Title
Asynchronous Parallel Coordinate Update Methods for Large Scale Problems

Permalink
https://escholarship.org/uc/item/1g16x4cb

Author
Peng, Zhimin

Publication Date
2016

Peer reviewed|Thesis/dissertation
Asynchronous Parallel Coordinate Update Methods for Large Scale Problems

A dissertation submitted in partial satisfaction of the requirements for the degree Doctor of Philosophy in Mathematics

by

Zhimin Peng

2016
ABSTRACT OF THE DISSERTATION

Asynchronous Parallel Coordinate Update Methods for Large Scale Problems

by

Zhimin Peng

Doctor of Philosophy in Mathematics

University of California, Los Angeles, 2016

Professor Wotao Yin, Chair

This thesis focuses on coordinate update methods (CU), which are useful for solving problems involving large or high-dimensional datasets. Coordinate update methods decompose a problem into simple subproblems, where each updates one, or a small block of, variables while fixing others. These methods can deal with linear and nonlinear mappings, smooth and nonsmooth functions, as well as convex and nonconvex problems. In addition, they are easy to parallelize.

The great performance of coordinate update methods depends on solving simple subproblems. To derive simple subproblems for several new classes of applications, this thesis systematically studies coordinate friendly (CF) operators that perform low-cost coordinate updates. Based on the discovered coordinate friendly operators, as well as operator splitting techniques, I obtained new coordinate update algorithms for a variety of problems in machine learning, image processing, as well as sub-areas of optimization. Several problems are treated with coordinate update for the first time in history.

CU can be further scaled up to solve larger problems through asynchronous parallel (async-parallel) techniques. Asynchrony is crucial to parallel computing since it reduces synchronization wait, relaxes communication bottleneck, and thus
speeds up computing significantly. I proposed ARock, an algorithmic framework in which multiple agents (machines, processors, or cores) update coordinates in an asynchronous parallel fashion. At each step of ARock, an agent updates a randomly selected coordinate based on possibly out-of-date information. Convergence results are established for solving fixed-point problems. Numerical tests on regression and equations show that ARock significantly outperforms its synchronous counterpart.

The other novel contribution of this thesis is the introduction of an abstract framework for implementing asynchronous parallel algorithms on shared memory platforms. The programming model adopts the thread library from C++ at its lowest level. At the highest level, the goal is to enable fast prototyping of async-parallel algorithms. I developed a multilevel approach to reduce the gap between expert to low-level programming and novice-level programming. A spectrum of applications have been implemented under this framework.
The dissertation of Zhimin Peng is approved.

Luminita Aura Vese

Lieven Vandenberghe

Jingsheng Jason Cong

Wotao Yin, Committee Chair

University of California, Los Angeles

2016
# Table of Contents

1 **Introduction** ......................................................... 1  
   1.1 Coordinate Update Algorithmic Frameworks  ................. 4  
      1.1.1 Sequential Update  ..................................... 4  
      1.1.2 Parallel Update  ...................................... 6  
   1.2 Some Key Concepts of Operators  ........................... 9  
   1.3 Overview of Multi-core Computer Systems ................. 11  
      1.3.1 Shared Memory Systems  .............................. 11  
      1.3.2 Distributed Memory Systems  ......................... 12  
   1.4 Outline .......................................................... 12  

2 **Coordinate Friendly Operators** ........................... 15  
   2.1 Single Coordinate Friendly Operators ..................... 16  
      2.1.1 Examples of CF Operators  .......................... 20  
   2.2 Composite Coordinate Friendly Operators ................. 22  
      2.2.1 Combinations of Operators  .......................... 22  
      2.2.2 Operator Splitting Schemes  ........................ 27  
      2.2.3 Examples in Optimization  .......................... 29  
   2.3 Primal-dual Coordinate Friendly Operators ............... 31  
      2.3.1 Primal-dual Coordinate Update Algorithms .......... 33  
      2.3.2 Extended Monotropic Programming ................... 34  
      2.3.3 Overlapping-Block Coordinate Updates .............. 36  

3 **ARock: Asynchronous Parallel Coordinate Update Framework** 38
3.1 Introduction .................................................. 38
  3.1.1 On Random Coordinate Selection ...................... 42
  3.1.2 Uncoordinated Memory Access ......................... 42
  3.1.3 Block Coordinate ....................................... 45
  3.1.4 Straightforward Generalization ......................... 45
  3.1.5 Special Cases .......................................... 46
  3.1.6 Related work ........................................... 46
  3.1.7 Notation ................................................. 49
3.2 Convergence .................................................. 49
  3.2.1 Almost Sure Convergence .............................. 49
  3.2.2 Linear Convergence .................................... 55

