(1) + \frac{\rho}{2}\left(\|\tilde{u}^t - u\|^2 - \|\tilde{u}^{t+1} - u\|^2\right). \tag{4.18}$$

We shall now sketch the construction of $\bar{w}$, and hence the proof of theorem 4.5.1, under the preceding hypotheses.

We start by setting $\bar{w} = \frac{1}{t}\sum_{k=2}^{t+1} \tilde{w}^k$. Then by convexity of the gap function (in its second argument), $\tilde{u}^1 = 0$, and (4.18), we reach

$$Q(x^\star, z^\star, u; \bar{w}) \leq O(1/t) + \frac{\rho}{2t}\left(\|\tilde{u}^1 - u\|^2 - \|\tilde{u}^{t+1} - u\|^2\right) \leq O(1/t) - \frac{\rho}{t}\langle \tilde{u}^1 - \tilde{u}^{t+1}, u\rangle.$$

Next we set $\tilde{v}^{t+1} = \frac{1}{t}(\tilde{u}^1 - \tilde{u}^{t+1})$ and observe that

$$\tilde{v}^{t+1} = \frac{-1}{t}\sum_{k=1}^{t}\left(\tilde{u}^{k+1} - \tilde{u}^k\right) = \frac{-1}{t}\sum_{k=1}^{k}\left(M\tilde{x}^{k+1} - \tilde{z}^{k+1}\right) = \bar{z} - M\bar{x}.$$

Now setting $u = u^\star$ into our bound on $Q(x^\star, z^\star, u; \bar{w})$, using $Q(w^\star, \bar{w}) \geq 0$, and performing some algebra, we find

$$\|\tilde{v}^{t+1}\|^2 \leq 2\left(\|\tilde{u}^{t+1} - u^\star\|^2 + \|\tilde{u}^1 - u^\star\|^2\right) \leq O(1/t^2).$$

Hence the infeasibility is bounded as $\|M\bar{x} - \bar{z}\| \leq O(1/t)$.

Returning to our bound on $Q(x^\star, z^\star, u; \bar{w})$, we see for any w that the following inequality holds

$$Q(w, \bar{w}) + \langle \tilde{v}^{t+1}, \rho u\rangle \leq O(1/t).$$

Hence setting $w = (x^\star, z^\star, u)$, we conclude

$$\ell_U(\tilde{v}^{t+1}, \bar{w}) \leq O(1/t),$$

which, by lemma 4.5.1, yields the theorem.

4.5.2 Controlling the gap function

The proof sketch shows, the key step in establishing convergence of GeNI-ADMM is the inequality (4.18). It is clear that in order to establish (4.18), we must achieve appropriate control over the gap function. To accomplish this, we make use of the optimality conditions of the x and z-subproblems. However, as the subproblems are only solved approximately, the inexact solutions satisfy *perturbed* optimality conditions. The precise form of these perturbed optimality conditions provide the content of lemmas 4.5.2 and 4.5.3. Detailed proofs establishing these lemmas are given in appendix C.2.

Lemma 4.5.2 (Inexact x-optimality condition)**.** Suppose $\tilde{x}^{k+1}$ is an ε_x^k-minimizer of the x-subproblem under assumption 4.4.4 and define the constant

$$\chi_1 := \left(2\tau_\zeta R_{x^\star,\Theta^1} + \sqrt{\rho}(3R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \beta_f + 2\alpha\rho R_{x^\star,M}\right).$$

Then for any $x \in X$, we have

$$\begin{aligned}\langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - x\rangle \leq \langle \eta^k \Theta^k(\tilde{x}^{k+1} - \tilde{x}^k), x - \tilde{x}^{k+1}\rangle + \rho\langle M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - \tilde{x}^{k+1})\rangle \\ + \alpha\rho\langle M(\tilde{x}^k - \tilde{x}^{k+1}), M(x - \tilde{x}^{k+1})\rangle \\ + \left(\chi_1 + \tau_\zeta\|x - \tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha)\|x - \tilde{x}^{k+1}\|_{\rho M^T M}\right)\varepsilon_x^k + \alpha(\varepsilon_x^k)^2.\end{aligned} \tag{4.19}$$

Lemma 4.5.3 (Inexact z-optimality condition)**.** Suppose $\tilde{z}^{k+1}$ is an ε_z^k-minimum of the z-subproblem under definition 4.3.1. Then for any $z \in Z$, we have

$$g(\tilde{z}^{k+1}) - g(z) \leq \rho\langle \tilde{u}^{k+1}, \tilde{z}^{k+1} - z\rangle + (1+2\rho)\varepsilon_z^k + \left(2(R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k}. \tag{4.20}$$

We note while lemma 4.5.2 (lemma 4.5.3) necessarily holds when $\tilde{x}^{k+1}$ ($\tilde{z}^{k+1}$) is an ε_x^k-approximate minimizer (ε_z^k-minimum), the converse does not hold. With

lemmas 4.5.2 and 4.5.3 in hand, we can establish control of the gap function for one iteration.

Lemma 4.5.4. Let $\tilde{w}^{k+1} = (\tilde{x}^{k+1}, \tilde{z}^{k+1}, \tilde{u}^{k+1})$ denote the iterates generated by algorithm 4.3.1 at iteration $k+1$. Then the gap function Q satisfies

$$\begin{aligned}
Q(w, \tilde{w}^{k+1}) \leq & \frac{\eta}{2}\left(\|x - \tilde{x}^k\|_{\Theta^k}^2 - \|x - \tilde{x}^{k+1}\|_{\Theta^k}^2\right) + \frac{\rho}{2}\left(\|\tilde{u}^k - u\|^2 - \|\tilde{u}^{k+1} - u\|^2\right) \\
& + \frac{\rho}{2}\left(\|\tilde{z}^k - Mx\|^2 - \|\tilde{z}^{k+1} - Mx\|^2 - \|\tilde{z}^k - M\tilde{x}^{k+1}\|^2\right) \\
& - \frac{\alpha\rho}{2}\left(\|M(\tilde{x}^k - x)\|^2 - \|M(\tilde{x}^{k+1} - x)\|^2\right) \\
& - \frac{1}{2}\left(\eta^k\|\tilde{x}^k - \tilde{x}^{k+1}\|_{\Theta^k}^2 - \hat{L}_\Theta\|\tilde{x}^k - \tilde{x}^{k+1}\|_{\Theta^k}^2 - \alpha\rho\|M\|^2\|\tilde{x}^k - \tilde{x}^{k+1}\|^2\right) \\
& + \left(\chi_1 + \tau_\zeta\|x - \tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha)\|x - \tilde{x}^{k+1}\|_{\rho M^T M}\right)\varepsilon_x^k + \alpha(\varepsilon_x^k)^2 \\
& + (1+2\rho)\varepsilon_z^k + \left(2(R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k}.
\end{aligned}$$

Proof. From the definition of Q,

$$Q(w, \tilde{w}^{k+1}) = f(\tilde{x}^{k+1}) - f(x) + g(\tilde{z}^{k+1}) - g(z) + \langle \rho u, M\tilde{x}^{k+1} - \tilde{z}^{k+1}\rangle - \langle \rho\tilde{u}^{k+1}, Mx - z\rangle.$$

Our goal is to upper bound $Q(w, \tilde{w}^{k+1})$. We start by bounding $f(\tilde{x}^{k+1}) - f(x)$ as follows:

$$\begin{aligned}
f(\tilde{x}^{k+1}) - f(x) &= f(\tilde{x}^{k+1}) - f(\tilde{x}^k) + f(\tilde{x}^k) - f(x) \\
&\overset{(1)}{\leq} \langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - \tilde{x}^k\rangle + \frac{\hat{L}_\Theta}{2}\|\tilde{x}^{k+1} - \tilde{x}^k\|_{\Theta_k}^2 + f(\tilde{x}^k) - f(x) \\
&\overset{(2)}{\leq} \langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - \tilde{x}^k\rangle + \frac{\hat{L}_\Theta}{2}\|\tilde{x}^{k+1} - \tilde{x}^k\|_{\Theta_k}^2 + \langle \nabla f(\tilde{x}^k), \tilde{x}^k - x\rangle \\
&= \langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - x\rangle + \frac{\hat{L}_\Theta}{2}\|\tilde{x}^{k+1} - \tilde{x}^k\|_{\Theta^k}^2,
\end{aligned}$$

where (1) uses relative smoothness and (2) uses convexity of f. Inserting the upper bound on $f(\tilde{x}^{k+1}) - f(x)$ into the expression for $Q(w, \tilde{w}^{k+1})$, we find

$$\begin{aligned}
Q(w, \tilde{w}^{k+1}) \leq & \langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - x\rangle + \frac{\hat{L}_\Theta}{2}\|\tilde{x}^{k+1} - \tilde{x}^k\|_{\Theta^k}^2 + g(\tilde{z}^{k+1}) - g(z) \\
& + \langle \rho u, M\tilde{x}^{k+1} - \tilde{z}^{k+1}\rangle - \langle \rho\tilde{u}^{k+1}, Mx - z\rangle.
\end{aligned} \tag{4.21}$$

Using approximate optimality of the iterates, we may insert the bounds of (4.19) and (4.20) from lemmas 4.5.2 and 4.5.3 into (4.21), which yields

$$\begin{aligned} Q(w, \tilde{w}^{k+1}) \leq\ & \langle \eta^k \Theta^k (\tilde{x}^{k+1} - \tilde{x}^k), x - \tilde{x}^{k+1} \rangle + \rho \langle M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - \tilde{x}^{k+1}) \rangle \\ & + \rho \langle \tilde{u}^{k+1}, \tilde{z}^{k+1} - z \rangle + \langle \rho u, M\tilde{x}^{k+1} - \tilde{z}^{k+1} \rangle - \langle \rho \tilde{u}^{k+1}, Mx - z \rangle \\ & + \alpha\rho \langle M(\tilde{x}^k - \tilde{x}^{k+1}), M(x - \tilde{x}^{k+1}) + \frac{\hat{L}_\Theta}{2} \|\tilde{x}^{k+1} - \tilde{x}^k\|^2_{\Theta^k} \\ & + \left(\chi_1 + \tau_\zeta \|x - \tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha) \|x - \tilde{x}^{k+1}\|_{\rho M^T M} \right) \varepsilon_x^k + \alpha (\varepsilon_x^k)^2 \\ & + (1 + 2\rho)\varepsilon_z^k + \left(2(R_{x^\star, u^\star, z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z\| \right) \sqrt{2\rho\varepsilon_z^k}. \end{aligned} \tag{4.22}$$

We now simplify (4.22) by combining terms. Some basic manipulations show the terms on line 2 of (4.22) may be rewritten as

$$\begin{aligned} & \rho \langle \tilde{u}^{k+1}, \tilde{z}^{k+1} - z \rangle + \rho \langle u, M\tilde{x}^{k+1} - \tilde{z}^{k+1} \rangle - \rho \langle \tilde{u}^{k+1}, Mx - z \rangle \\ & = \rho \langle \tilde{u}^{k+1} - u, \tilde{z}^{k+1} - M\tilde{x}^{k+1} \rangle - \rho \langle \tilde{u}^{k+1}, M(x - \tilde{x}^{k+1}) \rangle. \end{aligned}$$

We can combine the preceding display with the second term of line 1 in (4.22) to reach

$$\begin{aligned} & \rho \langle M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - \tilde{x}^{k+1}) \rangle - \rho \langle \tilde{u}^{k+1}, M(x - \tilde{x}^{k+1}) \rangle + \rho \langle \tilde{u}^{k+1} - u, \tilde{z}^{k+1} - M\tilde{x}^{k+1} \rangle \\ & = \langle \tilde{u}^{k+1} + \tilde{z}^{k+1} - \tilde{z}^k, M(x - \tilde{x}^{k+1}) \rangle - \rho \langle \tilde{u}^{k+1}, M(x - \tilde{x}^{k+1}) \rangle + \rho \langle \tilde{u}^{k+1} - u, \tilde{z}^{k+1} - M\tilde{x}^{k+1} \rangle \\ & = \langle \tilde{z}^{k+1} - \tilde{z}^k, M(x - \tilde{x}^{k+1}) \rangle + \rho \langle \tilde{u}^{k+1} - u, \tilde{z}^{k+1} - M\tilde{x}^{k+1} \rangle. \end{aligned}$$

Inserting the preceding simplification into (4.22), we reach

$$\begin{aligned} Q(w, \tilde{w}^{k+1}) \leq\ & \langle \eta^k \Theta^k (\tilde{x}^{k+1} - \tilde{x}^k), x - \tilde{x}^{k+1} \rangle + \rho \langle \tilde{u}^{k+1} - u, \tilde{z}^{k+1} - M\tilde{x}^{k+1} \rangle + \rho \langle \tilde{z}^{k+1} - \tilde{z}^k, M(x - \tilde{x}^{k+1}) \rangle \\ & + \alpha\rho \langle M(\tilde{x}^k - \tilde{x}^{k+1}), M(x - \tilde{x}^{k+1}) \rangle + \frac{\hat{L}_\Theta}{2} \|\tilde{x}^{k+1} - \tilde{x}^k\|^2_{\Theta^k} \\ & + \left(\chi_1 + \tau_\zeta \|x - \tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha) \|x - \tilde{x}^{k+1}\|_{\rho M^T M} \right) \varepsilon_x^k + \alpha (\varepsilon_x^k)^2 \\ & + (1 + 2\rho)\varepsilon_z^k + \left(2(R_{x^\star, u^\star, z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z\| \right) \sqrt{2\rho\varepsilon_z^k}. \end{aligned} \tag{4.23}$$

Now, we bound the first two leading terms in line 1 of (4.23) by invoking the identity $(a-b)^T\Upsilon(c-d) = 1/2\left(\|a-d\|^2_\Upsilon - \|a-c\|^2_\Upsilon\right) + 1/2\left(\|c-b\|^2_\Upsilon - \|d-b\|^2_\Upsilon\right)$ to obtain

$$\eta^k\langle\Theta^k(\tilde{x}^{k+1}-\tilde{x}^k), x-\tilde{x}^{k+1}\rangle + \rho\langle\tilde{u}^{k+1}-u, \tilde{z}^{k+1}-M\tilde{x}^{k+1}\rangle \leq$$
$$\frac{\eta^k}{2}\left(\|x-\tilde{x}^k\|^2_{\Theta^k} - \|x-\tilde{x}^{k+1}\|^2_{\Theta^k}\right) - \frac{\eta^k}{2}\|\tilde{x}^{k+1}-\tilde{x}^k\|^2_{\Theta^k} + \frac{\rho}{2}\left(\|\tilde{u}^k-u\|^2 - \|\tilde{u}^{k+1}-u\|^2 - \|\tilde{u}^k-\tilde{u}^{k+1}\|^2\right).$$

Similarly, to bound the third and fourth terms in (4.23), we again invoke $(a-b)^T\Upsilon(c-d) = 1/2\left(\|a-d\|^2_\Upsilon - \|a-c\|^2_\Upsilon\right) + 1/2\left(\|c-b\|^2_\Upsilon - \|d-b\|^2_\Upsilon\right)$ which yields

$$\begin{aligned}
\rho\langle\tilde{z}^{k+1}-\tilde{z}^k, M(x-\tilde{x}^{k+1})\rangle &= \frac{\rho}{2}\left(\|\tilde{z}^k-Mx\|^2 - \|\tilde{z}^{k+1}-Mx\|^2 + \|\tilde{z}^{k+1}-M\tilde{x}^{k+1}\|^2 - \|\tilde{z}^k-M\tilde{x}^{k+1}\|^2\right)\\
&= \frac{\rho}{2}\left(\|\tilde{z}^k-Mx\|^2 - \|\tilde{z}^{k+1}-Mx\|^2 - \|\tilde{z}^k-M\tilde{x}^{k+1}\|^2\right) + \frac{\rho}{2}\|\tilde{u}^k-\tilde{u}^{k+1}\|^2,\\
\alpha\rho\langle M(\tilde{x}^k-\tilde{x}^{k+1}), M(x-\tilde{x}^{k+1})\rangle &= -\frac{\alpha\rho}{2}\left(\|M(\tilde{x}^k-x)\|^2 - \|M(\tilde{x}^{k+1}-x)\|^2\right) + \frac{\alpha\rho}{2}\|M(\tilde{x}^k-\tilde{x}^{k+1})\|^2\\
&\leq -\frac{\alpha\rho}{2}\left(\|M(\tilde{x}^k-x)\|^2 - \|M(\tilde{x}^{k+1}-x)\|^2\right) + \frac{\alpha\rho}{2}\|M\|^2\|\tilde{x}^k-\tilde{x}^{k+1}\|^2.
\end{aligned}$$

Putting everything together, we conclude

$$\begin{aligned}
Q(w,\tilde{w}^{k+1}) \leq\ & \frac{\eta^k}{2}\left(\|\tilde{x}^k-x\|^2_{\Theta^k} - \|\tilde{x}^{k+1}-x\|^2_{\Theta^k}\right) + \frac{\rho}{2}\left(\|\tilde{u}^k-u\|^2 - \|\tilde{u}^{k+1}-u\|^2\right)\\
&+ \frac{\rho}{2}\left(\|\tilde{z}^k-Mx\|^2 - \|\tilde{z}^{k+1}-Mx\|^2 - \|\tilde{z}^k-M\tilde{x}^{k+1}\|^2\right)\\
&- \frac{\alpha\rho}{2}\left(\|M(\tilde{x}^k-x)\|^2 - \|M(\tilde{x}^{k+1}-x)\|^2\right)\\
&- \frac{1}{2}\left(\eta^k\|\tilde{x}^k-\tilde{x}^{k+1}\|^2_{\Theta^k} - \hat{L}_\Theta\|\tilde{x}^k-\tilde{x}^{k+1}\|^2_{\Theta^k} - \alpha\rho\|M\|^2\|\tilde{x}^k-\tilde{x}^{k+1}\|^2\right)\\
&+ \left(\chi_1 + \tau_\zeta\|x-\tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha)\|x-\tilde{x}^{k+1}\|_{\rho M^TM}\right)\varepsilon^k_x + \alpha(\varepsilon^k_x)^2\\
&+ (1+2\rho)\varepsilon^k_z + \left(2(R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1}-z\|\right)\sqrt{2\rho\varepsilon^k_z}.
\end{aligned}$$

as desired. □ □

4.5.3 Proof of theorem 4.5.1

With lemma 4.5.4 in hand, we are ready to prove theorem 4.5.1.

Proof. Let $\bar{w}^{t+1} = \frac{1}{t}\sum_{k=2}^{t+1} \tilde{w}^k$, where $\{\tilde{w}^k\}_{k\geq 1}$ are the iterates produced by algorithm 4.3.1. From lemma 4.5.4, we have for each k that

$$\begin{aligned}
Q(w, \tilde{w}^{k+1}) \leq{}& \frac{\eta^k}{2}\left(\|\tilde{x}^k - x\|_{\Theta^k}^2 - \|\tilde{x}^{k+1} - x\|_{\Theta^k}^2\right) + \frac{\rho}{2}\left(\|\tilde{u}^k - u\|^2 - \|\tilde{u}^{k+1} - u\|^2\right) \\
&+ \frac{\rho}{2}\left(\|\tilde{z}^k - Mx\|^2 - \|\tilde{z}^{k+1} - Mx\|^2 - \|\tilde{z}^k - M\tilde{x}^{k+1}\|^2\right) \\
&- \frac{\alpha\rho}{2}\left(\|M(\tilde{x}^k - x)\|^2 - \|M(\tilde{x}^{k+1} - x)\|^2\right) \\
&- \frac{1}{2}\left(\eta^k\|\tilde{x}^k - \tilde{x}^{k+1}\|_{\Theta^k}^2 - \hat{L}_\Theta\|\tilde{x}^k - \tilde{x}^{k+1}\|_{\Theta^k}^2 - \alpha\rho\|M\|^2\|\tilde{x}^k - \tilde{x}^{k+1}\|^2\right) \\
&+ \left(\chi_1 + \tau_\zeta\|x - \tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha)\|x - \tilde{x}^{k+1}\|_{\rho M^T M}\right)\varepsilon_x^k + \alpha(\varepsilon_x^k)^2 \\
&+ (1+2\rho)\varepsilon_z^k + \left(2(R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k}.
\end{aligned}$$

Now, summing up the preceding display from $k = 1$ to t and setting $w = (x^\star, z^\star, u)$, we obtain

$$\begin{aligned}
\sum_{k=2}^{t+1} Q(x^\star, z^\star, u; \tilde{w}^k) \leq{}& \underbrace{\frac{\eta^k}{2}\sum_{k=1}^{t}\left(\|\tilde{x}^k - x^\star\|_{\Theta^k}^2 - \|\tilde{x}^{k+1} - x^\star\|_{\Theta^k}^2\right)}_{T_1} + \underbrace{\frac{\rho}{2}\sum_{k=1}^{t}\left(\|\tilde{u}^k - u\|^2 - \|\tilde{u}^{k+1} - u\|^2\right)}_{T_2} \\
&+ \underbrace{\frac{\rho}{2}\sum_{k=1}^{t}\left(\|\tilde{z}^k - Mx^\star\|^2 - \|\tilde{z}^{k+1} - Mx^\star\|^2 - \|\tilde{z}^k - M\tilde{x}^{k+1}\|^2\right)}_{T_3} \\
&- \underbrace{\frac{\alpha\rho}{2}\sum_{k=1}^{t}\left(\|M(\tilde{x}^k - x^\star)\| - \|M(\tilde{x}^{k+1} - x^\star)\|\right)}_{T_4} \\
&- \underbrace{\frac{1}{2}\sum_{k=1}^{t}\left((\eta^k - \hat{L}_\Theta)\|\tilde{x}^k - \tilde{x}^{k+1}\|_{\Theta^k}^2 - \alpha\rho\|M\|^2\|\tilde{x}^k - \tilde{x}^{k+1}\|^2\right)}_{T_5} \\
&+ \underbrace{\sum_{k=1}^{t}\left(\left(\chi_1 + \tau_\zeta\|x^\star - \tilde{x}^{k+1}\|_{\eta\Theta^1} + (1+\alpha)\|x^\star - \tilde{x}^{k+1}\|_{\rho M^T M}\right)\varepsilon_x^k + \alpha(\varepsilon_x^k)^2\right)}_{T_6} \\
&+ \underbrace{\sum_{k=1}^{t}\left((1+2\rho)\varepsilon_z^k + \left(2(R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z^\star\|\right)\sqrt{2\rho\varepsilon_z^k}\right)}_{T_7}.
\end{aligned}$$

We start by bounding T_6 and T_7. For T_6, we have by summability of the

forcing sequences (assumption 4.4.3) that

$$\begin{aligned} T_6 &\leq \left(\alpha K_{\varepsilon_x} + \chi_1 + \tau_\zeta R_{x^\star,\Theta^1} + \rho(1+\alpha) R_{x^\star,M}\right) \mathcal{E}_x \\ &\leq \left(\alpha K_{\varepsilon_x} + \chi_1 + (\rho + \rho\alpha + \tau_\zeta) R_\star\right) \mathcal{E}_x \\ &\leq \left(\alpha K_{\varepsilon_x} + \beta_f + \left(\rho + 3\rho\alpha + 3\tau_\zeta + 4\sqrt{\rho}\right) R_\star\right) \mathcal{E}_x \\ &= (\alpha K_{\varepsilon_x} + \beta_f + \chi R_\star) \mathcal{E}_x. \end{aligned}$$

Here we define constant $\chi := \rho + 3\rho\alpha + 3\tau_\zeta + 4\sqrt{\rho}$ to simplify the notation. By analogous reasoning, we find for T_7 that

$$T_7 \leq \left((1+2\rho) K_{\varepsilon_z} + 5R_\star \sqrt{2\rho}\right) \mathcal{E}_z.$$

Thus we reach

$$T_6 + T_7 \leq (\alpha K_{\varepsilon_x} + \beta_f + \chi R_\star) \mathcal{E}_x + \left((1+2\rho) K_{\varepsilon_z} + 5R_\star \sqrt{2\rho}\right) \mathcal{E}_z.$$

Next we bound T_5. As $\Theta^k \succeq \theta I$ for all k and $\eta = \hat{L}_\Theta + \frac{\alpha\rho\|M\|^2}{\theta}$, we always have $T_5 \geq 0$ and thus we can drop this term from the analysis.

We now turn to bounding T_1. Using the definition of T_1, we find

$$\begin{aligned} T_1 &= \frac{\eta}{2} \sum_{k=1}^{t} \left(\|\tilde{x}^k - x^\star\|_{\Theta^k}^2 - \|\tilde{x}^{k+1} - x^\star\|_{\Theta^k}^2 \right) \\ &= \frac{\eta}{2} \left(\|\tilde{x}^1 - x^\star\|_{\Theta^1}^2 - \|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2 \right) + \frac{\eta}{2} \sum_{k=2}^{t} \left(\|\tilde{x}^k - x^\star\|_{\Theta^k}^2 - \|\tilde{x}^k - x^\star\|_{\Theta^{k-1}}^2 \right). \end{aligned}$$

Now using our hypotheses on the sequence $\{\Theta^k\}_k$ in (4.13), we obtain

$$\begin{aligned} \|\tilde{x}^k - x^\star\|_{\Theta^k}^2 - \|\tilde{x}^k - x^\star\|_{\Theta^{k-1}}^2 = (\tilde{x}^k - x^\star)^T (\Theta^k - \Theta^{k-1})(\tilde{x}^k - x^\star) &\leq \zeta^{k-1} \|\tilde{x}^k - x^\star\|_{\Theta^{k-1}}^2 \\ &\leq \tau_\zeta \zeta^{k-1} \|\tilde{x}^k - x^\star\|_{\Theta^1}^2. \end{aligned}$$

Inserting the previous bound into T_1, we reach

$$\begin{aligned}
T_1 &\le \frac{\eta}{2}\|\tilde{x}^1 - x^\star\|_{\Theta^1}^2 + \frac{\eta}{2}\tau_\zeta \sum_{k=1}^{t} \zeta^{k-1}\|\tilde{x}^k - x^\star\|_{\Theta^1}^2 - \frac{\eta}{2}\|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2 \\
&\le \frac{\eta}{2}\left(\|\tilde{x}^1 - x^\star\|_{\Theta^1}^2 + \mathcal{E}_\zeta R_{x^\star,\Theta^1}^2\right) - \frac{\eta}{2}\|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2 \\
&\le \frac{\eta}{2}\left(\|\tilde{x}^1 - x^\star\|_{\Theta^1}^2 + \mathcal{E}_\zeta R_\star^2\right) - \frac{\eta}{2}\|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2 = \frac{\eta}{2}\left(d_{x^\star,\Theta^1}^2 + \mathcal{E}_\zeta R_\star^2\right) - \frac{\eta}{2}\|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2.
\end{aligned}$$

Next we bound T_3. As we can ignore the last term in the summand, we obtain a telescoping sum. This yields the following bound for T_3:

$$\begin{aligned}
T_3 &\le \frac{\rho}{2}\sum_{k=1}^{t}\left(\|\tilde{z}^k - Mx^\star\|^2 - \|\tilde{z}^{k+1} - Mx^\star\|^2\right) = \frac{\rho}{2}\sum_{i=1}^{t}\left(\|\tilde{z}^k - z^\star\|^2 - \|\tilde{z}^{k+1} - z^\star\|^2\right) \\
&\le \frac{\rho}{2}\|\tilde{z}^1 - z^\star\|^2 = \frac{\rho}{2}d_{z^\star}^2.
\end{aligned}$$

Next, we observe T_4 is a telescoping sum, so

$$T_4 = \frac{\alpha\rho}{2}\sum_{k=1}^{t}\left(\|M(\tilde{x}^k - x^\star)\| - \|M(\tilde{x}^{k+1} - x^\star)\|\right) = \frac{\alpha\rho}{2}\left(\|M(\tilde{x}^1 - x^\star)\|^2 - \|M(\tilde{x}^{t+1} - x^\star)\|^2\right).$$

Last, T_2 is also a telescoping sum, hence

$$T_2 = \frac{\rho}{2}\sum_{k=1}^{t}\left(\|\tilde{u}^k - u\|^2 - \|\tilde{u}^{k+1} - u\|^2\right) = \frac{\rho}{2}\left(\|\tilde{u}^1 - u\|^2 - \|\tilde{u}^{t+1} - u\|^2\right).$$

By the definition of η, we have

$$\frac{\eta}{2}\|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2 \ge \frac{\alpha\rho}{2}\|M(\tilde{x}^{t+1} - x^\star)\|^2.$$

Thus,

$$\begin{aligned}
T_1 + T_3 - T_4 &\le \frac{\eta}{2}\mathcal{E}_\zeta R_\star^2 + \frac{\eta}{2}d_{x^\star,\Theta^1}^2 - \left(\frac{\eta}{2}\|\tilde{x}^{t+1} - x^\star\|_{\Theta^t}^2 - \frac{\alpha\rho}{2}\|M(\tilde{x}^{t+1} - x^\star)\|^2\right) + \left(\frac{\rho}{2}d_{z^\star}^2 - \frac{\alpha\rho}{2}\|M(\tilde{x}^1 - x^\star)\|^2\right) \\
&\le \frac{\eta}{2}\left(d_{x^\star,\Theta^1}^2 + \mathcal{E}_\zeta R_\star^2\right) + \frac{\rho(1-\alpha)}{2}d_{z^\star}^2.
\end{aligned}$$

The second inequality follows from the relation $M\tilde{x}^1 = \tilde{z}^1$. Using our bounds on T_1 through T_7, we find

$$\begin{aligned}\sum_{k=2}^{t+1} Q(x^\star, z^\star, u; \tilde{w}^k) &\leq \frac{\eta}{2}\left(d^2_{x^\star,\Theta^1} + \mathcal{E}_\zeta R_\star^2\right) + \frac{\rho(1-\alpha)}{2} d^2_{z^\star} + (\alpha K_{\varepsilon_x} + \beta_f + \chi R_\star)\mathcal{E}_x \\ &\quad + \left((1+2\rho)K_{\varepsilon_z} + 5R_\star\sqrt{2\rho}\right)\mathcal{E}_z + \frac{\rho}{2}\left(\|\tilde{u}^1 - u\|^2 - \|\tilde{u}^{t+1} - u\|^2\right) \\ &= \frac{\eta}{2} d^2_{x^\star,\Theta^1} + \frac{\rho(1-\alpha)}{2} d^2_{z^\star} + C_x\mathcal{E}_x + C_z\mathcal{E}_z + \frac{\eta}{2}\mathcal{E}_\zeta R_\star^2 + \frac{\rho}{2}\left(\|\tilde{u}^1 - u\|^2 - \|\tilde{u}^{t+1} - u\|^2\right),\end{aligned}$$

where $C_x \coloneqq \alpha K_{\varepsilon_x} + \beta_f + \chi R_\star$ and $C_z \coloneqq (1+2\rho)K_{\varepsilon_z} + 5R_\star\sqrt{2\rho}$. Now, as $\bar{w}^{t+1} = \frac{1}{t}\sum_{k=2}^{t+1}\tilde{w}^k$, the convexity of Q in its second argument yields

$$\begin{aligned}Q(x^\star, z^\star, u; \bar{w}^{t+1}) &\leq \frac{1}{t}\sum_{k=2}^{t+1} Q(x^\star, z^\star, u; \tilde{w}^k) \\ &\leq \frac{1}{t}\left(\frac{\eta}{2} d^2_{x^\star,\Theta^1} + \frac{\rho(1-\alpha)}{2} d^2_{z^\star} + C_x\mathcal{E}_x + C_z\mathcal{E}_z + \frac{\eta}{2}\mathcal{E}_\zeta R_\star^2 + \frac{\rho}{2}\left(\|\tilde{u}^1 - u\|^2 - \|\tilde{u}^{t+1} - u\|^2\right)\right).\end{aligned} \tag{4.24}$$

Define $\Gamma \coloneqq \frac{\eta}{2} d^2_{x^\star,\Theta^1} + \frac{\rho(1-\alpha)}{2} d^2_{z^\star} + C_x\mathcal{E}_x + C_z\mathcal{E}_z + \frac{\eta}{2}\mathcal{E}_\zeta R_\star^2$. Since $Q(w^\star, \bar{w}^{t+1}) \geq 0$, by (4.24) we reach

$$\|\tilde{u}^{t+1} - u^\star\|^2 \leq \frac{2}{\rho}\Gamma + d^2_{u^\star}.$$

Let $\tilde{v}^{t+1} = \frac{1}{t}\left(\tilde{u}^1 - \tilde{u}^{t+1}\right)$. Then we can bound $\|\tilde{v}_{t+1}\|^2$ as

$$\|\tilde{v}^{t+1}\|^2 \leq \frac{2}{t^2}\left(\|\tilde{u}^1 - u^\star\|^2 + \|\tilde{u}^{t+1} - u^\star\|^2\right) \leq \frac{4}{t^2}\left(\frac{\Gamma}{\rho} + d^2_{u^\star}\right).$$

By (4.24), given the fact $\tilde{u}^1 = 0$, we also have

$$Q(x^\star, z^\star, u; \bar{w}^{t+1}) \leq \frac{\Gamma - \rho\langle\tilde{u}^1 - \tilde{u}^{t+1}, u\rangle}{t} = \frac{\Gamma}{t} - \rho\langle\tilde{v}^{t+1}, u\rangle,$$

where the equality follows from the definition of $\tilde{v}^{t+1}$. Hence for any u

$$Q(x^\star, z^\star, u; \bar{w}^{t+1}) + \langle\tilde{v}^{t+1}, \rho u\rangle \leq \frac{\Gamma}{t},$$

and therefore

$$\ell_U(\tilde{v}^{t+1}, \bar{w}^{t+1}) \leq \frac{1}{t}\left(\frac{\eta}{2} d^2_{x^\star,\Theta^1} + \frac{\rho(1-\alpha)}{2} d^2_{z^\star} + C_x\mathcal{E}_x + C_z\mathcal{E}_z + \frac{\eta}{2}\mathcal{E}_\zeta R_\star^2\right).$$

We now finish the proof by invoking lemma 4.5.1. □ □

4.6 Linear convergence of GeNI-ADMM

In this section, we seek to establish linear convergence results for algorithm 4.3.1. In general, the linear convergence of ADMM relies on strong convexity of the objective function [12, 77, 82]. Consistently, the linear convergence of GeNI-ADMM also requires strong convexity. Many applications of GeNI-ADMM fit into this setting, such as elastic net [43]. However, linear convergence is not restricted to strongly convex problems. It has been shown that local linear convergence of ADMM can be guaranteed even without strong convexity [110]. Experiments in section 3.5 show the same phenomenon for GeNI-ADMM: it converges linearly after a couple of iterations when it reaches some neighborhood of the solution. The linear convergence theory of GeNI-ADMM provides a way to understand this phenomenon. We first list the assumptions required for linear convergence:

Assumption 4.6.1 (Regularity of f and g)**.** The function f is finite valued, strongly convex with parameter σ_f, and smooth with parameter L_f. The function g is finite valued, convex, and lower semi-continuous.

Assumption 4.6.1 modifies assumption 4.4.5 by measuring strong convexity and smoothness of f in the standard way rather than relative to $\{\Theta^k\}_{k\geq 1}$.

Assumption 4.6.2 (Gradient estimation error of f)**.** There exists a forcing sequence $\{\varepsilon_\nabla^k\}_{k\geq 1}$ such that

$$\|\nabla f(\tilde{x}^{k+1}) - (\nabla f(\tilde{x}^k) + \eta^k \Theta^k(\tilde{x}^{k+1} - \tilde{x}^k))\| \leq \varepsilon_\nabla^k. \tag{4.25}$$

Recall that GeNI-ADMM approximates f at iteration k by a generalized second-order Taylor expansion, as in (4.6). Assumption 4.6.2 shows the

first-order generalized Taylor expansion of $\nabla f(x)$ about $\tilde{x}^k$ well approximates $\nabla f(\tilde{x}^{k+1})$. Note, when f is quadratic eq. (4.25) holds with $\varepsilon_\nabla^k = 0$.

Assumption 4.6.3. The linear operator M has full row rank.

Assumption 4.6.4 (Geometric decay of the forcing sequences)**.** There exists a constant $q > 0$ such that the forcing sequences $\{\varepsilon_\nabla^k\}_{k\geq 1}$, $\{\varepsilon_x^k\}_{k\geq 1}$, and $\{\varepsilon_z^k\}_{k\geq 1}$ satisfy

$$\varepsilon_\nabla^{k+1} \leq \varepsilon_\nabla^k/(1+q), \quad \varepsilon_x^{k+1} \leq \varepsilon_x^k/(1+q), \text{ and } \varepsilon_z^{k+1} \leq \varepsilon_z^k/(1+q)^2. \tag{4.26}$$

Assumption 4.6.4 requires the gradient estimation error and inexactness sequences to decay geometrically. Compared with the sublinear convergence result, linear convergence requires more accurate approximations of f by its penalized linearization and solutions of the subproblems. Again, since the z-subproblem inexactness is weaker than the x-subproblem inexactness, $\{\varepsilon_z^k\}_{k\geq 1}$ should have faster decay rate ($(1+q)^2$) than the decay rate $(1+q)$ of $\{\varepsilon_x^k\}_{k\geq 1}$.

Inspired by [27], let $y = (z, u)$, $\tilde{y} = (\tilde{z}, \tilde{u})$, and $y^\star = (z^\star, u^\star)$. We define term $\|\tilde{y}^k - y^\star\|^2 = \|\tilde{z}^k - z^\star\|^2 + \|\tilde{u}^k - u^\star\|^2$, which can be viewed as a *Lyapunov function* [91, 99]. We show that $\|\tilde{y}^k - y^\star\|^2$ decreases geometrically, which guarantees the linear convergence of algorithm 4.3.1.

Theorem 4.6.1. Instate Assumptions 4.4.1, 4.4.4, and 4.4.6-4.6.4. There exist constants δ and $S > 0$ such that if $q > \delta$,

$$(1+\delta)^{k-1}\|\tilde{y}^{k+1} - y^\star\|^2 \leq \|\tilde{y}^2 - y^\star\|^2 + S. \tag{4.27}$$

Furthermore, for appropriate choice of stepsize $\rho = \sqrt{\frac{L_f \sigma_f}{\|M\|^2 \lambda_{\min}(MM^T)}}$, the decay rate satisfies $\delta = \frac{1}{\kappa_M \sqrt{\kappa_f}}$, where $\kappa_f = \frac{L_f}{\sigma_f}$ is the condition number of f and κ_M is the condition number of M.

4.6.1 Sufficient descent

From theorem 4.6.1, to establish linear convergence of GeNI-ADMM, it suffices to show $\|\tilde{y}^k - y^\star\|^2$ decreases geometrically. We take two steps to achieve this. First, we show that $\|\tilde{y}^k - y^\star\|^2$ decreases for every iteration k. Second, we bound the decrease $\|\tilde{y}^k - y^\star\|^2 - \|\tilde{y}^{k+1} - y^\star\|^2$ below by $\delta\|\tilde{y}^{k+1} - y^\star\|^2$ for some $\delta > 0$. Since the subproblems are only solved approximately, we should also consider the inexactness of the solutions in these two steps. For the first step, we use strong convexity of f and convexity of g with appropriate perturbations to account for the inexactness. We call these conditions *perturbed convexity* conditions, as outlined in lemma 4.6.1. A detailed proof of lemma 4.6.1 is presented in appendix C.3.

Lemma 4.6.1. Let $\tilde{x}^{k+1}$ and $\tilde{z}^{k+1}$ be the inexact solutions of x and z-subproblems under definition 4.3.1. Recall $(x^\star, z^\star, u^\star)$ is a saddle point of (4.1). Then the following inequalities are satisfied:

1. **(Semi-inexact f-strong convexity)**
$$\begin{aligned}\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + \frac{R_{x^\star,u^\star,z^\star}}{\rho}\varepsilon_\nabla^k + \\ \frac{\sqrt{\eta^k\|\Theta^k\|}R_{x^\star,u^\star,z^\star} + \sqrt{\rho}R_{x^\star,M}}{\rho}\varepsilon_x^k \geq \frac{\sigma_f}{\rho}\|\tilde{x}^{k+1} - x^\star\|^2,\end{aligned} \tag{4.28}$$

2. **(Semi-inexact g-convexity)**
$$\langle \tilde{z}^{k+1} - z^\star, \tilde{u}^{k+1} - u^\star\rangle + 2\sqrt{\frac{2}{\rho}}R_{x^\star,u^\star,z^\star}\sqrt{\varepsilon_z^k} \geq 0, \tag{4.29}$$

3. **(Inexact g-convexity)**
$$\langle \tilde{z}^k - \tilde{z}^{k+1}, \tilde{u}^k - \tilde{u}^{k+1}\rangle + 8\sqrt{\frac{2}{\rho}}R_{x^\star,u^\star,z^\star}\sqrt{\varepsilon_z^{k-1}} + \frac{4}{\rho}\varepsilon_z^{k-1} \geq 0. \tag{4.30}$$

In lemma 4.6.1, we call (4.28) and (4.29) *semi-inexact* (strong) convexity because $z^\star$ ($x^\star$) is part of the exact saddle point but $\tilde{z}^{k+1}$ ($\tilde{x}^{k+1}$) is an inexact subproblem solution. While in (4.30), both $\tilde{z}^k$ and $\tilde{z}^{k+1}$ are inexact subproblem solutions.

With lemma 4.6.1, we are able to show that $\|\tilde{y}^k - y^\star\|^2$ decreases for every iteration k considering inexactness as the following lemma.

Lemma 4.6.2. Define constants $C_1 = \frac{2R_{x^\star,u^\star,z^\star}}{\rho}$, $C_2 \geq \frac{2(\sqrt{\eta^k\|\Theta^k\|}R_{x^\star,u^\star,z^\star} + \sqrt{\rho}R_{x^\star,M})}{\rho}$ for all k, and $C_3 = 20\sqrt{\frac{2}{\rho}}R_{x^\star,u^\star,z^\star}$. The sufficient descent condition holds:

$$\|\tilde{y}^k - y^\star\|^2 - \|\tilde{y}^{k+1} - y^\star\|^2 + C_1\varepsilon_\nabla^k + C_2\varepsilon_x^k + C_3\sqrt{\varepsilon_z^{k-1}} + \frac{8}{\rho}\varepsilon_z^{k-1} \geq \|\tilde{y}^k - \tilde{y}^{k+1}\|^2 + \frac{2\sigma_f}{\rho}\|\tilde{x}^{k+1} - x^\star\|^2. \tag{4.31}$$

Proof. Proof. By adding the inequalities (4.28) and (4.29) together, using the relation $M(\tilde{x}^{k+1} - x^\star) - (\tilde{z}^{k+1} - z^\star) = \tilde{u}^{k+1} - \tilde{u}^k$, and rearranging, we have

$$\begin{aligned}
&\langle \tilde{u}^{k+1} - \tilde{u}^k, u^\star - \tilde{u}^{k+1}\rangle + \langle \tilde{z}^{k+1} - z^\star, \tilde{z}^k - \tilde{z}^{k+1}\rangle + \\
&\frac{R_{x^\star,u^\star,z^\star}}{\rho}\varepsilon_\nabla^k + \frac{\sqrt{\eta^k\|\Theta^k\|}R_{x^\star,u^\star,z^\star} + \sqrt{\rho}R_{x^\star,M}}{\rho}\varepsilon_x^k + 2\sqrt{\frac{2}{\rho}}R_{x^\star,u^\star,z^\star}\sqrt{\varepsilon_z^k} \geq \\
&\frac{\sigma_f}{\rho}\|\tilde{x}^{k+1} - x^\star\|^2 + \langle \tilde{u}^k - \tilde{u}^{k+1}, \tilde{z}^k - \tilde{z}^{k+1}\rangle.
\end{aligned}$$

Recall $y = (z, u)$ and definitions of constants C_1, C_2, and C_3. Using the identity $(a-b)^T\Upsilon(c-d) = 1/2\left(\|a-d\|_\Upsilon^2 - \|a-c\|_\Upsilon^2\right) + 1/2\left(\|c-b\|_\Upsilon^2 - \|d-b\|_\Upsilon^2\right)$, we arrive at

$$\begin{aligned}
&\|\tilde{y}^k - y^\star\|^2 - \|\tilde{y}^{k+1} - y^\star\|^2 + C_1\varepsilon_\nabla^k + C_2\varepsilon_x^k + \frac{C_3}{5}\sqrt{\varepsilon_z^k} \geq \\
&\|\tilde{y}^k - \tilde{y}^{k+1}\|^2 + 2\langle \tilde{z}^k - \tilde{z}^{k+1}, \tilde{u}^k - \tilde{u}^{k+1}\rangle + \frac{2\sigma_f}{\rho}\|\tilde{x}^{k+1} - x^\star\|^2
\end{aligned} \tag{4.32}$$

Inserting the lower bound (4.30) into (4.32) to remove the cross-term $2\langle \tilde{z}^k - \tilde{z}^{k+1}, \tilde{u}^k - \tilde{u}^{k+1}\rangle$, noticing the fact $\varepsilon_z^{k-1} \geq \varepsilon_z^k$, and rearranging, we arrive at the sufficient descent condition as desired. □ □

Given the sufficient descent condition (4.31), the next step to show linear convergence is to show the righthand side in (4.31) bounds $\|\tilde{y}^{k+1} - y^\star\|^2$ and thereby to show this distance decays geometrically.

Lemma 4.6.3. Define $C_4 \geq \frac{2\|\eta^k\Theta^k+\rho M^T M\|}{\rho L_f}$ for all k. There exists a constant $\delta > 0$ such that

$$\|\tilde{y}^k-y^\star\|^2-\|\tilde{y}^{k+1}-y^\star\|^2+C_1\varepsilon_\nabla^k+\frac{2}{\rho L_f}(\varepsilon_\nabla^k)^2+C_2\varepsilon_x^k+C_4(\varepsilon_x^k)^2+C_3\sqrt{\varepsilon_z^{k-1}}+\frac{8}{\rho}\varepsilon_z^{k-1} \geq \delta\|\tilde{y}^{k+1}-y^\star\|^2. \tag{4.33}$$

As in theorem 4.6.1, the constant δ can be regarded as the rate of linear convergence. It depends on the condition numbers κ_f and κ_M. For appropriate choice of stepsize ρ, it satisfies $\delta = \frac{1}{\kappa_M\sqrt{\kappa_f}}$. The proof of lemma 4.6.3, and an explicit expression for δ, appears in appendix C.3.

4.6.2 Proof of theorem 4.6.1

Proof. Proof. By induction on (4.33), we have

$$\|\tilde{y}^2 - y^\star\|^2 + C_1\sum_{\upsilon=2}^{k}(1+\delta)^{\upsilon-2}\varepsilon_\nabla^\upsilon + \frac{2}{\rho L_f}\sum_{\upsilon=2}^{k}(1+\delta)^{\upsilon-2}(\varepsilon_\nabla^\upsilon)^2 + C_2\sum_{\upsilon=2}^{k}(1+\delta)^{\upsilon-2}\varepsilon_x^\upsilon +$$
$$C_4\sum_{\upsilon=2}^{k}(1+\delta)^{\upsilon-2}(\varepsilon_x^\upsilon)^2 + C_3\sum_{\upsilon=1}^{k-1}(1+\delta)^{\upsilon-1}\sqrt{\varepsilon_z^\upsilon} + \frac{8}{\rho}\sum_{\upsilon=1}^{k-1}(1+\delta)^{\upsilon-1}\varepsilon_z^\upsilon \geq (1+\delta)^{k-1}\|\tilde{y}^{k+1}-y^\star\|^2. \tag{4.34}$$

From assumption 4.6.4, sequence $\{\varepsilon_\nabla^k\}_{k\geq 1}$ has geometric decay as $\varepsilon_\nabla^{k+1} \leq \varepsilon_\nabla^k/(1+q)$. The x-inexactness summation term in (4.34) can be further simplified as

$$C_1\sum_{\upsilon=2}^{k}(1+\delta)^{\upsilon-2}\varepsilon_\nabla^\upsilon \leq C_1\varepsilon_\nabla^2\sum_{\upsilon=0}^{k-2}\left(\frac{1+\delta}{1+q}\right)^\upsilon := S_1,$$
$$\frac{2}{\rho L_f}\sum_{\upsilon=2}^{k}(1+\delta)^{\upsilon-2}(\varepsilon_\nabla^\upsilon)^2 \leq \frac{2}{\rho L_f}(\varepsilon_\nabla^2)^2\sum_{\upsilon=0}^{k-2}\left(\frac{1+\delta}{(1+q)^2}\right)^\upsilon := S_2.$$

Similarly, we can simplify the x and z-inexactness summation terms as

$$
\begin{aligned}
C_2 \sum_{\upsilon=2}^{k} (1+\delta)^{\upsilon-2} \varepsilon_x^{\upsilon} &\leq C_2 \varepsilon_x^2 \sum_{\upsilon=0}^{k-2} \left(\frac{1+\delta}{1+q} \right)^{\upsilon} := S_3, \\
C_4 \sum_{\upsilon=2}^{k} (1+\delta)^{\upsilon-2} (\varepsilon_x^{\upsilon})^2 &\leq C_4 (\varepsilon_x^2)^2 \sum_{\upsilon=0}^{k-2} \left(\frac{1+\delta}{(1+q)^2} \right)^{\upsilon} := S_4, \\
C_3 \sum_{\upsilon=1}^{k-1} (1+\delta)^{\upsilon-1} \sqrt{\varepsilon_z^{\upsilon}} &\leq C_3 \sqrt{\varepsilon_z^1} \sum_{\upsilon=0}^{k-2} \left(\frac{1+\delta}{1+q} \right)^{\upsilon} := S_5, \\
\frac{8}{\rho} \sum_{\upsilon=1}^{k-1} (1+\delta)^{\upsilon-1} \varepsilon_z^{\upsilon} &\leq \frac{8}{\rho} \varepsilon_z^1 \sum_{\upsilon=0}^{k-2} \left(\frac{1+\delta}{(1+q)^2} \right)^{\upsilon} := S_6.
\end{aligned}
$$

Since $q > \delta > 0$, terms S_1 to S_6 are all bounded for any $k > 1$. Define constant $S := \sum_{i=1}^{6} S_i$. We know S is finite and

$$
(1+\delta)^{k-1} \|\tilde{y}^{k+1} - y^\star\|^2 \leq \|\tilde{y}^2 - y^\star\|^2 + S,
$$

as desired. □ □

4.7 Applications

In this section we apply our theory to establish convergence rates for NysADMM and sketch-and-solve ADMM.

4.7.1 Convergence of NysADMM

Algorithm 4.7.1 NysADMM

input: penalty parameter ρ, step-size η, forcing sequences $\{\varepsilon_x^k\}_{k\geq 1}$, $\{\varepsilon_z^k\}_{k\geq 1}$

repeat

Find ε_x^k-approximate solution $\tilde{x}^{k+1}$ of

$$(\eta H_f^k + \rho I)x = \eta H_f^k \tilde{x}^k - \nabla f(\tilde{x}^k) + \rho(\tilde{z}^k - \tilde{u}^k)$$

$\tilde{z}^{k+1} \overset{\varepsilon_z^k}{\approx} \underset{z\in Z}{\operatorname{argmin}}\{g(z) + \frac{\rho}{2}\|M\tilde{x}^{k+1} - z + \tilde{u}^k\|^2\}$

$\tilde{u}^{k+1} = \tilde{u}^k + M\tilde{x}^{k+1} - \tilde{z}^{k+1}$

until convergence

output: solution $(x^\star, z^\star)$ of problem (4.1)

We begin with the NysADMM scheme from [112]. As discussed above, NysADMM is obtained from algorithm 4.3.1 by setting $\alpha = 0$ and using the exact Hessian $\Theta^k = H_f(\tilde{x}^k) = H_f^k$. Instantiating these selections into algorithm 4.3.1, we obtain NysADMM, presented as algorithm 4.7.1. The x-subproblem for NysADMM is the linear system (4.9). NysADMM solves this system using the Nyström PCG method from [41]. The general convergence of NysADMM was left open in [112], which established convergence only for quadratic f. Moreover, the result in [112] does not provide an explicit rate of convergence. We shall now rectify this state of affairs using theorem 4.5.1. Substituting the parameters defining NysADMM into theorem 4.5.1, we obtain the following convergence guarantee.

Corollary 4.7.1. Instate the hypotheses of theorem 4.5.1. Set $\alpha = 0$, $\eta = \hat{L}_f$, and $\Theta^k = H_f(\tilde{x}^k) = H_f^k$ in algorithm 4.3.1. Then

$$f(\bar{x}^{t+1}) + g(\bar{z}^{t+1}) - p^\star \leq \frac{\Gamma}{t}$$

and

$$\|M\bar{x}^{t+1} - \bar{z}^{t+1}\| \leq \frac{2}{t}\sqrt{\frac{\Gamma}{\rho} + d_{u^\star}^2}.$$

Here Γ and $d_{u^\star}^2$ are the same as in theorem 4.5.1.

NysADMM converges at the same $O(1/t)$-rate as standard ADMM, despite all the approximations it makes. Thus, NysADMM offers the same level of performance as ADMM, but is much faster due to its use of inexactness. corollary 4.7.1 supports the empirical choice of a constant step-size $\eta = 1$, which was shown to have excellent performance uniformly across tasks in [112]: the theorem sets $\eta = \hat{L}_f$, and $\hat{L}_f = 1$ for quadratic functions and satisfies $\hat{L}_f = O(1)$ for loss functions such as the logistic loss. More generally, we recommend setting $\eta = 1$ as the default value for GeNI-ADMM. Given NysADMM's superb empirical performance in [112] and the firm theoretical grounding given by corollary 4.7.1, we conclude that NysADMM provides a reliable framework for solving large-scale machine learning problems.

4.7.2 Convergence of sketch-and-solve ADMM

Algorithm 4.7.2 Sketch-and-solve ADMM

input: penalty parameter ρ, step-size η, $\{\varepsilon_z^k\}_{k\geq 1}$

repeat

Find solution $\tilde{x}^{k+1}$ of

$$(\eta\hat{H}^k + \rho I)x = \eta\left(\hat{H}^k + \gamma^k I\right)\tilde{x}^k - \nabla f(\tilde{x}^k) + \rho(\tilde{z}^k - \tilde{u}^k)$$

$\tilde{z}^{k+1} \overset{\varepsilon_z^k}{\cong} \underset{z\in Z}{\operatorname{argmin}}\{g(z) + \frac{\rho}{2}\|M\tilde{x}^{k+1} - z + \tilde{u}^k\|^2\}$

$\tilde{u}^{k+1} = \tilde{u}^k + M\tilde{x}^{k+1} - \tilde{z}^{k+1}$

until convergence

output: solution $(x^\star, z^\star)$ of problem (4.1)

Sketch-and-solve ADMM is obtained from GeNI-ADMM by setting Θ^k to be an approximate Hessian computed by a sketching procedure. The two most popular sketch-and-solve methods are the Newton sketch- [28, 59, 83] and Nyström sketch-and-solve [4, 7, 41]. Both methods use an approximate Hessian that is cheap to compute and yield a linear system that is fast to solve. ADMM together with sketching techniques provide a compelling means to handle massive problem instances. The recent survey [16] suggests using sketching to solve a linear ADMM subproblem to efficiently solve large-scale inverse-problems. However, it was previously unknown whether the resulting method would converge. Here, we formally describe the sketch-and-solve ADMM method and prove convergence.

Sketch-and-solve ADMM is obtained from algorithm 4.3.1 by setting

$$\Theta^k = \hat{H}^k + \gamma^k I, \tag{4.35}$$

where $\hat{H}^k$ is an approximation to the Hessian $H_f(\tilde{x}^k)$ at the kth iteration, and $\gamma^k \geq 0$ is a constant chosen to ensure convergence. The term $\gamma^k I$ ensures that the approximate linearization satisfies the Θ-relative smoothness condition when γ^k is chosen appropriately, as in the following lemma:

Lemma 4.7.1. Suppose f is $\hat{L}_f$-relatively smooth with respect to its Hessian H_f. Construct $\{\Theta^k\}_{k\geq 1}$ as in (4.35) and select $\gamma^k > 0$ such that $\gamma^k \geq \|E^k\| = \|H_f(\tilde{x}^k) - \hat{H}^k\|$ for every k. Then

$$f(x) \leq f(\tilde{x}^k) + \langle \nabla f(\tilde{x}^k), x - \tilde{x}^k \rangle + \frac{\hat{L}_f}{2}\|x - \tilde{x}^k\|^2_{\Theta^k}.$$

Proof. By $\hat{L}_f$ relative smoothness we have,

$$f(x) \leq f(\tilde{x}^k) + \langle \nabla f(\tilde{x}^k), x - \tilde{x}^k \rangle + \frac{\hat{L}_f}{2}\|x - \tilde{x}^k\|^2_{H_f(\tilde{x}^k)}.$$

Now,

$$\begin{aligned}\|x-\tilde{x}^k\|^2_{H_f(\tilde{x}^k)} &= \|x-\tilde{x}^k\|^2_{\hat{H}^k} + (x-\tilde{x}^k)^T(H_f(\tilde{x}^k)-\hat{H}^k)(x-\tilde{x}^k)\\ &\leq \|x-\tilde{x}^k\|^2_{\hat{H}^k} + \|E^k\|\|x-\tilde{x}^k\|^2 \leq \|x-\tilde{x}^k\|^2_{\hat{H}^k} + \gamma^k\|x-\tilde{x}^k\|^2\\ &= \|x-\tilde{x}^k\|^2_{\Theta^k}.\end{aligned}$$

The desired claim now immediately follows. □

Lemma 4.7.1 shows we can ensure relative smoothness by selecting $\gamma^k > 0$ appropriately. Assuming relative smoothness, we may invoke theorem 4.5.1 to guarantee that sketch-and-solve ADMM converges.

Corollary 4.7.2. Suppose f is $\hat{L}_f$-relatively smooth with respect to its Hessian H_f. In algorithm 4.3.1, select $\alpha = 0$ and for each k, set $\Theta^k = \hat{H}^k + \gamma^k I$ with $\gamma^k = \|E^k\| = \|\hat{H}^k - H_f(\tilde{x}^k)\|$ and $\eta^k = \hat{L}_f$. Instate assumptions 4.4.1-4.4.6. (Assumption 4.4.5 is slightly modified here as f is relatively smooth with respect to its Hessian H_f instead of Θ.) Then

$$f(\bar{x}^{t+1})+g(\bar{z}^{t+1})-p^\star \leq \frac{1}{t}\left(\hat{L}_f\left(d^2_{x^\star,H_f(\tilde{x}^1)} + \|E^1\|\|\tilde{x}^1-x^\star\|^2\right) + \frac{\rho}{2}d^2_{z^\star} + C_x\mathcal{E}_x + C_z\mathcal{E}_z + \frac{\hat{L}_f}{2}\mathcal{E}_\zeta R^2_\star\right) =: \frac{1}{t}\hat{\Gamma}$$

and

$$\|M\bar{x}^{t+1}-\bar{z}^{t+1}\| \leq \frac{2}{t}\sqrt{\frac{\hat{\Gamma}}{\rho}+d^2_{u^\star}}.$$

Here variables $\bar{x}^{t+1}$ and $\bar{z}^{t+1}$, diameters $d^2_{x^\star,H_f(\tilde{x}^1)}$, $d^2_{z^\star}$, and $d^2_{u^\star}$, and constants C_x, C_z, $\mathcal{E}_x$, $\mathcal{E}_z$, $\mathcal{E}_\zeta$, $R_\star$, and $p^\star$ are all defined as in theorem 4.5.1.

Remark 4.7.1. It is not hard to see that corollary 4.7.2 also holds for any $\gamma > 0$ if the $\{\Theta^k\}_{k\geq 1}$ satisfy

$$(1-\tau)(H^k_f+\gamma I) \preceq \Theta^k \preceq (1+\tau)(H^k_f+\gamma I),$$

for some $\tau \in (0,1)$, and $\eta \geq \frac{\hat{L}_f}{1-\tau}$. Many sketching methods can ensure this relative error condition with high probability [56] by choosing the sketch size proportional to the effective dimension of $H_f + \gamma I$ [41, 59].

Corollary 4.7.2 shows sketch-and-solve ADMM obtains an $O(1/t)$-convergence rate. The main difference between NysADMM and sketch-and-solve is the additional error term due to the use of the approximate Hessian, which results in a slightly slower convergence rate. In this sense, sketch-and-solve ADMM can be regarded as a compromise between NysADMM ($\Theta^k = H_f(\tilde{x}^k)$) and gradient descent ADMM ($\Theta^k = I$). A more accurate Hessian approximation generally yields faster convergence. Moreover, with sketch-and-solve methods, it is usually cheap to compute Θ^k and its inverse, so the x-subproblem of sketch-and-solve ADMM is easy to solve accurately and $\mathcal{E}_x \approx 0$.

4.8 Numerical experiments

We include two numerical experiments where we apply our method to solve a machine learning problem with real-world data. In this section, we numerically illustrate the convergence results highlighted in section 4.7 for the several methods highlighted in section 4.3.2 that fit into our framework: sketch-and-solve ADMM (algorithm 4.7.2), NysADMM (algorithm 4.7.1), and "gradient descent" ADMM, obtained by setting $\alpha = 0$ in (4.7). In general, we will see that curvature information, used in (exact) ADMM and NysADMM, speeds up convergence, and NysADMM enjoys the same convergence rate as ADMM with exact subproblem solves. In fact, NysADMM often linearly convergences to very high accuracy, as illustrated in Figure 4.1 and Figure 4.2.

For both experiments, we use $10,000$ random samples from the `realsim` dataset from LIBSVM [21], accessed through OpenML [105], which has $72,309$ samples and $20,958$ features. For NysADMM, we choose a sketch size $s = 50$,

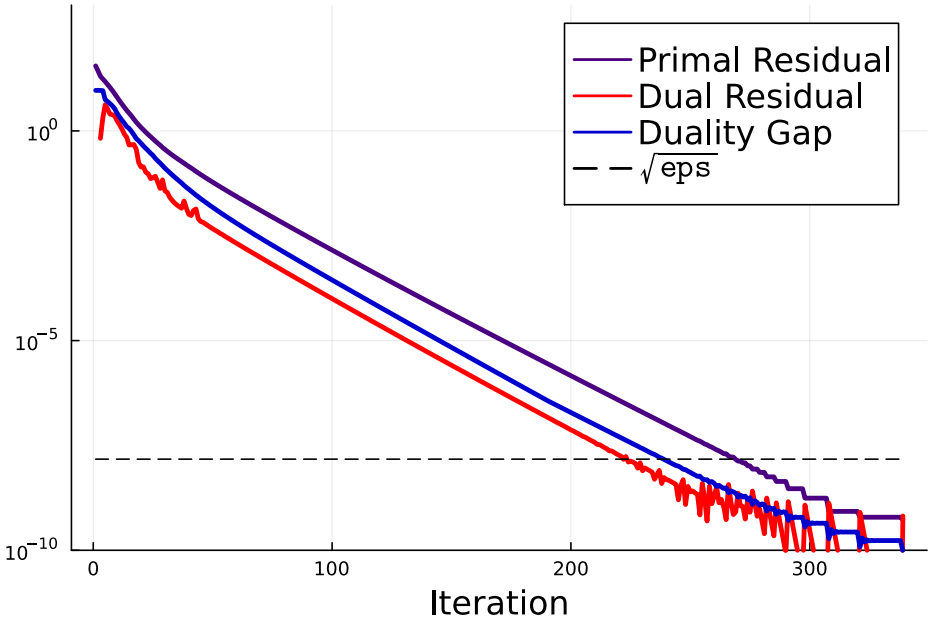

Figure 4.1: Linear convergence of NysADMM applied to lasso.

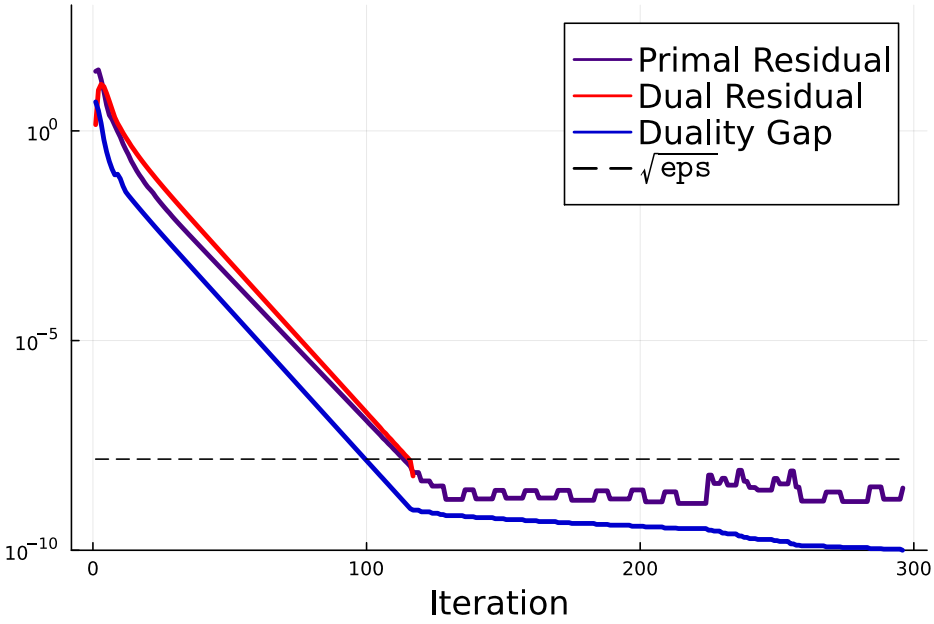

Figure 4.2: Linear convergence of NysADMM applied to logistic regression.

and for sketch-and-solve ADMM, we choose a sketch size $s = 500$. For sketch-and-solve ADMM, the parameter γ^k in (4.35) is chosen by estimating the error of the Nyström sketch using power iteration. For NysADMM, the linear system

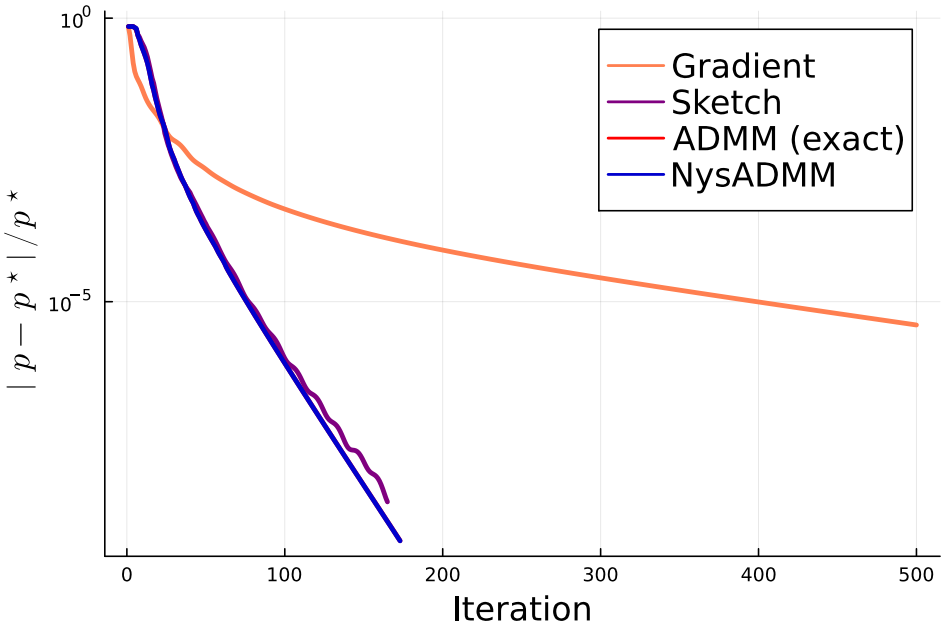

Figure 4.3: Convergence (in objective function value) of lasso regression for NysADMM, sketch-and-solve ADMM, and gradient descent ADMM.

solve tolerance at each iteration is set to be the geometric mean of the primal and dual residuals at the previous iteration, *i.e.*, $\varepsilon_x^{k+1} = \sqrt{r_p^k r_d^k}$, as in [112]. We use a bound on the duality gap as a stopping criterion, and stop when this bound (see **??** of the appendix) is under a tolerance ϵ or when the maximum number of iterations has been reached. All experiments are performed in the Julia programming language [11] on a MacBook Pro with a M1 Max processor and 64GB of RAM. To compute the "true" optimal values, we use the commercial solver Mosek [6] (with tolerances set low for high accuracy and presolve turned off to preserve the problem scaling) and the modeling language JuMP [33, 60].

4.8.1 Lasso regression

The lasso regression problem is to minimize the ℓ_2 error of a linear model with an ℓ_1 penalty on the weights:

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \gamma\|x\|_1.$$

This can be easily transformed into the form (4.1) by taking $f(x) = (1/2)\|Ax-b\|_2^2$, $g(z) = \gamma\|z\|_1$, $M = I$, and $X = Z = \mathbf{R}^n$. We set $\gamma = 0.05 \cdot \gamma_{\max}$, where $\gamma_{\max} = \|A^T b\|_\infty$ is the value above which the all zeros vector is optimal. (This can be seen from the optimality conditions, which we state in **??** of the appendix.) We stop the algorithm when the gap is less than 10^{-4}, or after 500 iterations.

The results of lasso regression are illustrated in Figure 4.3. We see that all methods enjoy a linear convergence rate; however, curvature information, used by (exact) ADMM and NysADMM, greatly speeds up convergence. The difference in convergence between NysADMM and (exact) ADMM is negligible, despite the former having a much cheaper iteration complexity due to the use of inexact linear system solves.

We also examine our method applied to the closely related but strongly convex elastic net problem,

$$\text{minimize} \quad (1/2)\|Ax - b\|_2^2 + \gamma\|x\|_1 + (\mu/2)\|x\|_2^2.$$

We use the same problem data and parameter as the Lasso experiment (and set $\mu = 1$), and we show the results in Figure 4.4. Comparing Figures 4.3 and 4.4, we clearly see the linear convergence we expect (section 4.6) for both NysADMM and exact ADMM. However, strong convexity clearly leads to an improvement in the number of iterations required to reach convergence.

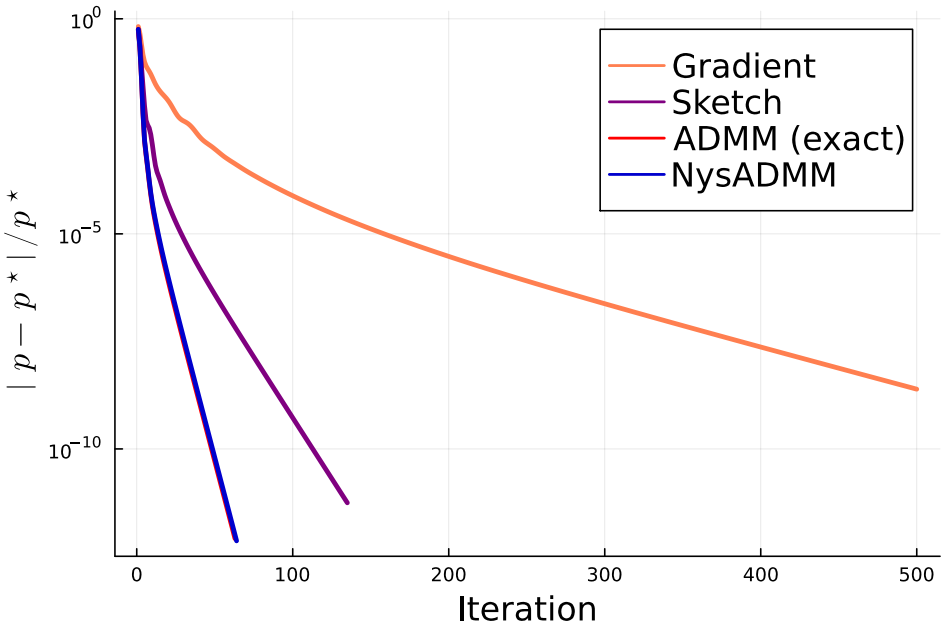

Figure 4.4: Convergence (in objective function value) of elastic net regression for NysADMM, sketch-and-solve ADMM, and gradient descent ADMM.

4.8.2 Logistic Regression

For the logistic regression problem, with data $(\tilde{a}_i, \tilde{b}_i)$, where $\tilde{b}_i \in \{\pm 1\}$. We define

$$\mathbb{P}\left(\tilde{b}_i \mid \tilde{a}_i\right) = \frac{1}{1 + \exp\left(\tilde{b}_i(\tilde{a}_i^T x)\right)} = \frac{1}{1 + \exp(a_i^T x)},$$

where we define $a_i = \tilde{b}_i \tilde{a}_i$. The loss is the negative of the log likelihood, $-\log\left(\prod_{i=1}^m \mathbb{P}(\tilde{b}_i \mid \tilde{a}_i)\right)$. Thus, the optimization problem is

$$\text{minimize} \quad \sum_{i=1}^m \log\left(1 + \exp(a_i^T x)\right) + \gamma \|x\|_1.$$

This problem can be similarly transformed into the form of (4.1) (see more details in **??** of the appendix). We set $\gamma = 0.05 \cdot \gamma_{\max}$, where $\gamma_{\max} = (1/2)\|A^T \mathbf{1}\|_\infty$ is the value above which the all zeros vector is optimal. We stop the algorithm when the gap is less than 10^{-4}, or after 500 iterations. For NysADMM, the preconditioner is re-constructed after every 20 iterations. For sketch-and-solve

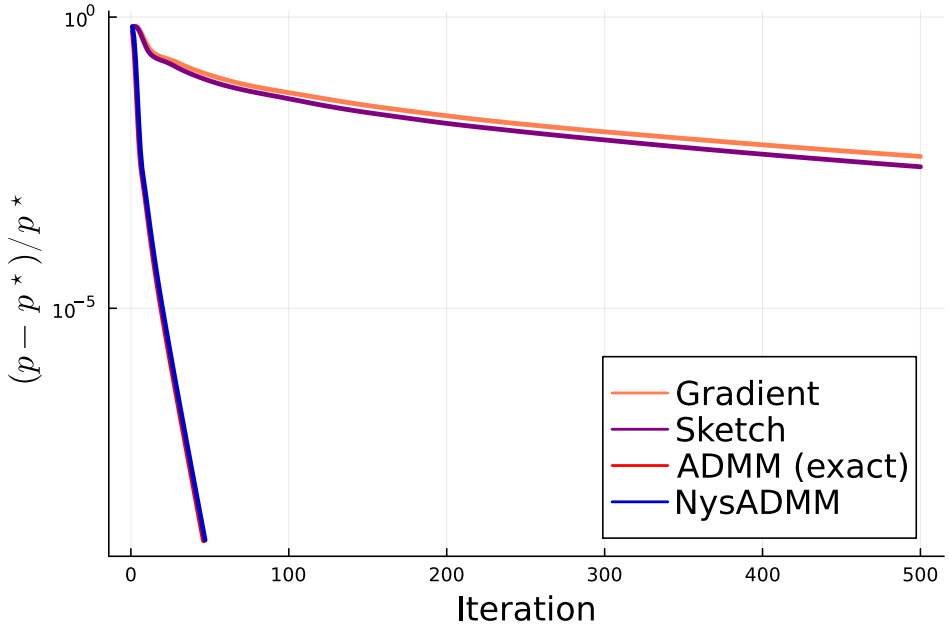

Figure 4.5: Convergence (in objective function value) of logistic regression for NysADMM, sketch-and-solve ADMM, and gradient descent ADMM.

ADMM, we re-construct the approximate Hessian at every iteration. Since the x-subproblem is not a quadratic program, we use the LBFGS [64] implementation from the `Optim.jl` package [71] to solve the x-subproblem.

The results of logistic regression are illustrated in Figure 4.5. We see that all methods enjoy a linear convergence rate; however, curvature information, used by (exact) ADMM and NysADMM, greatly speeds up convergence. The difference in convergence between NysADMM and standard ADMM is negligible, despite the former having a much cheaper iteration complexity due to the approximation of the x subproblem and use of inexact linear system solves.

4.9 Conclusion and future work

In this chapter, we have developed a novel framework, GeNI-ADMM, that allows for efficient theoretical analysis of approximate ADMM schemes and inspires the design of new practical approximate ADMM schemes. GeNI-ADMM generalizes many existing prior approximate ADMM schemes as special cases. It replaces the x-subproblem by a generalized second-order Taylor expansion in which the Hessian is replaced by a sequence of psd matrices $\{\Theta^k\}_{k\geq 1}$. The solution to this subproblem yields a generalized Newton step. Both the x and z-subproblems of GeNI-ADMM may be solved inexactly. We have established the usual $O(1/t)$-convergence for GeNI-ADMM under standard hypotheses, and linear convergence under additional hypotheses such as strong convexity. We have shown how to derive explicit rates of convergence for ADMM variants that exploit randomized numerical linear algebra using the GeNI-ADMM framework, and have provided convergence results for NysADMM and sketch-and-solve ADMM as special cases. Numerical experiments on real-world data generally show an initial sublinear phase followed by linear convergence, validating the theory and providing an interesting direction for future research.

There are several interesting directions for future work. Our analysis shows that in approximate ADMM schemes, a better approximation usually leads to a faster convergence. However, a better approximation usually makes $\{\Theta^k\}_{k=0}^{\infty}$ more complex and the resulting subproblem is hard to solve. A comprehensive study of the trade-off between theoretical convergence speed and subproblem complexity depending on the choice of $\{\Theta^k\}_{k=0}^{\infty}$ will be an important future research direction. Our numerical experiments show that GeNI-ADMM can achieve linear convergence locally in the neighborhood of the optimal so-

lution even for problems that are not strongly convex. Existing studies have proved that exact ADMM exhibits local linear convergence without strong convexity [110]. It would be interesting to establish local linear convergence for approximate ADMM schemes without strong convexity. Our theoretical analysis shows that curvature information provides an advantage in approximate ADMM schemes. This observation inspires the design of efficient ADMM-type methods for a broader range of applications, including stochastic optimization and nonconvex optimization.

APPENDIX A

APPENDIX OF CHAPTER 2

A.1 Proof of (2.14)

Since $\mathbf{u}$ and $\mathbf{y}$ are partitioned as $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2)$ and $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2)$, the state-space realization can be partitioned accordingly as

$$\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right].$$

We can express the transfer function $\hat{H}(z)$ as

$$\hat{H}(z) = \begin{bmatrix} \hat{H}_{11}(z) & \hat{H}_{12}(z) \\ \hat{H}_{21}(z) & \hat{H}_{22}(z) \end{bmatrix} = \begin{bmatrix} C_1(zI - A)^{-1}B_1 + D_{11} & C_1(zI - A)^{-1}B_2 + D_{12} \\ C_2(zI - A)^{-1}B_1 + D_{21} & C_2(zI - A)^{-1}B_2 + D_{22} \end{bmatrix}.$$

The system equations show as

$$\begin{aligned} x^{k+1} &= Ax^k + B_1u_1^k + B_2u_2^k \\ y_1^k &= C_1x^k + D_{11}u_1^k + D_{12}u_2^k \\ y_2^k &= C_2x^k + D_{21}u_1^k + D_{22}u_2^k. \end{aligned} \tag{A.1}$$

As we invert the input-output map corresponding to $\mathbf{y}_1$ and $\mathbf{u}_1$, the input of this system becomes (y_1^k, u_2^k) and the output is (u_1^k, y_2^k) at time k. From eq. (A.1), as D_{11} is invertibe, we have

$$u_1^k = -D_{11}^{-1}C_1x^k + D_{11}^{-1}y_1^k - D_{11}^{-1}D_{12}u_2^k.$$

The new system equations change to

$$\begin{aligned} x^{k+1} &= (A - B_1D_{11}^{-1}C_1)x^k + B_1D_{11}^{-1}y_1^k + (B_2 - B_1D_{11}^{-1}D_{12})u_2^k \\ u_1^k &= -D_{11}^{-1}C_1x^k + D_{11}^{-1}y_1^k - D_{11}^{-1}D_{12}u_2^k \\ y_2^k &= (C_2 - D_{21}D_{11}^{-1}C_1)x^k + D_{21}D_{11}^{-1}y_1^k + (D_{22} - D_{21}D_{11}^{-1}D_{12})u_2^k, \end{aligned}$$

which correspond to state-space realization

$$\left[\begin{array}{c|cc} A - B_1 D_{11}^{-1} C_1 & B_1 D_{11}^{-1} & B_2 - B_1 D_{11}^{-1} D_{12} \\ \hline -D_{11}^{-1} C_1 & D_{11}^{-1} & -D_{11}^{-1} D_{12} \\ (C_2 - D_{21} D_{11}^{-1} C_1) & D_{21} D_{11}^{-1} & D_{22} - D_{21} D_{11}^{-1} D_{12} \end{array}\right]. \tag{A.2}$$

To calculate the transfer function, note that

$$\begin{aligned}(zI - A + B_1 D_{11}^{-1} C_1)^{-1} &= (zI - A)^{-1} + (zI - A)^{-1} B_1 (-D_{11} - C_1 (zI - A)^{-1} B_1)^{-1} C_1 (zI - A)^{-1} \\ &= (zI - A)^{-1} - (zI - A)^{-1} B_1 \hat{H}_{11}^{-1}(z) C_1 (zI - A)^{-1}.\end{aligned}$$

We have

$$
\begin{aligned}
\hat{H}'_{11}(z) &= -D_{11}^{-1}C_1((zI-A)^{-1} - (zI-A)^{-1}B_1\hat{H}_{11}^{-1}(z)C_1(zI-A)^{-1})B_1D_{11}^{-1} + D_{11}^{-1} \\
&= -D_{11}^{-1}(\hat{H}_{11}(z) - D_{11} - (\hat{H}_{11}(z) - D_{11})\hat{H}_{11}^{-1}(z)(\hat{H}_{11}(z) - D_{11}))D_{11}^{-1} + D_{11}^{-1} \\
&= -D_{11}^{-1}(\hat{H}_{11}(z) - D_{11})(I - \hat{H}_{11}^{-1}(z)(\hat{H}_{11}(z) - D_{11}))D_{11}^{-1} + D_{11}^{-1} \\
&= -D_{11}^{-1}(\hat{H}_{11}(z) - D_{11})\hat{H}_{11}^{-1}(z) + D_{11}^{-1} \\
&= \hat{H}_{11}^{-1}(z) \\
\hat{H}'_{12}(z) &= -D_{11}^{-1}C_1((zI-A)^{-1} - (zI-A)^{-1}B_1\hat{H}_{11}^{-1}(z)C_1(zI-A)^{-1})B_2 - \hat{H}_{11}^{-1}(z)D_{12} \\
&= -D_{11}^{-1}(\hat{H}_{12}(z) - D_{12} - (\hat{H}_{11}(z) - D_{11})\hat{H}_{11}^{-1}(z)(\hat{H}_{12}(z) - D_{12})) - \hat{H}_{11}^{-1}(z)D_{12} \\
&= -D_{11}^{-1}(I - (\hat{H}_{11}(z) - D_{11})\hat{H}_{11}^{-1}(z))(\hat{H}_{12}(z) - D_{12}) - \hat{H}_{11}^{-1}(z)D_{12} \\
&= -\hat{H}_{11}^{-1}(z)(\hat{H}_{12}(z) - D_{12}) - \hat{H}_{11}^{-1}(z)D_{12} \\
&= -\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \\
\hat{H}'_{21}(z) &= C_2((zI-A)^{-1} - (zI-A)^{-1}B_1\hat{H}_{11}^{-1}(z)C_1(zI-A)^{-1})B_1D_{11}^{-1} + D_{21}\hat{H}_{11}^{-1}(z) \\
&= (\hat{H}_{21}(z) - D_{21} - (\hat{H}_{21}(z) - D_{21})\hat{H}_{11}^{-1}(z)(\hat{H}_{11}(z) - D_{11}))D_{11}^{-1} + D_{21}\hat{H}_{11}^{-1}(z) \\
&= (\hat{H}_{21}(z) - D_{21})(I - \hat{H}_{11}^{-1}(z)(\hat{H}_{11}(z) - D_{11}))D_{11}^{-1} + D_{21}\hat{H}_{11}^{-1}(z) \\
&= (\hat{H}_{21}(z) - D_{21})\hat{H}_{11}^{-1}(z) + D_{21}\hat{H}_{11}^{-1}(z) \\
&= \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z) \\
\hat{H}'_{22}(z) &= \hat{H}_{22}(z) - (\hat{H}_{21}(z) - D_{21})\hat{H}_{11}^{-1}(z)(\hat{H}_{12}(z) - D_{12}) - D_{21}\hat{H}_{11}^{-1}(z)(\hat{H}_{12}(z) - D_{12}) \\
&\quad - (\hat{H}_{21}(z) - D_{21})\hat{H}_{11}^{-1}(z)D_{12} - D_{21}\hat{H}_{11}^{-1}(z)D_{12} \\
&= \hat{H}_{22}(z) - \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z).
\end{aligned}
$$

Thus, we get the desired results as eq. (2.14).

A.2 Proof of proposition 2.6.1

Given an algorithm $\mathcal{A}$ with state-space realization (A, B, C, D), the relation between the input u and output y can be expressed as

$$y^k = C(A)^k x^0 + \sum_{j=0}^{k-1} C(A)^{k-(j+1)} B u^j + D u^k. \tag{A.3}$$

Relation eq. (A.3) is obtained by eq. (2.5), without the assumption that $x^0 = 0$. The output y^k is the sum of $C(A)^k x^0$, which is due to the initial condition x^0, and $\sum_{j=0}^{k-1} C(A)^{k-(j+1)} B u^j + D u^k$, which is due to the inputs $\{u^0, \ldots, u^k\}$. The linearity of $\mathcal{A}$ ($\mathcal{A}$ is treated as a linear system) allows the decomposition of two contributions and they can be studied separately:

$$\text{(total response)} = \underbrace{\text{(zero input response)}}_{\text{set } u^k = 0 \text{ for } k \geq 0} + \underbrace{\text{(zero state response)}}_{\text{set } x^0 = 0}.$$

Since we would like to characterize $\mathcal{A}$ with its input-output map, we can only focus on the zero state response, which allows us to avoid details about initialization. With the definition of impulse response,

$$H^k = \begin{cases} D & k = 0 \\ C(A)^{k-1} B & k \geq 1 \end{cases},$$

we can express the zero state response as

$$y^k = H^k u^0 + H^{k-1} u^1 + \cdots + H^1 u^{k-1} + H^0 u^k.$$

Transfer function provides a compact form to represent the H^k series by taking the z-transform. The transfer function $H(z)$ shows as follows,

$$\hat{H}(z) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = D + \sum_{k=1}^{\infty} C(A)^{k-1} B z^{-k} = C(zI - A)^{-1} B + D.$$

Therefore, we know that algorithm $\mathcal{A}$ is uniquely characterized by its input-output map, and thus uniquely characterized by its impulse response and transfer function.

From the definition of oracle equivalence, oracle-equivalent algorithms have identical sequences of output $\mathbf{y}$ for each possible sequence of input $\mathbf{u}$ if initialized properly. Here $\mathbf{y} = \{u^k\}_{k=0}^{\infty}$ and $\mathbf{u} = \{y^k\}_{k=0}^{\infty}$. Thus, they must have identical impulse responses and consequently identical transfer functions. This completes the proof.

A.3 Proof of proposition 2.7.1

Algorithm A.3.1 General form of algorithm $\mathcal{A}$

for $k = 0, 1, 2, \ldots$ **do**

$$
\begin{aligned}
x_1^{k+1} &= L_1(x_1^k, \ldots, x_m^k) \\
x_2^{k+1} &= L_2(x_1^{k+1}, x_2^k, \ldots, x_m^k) \\
&\vdots \\
x_i^{k+1} &= L_i(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^k, \ldots, x_m^k, u_1^{k+1}) \\
&\vdots \\
x_{\tilde{i}}^{k+1} &= L_{\tilde{i}}(x_1^{k+1}, \ldots, x_{\tilde{i}-1}^{k+1}, x_{\tilde{i}}^k, \ldots, x_m^k, u_n^{k+1}) \\
&\vdots \\
x_m^{k+1} &= L_m(x_1^{k+1}, \ldots, x_{m-1}^{k+1}, x_m^k)
\end{aligned}
$$

end for

First, we prove that cyclic permutation implies shift equivalence. Without loss of generality, we can express algorithm $\mathcal{A}$ in the general form as algorithm A.3.1. Since $\mathcal{A}$ is an linear algorithm, $L_1, \ldots, L_m$ are linear functions. Given an initialization $\{x_1^0, \ldots, x_m^0\}$, $\mathcal{A}$ generates state sequence $(x_1^k, \ldots, x_m^k)_{k\geq 0}$, input sequence $(u_1^k, \ldots, u_n^k)_{k\geq 1}$, and output sequence $(y_1^k, \ldots, y_n^k)_{k\geq 1}$. The ith update equation is the

first update equation that contains an oracle call, corresponding to u_1^k and y_1^k. The $\tilde{i}$th update equation is the last update equation that contains an oracle call, corresponding to u_n^k and y_n^k. The outputs are also linear functions of the states. Specifically, we have

$$\begin{aligned} y_1^k =& Y_1(x_1^{k+1}, \ldots, x_{i-1}^{k+1}, x_i^k, \ldots, x_m^k), \quad u_1^k = \phi_1(y_1^k) \\ & \vdots \\ y_n^k =& Y_n(x_1^{k+1}, \ldots, x_{\tilde{i}-1}^{k+1}, x_{\tilde{i}}^k, \ldots, x_m^k), \quad u_n^k = \phi_n(y_n^k). \end{aligned}$$

Functions $Y_1, \ldots, Y_n$ are linear functions and $\phi_1, \ldots, \phi_n$ denote the oracle calls. Without loss of generality, suppose permutation $\tilde{\pi} = (\tilde{l}+1, \ldots, m, 1, \ldots, \tilde{l})$ with $1 < \tilde{l} < m$.

First case. Suppose the new order of oracle calls within one iteration is a cyclic permutation π of (n) (not identical to (n)). Without loss of generality, suppose $\pi = (j+1, \ldots, n, 1, \ldots, j)$ with $1 < j < n$, the jth oracle call corresponds to the $\tilde{j}$th update equation, and the $j+1$th oracle call corresponds to the $\tilde{p}$th update equation. By definition, we have $i \le \tilde{j} < \tilde{l}+1 \le \tilde{p} \le \tilde{i}$. At the first time step $k=1$, the first input is u_1^1 and the first output is y_1^1, and the $j+1$th input and output are u_{j+1}^1 and y_{j+1}^1. We have

$$\begin{aligned} y_1^1 &= Y_1(x_1^1, \ldots, x_{i-1}^1, x_i^0, \ldots, x_m^0) \\ x_{\tilde{l}+1}^1 &= L_{\tilde{l}+1}(x_1^1, \ldots, x_{\tilde{l}}^1, x_{\tilde{l}+1}^0, \ldots, x_m^0) \\ y_{j+1}^1 &= Y_1(x_1^1, \ldots, x_{\tilde{p}-1}^1, x_{\tilde{p}}^0, \ldots, x_m^0). \end{aligned}$$

Here without loss of generality, suppose the $\tilde{l}+1$th update equation does not contain an oracle call. In other words, $\tilde{j} < \tilde{l}+1 < \tilde{p}$. By definition, $\mathcal{B}$ calls the update equations in the order $\tilde{\pi}$. At the first time step, the $\tilde{l}+1$th update equation is first called. If $\mathcal{B}$ is suitably initialized with states $\{x_1^1, \ldots, x_{\tilde{l}}^1, x_{\tilde{l}+1}^0, \ldots, x_m^0\}$, it will generate state sequence $(x_{\tilde{l}+1}^k, \ldots, x_m^k, x_1^{k+1}, \ldots, x_{\tilde{l}}^{k+1})_{k\ge 0}$, input sequence

$(u^k_{j+1}, \ldots, u^k_n, u^{k+1}_1, \ldots, u^{k+1}_j)_{k\geq 1}$, and output sequence $(y^k_{j+1}, \ldots, y^k_n, y^{k+1}_1, \ldots, y^{k+1}_j)_{k\geq 1}$. The input and output sequences of $\mathcal{A}$ and $\mathcal{B}$ match up to prefixes $(u^1_1, \ldots, u^1_j)$ and $(y^1_1, \ldots, y^1_j)$ respectively. Therefore, $\mathcal{A}$ and $\mathcal{B}$ are shift-equivalent.

Second case. Suppose the order of oracle calls within one iteration remain unchanged (identical to (n)). By definition, we have $1 < \tilde{l} + 1 \leq i$. We have

$$x^1_{\tilde{l}+1} = L_{\tilde{l}+1}(x^1_1, \ldots, x^1_{\tilde{l}}, x^0_{\tilde{l}+1}, \ldots, x^0_m)$$
$$y^1_1 = Y_1(x^1_1, \ldots, x^1_{i-1}, x^0_i, \ldots, x^0_m).$$

Here without loss of generality, suppose the $\tilde{l}+1$th update equation does not contain an oracle call. In other words, $\tilde{l}+1 < i$. By definition, $\mathcal{B}$ calls the update equations in the order $\tilde{\pi}$. At the first time step, the $\tilde{l}+1$th update equation is first called. If $\mathcal{B}$ is suitably initialized with states $\{x^1_1, \ldots, x^1_{\tilde{l}}, x^0_{\tilde{l}+1}, \ldots, x^0_m\}$, it will generate state sequence $(x^k_{\tilde{l}+1}, \ldots, x^k_m, x^{k+1}_1, \ldots, x^{k+1}_{\tilde{l}})_{k\geq 0}$. The input and output sequences remain unchanged. Therefore, $\mathcal{A}$ and $\mathcal{B}$ are oracle-equivalent. Meanwhile, since oracle equivalence can be regarded as a special case of shift equivalence, $\mathcal{A}$ and $\mathcal{B}$ are also shift-equivalent.

Next, we prove that shift equivalence implies cyclic permutation. Suppose algorithms $\mathcal{A}$ and $\mathcal{B}$ are shift-equivalent. If they are also oracle-equivalent, then they can be written using the same set of update equations, which is trivially related by a cyclic permutation (where the permutation is the identity). Now suppose they are not oracle-equivalent. Let $(u^k_1, \ldots, u^k_m)_{k\geq 0}$ and $(\tilde{u}^k_1, \ldots, \tilde{u}^k_m)_{k\geq 0}$ be the input sequences for $\mathcal{A}$ and $\mathcal{B}$. The input sequences match up to a non-empty prefix. Without loss of generality, suppose the length of this prefix is q: that is, if we remove a prefix of length q from the input sequence of $\mathcal{A}$, then $\mathcal{A}$ and $\mathcal{B}$ have the same input sequence. Recall we only need to consider the case $q < m$. If $q > m$, it is equivalent to consider $q = q \bmod m$. Comparing

the input sequences of $\mathcal{A}$ and $\mathcal{B}$, and using the prefix length q, we can write $(u_{q+1}^{k-1}, \ldots, u_m^{k-1}, u_1^k, \ldots, u_q^k) = (\tilde{u}_1^k, \ldots, \tilde{u}_m^k)$ for $k \geq 1$. The output sequences of $\mathcal{A}$ and $\mathcal{B}$ have the same relation. Therefore, $\mathcal{B}$ and this shifted version of $\mathcal{A}$ are oracle equivalent, and so we can write $\mathcal{B}$ and this shifted version of $\mathcal{A}$ using the same set of update equations. To undo the shift of $\mathcal{A}$, we simply move the first q update equations to the end of the algorithm.

A.4 Proof of proposition 2.7.3

The state-space realization of $\mathcal{A}$ corresponds to the state update equations

$$\begin{aligned} x^{k+1} &= Ax^k + B_1\bar{u}_1^k + B_2\bar{u}_2^k \\ \bar{y}_1^k &= C_1x^k + D_{11}\bar{u}_1^k + D_{12}\bar{u}_2^k \\ \bar{y}_2^k &= C_2x^k + D_{21}\bar{u}_1^k + D_{22}\bar{u}_2^k. \end{aligned} \tag{A.4}$$

Sufficiency. We will derive the state-space realization of $P_\pi\mathcal{A}$:

$$\left[\begin{array}{cc:cc} A & B_1 & 0 & B_2 \\ 0 & 0 & I & 0 \\ \hdashline C_1A & C_1B_1 & D_{11} & C_1B_2 \\ C_2 & D_{21} & 0 & D_{22} \end{array}\right].$$

To verify this realization is correct, we can write the system equations of this state-space realization as

$$\begin{aligned} x^{k+1} &= Ax^k + B_1\bar{u}_1^k + B_2\bar{u}_2^k \\ \bar{u}_1^{k+1} &= \bar{u}_1^{k+1} \\ \bar{y}_1^{k+1} &= C_1Ax^k + C_1B_1\bar{u}_1^k + D_{11}\bar{u}_1^{k+1} + C_1B_2\bar{u}_2^k \\ \bar{y}_2^k &= C_2x^k + D_{21}\bar{u}_1^k + D_{12}\bar{u}_2^k. \end{aligned} \tag{A.5}$$

Note that equations eq. (A.5) are the results of equations eq. (A.4) after applying permutation π. As we perform cyclic permutation π, within each iteration, the update order of the oracles is shifted as $(j+1,\ldots,n,1,\ldots,j)$, indicating oracles $(j+1,\ldots,n)$ are updated before $(1,\ldots,j)$. Further, the input and output sequences within one iteration at time step k become $(\bar{u}_2^k, \bar{u}_1^{k+1})$ and $(\bar{y}_2^k, \bar{y}_1^{k+1})$. From the state-space realization, we may compute the transfer function as

$$\hat{H}_{\mathcal{B}}(z) = \begin{bmatrix} C_1(zI-A)^{-1}B_1 + D_{11} & zC_1(zI-A)^{-1}B_2 \\ C_2(zI-A)^{-1}B_1/z + D_{21}/z & C_2(zI-A)^{-1}B_2 + D_{22} \end{bmatrix} = \begin{bmatrix} \hat{H}_{11}(z) & z\hat{H}_{12}(z) \\ \hat{H}_{21}(z)/z & \hat{H}_{22}(z) \end{bmatrix}. \tag{A.6}$$

To arrive at eq. (A.6), we have used the fact that $D_{12} = 0$ by assumption, and

$$\left(zI - \begin{bmatrix} A & B_1 \\ 0 & 0 \end{bmatrix} \right)^{-1} = \begin{bmatrix} (zI-A)^{-1} & \frac{1}{z}(zI-A)^{-1}B_1 \\ 0 & \frac{1}{z}I \end{bmatrix}.$$

Necessity is provided by proposition 2.6.1. Equivalent algorithms must have identical transfer functions. Thus, if we find an algorithm and its transfer function is the same as eq. (2.21), it must be equivalent to $\mathcal{B}$.

A.5 Discussions on permutation and its generalization

To take a revisit of proposition 2.7.3, it can be found that as algorithm $\mathcal{A}$ is permuted to make the order of oracle calls within one iteration as $(j+1,\ldots,n,1,\ldots,j)$ from $(1,\ldots,n)$, the resulting transfer function is exactly the same as adding a one-step time delay to channels (oracles) $(1,\ldots,j)$ according to results in control theory. Another interpretation of adding a one-step time delay comes from the system equations eq. (A.5). We can see that the input and output corresponding to channels (oracles) $(1,\ldots,j)$ are the input and output for

the next time step $\bar{u}_1^{k+1}$ and $\bar{y}_1^{k+1}$, however, the input and output of channels (oracles) $(j{+}1, \ldots, n)$ are still the ones for the current time step $\bar{u}_2^k$ and $\bar{y}_2^k$. Intrinsically, after cyclic permutation, the intrinsic update order of oracles does not change, but a one-step time delay is added to the oracles that we would like to update latterly.

Using the idea of time delay, we can generalize algorithm permutation as adding any step of time delay to any channel (oracle) of an algorithm. Suppose we add time delay to oracle i of algorithm $\mathcal{A}$ by d_i for any $i \in (n)$, where d_i can be any integer, the resulting algorithm $\mathcal{B}$ has transfer function $\hat{H}_{\mathcal{B}}(z)$ as

$$\hat{H}_{\mathcal{B}}(z) = \begin{bmatrix} z^{d_1} & & & \\ & z^{d_2} & & \\ & & \ddots & \\ & & & z^{d_n} \end{bmatrix} \hat{H}_{\mathcal{A}}(z) \begin{bmatrix} z^{-d_1} & & & \\ & z^{-d_2} & & \\ & & \ddots & \\ & & & z^{-d_n} \end{bmatrix}, \tag{A.7}$$

where $\hat{H}_{\mathcal{A}}(z)$ is the transfer function of $\mathcal{A}$.

To be more specific, suppose we add time delay d_i to oracle i for algorithm $\mathcal{A}$, $h_{\mathcal{A}}^{kl}(z)$ with $1 \le k \le n$ and $1 \le l \le n$ denotes the entry of $\hat{H}_{\mathcal{A}}(z)$. The transfer function of the resulting algorithm $\mathcal{B}$ can be expressed entrywise as

$$h_{\mathcal{B}}^{kl}(z) = \begin{cases} h_{\mathcal{A}}^{ii}(z) & k = i \; l = i \\ h_{\mathcal{A}}^{il}(z) z^{d_i} & k = i \; l \neq i \\ h_{\mathcal{A}}^{ki}(z) z^{-d_i} & k \neq i \; l = i \\ h_{\mathcal{A}}^{kl}(z) & k \neq i \; l \neq i \end{cases} \tag{A.8}$$

In this way, we know that proposition 2.7.3 is a special case of eq. (A.7) with $d_1 = \cdots = d_j = 1$.

However, there are restrictions so that we cannot add any arbitrary step of time delay to any oracle. From section 2.4.3, transfer functions are rational (ma-

trix) functions with respect to z. Further, the rational functions must be proper in order to make the transfer function realizable. From eq. (A.8), as we add time delay d_i to oracle i for $\mathcal{A}$, the off-diagonal entries in the ith row of $\hat{H}_{\mathcal{A}}(z)$ are multiplied by z^{d_i} and the off-diagonal entries in the ith column of $\hat{H}_{\mathcal{A}}(z)$ are multiplied by z^{-d_i} while the ith diagonal entry remains unchanged. From the perspective of relative degrees, as relative degree is the difference between the degree of denominator and the degree of numerator, the relative degrees of the off-diagonal entries in the ith row are decreased by d_i but the relative degrees of the off-diagonal entries in the ith column are increased by d_i. Suppose the smallest relative degree among the off-diagonal entries in the ith row is r_i, then d_i must satisfy $d_i \leq r_i$ to maintain properness of the resulting off-diagonal entries in the ith row. Similarly, suppose the smallest relative degree among the off-diagonal entries of the ith column is c_i, then d_i must satisfy $-d_i \leq c_i$ to maintain properness of the resulting off-diagonal entries in the ith column. In other words, we can add time delay d_i to oracle i only if $-c_i \leq d_i \leq r_i$. Otherwise, at least one off-diagonal entry in the ith row or the ith column is no longer proper, leading to an invalid transfer function.

For any algorithm with state-space realization (A, B, C, D), the transfer function is calculated by $C(zI-A)^{-1}B+D$. Term $C(zI-A)^{-1}B$ is a strictly proper (matrix) function, where strictly proper means that the degree of z in the numerator polynomial is strictly less than the degree of z in the denominator polynomial. Thus, for any nonzero entry of D, the corresponding entry in the transfer function has relative degree zero. Take a revisit of cyclic permutation, for any causal algorithm, the entries above diagonal of the D matrix must be zero, especially after necessary reordering. Thus, the entries above diagonal in the transfer function have strictly positive relative degrees. This implies that any cyclic permutation

of an algorithm always exists. Note that before performing cyclic permutation, we are required to reorder the state-space realization if needed.

Reconsider algorithms 2.7.5 and 2.7.6, in eq. (2.24) and eq. (2.25), comparing $\hat{H}_{10}(z)$ to $\hat{H}_{11}(z)$, we add a one-step time delay to the first channel. Term $\frac{1}{z-1}$ in $\hat{H}_{10}(z)$ is multiplied by z and term $\frac{2z-1}{z-1}$ is multiplied by z^{-1}. Further, the off-diagonal entry in the first row of $\hat{H}_{10}(z)$ has relative degree 1 and the off-diagonal entry in the first column of $\hat{H}_{10}(z)$ has relative degree 0. Thus, we can only add time delay $d_1 = 1$ to the first oracle of algorithm 2.7.5, as $0 \leq d_1 \leq 1$ to maintain properness.

A.6 Proof of proposition 2.7.4

Partition the oracle calls of algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{X}$ into two (nonlinear) oracles ϕ_1 and ϕ_2. Formally, write the update equations as

$$\begin{aligned} x^\star &= Ax^\star + B_1\bar{u}_1^\star + B_2\bar{u}_2^\star \\ \bar{y}_1^\star &= C_1x^\star + D_{11}\bar{u}_1^\star + D_{12}\bar{u}_2^\star \\ \bar{y}_2^\star &= C_2x^\star + D_{21}\bar{u}_1^\star + D_{22}\bar{u}_2^\star \\ u_1^\star &= \phi_1(y_1^\star) \\ u_2^\star &= \phi_2(y_2^\star). \end{aligned} \tag{A.9}$$

Here the cyclic permutation π swaps the first and second set of oracle calls. Then the cyclic permutation $P_\pi\mathcal{A}$ converges to fixed point $(\bar{y}_2^\star, \bar{y}_1^\star, \bar{u}_2^\star, \bar{u}_1^\star, x^\star)$. To verify this, since $D_{12} = 0$, plugging in the fixed point conditions eq. (A.9) to the system

equations of the shifted algorithm eq. (A.5), we have

$$
\begin{aligned}
x^\star &= Ax^\star + B_1\bar{u}_1^\star + B_2\bar{u}_2^\star \\
\bar{u}_1^\star &= \bar{u}_1^\star \\
\bar{y}_1^\star &= C_1Ax^\star + C_1B_1\bar{u}_1^\star + D_{11}\bar{u}_1^\star + C_1B_2\bar{u}_2^\star = C_1x^\star + D_{11}\bar{u}_1^\star \\
\bar{y}_2^\star &= C_2x^\star + D_{21}\bar{u}_1^\star + D_{12}\bar{u}_2^\star \\
u_1^\star &= \phi_1(y_1^\star) \\
u_2^\star &= \phi_2(y_2^\star).
\end{aligned}
$$

This completes the proof.

A.7 Proof of shift-equivalence of DR and ADMM continued

Suppose the oracles for both DR (algorithm 2.7.5) and ADMM (algorithm 2.7.6) are subgradients of f and g. Oracles prox and argmin can be expanded as inclusions involving subgradients. The update equations of DR and ADMM can be rewritten into formations of algorithms A.7.1 and A.7.2 respectively. Note that the update equations involving subgradients are inclusions.

Algorithm A.7.1 DR

for $k = 0, 1, 2, \ldots$ **do**
$x_1^{k+1} \in x_3^k - t\partial f(x_1^{k+1})$
$x_2^{k+1} \in 2x_1^{k+1} - x_3^k - tL^T\partial g(Lx_2^{k+1})$
$x_3^{k+1} = x_3^k + x_2^{k+1} - x_1^{k+1}$
end for

Algorithm A.7.2 ADMM

for $k = 0, 1, 2, \ldots$ **do**
$\xi_1^{k+1} \in L\xi_2^k - L\xi_3^k - \frac{1}{\rho}LL^T\partial g(L\xi_1^{k+1})$
$\xi_2^{k+1} \in L^{-1}\xi_1^{k+1} + \xi_3^k - \frac{1}{\rho}\partial f(\xi_2^{k+1})$
$\xi_3^{k+1} = \xi_3^k + L^{-1}\xi_1^{k+1} - \xi_2^{k+1}$
end for

We still assume $\rho = 1/t$ in ADMM. The transfer functions are computed as

$\hat{H}_{18}(z)$ and $\hat{H}_{19}(z)$ respectively. Note that $\hat{H}_{19}(z)$ is not written in the causal order.

$$\hat{H}_{18}(z) = \left[\begin{array}{ccc|cc} 0 & 0 & I & -tI & 0 \\ 0 & 0 & I & -2tI & -tL^T \\ 0 & 0 & I & -tI & -tL^T \\ \hline 0 & 0 & I & -tI & 0 \\ 0 & 0 & L & -2tL & -tLL^T \end{array}\right] = \left[\begin{array}{cc} -\frac{tz}{z-1}I & -\frac{t}{z-1}L^T \\ \frac{t-2tz}{z-1}L & -\frac{tz}{z-1}LL^T \end{array}\right]$$

$$\hat{H}_{19}(z) = \left[\begin{array}{ccc|cc} 0 & L & -L & 0 & -tLL^T \\ 0 & I & 0 & -tI & -tL^T \\ 0 & 0 & 0 & tI & 0 \\ \hline 0 & I & 0 & -tI & -tL^T \\ 0 & L & -L & 0 & -tLL^T \end{array}\right] = \left[\begin{array}{cc} -\frac{tz}{z-1}I & -\frac{tz}{z-1}L^T \\ \frac{t-2tz}{z(z-1)}L & -\frac{tz}{z-1}LL^T \end{array}\right]$$

From propositions 2.7.1 and 2.7.3, we know that they are still shift-equivalent.

A.8 Proof of proposition 2.8.1

Sufficiency. The update equations of $\mathcal{B}$ can be written as

$$\begin{aligned} x_1^k &= Ax_{\mathcal{B}}^k + Bu_1^k \\ y_1^k &= Cx_{\mathcal{B}}^k + Du_1^k \\ x_{\mathcal{B}}^{k+1} &= Ax_1^k + Bu_2^k \\ y_2^k &= Cx_1^k + Du_2^k, \end{aligned}$$

where x_1^k is an intermediate state. Eliminating the intermediate state x_1^k, we arrive at the new update equations:

$$\begin{aligned} x_{\mathcal{B}}^{k+1} &= A^2x_{\mathcal{B}}^k + ABu_1^k + Bu_2^k \\ y_1^k &= Cx_{\mathcal{B}}^k + Du_1^k \\ y_2^k &= CAx_{\mathcal{B}}^k + CBu_1^k + Du_2^k. \end{aligned}$$

The corresponding state-space realization has transfer function

$$\left[\begin{array}{c|cc} A^2 & AB & B \\ \hline C & D & 0 \\ CA & CB & D \end{array}\right] = \begin{bmatrix} C(zI - A^2)^{-1}AB + D & C(zI - A^2)^{-1}B \\ CA(zI - A^2)^{-1}AB + CB & CA(zI - A^2)^{-1}B + D \end{bmatrix}.$$

Necessity is provided by proposition 2.6.1 since the transfer function uniquely characterizes an equivalence class of algorithms.

A.9 Proof of proposition 2.8.3

Suppose the oracles of algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{X}$ can be represented as $\phi : \mathcal{X} \to \mathcal{X}$. Since $\mathcal{A}$ converges to fixed point $(y^\star, u^\star, x^\star)$, it satisfies

$$x^\star = Ax^\star + Bu^\star$$

$$y^\star = Cx^\star + Du^\star$$

$$u^\star = \phi(y^\star).$$

Therefore, we have

$$
\begin{aligned}
x^\star &= Ax^\star + Bu^\star \\
&= A(Ax^\star + Bu^\star) + Bu^\star \\
&= A^2x^\star + ABu^\star + Bu^\star \\
&= \dots \\
&= A^{n-1}x^\star + A^{n-2}Bu^\star + \cdots + ABu^\star + Bu^\star \\
&= A^nx^\star + A^{n-1}Bu^\star + \cdots + ABu^\star + Bu^\star \\
y^\star &= Cx^\star + Du^\star \\
&= C(Ax^\star + Bu^\star) + Du^\star \\
&= CAx^\star + CBu^\star + Du^\star \\
&= \dots \\
&= CA^{n-1}x^\star + CA^{n-2}Bu^\star + \cdots + CBu^\star + Du^\star.
\end{aligned}
$$

With eq. (2.27), we have

$$
\begin{aligned}
x^\star &= A^nx^\star + A^{n-1}Bu^\star + \cdots + ABu^\star + Bu^\star \\
y^\star &= Cx^\star + Du^\star \\
y^\star &= CAx^\star + CBu^\star + Du^\star \\
&\vdots \\
y^\star &= CA^{n-1}x^\star + CA^{n-2}Bu^\star + \dots CBu^\star + Du^\star,
\end{aligned}
$$

which indicates that $\mathcal{A}^n$ converges to fixed point $(y', u', x^\star)$ with $y' = y^\star \bigotimes \mathbb{1}^n$ and $u' = u^\star \bigotimes \mathbb{1}^n$.

A.10 Proof of proposition 2.9.3

Without loss of generality, let the permutation matrix equal to the identity as proposition 2.9.1. To simplify the notations, let

$$\left[\begin{array}{c|cc} A & B[[n],\kappa] & B[[n],\bar{\kappa}] \\ \hline C[\kappa,[n]] & D[\kappa] & D[\kappa,\bar{\kappa}] \\ C[\bar{\kappa},[n]] & D[\bar{\kappa},\kappa] & D[\bar{\kappa}] \end{array}\right] = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right],$$

$$\begin{bmatrix} \hat{H}[\kappa](z) & \hat{H}[\kappa,\bar{\kappa}](z) \\ \hat{H}[\bar{\kappa},\kappa](z) & \hat{H}[\bar{\kappa}](z) \end{bmatrix} = \begin{bmatrix} \hat{H}_{11}(z) & \hat{H}_{12}(z) \\ \hat{H}_{21}(z) & \hat{H}_{22}(z) \end{bmatrix} = \begin{bmatrix} C_1(zI-A)^{-1}B_1 + D_{11} & C_1(zI-A)^{-1}B_2 + D_{12} \\ C_2(zI-A)^{-1}B_1 + D_{21} & C_2(zI-A)^{-1}B_2 + D_{22} \end{bmatrix}.$$

In this way, $(y[\kappa]^\star, y[\bar{\kappa}]^\star, u[\kappa]^\star, u[\bar{\kappa}]^\star, x^\star)$ can be written as $(y_1^\star, y_2^\star, u_1^\star, u_2^\star, x^\star)$, and $(u[\kappa]^\star, y[\bar{\kappa}]^\star, y[\kappa]^\star, u[\bar{\kappa}]^\star, x^\star)$ can be written as $(u_1^\star, y_2^\star, y_1^\star, u_2^\star, x^\star)$.

Partition the oracle calls of algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{X}$ into two nonlinear oracles ϕ_1 and ϕ_2. Oracle ϕ_1 corresponds to the oracle calls in set κ, and ϕ_2 corresponds to the remaining oracle calls. Since $\mathcal{A}$ converges to fixed point $(y_1^\star, y_2^\star, u_1^\star, u_2^\star, x^\star)$, it satisfies

$$\begin{aligned} x^\star &= Ax^\star + B_1u_1^\star + B_2u_2^\star \\ y_1^\star &= C_1x^\star + D_{11}u_1^\star + D_{12}u_2^\star \\ y_2^\star &= C_2x^\star + D_{21}u_1^\star + D_{22}u_2^\star \\ u_1^\star &= \phi_1(y_1^\star) \\ u_2^\star &= \phi_2(y_2^\star). \end{aligned}$$

The state-space realization of $C_\kappa\mathcal{A}$ is the same as eq. (A.2). Note that D_{11} is

invertible, we have

$$
\begin{aligned}
x^\star &= Ax^\star + B_2u_2^\star + B_1u_1^\star \\
&= Ax^\star + B_2u_2^\star + B_1(-D_{11}^{-1}C_1x^\star + D_{11}^{-1}y_1^\star - D_{11}^{-1}D_{12}u_2^\star) \\
&= (A - B_1D_{11}^{-1}C_1)x^\star + B_1D_{11}^{-1}y_1^\star + (B_2 - B_1D_{11}^{-1}D_{12})u_2^\star \\
u_1^\star &= -D_{11}^{-1}C_1x^\star + D_{11}^{-1}y_1^\star - D_{11}^{-1}D_{12}u_2^\star \\
y_2^\star &= C_2x^\star + D_{22}u_2^\star + D_{21}u_2^\star \\
&= C_2x^\star + D_{22}u_2^\star + D_{21}(D_{11}^{-1}y_1^\star - D_{11}^{-1}C_1x^\star - D_{11}^{-1}D_{12}u_2^\star) \\
&= (C_2 - D_{21}D_{11}^{-1}C_1)x^\star + D_{21}D_{11}^{-1}y_1^\star + (D_{22} - D_{21}D_{11}^{-1}D_{12})u_2^\star \\
y_1^\star &= \phi_1^{-1}(u_1^\star) \\
u_2^\star &= \phi_2(y_2^\star).
\end{aligned}
$$

Oracle ϕ_1^{-1} is the inverse oracle of oracle ϕ_1. Therefore, we get the desired results that algorithm $C_\kappa\mathcal{A}$ converges to fixed point $(u_1^\star, y_2^\star, y_1^\star, u_2^\star, x^\star)$.

A.11 Commutativity between conjugation and cyclic permutation

Proposition A.11.1. Conjugation and cyclic permutation commute.

Proof. Given an algorithm $\mathcal{A}$ with transfer function $\hat{H}(z)$. Suppose κ is a subset of the oracles of $\mathcal{A}$, D_κ is invertible, and $\pi = (m+1, \ldots, n, 1, \ldots, m)$ is an arbitrary cyclic permutation of the oracles of $\mathcal{A}$. We will show that the transfer functions of $C_\kappa P_\pi\mathcal{A}$ and $P_\pi C_\kappa\mathcal{A}$ are identical.

Suppose $\hat{H}^\star(z)$ is the transfer function of $P_\pi\mathcal{A}$, the results in proposition 2.7.3

can be written as

$$\hat{H}^{\star}(z) = Q\hat{H}(z)Q^{-1}. \tag{A.10}$$

Here Q is a diagonal matrix where the first m diagonal entries are all z and the rest of the diagonal entries are all ones. We will use the same settings and notations as proposition 2.9.1 to express changes in transfer function of conjugation C_κ. Without loss of generality, the transfer function $\hat{H}'(z)$ of $C_\kappa\mathcal{A}$ satisfies

$$\hat{H}'(z) = \begin{bmatrix} \hat{H}_{11}^{-1}(z) & -\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \\ \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z) & \hat{H}_{22}(z) - \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \end{bmatrix}. \tag{A.11}$$

Thus we can partition matrix Q as $\mathrm{diag}(Q_1, Q_2)$, where Q_1 corresponds to the oracles in κ and Q_2 corresponds to the rest part of oracles. Consequently, Q^{-1} can be written as $\mathrm{diag}({Q_1}^{-1}, {Q_2}^{-1})$.

From eq. (A.10) and eq. (A.11), we have

$$\begin{aligned}
\hat{H}(z) &\xrightarrow{C_\kappa} \begin{bmatrix} \hat{H}_{11}^{-1}(z) & -\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \\ \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z) & \hat{H}_{22}(z) - \hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z) \end{bmatrix} \\
&\xrightarrow{P_\pi} \begin{bmatrix} Q_1\hat{H}_{11}^{-1}(z)Q_1^{-1} & -Q_1\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z)Q_2^{-1} \\ Q_2\hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)Q_1^{-1} & Q_2\hat{H}_{22}(z)Q_2^{-1} - Q_2\hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z)Q_2^{-1} \end{bmatrix}, \\
\hat{H}(z) &\xrightarrow{P_\pi} \begin{bmatrix} Q_1\hat{H}_{11}(z)Q_1^{-1} & Q_1\hat{H}_{12}(z)Q_2^{-1} \\ Q_2\hat{H}_{21}(z)Q_1^{-1} & Q_2\hat{H}_{22}(z)Q_2^{-1} \end{bmatrix} \\
&\xrightarrow{C_\kappa} \begin{bmatrix} Q_1\hat{H}_{11}^{-1}(z)Q_1^{-1} & -Q_1\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z)Q_2^{-1} \\ Q_2\hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)Q_1^{-1} & Q_2\hat{H}_{22}(z)Q_2^{-1} - Q_2\hat{H}_{21}(z)\hat{H}_{11}^{-1}(z)\hat{H}_{12}(z)Q_2^{-1} \end{bmatrix}.
\end{aligned}$$

We get the desired results to show C_κ and P_π commute. Therefore, conjugation and cyclic permutation commute. □

A.12 Proof of (2.36) and (2.37)

For each $i \in \{1, \dots, n\}$ we have

$$z_i = \text{argmin}_x \left\{ \lambda_i f_i(x) + \frac{1}{2} \begin{bmatrix} x \\ y_i \end{bmatrix}^T \begin{bmatrix} Q_{11}^i & Q_{12}^i \\ Q_{21}^i & Q_{22}^i \end{bmatrix} \begin{bmatrix} x \\ y_i \end{bmatrix} \right\}.$$

Besides, Q_{11}^i is invertible for any $i \in \{1, \dots, n\}$. Since f_i is a convex function, the argmin oracle can be written as $z_i \in -{Q_{11}^i}^{-1} \lambda \partial f_i(z_i) - {Q_{11}^i}^{-1} Q_{12}^i y_i$ by treating ∂f_i as the oracle. Written into matrix form, we have

$$\bar{u}_1 = -{Q_1}^{-1} \lambda \tilde{u}_1 - {Q_1}^{-1} Q_2 \bar{y}_1, \tag{A.12}$$

where $\tilde{u}_1 = [\partial f_1(z_1), \dots, \partial f_n(z_n)]^T$. Combine eq. (A.12) with the state-space realization eq. (2.34), we get the desired results for eq. (2.36). The corresponding system equations show as

$$\begin{aligned}
x^{k+1} &= (A - B_1(I + M_1 D_{11})^{-1} M_1 C_1) x^k - B_1(I + M_1 D_{11})^{-1} M_2 \tilde{u}_1^k + (B_2 - B_1(I + M_1 D_{11})^{-1} M_1 D_{12}) \bar{u}_2^k \\
\bar{u}_1^k &= -(I + M_1 D_{11})^{-1} M_1 C_1 x^k - (I + M_1 D_{11})^{-1} M_2 \tilde{u}_1^k - (I + M_1 D_{11})^{-1} M_1 D_{12} \bar{u}_2^k \\
\bar{y}_2^k &= (C_2 - D_{21}(I + M_1 D_{11})^{-1} M_1 C_1) x^k - D_{21}(I + M_1 D_{11})^{-1} M_2 \tilde{u}_1^k + (D_{22} - D_{21}(I + M_1 D_{11})^{-1} M_1 D_{12}) \bar{u}_2^k.
\end{aligned}$$

To calculate the transfer function, note that

$$(zI - A + B_1(I + M_1 D_{11})^{-1} M_1 C_1)^{-1} = (zI - A)^{-1} - (zI - A)^{-1} B_1 (I + M_1 \hat{H}_{11}(z))^{-1} M_1 C_1 (zI - A)^{-1}.$$

We have

$$
\begin{aligned}
\hat{H}'_{11}(z) &= (I + M_1 D_{11})^{-1} M_1 C_1 (zI - A + B_1 (I + M_1 D_{11})^{-1} M_1 C_1)^{-1} B_1 (I + M_1 D_{11})^{-1} M_2 - (I + M_1 D_{11})^{-1} M_2 \\
&= (I + M_1 D_{11})^{-1} (M_1 \hat{H}_{11}(z) - M_1 D_{11})(I - (I + M_1 \hat{H}_{11}(z))^{-1} (M_1 \hat{H}_{11}(z) - M_1 D_{11}))(I + M_1 D_{11})^{-1} M_2 \\
&\quad - (I + M_1 D_{11})^{-1} M_2 \\
&= (I + M_1 D_{11})^{-1} (M_1 \hat{H}_{11}(z) - M_1 D_{11})(I + M_1 \hat{H}_{11}(z))^{-1} M_2 - (I + M_1 D_{11})^{-1} M_2 \\
&= -(I + M_1 \hat{H}_{11}(z))^{-1} M_2 \\
\hat{H}'_{12}(z) &= -(I + M_1 D_{11})^{-1} M_1 C_1 (zI - A + B_1 (I + M_1 D_{11})^{-1} M_1 C_1)^{-1} B_2 - (I + M_1 \hat{H}_{11}(z))^{-1} M_1 D_{12} \\
&= -(I + M_1 D_{11})^{-1} (I - (M_1 \hat{H}_{11}(z) - M_1 D_{11})(I + M_1 \hat{H}_{11}(z))^{-1})(M_1 \hat{H}_{12}(z) \\
&\quad - M_1 D_{12}) - (I + M_1 \hat{H}_{11}(z))^{-1} M_1 D_{12} \\
&= -(I + M_1 \hat{H}_{11}(z))^{-1} (M_1 \hat{H}_{12}(z) - M_1 D_{12}) - (I + M_1 \hat{H}_{11}(z))^{-1} M_1 D_{12} \\
&= -(I + M_1 \hat{H}_{11}(z))^{-1} M_1 \hat{H}_{12}(z) \\
\hat{H}'_{21}(z) &= -C_2 (zI - A + B_1 (I + M_1 D_{11})^{-1} M_1 C_1)^{-1} B_1 (I + M_1 D_{11})^{-1} M_2 - D_{21} (I + M_1 \hat{H}_{11}(z))^{-1} M_2 \\
&= -(\hat{H}_{21}(z) - D_{21})(I - (I + M_1 \hat{H}_{11}(z))^{-1} (M_1 \hat{H}_{11}(z) - M_1 D_{11}))(I + M_1 D_{11})^{-1} M_2 \\
&\quad - D_{21} (I + M_1 \hat{H}_{11}(z))^{-1} M_2 \\
&= -(\hat{H}_{21}(z) - D_{21})(I + M_1 \hat{H}_{11}(z))^{-1} M_2 - D_{21} (I + M_1 \hat{H}_{11}(z))^{-1} M_2 \\
&= -\hat{H}_{21}(z)(I + M_1 \hat{H}_{11}(z))^{-1} M_2 \\
\hat{H}'_{22}(z) &= \hat{H}_{22}(z) - (\hat{H}_{21}(z) - D_{21})(I + M_1 \hat{H}_{11}(z))^{-1} M_1 (\hat{H}_{12}(z) - D_{12}) - D_{21} (I + M_1 \hat{H}_{11}(z))^{-1} M_1 (\hat{H}_{12}(z) - D_{12}) \\
&\quad - (\hat{H}_{21}(z) - D_{21})(I + M_1 \hat{H}_{11}(z))^{-1} M_1 D_{12} - D_{21} (I + M_1 \hat{H}_{11}(z))^{-1} M_1 D_{12} \\
&= \hat{H}_{22}(z) - \hat{H}_{21}(z)(I + M_1 \hat{H}_{11}(z))^{-1} M_1 \hat{H}_{12}(z).
\end{aligned}
$$

Thus, we get the desired results as eq. (2.37).

APPENDIX B

APPENDIX OF CHAPTER 3

B.1 Proofs of main results

In this section we give the proofs for the main results of the paper: theorem 3.4.1, theorem 3.4.2, and theorem 3.4.3.

B.1.1 Preliminaries

We start by recalling some useful background information and technical results that are useful for proving the main theorems. In order to obtain the exponentially small failure probabilities in theorem 3.4.1 and theorem 3.4.3 we take a different approach from the one in [41]. The proofs are based on regularized Schur complements and approximate matrix multiplication. Our arguments are inspired by the techniques used to establish statistical guarantees for approximate kernel ridge regression via column sampling schemes [4, 7].

Nyström Approximation: Properties

We start by recalling some important properties of the Nyström approximation (B.2.1). We shall also need the regularized Nyström approximation. Recall that $\Omega \in \mathbb{R}^{d\times s}$ denotes the test matrix from which we construct the Nyström approximation. Given $\sigma > 0$, the regularized Nyström approximation with respect to Ω is defined as

$$H\langle\Omega\rangle_\sigma = (H\Omega)(\Omega^T H\Omega + \sigma I)^{-1}(H\Omega)^T. \tag{B.1}$$

Furthermore, let $H = V\Lambda V^T$ be the eigendecomposition of H and define $D_\sigma = H(H + \sigma I)^{-1} = \Lambda(\Lambda + \sigma I)^{-1}$. We shall see below that D_σ plays a crucial role in the analysis. The following lemmas are well known in the literature and summarize the properties of the Nyström and regularized Nyström approximation. Lemma B.1.1 may be found in [41] and lemma B.1.2 in [4].

Lemma B.1.1. Let $H\langle\Omega\rangle$ be a Nyström approximation of a symmetric psd matrix H. Then

1. The approximation $H\langle\Omega\rangle$ is psd and has rank at most s.
2. The approximation $H\langle\Omega\rangle$ depends only on range(Ω).
3. In the Loewner order, $H\langle\Omega\rangle \preceq H$.
4. In particular, the eigenvalues satisfy $\lambda_j(H\langle\Omega\rangle) \leq \lambda_j(H)$ for each $1 \leq j \leq d$.

Lemma B.1.2. Let H be a symmetric psd matrix, $\sigma > 0$. Define $E = H - H\langle\Omega\rangle$ and $E_\sigma = H - H\langle\Omega\rangle_\sigma$. Then the following hold.

1. $H\langle\Omega\rangle_\sigma \preceq H\langle\Omega\rangle \preceq H$.
2. $0 \preceq E \preceq E_\sigma$.
3. If $\|D_\sigma^{1/2} V^T (\frac{1}{s}\Omega\Omega^T) V D_\sigma^{1/2} - D_\sigma\| \leq \eta < 1$, then

$$0 \preceq E_\sigma \preceq \frac{\sigma}{1-\eta} I. \tag{B.2}$$

Lemma B.1.2 relates $H\langle\Omega\rangle_\sigma$ to $H\langle\Omega\rangle$ and H. In particular, item 2 implies that $\|E\| \leq \|E_\sigma\|$, so controlling E_σ controls E. Item 3 shows that E_σ can be controlled by the spectral norm of the matrix

$$D_\sigma^{1/2} V^T \frac{1}{s}\Omega\Omega^T V D_\sigma^{1/2} - D_\sigma. \tag{B.3}$$

The spectral norm of (B.3) can be bounded by observing

$$\mathbb{E}\left[D_\sigma^{1/2} V^T \frac{1}{s}\Omega\Omega^T V D_\sigma^{1/2}\right] = D_\sigma^{1/2} V^T \mathbb{E}\left[\frac{1}{s}\Omega\Omega^T\right] V D_\sigma^{1/2} = D_\sigma^{1/2} V^T V D_\sigma^{1/2} = D_\sigma. \tag{B.4}$$

Thus, $D_\sigma^{1/2} V^T \frac{1}{s}\Omega\Omega^T V D_\sigma^{1/2}$ is an unbiased estimator of D_σ, and may be viewed as approximating the product of the matrices $D_\sigma^{1/2} V^T$ and $V D_\sigma^{1/2}$. Hence results from randomized linear algebra can bound the spectral norm of this difference. In particular, it suffices to take a sketch size that scales with the effective dimension, using results on approximate matrix multiplication in terms of stable rank [24].

Approximate matrix multiplication in terms of the effective dimension

The condition in item 3 of lemma B.1.2 follows immediately from theorem 1 of [24]. Unfortunately, the analysis in that paper does not yield explicit constants. Instead we use a special case of their results due to [58] that provides explicit constants. Theorem B.1.1 simplifies theorem 5.2 in [58].

Theorem B.1.1. Let $\Psi \in \mathbb{R}^{s\times d}$ be a matrix with i.i.d. $N(0, \frac{1}{s})$ entries. Given $\delta > 0$, and $\tau \in (0, 1)$ it holds with probability at least $1 - \delta$ that

$$\sup_{v\in\mathbb{S}^{d-1}} \langle v, (D_\sigma^{1/2} V^T \Psi^T \Psi V D_\sigma^{1/2} - D_\sigma) v\rangle \leq \tau + 2\sqrt{\tau}, \tag{B.5}$$

$$\inf_{v\in\mathbb{S}^{d-1}} \langle v, (D_\sigma^{1/2} V^T \Psi^T \Psi V D_\sigma^{1/2} - D_\sigma) v\rangle \geq \tau - 2\sqrt{\tau}, \tag{B.6}$$

provided $s \geq \frac{\left(\sqrt{d_{\text{eff}}(\sigma)} + \sqrt{8\log(16/\delta)}\right)^2}{\tau}$.

Setting $\Psi = \frac{1}{\sqrt{s}}\Omega^T$, where $\Omega \in \mathbb{R}^{d\times s}$ has i.i.d. $N(0, 1)$ entries, theorem B.1.1 yields the following corollary.

Corollary B.1.1. Let $\Omega \in \mathbb{R}^{d\times s}$ be a matrix with i.i.d. $N(0,1)$ entries. Given $\delta > 0$, and $\tau \in (0,1)$ it holds with probability at least $1-\delta$ that

$$\left\| D_\sigma^{1/2} V^T \frac{1}{s} \Omega\Omega^T V D_\sigma^{1/2} - D_\sigma \right\| \leq \tau + 2\sqrt{\tau} \tag{B.7}$$

provided $s \geq \frac{\left(\sqrt{d_{\text{eff}}(\rho)}) + \sqrt{8\log(16/\delta)}\right)^2}{\tau}$.

Condition number of Nyström preconditoned linear system

The following result is a simpler version of proposition 5.2 in [41].

Proposition B.1.1. Let $\hat{H} = U\hat{\Lambda}U^T$ be any rank-s Nyström approximation, with sth largest eigenvalue $\hat{\lambda}_s$, and let $E = H - \hat{H}$ be the approximation error. Construct the Nyström preconditioner P as in (3.6). Then the condition number of the preconditioned matrix $P^{-1/2}H_\rho P^{-1/2}$ satisfies

$$\kappa_2(P^{-1/2}H_\rho P^{-1/2}) \leq \frac{\hat{\lambda}_s + \rho + \|E\|}{\rho}. \tag{B.8}$$

Proposition B.1.1 bounds the condition condition number of the Nyström preconditioned linear system in terms of $\hat{\lambda}_s, \rho$ and the approximation error $\|E\|$. We would like to emphasize that the bound in proposition B.1.1 is deterministic.

B.1.2 Proofs of Theorem 3.4.1 and Corollary 3.4.1

We start with two lemmas from which theorem 3.4.1 follows easily. The first lemma and its proof appear in [41].

Lemma B.1.3. Let $H \in \mathbb{S}_n^+(\mathbb{R})$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$. Let $\rho > 0$ be regularization parameter, and define the effective dimension as in (3.7). Then the following statement holds.

Fix $\gamma > 0$. If $j \geq (1 + \gamma^{-1}) d_{\text{eff}}(\rho)$, then $\lambda_j \leq \gamma\rho$.

Lemma B.1.4. Let $\epsilon > 0$ and $E = H - H\langle\Omega\rangle$. Suppose we construct a randomized Nyström approximation from a standard Gaussian random matrix Ω with sketch size $s \geq 8\left(\sqrt{d_{\text{eff}}(\epsilon)} + \sqrt{8\log(\frac{16}{\delta})}\right)^2$. Then the event

$$\mathcal{E} = \{\|E\| \leq 6\epsilon\}, \tag{B.9}$$

holds with probability at least $1 - \delta$.

Proof. Let $\Omega_s = \frac{1}{\sqrt{s}}\Omega$ and observe that $H\langle\Omega_s\rangle = H\langle\Omega\rangle$. Now the conditions of corollary B.1.1 are satisfied with $\sigma = \epsilon$ and $\tau = 8$. Consequently with probability at least $1 - \delta$,

$$\left\| D_\epsilon^{1/2} V^T \frac{1}{s} \Omega\Omega^T V D_\epsilon^{1/2} - D_\epsilon \right\| \leq \frac{1}{8} + \frac{\sqrt{2}}{2}.$$

Hence applying lemma B.1.2 with $\sigma = \epsilon$ and $\eta = \frac{1}{8} + \frac{\sqrt{2}}{2}$, we obtain

$$\left\| H - H\langle\Omega_s\rangle_\epsilon \right\| \leq 6\epsilon,$$

with probability at least $1 - \delta$. Recalling our initial observation, we conclude the desired result. □

Proof of theorem 3.4.1

Proof. As $s \geq 8\left(\sqrt{d_{\text{eff}}(\rho)} + \sqrt{8\log(\frac{16}{\delta})}\right)^2$ we have that $\|E\| \leq 6\rho$ with probability at least $1 - \delta$ by lemma B.1.4. Furthermore, $\hat{\lambda}_s \leq \frac{\rho}{7}$ by item 3 of lemma B.1.1 and lemma B.1.3 with $\gamma = 1/7$. Combining this with proposition B.1.1, we conclude with probability at least $1 - \delta$,

$$\kappa_2(P^{-1/2} H_\rho P^{-1/2}) \leq \frac{\hat{\lambda}_s + \rho + \|E\|}{\rho} \leq 1 + 6 + \frac{1}{7} \leq 8$$

as desired. □

Proof of Corollary 3.4.1

Proof. Let $A = P^{-1/2} H_\rho P^{-1/2}$ and condition on the event that $\kappa_2(A) \leq 8$, which holds with probability at least $1 - \delta$. The standard theory for convergence of CG [100] guarantees after t iterations that,

$$\frac{\|x_t - \tilde{x}_\star\|_A}{\|\tilde{x}_\star\|_A} \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^t \tag{B.10}$$

where $\|x\|_A = x^T A x$. Theorem 3.4.1 guarantees that the Nyström preconditioned matrix satisfies $\kappa_2(A) \leq 8$, so the above display may be majorized as

$$\frac{\|x_t - \tilde{x}_\star\|_A}{\|\tilde{x}_\star\|_A} \leq \left(\frac{1}{2}\right)^{t-1}. \tag{B.11}$$

Now, from the elementary inequality

$$\lambda_d(A)\|x\|_2 \leq \|x\|_A \leq \lambda_1(A)\|x\|_2, \tag{B.12}$$

we conclude

$$\frac{\|x_t - \tilde{x}_\star\|_2}{\|\tilde{x}_\star\|_2} \leq \kappa_2(A)\left(\frac{1}{2}\right)^{t-1} \leq \left(\frac{1}{2}\right)^{t-4}. \tag{B.13}$$

To obtain the claimed result, multiply both sides by $\|\tilde{x}_\star\|_2$ and solve $\|\tilde{x}_\star\|_2 \left(\frac{1}{2}\right)^{t-4} = \epsilon$ for t. □

B.1.3 Proof of Theorem 3.4.2

This proof is a natural consequence of the following theorem from [35].

Theorem B.1.2. Consider a convex optimization problem in the primal form (P), minimize $f(x) + h(Mx)$, where $x \in \mathbb{R}^d$, $M \in \mathbb{R}^{m \times d}$ has full column rank. Pick any

$y^0, z^0 \in \mathbb{R}^m$, and $\rho > 0$, and summable sequences

$$\{\varepsilon^k\}_{k=0}^{\infty} \subseteq [0,\infty),\ \sum_{k=0}^{\infty} \varepsilon^k < \infty,$$

$$\{\nu^k\}_{k=0}^{\infty} \subseteq [0,\infty),\ \sum_{k=0}^{\infty} \nu^k < \infty,$$

$$\{\lambda^k\}_{k=0}^{\infty} \subseteq (0,2),\ 0 < \inf \lambda^k \leq \sup \lambda^k < 2.$$

The dual problem (D) of primal problem (P) is

$$\text{maximize}_{y\in\mathbb{R}^m}\ -(f^*(-M^T y) + g^*(y)).$$

Suppose the primal and dual ADMM iterates $\{x^k\}_{k=0}^{\infty}$, $\{z^k\}_{k=0}^{\infty}$, and $\{y^k\}_{k=0}^{\infty}$ satisfy the update equations to within errors given by conform, for all k to

$$\begin{aligned}
&\left\| x^{k+1} - \text{argmin}_x\{f(x) + \langle y^k, Mx\rangle + \frac{1}{2}\rho\|Mx - z^k\|_2^2\}\right\|_2 \leq \varepsilon^k,\\
&\left\| z^{k+1} - \text{argmin}_z\{h(z) - \langle y^k, z\rangle + \frac{1}{2}\rho\|\lambda^k M x^{k+1} - z + (1-\lambda^k)z^k\|_2^2\}\right\|_2 \leq \nu^k, \qquad \text{(B.14)}\\
&y^{k+1} = y^k + \rho(\lambda^k M x^{k+1} + (1-\lambda^k)z^k - z^{k+1}).
\end{aligned}$$

Then if (P) has a Kuhn-Tucker pair, $\{x^k\}$ converges to a solution of (P) and $\{y^k\}$ converges to a solution of (D).

Proof of theorem 3.4.2

Proof. Consider optimization problem (3.1) and the associated NysADMM algorithm algorithm 3.2.3. Suppose $\{\tilde{x}^k\}_{k=0}^{\infty}$, $\{\tilde{z}^k\}_{k=0}^{\infty}$, and $\{\tilde{u}^k\}_{k=0}^{\infty}$ are generated by NysADMM iterations. Since $\ell(Ax, b)$ is quadratic with respect to x and the smooth part g of regularizer r has constant Hessian, the x-subproblem of (3.1) is exactly the linear system (3.4).

Let x^{k+1} be the exact solution for the x-subproblem at iteration k. For all $k \geq 0$, NysADMM iterate $\tilde{x}^{k+1}$ satisfies $\|\tilde{x}^{k+1} - x^{k+1}\|_2 \leq \varepsilon^k$. Let $M = I$, $\nu^k = 0$, $\lambda^k = 1$,

$y^k = \rho\tilde{u}^k$ for all k, and $f(x) = \ell(Ax, b) + g(x)$. By theorem B.1.2, $\{\tilde{x}^k\}_{k=0}^{\infty}$, $\{\tilde{z}^k\}_{k=0}^{\infty}$, and $\{\rho\tilde{u}^k\}_{k=0}^{\infty}$ satisfy condition (B.14). Therefore, if optimization problem (3.1) has a Kuhn-Tucker pair, $\{\tilde{x}^k\}$ converges to a solution of (3.1) and $\{\rho\tilde{u}^k\}$ converges to a solution of the dual problem of (3.1).

Next, we derive the bound for the number of Nyström PCG iterations T^k required at NysADMM iteration k. Note that in this case the Hessians of ℓ and g are constant. We only need to sketch once for the constant linear system matrix $A^T H^{\ell}(A\tilde{x}^k; b)A + H^g(\tilde{x}^k)$ and can reuse the sketch for all NysADMM iterations. Since the Nyström preconditioner is constructed with sketch size $s \geq 8\left(\sqrt{d_{\text{eff}}(\rho)} + \sqrt{8\log(\frac{16}{\delta})}\right)^2$, by corollary 3.4.1, with probability at least $1 - \delta$, after

$$T^k \geq \left\lceil \frac{\log\left(\frac{16\|x^{k+1}\|_2}{\varepsilon^k}\right)}{\log(2)} \right\rceil$$

Nyström PCG iterations, we have $\|\tilde{x}^{k+1} - x^{k+1}\|_2 \leq \varepsilon^k$. Recall the righthand side of linear system (3.4) r^k. The exact solution for the x-subproblem x^{k+1} at iteration k satisfies $\|x^{k+1}\|_2 \leq \frac{\|r^k\|_2}{\rho}$. We have

$$\left\lceil \frac{\log\left(\frac{16\|x^{k+1}\|_2}{\varepsilon^k}\right)}{\log(2)} \right\rceil \leq \left\lceil \frac{\log\left(\frac{16\|r^k\|_2}{\varepsilon^k\rho}\right)}{\log(2)} \right\rceil.$$

Further, by assumption, as $\|r^k\|_2$ is bounded by a constant R for all k, we have

$$\left\lceil \frac{\log\left(\frac{16\|r^k\|_2}{\varepsilon^k\rho}\right)}{\log(2)} \right\rceil \leq 4 + \left\lceil \frac{\log\left(\frac{R}{\varepsilon^k\rho}\right)}{\log(2)} \right\rceil \leq 4 + \left\lceil 2\log\left(\frac{R}{\varepsilon^k\rho}\right) \right\rceil.$$

This gives the bound for the number of Nyström PCG iterations T^k required at NysADMM iteration k □

B.1.4 Proof of Theorem 3.4.3

Proof. By hypothesis we have $s > 8d_{\text{eff}}(\frac{\epsilon\rho}{6})$, so lemma B.1.3 with $\gamma = 7$ yields

$$\hat{\lambda}_s \leq \lambda_s \leq \frac{1}{7}\frac{\epsilon\rho}{6} = \frac{\epsilon\rho}{42},$$

Thus,

$$\frac{\hat{\lambda}_s + \rho}{\rho} \leq 1 + \frac{\epsilon}{42}.$$

This gives the first statement. For the second statement we use our hypothesis on s to apply lemma B.1.4 with tolerance $\epsilon\rho/6$. From this we conclude $\|E\| \leq \epsilon\rho$ with probability at least $1 - \delta$. Combining this with proposition B.1.1 yields

$$\kappa_2(P^{-1/2}H_\rho P^{-1/2}) - \frac{\hat{\lambda}_s + \rho}{\rho} \leq \epsilon,$$

with probability at least $1 - \delta$. On the other hand, condition numbers always satisfy

$$\kappa_2(P^{-1/2}H_\rho P^{-1/2}) \geq 1.$$

Combining this with our upper bound on $\hat{\lambda}_s$ gives

$$\kappa_2(P^{-1/2}H_\rho P^{-1/2}) - \frac{\hat{\lambda}_s + \rho}{\rho} \geq 1 - (1 + \epsilon/42) = -\epsilon/42.$$

Hence with probability at least $1 - \delta$

$$\left|\kappa_2(P^{-1/2}H_\rho P^{-1/2}) - \frac{\hat{\lambda}_s + \rho}{\rho}\right| \leq \epsilon.$$

□

B.2 Randomized Nyström approximation and Nyström PCG

In this section we give the algorithms from [41] for the randomized Nyström approximation and Nyström PCG.

Algorithm B.2.2 Nyström PCG

input: psd matrix H, righthand side r, initial guess x_0, regularization parameter ρ, sketch size s, tolerance ε

$[U, \hat{\Lambda}]$ = RandomizedNyströmApproximation(H, s)
$w_0 = r - (H + \rho I)x_0$
$y_0 = P^{-1}w_0$
$p_0 = y_0$
while $\|w\|_2 > \varepsilon$ **do**
  $v = (H + \rho I)p_0$
  $\alpha = (w_0^T y_0)/(p_0^T v)$
  $x = x_0 + \alpha p_0$
  $w = w_0 - \alpha v$
  $y = P^{-1}w$
  $\beta = (w^T y)/(w_0^T y_0)$
  $x_0 \leftarrow x$, $w_0 \leftarrow w$, $p_0 \leftarrow y + \beta p_0$, $y_0 \leftarrow y$
end while

output: approximate solution $\hat{x}$

Algorithm B.2.1 Randomized Nyström Approximation

input: psd matrix $H \in \mathbb{S}_d^+(\mathbb{R})$, sketch size s

Ω = randn(d, s) ▷ Gaussian test matrix
Ω = qr$(\Omega, 0)$ ▷ thin QR decomposition
$Y = H\Omega$ ▷ s matvecs with H
ν = eps(norm$(Y, 2)$) ▷ compute shift
$Y_\nu = Y + \nu\Omega$ ▷ add shift for stability
C = chol$(\Omega^T Y_\nu)$ ▷ Cholesky decomposition
$B = Y_\nu / C$ ▷ triangular solve
$[U, \Sigma, \sim]$ = svd$(B, 0)$ ▷ thin SVD
$\hat{\Lambda} = \max\{0, \Sigma^2 - \nu I\}$ ▷ remove shift, compute eigs

output: Nyström approximation $\hat{H}_{\text{nys}} = U\hat{\Lambda}U^T$

B.3 AdaNysADMM

In this section we give the adaptive algorithm for computing the randomized Nyström approximation adopted from [41]. The adaptive algorithm has the benefit of reusing computation, in particular, we do not need to compute the

sketch Y from scratch. We simply add onto the sketch that we have already computed. We also give the pseudo-code for AdaNysADMM that uses algorithm B.3.1 to compute the Nyström preconditioner.

Algorithm B.3.1 AdaptiveRandNysAppx

input: symmetric psd matrix H, initial rank s_0, tolerance Tol
  $Y = [\ ], \Omega = [\ ]$, and $(\hat{\lambda}_s + \rho)/\rho = \text{Inf}$
  $m = s_0$
  while $(\hat{\lambda}_s + \rho)/\rho > \text{Tol}$ **do**
    generate Gaussian test matrix $\Omega_0 \in \mathbb{R}^{n \times m}$
    $[\Omega_0, \sim] = \text{qr}(\Omega_0, 0)$
    $Y_0 = H\Omega_0$
    $\Omega = [\Omega\ \ \Omega_0]$ and $Y = [Y\ \ Y_0]$
    $\nu = \sqrt{n}\,\text{eps}(\text{norm}(Y, 2))$
    $Y_\nu = Y + \nu\Omega$,
    $C = \text{chol}(\Omega^T Y_\nu)$
    $B = Y_\nu / C$
    compute $[U, \Sigma, \sim] = \text{svd}(B, 0)$
    $\hat{\Lambda} = \max\{0, \Sigma^2 - \nu I\}$ ▷ remove shift
    compute $(\hat{\lambda}_s + \rho)/\rho$
    $m \leftarrow s_0$, $s_0 \leftarrow 2s_0$ ▷ double rank if tolerance is not met
    if $s_0 > s_{\max}$ **then**
      $s_0 = s_0 - m$ ▷ when $s_0 > s_{\max}$, reset to $s_0 = s_{\max}$
      $m = s_{\max} - s_0$
      generate Gaussian test matrix $\Omega_0 \in \mathbb{R}^{n \times m}$
      $[\Omega_0, \sim] = \text{qr}(\Omega_0, 0)$
      $Y_0 = H\Omega_0$
      $\Omega = [\Omega\ \ \Omega_0]$ and $Y = [Y\ \ Y_0]$
      $\nu = \sqrt{n}\,\text{eps}(\text{norm}(Y, 2))$ ▷ compute final approximation and break
      $Y_\nu = Y + \nu\Omega$,
      $C = \text{chol}(\Omega^T Y_\nu)$
      $B = Y_\nu / C$
      compute $[U, \Sigma, \sim] = \text{svd}(B, 0)$
      $\hat{\Lambda} = \max\{0, \Sigma^2 - \nu I\}$
      break
    end if
  end while
output: Nyström approximation $(U, \hat{\Lambda})$

Algorithm B.3.2 AdaNysADMM

input: feature matrix A, response b, loss function ℓ, regularization g and h, step-size ρ, positive summable sequence $\{\varepsilon^k\}_{k=0}^{\infty}$

$[U, \hat{\Lambda}]$ = AdaptiveRandNysAppx$(A^T H^{\ell} A + H^s, s)$ ▷ use algorithm B.3.1

repeat

find $\tilde{x}^{k+1}$ that solves (3.4) within tolerance ε^k by Nyström PCG

$\tilde{z}^{k+1} = \operatorname{argmin}_z \{h(z) + \frac{\rho}{2}\|\tilde{x}^{k+1} - z + \tilde{u}^k\|_2^2\}$

$\tilde{u}^{k+1} = \tilde{u}^k + \tilde{x}^{k+1} - \tilde{z}^{k+1}$

until convergence

APPENDIX C

APPENDIX OF CHAPTER 4

C.1 Proofs for section 4.4

We begin by proving the following lemma, which plays a key role in the proof of lemma 4.4.1.

Lemma C.1.1. Let $h(z) = \frac{\rho}{2}\|w - z\|^2$. Then

$$\begin{aligned}\partial_\varepsilon h(z) &= \left\{ s \,\middle|\, \frac{\rho}{2}\left\|w - z + \frac{s}{\rho}\right\|^2 \leq \varepsilon \right\} \\ &= \left\{ v = \rho(z - w + \tilde{s}) \,\middle|\, \frac{\rho\|\tilde{s}\|^2}{2} \leq \varepsilon \right\}.\end{aligned}$$

Proof. The second equality follows immediately from the first, hence we shall prove the first equality. By definition of being an ε-subgradient, s satisfies the following inequality for all t

$$h(t) - h(z) \geq s^T(t - z) - \varepsilon.$$

Setting $t = w + \frac{s}{\rho}$ in the preceding inequality obtains,

$$\frac{\rho}{2}\left(\frac{\|s\|^2}{\rho^2} - \|w - z\|^2\right) \geq s^T\left(w - z + \frac{s}{\rho}\right) - \varepsilon.$$

So,

$$\varepsilon \geq \frac{\rho}{2}\|w - z\|^2 + s^T(w - z) + \frac{\|s\|^2}{2\rho}.$$

Simple algebra yields

$$\frac{\rho}{2}\|w - z\|^2 + s^T(w - z) + \frac{\|s\|^2}{2\rho} = \frac{\rho}{2}\left\|w - z + \frac{s}{\rho}\right\|^2.$$

Thus

$$\frac{\rho}{2}\left\|w - z + \frac{s}{\rho}\right\|^2 \leq \varepsilon,$$

The preceding argument establishes $\partial_\varepsilon h(z) \subset \left\{ s \,\middle|\, \frac{\rho}{2}\left\|w - z + \frac{s}{\rho}\right\|^2 \leq \varepsilon \right\}$. The reverse inclusion may be established by starting from the last display and reversing the argument used to arrive at it. Hence we conclude the desired result. □

Proof of lemma 4.4.1

Proof. Observe the function defining the z-subproblme may be decomposed as $G(z) = g(z) + h(z)$ with $h(z) = \frac{\rho}{2}\|M\tilde{x}^{k+1} - z + \tilde{u}^k\|_2^2$. Now, by hypothesis $\tilde{z}^{k+1}$ gives an ε_z^k-minimum of $G(z)$. Hence by proposition 4.4.1, $0 \in \partial_{\varepsilon_z^k} G(\tilde{z}^{k+1})$ and $\partial_{\varepsilon_z^k} G(\tilde{z}^{k+1}) \subset \partial_{\varepsilon_z^k} g(\tilde{z}^{k+1}) + \partial_{\varepsilon_z^k} h(\tilde{z}^{k+1})$. Thus, we have $0 = s + s_h$ where $s \in \partial_{\varepsilon_z^k} g(\tilde{z}^{k+1})$ and $s_h \in \partial_{\varepsilon_z^k} h(\tilde{z}^{k+1})$. Applying lemma C.1.1 we reach $s_h = \rho(\tilde{z}^{k+1} - M\tilde{x}^{k+1} - u^k + \tilde{s})$ with $\|\tilde{s}\| \leq \sqrt{\frac{2\varepsilon_z^k}{\rho}}$. The desired claim now immediately follows from using $s = -s_h$.

C.2 Proofs for section 4.5

Proof of proposition 4.5.1

Proof. Using the eigendecomposition, we may decompose H_f as

$$H_f = \lambda_1(H_f)\mathbf{v}\mathbf{v}^T + \sum_{i=2}^{d} \lambda_i(H_f)\mathbf{v}_i\mathbf{v}_i^T.$$

So

$$\begin{aligned}\|x^\star\|_{H_f}^2 &= \lambda_1(H_f)\langle \mathbf{v}, x^\star\rangle^2 + \sum_{i=2}^{d} \lambda_i(H_f)\langle \mathbf{v}_i, x_\star\rangle^2 \\ &\leq \lambda_1(H_f)\mu_{\mathbf{v}}\|x^\star\|^2 + \lambda_2(H_f)\sum_{i=2}^{d}\langle \mathbf{v}_i, x^\star\rangle^2 \\ &\leq \left(\mu_{\mathbf{v}} + \frac{\lambda_2(H_f)}{\lambda_1(H_f)}\right)\lambda_1(H_f)\|x^\star\|^2.\end{aligned}$$

Hence we reach

$$\frac{\lambda_1(H_f)\|x^\star\|^2}{\|x^\star\|_{H_f}^2} \geq \left(\mu_{\mathbf{v}} + \frac{\lambda_2(H_f)}{\lambda_1(H_f)}\right)^{-1}.$$

As we always have $\lambda_1(H_f)\|x^\star\|^2 \geq \|x^\star\|_{H_f}^2$, we conclude from the previous display that

$$\frac{\lambda_1(H_f)\|x^\star\|^2}{\|x^\star\|_{H_f}^2} \geq \max\left\{\left(\mu_{\mathbf{v}} + \frac{\lambda_2(H_f)}{\lambda_1(H_f)}\right)^{-1}, 1\right\}.$$

The last claim follows from elementary calculation. □

Proof of lemma 4.5.1

Proof. By definition of $\ell_U(v, w)$

$$\ell_U(v, w) = f(x) + g(z) - p^\star + \sup_{\hat{u}\in U}\langle \rho\hat{u}, v - (Mx - z)\rangle. \tag{C.1}$$

Hence if $v = Mx - z$ and $\ell_U(v, w) \leq \epsilon$, (C.1) yields

$$f(x) + g(z) - p^\star \leq \epsilon.$$

Thus we obtain the first statement.

Now, suppose $\ell_U(v, w) \leq \epsilon$, $\|v\| \leq \delta$ and $U = \mathcal{Z}$. Then combining these hypotheses with (C.1) and using $p^\star$ that is the optimum, we reach

$$\sup_{\hat{u}\in\mathcal{Z}}\langle \rho\hat{u}, v - (Mx - z)\rangle \leq \epsilon.$$

Observe if $v \neq Mx - z$ then $\sup_{\hat{u} \in \mathcal{Z}} \langle \rho \hat{u}, v - (Mx - z) \rangle = \infty$, contradicting the preceding inequality. Thus $v = Mx - z$, which yields the second statement, and completes the proof. □

Proof of lemma 4.5.2

Proof. Throughout the proof, we shall use x^{k+1} and z^{k+1} to represent the exact solutions of the x-subproblem and z-subproblem at iteration k.

We start by observing the optimality condition satisfied by the exact solution of the x-subproblem is given by

$$\begin{aligned} \langle \nabla f(\tilde{x}^k), x^{k+1} - x \rangle \leq \langle \eta^k \Theta^k (x^{k+1} - \tilde{x}^k), x - x^{k+1} \rangle + \rho \langle M x^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - x^{k+1}) \rangle \\ + \alpha \rho \langle M(\tilde{x}^k - x^{k+1}), M(x - x^{k+1}) \rangle. \end{aligned} \tag{C.2}$$

To obtain the perturbed optimality condition satisfied by the *inexact solution*, we introduce the inexact solution in (C.2) by writing $x^{k+1} = x^{k+1} - \tilde{x}^{k+1} + \tilde{x}^{k+1}$, which yields

$$\begin{aligned} \langle \nabla f(\tilde{x}^k), x^{k+1} - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x \rangle \leq \langle \eta^k \Theta^k (x^{k+1} - \tilde{x}^{k+1} + \tilde{x}^{k+1} - \tilde{x}^k), x - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1} \rangle \\ + \rho \langle M(x^{k+1} - \tilde{x}^{k+1}) + M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1}) \rangle \\ + \alpha \rho \langle M(\tilde{x}^k - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1}), M(x - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1}) \rangle. \end{aligned} \tag{C.3}$$

Rearranging (C.3) we reach

$$\begin{aligned}\langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - x\rangle \leq \langle \eta^k \Theta^k(\tilde{x}^{k+1} - \tilde{x}^k), x - \tilde{x}^{k+1}\rangle + \rho\langle M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - \tilde{x}^{k+1})\rangle \\ + \alpha\rho\langle M(\tilde{x}^k - \tilde{x}^{k+1}), M(x - \tilde{x}^{k+1})\rangle \\ + \underbrace{\langle \eta^k \Theta^k(x^{k+1} - \tilde{x}^{k+1}), x - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1}\rangle}_{E_1} + \underbrace{\langle \eta^k \Theta^k(\tilde{x}^{k+1} - \tilde{x}^k), \tilde{x}^{k+1} - x^{k+1}\rangle}_{E_2} \\ + \underbrace{\rho\langle M(x^{k+1} - \tilde{x}^{k+1}), M(x - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1})\rangle}_{E_3} \\ + \underbrace{\rho\langle M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(\tilde{x}^{k+1} - x^{k+1})\rangle}_{E_4} + \underbrace{\alpha\rho\langle M(\tilde{x}^k - \tilde{x}^{k+1}), M(\tilde{x}^{k+1} - x^{k+1})\rangle}_{E_5} \\ + \underbrace{\alpha\rho\langle M(\tilde{x}^{k+1} - x^{k+1}), M(x - \tilde{x}^{k+1} + \tilde{x}^{k+1} - x^{k+1})\rangle}_{E_6} \\ + \underbrace{\langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - x^{k+1}\rangle}_{E_7}.\end{aligned}$$

We see $\tilde{x}^{k+1}$ satisfies the original optimality condition with the addition of seven error terms: E_1 to E_7, which arise from inexactness. We proceed to control these terms by using $\tilde{x}^{k+1}$ is an ε_x^k-approximate minimum of the x-subproblem. In this vein, recall at each iteration, assumption 4.4.4 indicates the inexact solution satisfies

$$\max\{\|\tilde{x}^{k+1} - x^{k+1}\|_{\eta^k\Theta^k + \rho M^T M}, \|\tilde{x}^{k+1} - x^{k+1}\|\} \leq \varepsilon_x^k. \tag{C.4}$$

Observe that (C.4) implies the important inequalities

$$\|\tilde{x}^{k+1} - x^{k+1}\|_{\eta^k\Theta^k} \leq \varepsilon_x^k, \quad \|\tilde{x}^{k+1} - x^{k+1}\|_{\rho M^T M} \leq \varepsilon_x^k.$$

Furthermore, the boundedness hypothesis assumption 4.4.6 ensures for all k that

$$\|\tilde{x}^k - x^\star\|_{\Theta^1} \leq R_{x^\star,\Theta^1}, \quad \|\tilde{x}^k - x^\star\|_{M^T M} \leq R_{x^\star,M}, \quad \|\tilde{z}^k - z^\star\|, \|\tilde{u}^k - u^\star\| \leq R_{x^\star,u^\star,z^\star}.$$

We now apply the preceding observations to bound terms E_1 through E_6.

For term E_1, we find

$$E_1 = \langle \eta^k \Theta^k (x^{k+1} - \tilde{x}^{k+1}), x - \tilde{x}^{k+1} \rangle - \|x^{k+1} - \tilde{x}^{k+1}\|_{\Theta^k}^2 \leq \langle \eta^k \Theta^k (x^{k+1} - \tilde{x}^{k+1}), x - \tilde{x}^{k+1} \rangle$$
$$\leq \|x^{k+1} - \tilde{x}^{k+1}\|_{\eta^k \Theta^k} \|x - \tilde{x}^{k+1}\|_{\eta^k \Theta^k} \leq \tau_\zeta \varepsilon_x^k \|x - \tilde{x}^{k+1}\|_{\eta^k \Theta^1}.$$

Next we bound E_2. Invoking Cauchy-Schwarz, we find

$$E_2 = \langle \eta^k \Theta^k (\tilde{x}^{k+1} - \tilde{x}^k), \tilde{x}^{k+1} - x^{k+1} \rangle \leq \|\tilde{x}^{k+1} - \tilde{x}^k\|_{\eta^k \Theta^k} \|\tilde{x}^{k+1} - x^{k+1}\|_{\eta^k \Theta^k} \leq \varepsilon_x^k \|\tilde{x}^{k+1} - \tilde{x}^k\|_{\eta^k \Theta^k}$$
$$\leq \varepsilon_x^k \tau_\zeta \left(\|\tilde{x}^{k+1} - x^\star\|_{\eta \Theta^1} + \|\tilde{x}^k - x^\star\|_{\eta \Theta^1} \right) \leq 2\tau_\zeta R_{x^\star, \Theta^1} \varepsilon_x^k.$$

The logic for the remaining terms is similar. Combining Cauchy-Schwarz, inexactness, and boundedness we reach

$$E_3 \leq \|x^{k+1} - \tilde{x}^{k+1}\|_{\rho M^T M} \|x - \tilde{x}^{k+1}\|_{\rho M^T M} \leq \varepsilon_x^k \|x - \tilde{x}^{k+1}\|_{\rho M^T M},$$
$$E_4 = \rho \langle \tilde{u}^{k+1} + \tilde{z}^{k+1} - \tilde{z}^k, M(\tilde{x}^{k+1} - x^{k+1}) \rangle \leq \sqrt{\rho} \|\tilde{u}^{k+1} + \tilde{z}^{k+1} - \tilde{z}^k\| \|\tilde{x}^{k+1} - x^{k+1}\|_{\rho M^T M}$$
$$\leq \sqrt{\rho} (3R_{x^\star, u^\star, z^\star} + \|u^\star\|) \varepsilon_x^k,$$
$$E_5 \leq \alpha \|\tilde{x}^k - \tilde{x}^{k+1}\|_{\rho M^T M} \|\tilde{x}^{k+1} - x^{k+1}\|_{\rho M^T M} \leq \alpha \varepsilon_x^k \|\tilde{x}^k - \tilde{x}^{k+1}\|_{\rho M^T M} \leq 2\alpha\rho R_{x^\star, M} \varepsilon_x^k,$$
$$E_6 \leq \alpha \varepsilon_x^k (\|x - \tilde{x}^{k+1}\|_{\rho M^T M} + \varepsilon_x^k),$$
$$E_7 \leq \|\nabla f(\tilde{x}^k)\| \|\tilde{x}^{k+1} - x^{k+1}\| \leq \beta_f \varepsilon_x^k.$$

Substituting the preceding bounds on the error terms, we conclude

$$\langle \nabla f(\tilde{x}^k), \tilde{x}^{k+1} - x \rangle \leq \langle \eta^k \Theta^k (\tilde{x}^{k+1} - \tilde{x}^k), x - \tilde{x}^{k+1} \rangle + \rho \langle M\tilde{x}^{k+1} - \tilde{z}^k + \tilde{u}^k, M(x - \tilde{x}^{k+1}) \rangle$$
$$+ \alpha\rho \langle M(\tilde{x}^k - \tilde{x}^{k+1}), M(x - \tilde{x}^{k+1}) \rangle$$
$$+ \left(\chi_1 + \tau_\zeta \|x - \tilde{x}^{k+1}\|_{\eta \Theta^1} + (1 + \alpha) \|x - \tilde{x}^{k+1}\|_{\rho M^T M} \right) \varepsilon_x^k + \alpha (\varepsilon_x^k)^2,$$

where $\chi_1 = \left(2\tau_\zeta R_{x^\star, \Theta^1} + \sqrt{\rho}(3R_{x^\star, u^\star, z^\star} + \|u^\star\|) + \beta_f + 2\alpha\rho R_{x^\star, M} \right)$. □

Proof of lemma 4.5.3

Proof. We start by the observing the optimality condition satisfied by the exact solution of the z-subproblem is given by

$$g(z^{k+1}) - g(z) + \rho\langle (z^{k+1} - M\tilde{x}^{k+1}) - \tilde{u}^k, z^{k+1} - z\rangle \leq 0. \tag{C.5}$$

To arrive at the inexact optimality condition for the z-subproblem, we follow the same strategy as in lemma 4.5.2. We write $g(z^{k+1}) = g(z^{k+1}) - g(\tilde{z}^{k+1}) + g(\tilde{z}^{k+1})$ and $z^{k+1} = z^{k+1} - \tilde{z}^{k+1} + \tilde{z}^{k+1}$ in (C.5), which yields

$$\begin{aligned} &g(z^{k+1}) - g(\tilde{z}^{k+1}) + g(\tilde{z}^{k+1}) - g(z) \\ &+ \rho\langle z^{k+1} - \tilde{z}^{k+1} + \tilde{z}^{k+1} - M\tilde{x}^{k+1} - \tilde{u}^k, z^{k+1} - \tilde{z}^{k+1} + \tilde{z}^{k+1} - z\rangle \leq 0. \end{aligned}$$

Using $\tilde{u}^{k+1} = M\tilde{x}^{k+1} - \tilde{z}^{k+1} + \tilde{u}^k$ in the preceding display, and rearranging we reach

$$\begin{aligned} g(\tilde{z}^{k+1}) - g(z) - \rho\langle \tilde{u}^{k+1}, \tilde{z}^{k+1} - z\rangle \leq g(\tilde{z}^{k+1}) - g(z^{k+1}) + \rho\langle \tilde{u}^{k+1}, z^{k+1} - \tilde{z}^{k+1}\rangle \\ + \rho\langle z^{k+1} - \tilde{z}^{k+1}, z - \tilde{z}^{k+1}\rangle. \end{aligned}$$

Now, invoking $\tilde{z}^{k+1}$ is ε_z^k-approximate minimizer of the z-subproblem, we obtain from ρ-strong convexity of the z-subproblem that $\|\tilde{z}^{k+1} - z^{k+1}\| \leq \sqrt{\frac{2\varepsilon_z^k}{\rho}}$. Applying this observation in the preceding display yields

$$g(\tilde{z}^{k+1}) - g(z) - \rho\langle \tilde{u}^{k+1}, \tilde{z}^{k+1} - z\rangle \leq g(\tilde{z}^{k+1}) - g(z^{k+1}) + \left(R_{x^\star,u^\star,z^\star} + \|u^\star\| + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_x^k}.$$

Next we apply lemma 4.4.1 to obtain, $s = \rho(M\tilde{x}^{k+1} - \tilde{z}^{k+1} + \tilde{u}^k - \tilde{s}) \in \partial_{\varepsilon_z^k} g(\tilde{z}_{k+1})$ such that

$$\|\tilde{s}\| \leq \sqrt{\frac{2\varepsilon_z^k}{\rho}}.$$

Using this in conjunction with the definition of the ε_z^k-subgradient, we reach

$$\begin{aligned} g(\tilde{z}^{k+1}) - g(z) - \rho\langle \tilde{u}^{k+1}, \tilde{z}^{k+1} - z\rangle &\overset{(1)}{\leq} \langle s, \tilde{z}^{k+1} - z^{k+1}\rangle + \varepsilon_z^k + \left(R_{x^\star,u^\star,z^\star} + \|u^\star\| + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k} \\ &\overset{(2)}{=} \rho\langle \tilde{u}^{k+1} - \tilde{s}, \tilde{z}^{k+1} - z^{k+1}\rangle + \varepsilon_z^k + \left(R_{x^\star,u^\star,z^\star}\|u^\star\| + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k} \\ &\overset{(3)}{\leq} \left(R_{x^\star,u^\star,z^\star} + \|u^\star\| + \sqrt{2\rho\varepsilon_z^k}\right)\sqrt{2\rho\varepsilon_z^k} + \\ &\quad \left(R_{x^\star,u^\star,z^\star} + \|u^\star\| + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k} \\ &= (1 + 2\rho)\varepsilon_z^k + \left(2(R_{x^\star,u^\star,z^\star} + \|u^\star\|) + \|\tilde{z}^{k+1} - z\|\right)\sqrt{2\rho\varepsilon_z^k}. \end{aligned}$$

Where (1) uses s is a ε_z^k-subgradient, (2) uses the explicit form of s, and (3) uses Cauchy-Schwarz. The desired claim now follows by rearrangement. □

C.3 Proofs for section 4.6

Proof of lemma 4.6.1

Proof. We start by the optimality condition of the x-subproblem,

$$-\rho M^T \tilde{u}^{k+1} + \rho M^T M(\tilde{x}^{k+1} - x^{k+1}) + \rho M^T(\tilde{z}^k - \tilde{z}^{k+1}) = \nabla f(\tilde{x}^k) + \eta^k \Theta^k (x^{k+1} - \tilde{x}^k). \quad \text{(C.6)}$$

Introducing the inexact solution $\tilde{x}^{k+1}$ in (C.6) by writing $x^{k+1} = x^{k+1} + \tilde{x}^{k+1} - \tilde{x}^{k+1}$ and $\nabla f(x^{k+1}) = \nabla f(x^{k+1}) + \nabla f(\tilde{x}^{k+1}) - \nabla f(\tilde{x}^{k+1})$ and rearranging, we obtain the optimality condition of the x-subproblem with the inexact solution,

$$\begin{aligned}\nabla f(\tilde{x}^{k+1}) = &-\rho M^T \tilde{u}^{k+1} + \rho M^T(\tilde{z}^k - \tilde{z}^{k+1}) + \nabla f(\tilde{x}^{k+1}) - (\nabla f(\tilde{x}^k) + \eta^k \Theta^k(\tilde{x}^{k+1} - \tilde{x}^k)) \\ &+ \rho M^T M(\tilde{x}^{k+1} - x^{k+1}) + \eta^k \Theta^k(\tilde{x}^{k+1} - x^{k+1})\end{aligned} \quad \text{(C.7)}$$

Similarly, we can get the optimality condition of the z-subproblem with the inexact solution $\tilde{z}^{k+1}$ as

$$\rho \tilde{u}^{k+1} + \rho(\tilde{z}^{k+1} - z^{k+1}) \in \partial g(z^{k+1}). \quad \text{(C.8)}$$

Recall that the saddle point $(x^\star, z^\star, u^\star)$ of problem (4.1) satisfies the KKT conditions

$$\begin{aligned} -\rho M^T u^\star &= \nabla f(x^\star), \\ \rho u^\star &\in \partial g(z^\star), \\ M x^\star - z^\star &= 0. \end{aligned} \quad \text{(C.9)}$$

Now, by strong convexity of f

$$\langle \tilde{x}^{k+1} - x^\star, \nabla f(\tilde{x}^{k+1}) - \nabla f(x^\star) \rangle \geq \sigma_f \|\tilde{x}^{k+1} - x^\star\|^2.$$

Combining the preceding display with (C.7) and (C.9), we have

$$\begin{aligned}
&\langle \tilde{x}^{k+1} - x^\star, \nabla f(\tilde{x}^{k+1}) - \nabla f(x^\star)\rangle \\
= \;& \rho\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1}\rangle + \rho\langle M(\tilde{x}^{k+1} - x^\star), \tilde{z}^k - \tilde{z}^{k+1}\rangle \\
& + \langle \tilde{x}^{k+1} - x^\star, \nabla f(\tilde{x}^{k+1}) - (\nabla f(\tilde{x}^k) + \eta^k\Theta^k(\tilde{x}^{k+1} - \tilde{x}^k))\rangle + \langle \tilde{x}^{k+1} - x^\star, (\eta^k\Theta^k + \rho M^T M)(\tilde{x}^{k+1} - x^{k+1})\rangle \\
\leq \;& \rho\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + R_{x^\star,u^\star,z^\star}\varepsilon_\nabla^k + (\sqrt{\eta^k\|\Theta^k\|}R_{x^\star,u^\star,z^\star} + \sqrt{\rho}R_{x^\star,M})\varepsilon_x^k,
\end{aligned}$$

where the last inequality follows from Cauchy-Schwartz, gradient estimation error of f (assumption 4.6.2), and $\|\tilde{x}^{k+1} - x^{k+1}\|_{\eta^k\Theta^k+\rho M^T M} \leq \varepsilon_x^k$. Thus we prove inequality (4.28),

$$\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + \frac{R_{x^\star,u^\star,z^\star}}{\rho}\varepsilon_\nabla^k + \frac{\sqrt{\eta^k\|\Theta^k\|}R_{x^\star,u^\star,z^\star} + \sqrt{\rho}R_{x^\star,M}}{\rho}\varepsilon_x^k \geq \frac{\sigma_f}{\rho}\|\tilde{x}^{k+1} - x^\star\|^2.$$

Next, we combine (C.8) and (C.9), and use convexity of g to obtain

$$\langle \tilde{z}^{k+1} - z^\star + z^{k+1} - \tilde{z}^{k+1}, \tilde{u}^{k+1} - u^\star + \tilde{z}^{k+1} - z^{k+1}\rangle \geq 0.$$

Given the fact that $\tilde{z}^{k+1}$ is an ε_z^k-minimum of the z-subproblem we know that $\|\tilde{z}^{k+1} - z^{k+1}\|_2 \leq \sqrt{\frac{2\varepsilon_z^k}{\rho}}$. Applying Cauchy-Schwarz, we get inequality (4.29)

$$\langle \tilde{z}^{k+1} - z^\star, \tilde{u}^{k+1} - u^\star\rangle + 2\sqrt{\frac{2}{\rho}}R_{x^\star,u^\star,z^\star}\sqrt{\varepsilon_z^k} \geq 0.$$

Observe by convexity of g and using (C.8),

$$\langle z^k - z^{k+1}, \tilde{u}^k + \tilde{z}^k - z^k - \tilde{u}^{k+1} - \tilde{z}^{k+1} + z^{k+1}\rangle \geq 0.$$

Now, arguing as we did to arrrive at (4.29) with the fact $\varepsilon_z^{k-1} > \varepsilon_z^k$, we conclude inequality (4.30)

$$\langle \tilde{z}^k - \tilde{z}^{k+1}, \tilde{u}^k - \tilde{u}^{k+1}\rangle + 8\sqrt{\frac{2}{\rho}}R_{x^\star,u^\star,z^\star}\sqrt{\varepsilon_z^{k-1}} + \frac{4}{\rho}\varepsilon_z^{k-1} \geq 0.$$

□

Proof of lemma 4.6.3

Proof. Recall that we use strong convexity of f to prove (4.28). Now we first use Lipschitz continuity of ∇f to prove another similar inequality. By Lipschitz continuity,

$$\langle \tilde{x}^{k+1} - x^\star, \nabla f(\tilde{x}^{k+1}) - \nabla f(x^\star)\rangle \geq \frac{1}{L_f}\|\nabla f(\tilde{x}^{k+1}) - \nabla f(x^\star)\|^2.$$

Using the same step as the proof of (4.28), we know

$$\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + \frac{C_1}{2}\varepsilon_\nabla^k + \frac{C_2}{2}\varepsilon_x^k \geq \frac{1}{\rho L_f}\|\nabla f(\tilde{x}^{k+1}) - \nabla f(x^\star)\|^2. \quad \text{(C.10)}$$

Combining (C.7) and (C.9), the righthand side of (C.10) becomes

$$\frac{1}{\rho L_f}\|\rho M^T(u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}) + \nabla f(\tilde{x}^{k+1}) - (\nabla f(\tilde{x}^k) + \eta^k\Theta^k(\tilde{x}^{k+1} - \tilde{x}^k)) + (\eta^k\Theta^k + \rho M^T M)(\tilde{x}^{k+1} - x^{k+1})\|^2.$$

And by the reverse triangle inequality,

$$\begin{aligned}\frac{1}{\rho L_f}\|\nabla f(\tilde{x}^{k+1}) - \nabla f(x^\star)\|^2 &\geq \frac{1}{\rho L_f}\|\rho M^T(u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1})\|^2 \\ - \frac{1}{\rho L_f}\|\nabla f(\tilde{x}^{k+1}) - (\nabla f(\tilde{x}^k) + \eta^k\Theta^k(\tilde{x}^{k+1} - \tilde{x}^k))\|^2 &- \frac{1}{\rho L_f}\|(\eta^k\Theta^k + \rho M^T M)(\tilde{x}^{k+1} - x^{k+1})\|^2\end{aligned} \quad \text{(C.11)}$$

Putting (C.10) and (C.11) together, we have

$$\begin{aligned}&\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + \frac{C_1}{2}\varepsilon_\nabla^k + \frac{1}{\rho L_f}(\varepsilon_\nabla^k)^2 + \frac{C_2}{2}\varepsilon_x^k + \frac{C_4}{2}(\varepsilon_x^k)^2 \geq \\ &\frac{1}{\rho L_f}\|\rho M^T(u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1})\|^2 \geq \frac{\rho\lambda_{\min}(MM^T)}{L_f}\|u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\|^2,\end{aligned} \quad \text{(C.12)}$$

where term $\frac{1}{\rho L_f}(\varepsilon_\nabla^k)^2$ is obtained by the gradient estimation error of f (assumption 4.6.2) and term $\frac{C_4}{2}(\varepsilon_x^k)^2$ comes from Cauchy-Schwarz and x-inexactness condition. Adding (4.28) and (C.12) together, we know for any $0 \leq \mu \leq 1$

$$\begin{aligned}&\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + \frac{C_1}{2}\varepsilon_\nabla^k + \frac{1}{\rho L_f}(1-\mu)(\varepsilon_\nabla^k)^2 + \frac{C_2}{2}\varepsilon_x^k + \frac{C_4}{2}(1-\mu)(\varepsilon_x^k)^2 \geq \\ &\frac{\sigma_f\mu}{\rho}\|\tilde{x}^{k+1} - x^\star\|^2 + \frac{\rho\lambda_{\min}(MM^T)}{L_f}(1-\mu)\|u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\|^2.\end{aligned} \quad \text{(C.13)}$$

Given the relation $M\tilde{x}^{k+1} - Mx^\star - (\tilde{z}^{k+1} - z^\star) = \tilde{u}^{k+1} - \tilde{u}^k$, we know

$$\|\tilde{x}^{k+1} - x^\star\|^2 \geq \frac{1}{\|M\|^2}\|\tilde{u}^k - \tilde{u}^{k+1} + z^\star - \tilde{z}^{k+1}\|^2.$$

Thus, (C.13) becomes

$$\langle M(\tilde{x}^{k+1} - x^\star), u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\rangle + \frac{C_1}{2}\varepsilon_\nabla^k + \frac{1}{\rho L_f}(1-\mu)(\varepsilon_\nabla^k)^2 + \frac{C_2}{2}\varepsilon_x^k + \frac{C_4}{2}(1-\mu)(\varepsilon_x^k)^2 \geq$$
$$\frac{\sigma_f}{\rho\|M\|^2}\mu\|\tilde{u}^k - \tilde{u}^{k+1} + z^\star - \tilde{z}^{k+1}\|^2 + \frac{\rho\lambda_{\min}(MM^T)}{L_f}(1-\mu)\|u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\|^2. \tag{C.14}$$

With same steps to arrive at (4.31) but replacing (4.28) with (C.14), we can derive a modified sufficient descent condition

$$\|\tilde{y}^k - y^\star\|^2 - \|\tilde{y}^{k+1} - y^\star\|^2 + C_1\varepsilon_\nabla^k + \frac{2}{\rho L_f}(1-\mu)(\varepsilon_\nabla^k)^2 + C_2\varepsilon_x^k + C_4(1-\mu)(\varepsilon_x^k)^2 + C_3\sqrt{\varepsilon_z^{k-1}} + \frac{8}{\rho}\varepsilon_z^{k-1} \geq$$
$$\|\tilde{y}^k - \tilde{y}^{k+1}\|^2 + \frac{2\sigma_f}{\rho\|M\|^2}\mu\|\tilde{u}^k - \tilde{u}^{k+1} + z^\star - \tilde{z}^{k+1}\|^2 + \frac{2\rho\lambda_{\min}(MM^T)}{L_f}(1-\mu)\|u^\star - \tilde{u}^{k+1} + \tilde{z}^k - \tilde{z}^{k+1}\|^2. \tag{C.15}$$

Using the reverse triangle inequality, the righthand side of (C.15) is greated than or equal to

$$\frac{2\sigma_f}{\rho\|M\|^2}\mu\|\tilde{z}^{k+1} - z^\star\|^2 + \left(1 - \frac{2\sigma_f}{\rho\|M\|^2}\mu\right)\|\tilde{u}^k - \tilde{u}^{k+1}\|^2$$
$$+ \frac{2\rho\lambda_{\min}(MM^T)}{L_f}(1-\mu)\|\tilde{u}^{k+1} - u^\star\|^2 + \left(1 - \frac{2\rho\lambda_{\min}(MM^T)}{L_f}(1-\mu)\right)\|\tilde{z}^k - \tilde{z}^{k+1}\|^2. \tag{C.16}$$

Therefore, as long as there exists $0 \leq \mu \leq 1$ such that

$$1 - \frac{2\sigma_f}{\rho\|M\|^2}\mu \geq 0 \text{ and } \; 1 - \frac{2\rho\lambda_{\min}(MM^T)}{L_f}(1-\mu) \geq 0, \tag{C.17}$$

inequality (4.33) must be satisfied.

Let

$$\mu = \frac{\rho^2\|M\|^2\lambda_{\min}(MM^T)}{L_f\sigma_f + \rho^2\|M\|^2\lambda_{\min}(MM^T)},$$

then we have

$$\delta = \frac{2\rho\sigma_f\lambda_{\min(MM^T)}}{L_f\sigma_f + \rho^2\|M\|^2\lambda_{\min}(MM^T)}.$$

Condition (C.17) simplifies to $1 - \delta \geq 0$. As we pick

$$\rho = \sqrt{\frac{L_f \sigma_f}{\|M\|^2 \lambda_{\min}(MM^T)}},$$

it maximizes δ as

$$\delta_{\max} = \frac{1}{\kappa_M \sqrt{\kappa_f}}.$$

Recall that κ_M is the condition number of M and κ_f is the condition number of f. We must have $\kappa_M \geq 1$, $\kappa_f \geq 1$, and $\delta_{\max} \leq 1$. Condition (C.17) is naturally satisfied for any δ. This completes the proof of lemma 4.6.3. □

BIBLIOGRAPHY

[1] Awesome gpt-3. `https://github.com/elyase/awesome-gpt3`, 2020.

[2] Openai api. `https://beta.openai.com/`, 2020.

[3] Alekh Agarwal, Sahand Negahban, and Martin J Wainwright. Fast global convergence of gradient methods for high-dimensional statistical recovery. *The Annals of Statistics*, 40(5):2452–2482, 2012.

[4] Ahmed Alaoui and Michael W Mahoney. Fast randomized kernel ridge regression with statistical guarantees. In *Advances in Neural Information Processing Systems*, 2015.

[5] Panos J Antsaklis and Anthony N Michel. *Linear systems*. Birkhäuser, 2006.

[6] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 10.0.*, 2022.

[7] Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on Learning Theory*, 2013.

[8] Amir Beck. *First-order methods in optimization*. SIAM, 2017.

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

[10] Dimitri Bertsekas, Angelia Nedic, and Asuman Ozdaglar. *Convex analysis and optimization*, volume 1. Athena Scientific, 2003.

[11] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

[12] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3:1–122, 2011.

[13] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends® in Machine Learning*, 8(3-4):231–357, 2015.

[16] Aydin Buluc, Tamara G Kolda, Stefan M Wild, Mihai Anitescu, Anthony Degennaro, John D Jakeman, Chandrika Kamath, Ramakrishnan-Ramki Kannan, Miles E Lopes, Per-Gunnar Martinsson, et al. Randomized algorithms for scientific computing (RASC). Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States), 2021.

[17] Emmanuel Candes and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012.

[18] Emmanuel Candes and Justin Romberg. Sparsity and incoherence in compressive sampling. *Inverse problems*, 23(3):969, 2007.

[19] Yair Censor, Aviv Gibali, and Simeon Reich. The subgradient extragradient method for solving variational inequalities in hilbert space. *Journal of Optimization Theory and Applications*, 148(2):318–335, 2011.

[20] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of mathematical imaging and vision*, 40:120–145, 2011.

[21] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):1–27, 2011.

[22] Chao-Kai Chiang, Tianbao Yang, Chia-Jung Lee, Mehrdad Mahdavi, Chi-Jen Lu, Rong Jin, and Shenghuo Zhu. Online optimization with gradual variations. In *Conference on Learning Theory*, pages 1–6, 2012.

[23] Agniva Chowdhuri, Palma London, Haim Avron, and Petros Drineas. Speeding up linear programming using randomized linear algebra. In *Advances in Neural Information Processing Systems*, 2020.

[24] Michael B Cohen, Jelani Nelson, and David P Woodruff. Optimal approximate matrix product in terms of stable rank. In *43rd International Colloquium on Automata, Languages, and Programming*, 2016.

[25] Constantinos Daskalakis, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. Training GANs with optimism. In *International Conference on Learning Representations*, 2018.

[26] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In *Advances in Neural Information Processing Systems*, 2014.

[27] Wei Deng and Wotao Yin. On the global and linear convergence of the generalized alternating direction method of multipliers. *Journal of Scientific Computing*, 66(3):889–916, 2016.

[28] Michal Derezinski, Jonathan Lacotte, Mert Pilanci, and Michael W Mahoney. Newton-less: Sparsification without trade-offs for the sketched newton update. *Advances in Neural Information Processing Systems*, 34:2835–2847, 2021.

[29] Michal Derezinski, Feynman T. Liang, Zhenyu Liao, and Michael W. Mahoney. Precise expressions for random projections: Low-rank approximation and randomized Newton. In *Advances in Neural Information Processing Systems*, 2020.

[30] Steven Diamond and Stephen Boyd. Cvxpy: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

[31] Jim Douglas and Henry H Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American mathematical Society*, 82:421–439, 1956.

[32] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[33] Iain Dunning, Joey Huchette, and Miles Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[34] Jonathan Eckstein. Splitting methods for monotone operators with applications to parallel optimization. *PhD thesis, MIT*, 1989.

[35] Jonathan Eckstein and Dimitri P Bertsekas. On the douglas-rachford splitting method and the proximal point algorithm for maximal monotone operators. *Mathematical Programming*, 55:293–318, 1992.

[36] Jonathan Eckstein and Masao Fukushima. Some reformulations and applications of the alternating direction method of multipliers. *Large Scale Optimization: State of the Art*, pages 119–138, 1993.

[37] Jonathan Eckstein and Wang Yao. Approximate versions of the alternating direction method of multipliers. *Optimization Online*, 2016.

[38] Jonathan Eckstein and Wang Yao. Relative-error approximate versions of douglas–rachford splitting and special cases of the admm. *Mathematical Programming*, 170(2):417–444, 2018.

[39] W Fenchel. *Convex cones, sets and functions, mimeographed notes*. Princeton University, 1953.

[40] Kimon Fountoulakis, Jacek Gondzio, and Pavel Zhlobich. Matrix-free interior point method for compressed sensing problems. *Mathematical Programming Computation*, 6(1):1–31, 2014.

[41] Zachary Frangella, Joel A. Tropp, and Madeleine Udell. Randomized Nyström preconditioning. *arXiv preprint arXiv:2110.02820*, 2021.

[42] Zachary Frangella, Shipu Zhao, Theo Diamandis, Bartolomeo Stellato, and Madeleine Udell. On the (linear) convergence of generalized newton inexact admm. *arXiv preprint arXiv:2302.03863*, 2023.

[43] Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1, 2010.

[44] Masao Fukushima. Application of the alternating direction method of multipliers to separable convex programming problems. *Computational Optimization and Applications*, 1:93–111, 1992.

[45] Gauthier Gidel, Hugo Berard, Gaëtan Vignoud, Pascal Vincent, and Simon Lacoste-Julien. A variational inequality perspective on generative adversarial networks. In *International Conference on Learning Representations*, 2019.

[46] James H. Goodnight. A tutorial on the sweep operator. *The American Statistician*, 33:149–158, 1979.

[47] Robert M Gower, Dmitry Kovalev, Felix Lieder, and Peter Richtárik. RSN: randomized subspace Newton. In *Advances in Neural Information Processing Systems*, 2019.

[48] Michael Grant and Stephen Boyd. Graph implementations for nonsmooth convex programs. In *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008.

[49] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, 2014.

[50] Bingsheng He and Xiaoming Yuan. On the o(1/n) convergence rate of the Douglas–Rachford alternating direction method. *SIAM Journal on Numerical Analysis*, 50(2):700–709, 2012.

[51] Ben Hermans, Andreas Themelis, and Panagiotis Patrinos. QPALM: a Newton-type proximal augmented Lagrangian method for quadratic programs. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4325–4330. IEEE, 2019.

[52] Ben Hermans, Andreas Themelis, and Panagiotis Patrinos. QPALM: a proximal augmented Lagrangian method for nonconvex quadratic programs. *Mathematical Programming Computation*, pages 1–45, 2022.

[53] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex Analysis and Minimization Algorithms II*, volume 305. Springer, 1993.

[54] Bin Hu, Peter Seiler, and Laurent Lessard. Analysis of biased stochastic gradient descent using sequential semidefinite programs. *Mathematical Programming*, pages 1–26, 2020.

[55] E. W. Kamen, P. P. Khargonekar, and K. R. Poolla. A transfer-function approach to linear time-varying discrete-time systems. *SIAM Journal on Control and Optimization*, 23(4):550–565, 1985.

[56] Sai Praneeth Karimireddy, Sebastian U. Stich, and Martin Jaggi. Global linear convergence of Newton's method without strong-convexity or lipschitz gradients, 2018.

[57] Jonathan Lacotte and Mert Pilanci. Effective dimension adaptive sketching methods for faster regularized least-squares optimization. In *Advances in Neural Information Processing Systems*, 2020.

[58] Jonathan Lacotte and Mert Pilanci. Fast convex quadratic optimization solvers with adaptive sketching-based preconditioners. *arXiv preprint arXiv:2104.14101*, 2021.

[59] Jonathan Lacotte, Yifei Wang, and Mert Pilanci. Adaptive Newton sketch: Linear-time optimization with quadratic convergence and effective Hessian dimensionality. In *International Conference on Machine Learning*, pages 5926–5936. PMLR, 2021.

[60] Benoît Legat, Oscar Dowson, Joaquim Dias Garcia, and Miles Lubin. MathOptInterface: a data structure for mathematical optimization problems. *INFORMS Journal on Computing*, 34(2):672–689, 2021.

[61] Laurent Lessard, Benjamin Recht, and Andrew Packard. Analysis and design of optimization algorithms via integral quadratic constraints. *SIAM Journal on Optimization*, 26(1):57–95, 2016.

[62] Xudong Li, Defeng Sun, and Kim-Chuan Toh. A highly efficient semismooth newton augmented lagrangian method for solving lasso problems. *SIAM Journal on Optimization*, 28(1):433–458, 2018.

[63] Pierre-Louis Lions and Bertrand Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[64] Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989.

[65] Po-Ling Loh. On lower bounds for statistical learning theory. *Entropy*, 19(11):617, 2017.

[66] Po-Ling Loh and Martin J Wainwright. Regularized M-estimators with nonconvexity: Statistical and algorithmic theory for local optima. *Journal of Machine Learning Research*, 16(19):559–616, 2015.

[67] Yu Malitsky. Projected reflected gradient methods for monotone variational inequalities. *SIAM Journal on Optimization*, 25(1):502–520, 2015.

[68] Per-Gunnar Martinsson and Joel A Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

[69] Maike Meier and Yuji Nakatsukasa. Randomized algorithms for Tikhonov regularization in linear least squares. *arXiv preprint arXiv:2203.07329*, 2022.

[70] Xiangrui Meng, Michael A Saunders, and Michael W Mahoney. LSRN: A parallel iterative solver for strongly over-or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

[71] Patrick Kofod Mogensen and Asbjørn Nilsen Riseth. Optim: A mathematical optimization package for Julia. *Journal of Open Source Software*, 3(24):615, 2018.

[72] J. Mohammadpour and C.W. Scherer. *Control of linear parameter varying systems with applications*. Springer New York, 2012.

[73] Renato DC Monteiro and Benar F Svaiter. Iteration-complexity of block-decomposition algorithms and the alternating direction method of multipliers. *SIAM Journal on Optimization*, 23(1):475–507, 2013.

[74] Jean Jacques Moreau. Décomposition orthogonale d'un espace hilbertien selon deux cônes mutuellement polaires. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 255:238–240, 1962.

[75] Yurii Nesterov. Gradient methods for minimizing composite functions. *Mathematical programming*, 140(1):125–161, 2013.

[76] Yurii Nesterov. *Lectures on convex optimization*. Springer, 2018.

[77] Robert Nishihara, Laurent Lessard, Ben Recht, Andrew Packard, and Michael Jordan. A general analysis of the convergence of admm. In *International Conference on Machine Learning*, pages 343–352. PMLR, 2015.

[78] Hua Ouyang, Niao He, Long Tran, and Alexander Gray. Stochastic alternating direction method of multipliers. In *International Conference on Machine Learning*, pages 80–88. PMLR, 2013.

[79] Yuyuan Ouyang, Yunmei Chen, Guanghui Lan, and Eduardo Pasiliao. An accelerated linearized alternating direction method of multipliers. *SIAM Journal on Imaging Sciences*, 8(1):644–681, 2015.

[80] Daniel O'Connor and Lieven Vandenberghe. On the equivalence of the primal-dual hybrid gradient method and douglas–rachford splitting. *Mathematical Programming*, 179:85–108, 2020.

[81] Brendan O'Donoghue and Emmanuel Candes. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics*, 15(3):715–732, 2015.

[82] Neal Parikh and Stephen Boyd. Proximal algorithms. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.

[83] Mert Pilanci and Martin J Wainwright. Newton sketch: A near linear-time optimization algorithm with linear-quadratic convergence. *SIAM Journal on Optimization*, 27(1):205–245, 2017.

[84] John Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*, 1998.

[85] Leonid Denisovich Popov. A modification of the arrow-hurwicz method for search of saddle points. *Mathematical notes of the Academy of Sciences of the USSR*, 28(5):845–848, 1980.

[86] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems*, 2008.

[87] Ali Rahimi and Benjamin Recht. Uniform approximation of functions with random bases. In *2008 46th Annual Allerton Conference on Communication, Control, and Computing*, 2008.

[88] Alexander Rakhlin and Karthik Sridharan. Online learning with predictable sequences. volume 30 of *Proceedings of Machine Learning Research*, pages 993–1019, 2013.

[89] Vladimir Rokhlin and Mark Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proceedings of the National Academy of Sciences*, 105(36):13212–13217, 2008.

[90] Ernest K Ryu and Stephen Boyd. Primer on monotone operator methods. *Appl. Comput. Math*, 15(1):3–43, 2016.

[91] Ernest K Ryu and Wotao Yin. *Large-scale convex optimization: algorithms & analyses via monotone operators*. Cambridge University Press, 2022.

[92] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *Mathematical Programming*, 162(1-2):83–112, 2017.

[93] Mark Schmidt, Nicolas Roux, and Francis Bach. Convergence rates of inexact proximal-gradient methods for convex optimization. *Advances in Neural Information Processing Systems*, 24, 2011.

[94] Shreya Shankar, Bora Uyumazturk, Devin Stein, Gulan, and Michael Lavelle. Gpt-3 sandbox. `https://github.com/shreyashankar/gpt3-sandbox`, 2020.

[95] Xinyue Shen, Steven Diamond, Madeleine Udell, Yuantao Gu, and Stephen Boyd. Disciplined multi-convex programming. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 895–900. IEEE, 2017.

[96] Neil Sloane. The on-line encyclopedia of integer sequences. `https://oeis.org/`, 1996.

[97] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[98] Zhendong Sun. *Switched linear systems: control and design*. Springer Science & Business Media, 2006.

[99] Adrien Taylor, Bryan Van Scoy, and Laurent Lessard. Lyapunov functions for first-order methods: Tight automated convergence guarantees. In *International Conference on Machine Learning*, pages 4897–4906. PMLR, 2018.

[100] Lloyd N Trefethen and David Bau III. *Numerical linear algebra*, volume 50. SIAM, 1997.

[101] Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. Fixed-rank approximation of a positive-semidefinite matrix from streaming data. In *Advances in Neural Information Processing Systems*, 2017.

[102] Michael J. Tsatsomeros. Principal pivot transforms: properties and applications. *Linear Algebra and its Applications*, 307:151–165, 2000.

[103] Madeleine Udell, Karanveer Mohan, David Zeng, Jenny Hong, Steven Diamond, and Stephen Boyd. Convex optimization in julia. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28. IEEE, 2014.

[104] Bryan Van Scoy, Randy A Freeman, and Kevin M Lynch. The fastest known globally convergent first-order method for minimizing strongly convex functions. *IEEE Control Systems Letters*, 2(1):49–54, 2017.

[105] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013.

[106] FP Vasilyev, EV Khoroshilova, and AS Antipin. An extragradient method for finding the saddle point in an optimal control problem. *Moscow University Computational Mathematics and Cybernetics*, 34(3):113–118, 2010.

[107] Zaiwen Wen, Donald Goldfarb, and Wotao Yin. Alternating direction augmented lagrangian methods for semidefinite programming. *Mathematical Programming Computation*, 2(3-4):203–230, 2010.

[108] Robert L Williams, Douglas A Lawrence, et al. *Linear state-space control systems*. John Wiley & Sons, 2007.

[109] David P Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends® in Theoretical Computer Science*, 10(1–2):1–157, 2014.

[110] Xiaoming Yuan, Shangzhi Zeng, and Jin Zhang. Discerning the linear convergence of admm for structured convex optimization through the lens of variational analysis. *J. Mach. Learn. Res.*, 21:83–1, 2020.

[111] Hangrui Yue, Qingzhi Yang, Xiangfeng Wang, and Xiaoming Yuan. Implementing the alternating direction method of multipliers for big datasets: A case study of least absolute shrinkage and selection operator. *SIAM Journal on Scientific Computing*, 40(5):A3121–A3156, 2018.

[112] Shipu Zhao, Zachary Frangella, and Madeleine Udell. NysADMM: faster composite convex optimization via low-rank approximation. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 26824–26840. PMLR, 17–23 Jul 2022.

[113] Shipu Zhao, Laurent Lessard, and Madeleine Udell. An automatic system to detect equivalence between iterative algorithms. *arXiv preprint arXiv:2105.04684*, 2021.