4 Applications ................................................... 60
  4.1 Linear Equations .......................................... 60
  4.2 Convex Smooth Optimization ............................. 61
  4.3 Convex Nonsmooth Optimization .......................... 62
    4.3.1 Minimize Smooth + Nonsmooth Functions ............ 62
    4.3.2 Minimize Nonsmooth + Nonsmooth Functions ........ 64
    4.3.3 Group Lasso .......................................... 65
    4.3.4 Empirical Risk Minimization (ERM) .................. 66
    4.3.5 DRS for Image Processing in the Primal-dual Form ... 67
    4.3.6 Feasibility Problem .................................. 70
    4.3.7 3D Mesh Denoising .................................. 71
  4.4 Constrained Optimization ................................ 72
    4.4.1 Support Vector Machine .............................. 72
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>Parallel Experiments</td>
<td>98</td>
</tr>
<tr>
<td>6.2.1</td>
<td>Linear Equations</td>
<td>99</td>
</tr>
<tr>
<td>6.2.2</td>
<td>Elastic Net Regularized Logistic Regression</td>
<td>101</td>
</tr>
<tr>
<td>6.3</td>
<td>Distributed Experiments</td>
<td>105</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion and Future Works</td>
<td>106</td>
</tr>
<tr>
<td>8</td>
<td>Appendix</td>
<td>107</td>
</tr>
<tr>
<td>8.1</td>
<td>Derivation of ADMM from the DRS update</td>
<td>107</td>
</tr>
<tr>
<td>8.2</td>
<td>Representing the Condat-Vũ Algorithm as a Nonexpansive Operator</td>
<td>108</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>110</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

1.1 Gradient descent: the coordinate updates are faster than the full update since the former can take larger steps at each step. ... 7
1.2 Sync-parallel computing (left) versus async-parallel computing (right). On the left, all the agents must wait at idle (white boxes) until the slowest agent has finished. ... 8
1.3 Architectures for UMA and NUMA systems. ... 12
1.4 Architectures for distributed memory systems. ... 13

3.1 Sync-parallel computing (left) versus async-parallel computing (right). ... 38
3.2 Consistent read versus inconsistent read: A demonstration. ... 43

5.1 Architectures for distributed memory systems. Linear algebra serves as the foundation for ARock. Other layers are structured according to the discussions in Chapter 2 and Chapter 3. ... 89
5.2 Relationship between workers, controller and shared memory. ... 94

6.1 Compare the convergence of 3S full update with 3S coordinate update algorithms. ... 97
6.2 CT image reconstruction. ... 98
6.3 Sparsity patterns of the three datasets. ... 100
6.4 Residue vs number of epochs. ... 101
6.5 Speedup vs wall clock time. ... 101
6.6 The distribution of coordinate sparsity. Each dot represents the total number of nonzeros in the vectors \( a_i \) that correspond to each coordinate. ... 102
6.7 Objective vs wall clock time. ................................. 103
6.8 Speedup vs number of threads. ............................... 103
6.9 Profile of sync-parallel and async-parallel: red indicates waiting
time, and brown indicates computing time. .................... 104
\section*{List of Tables}

2.1 $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits the weaker separability property from those of $\mathcal{T}_1$
\hspace{1cm} and $\mathcal{T}_2$. ................................. 25

2.2 Summary of how $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits CF properties from those of $\mathcal{T}_1$
\hspace{1cm} and $\mathcal{T}_2$. ................................. 25

5.1 Proximal operator. ................................. 91

5.2 Projection operator. ................................. 91

5.3 Forward operator. ................................. 92

6.1 Two datasets for portfolio optimization ................................. 96

6.2 Linear equation datasets ................................. 99

6.3 Two datasets for sparse logistic regression. ................................. 102

6.4 Results for distributed Lasso test. ................................. 105
ACKNOWLEDGMENTS

I wish to express my deepest appreciation to my family, friends, collaborators, and committee, but in particular to my advisor Professor Wotao Yin. He has been a tremendous mentor for me. He was always encouraging, supportive, and extremely patient in all situations. It has been a fun and pleasant five year journey with him.

I would also like to thank my committee, Prof. Jason Cong, Prof. Lieven Vandenberghe and Prof. Luminita Vese for their advise, feedback and support. Your knowledge and insights greatly enriched my work.

I am also indebted to my collaborators, including Dr. Wei Deng, Brent Edmunds, Prof. Ming-Jun Lai, Tianyu Wu, Dr. Yangyang Xu, Prof. Ming Yan, and Prof. Jinshan Zeng. I enjoyed the interaction and collaboration with them.

Furthermore, I am filled with gratitude to the following list of people: Etienne Ackermann, Sebastian Acosta, Yat Tin Chow, Damek Davis, Robert Hannah, Lanchao Liu, David Medina, and Kun Yuan.

Finally, I am very grateful for my family back home for their unconditional love and support. Most importantly to me, I would like to express my appreciation to Xinlu for her constant help, understanding and her faith in me.
Vita

2011 Bachelor of Science (Computational Mathematics),
Xi’an Jiaotong University, Xi’an, China.

2013 Master of Arts (Computational & Applied Mathematics),
Rice University, Houston, TX.

2013 – 2016 Teaching and Research Assistant, Department of Mathematics
University of California, Los Angeles, Los Angeles, CA.

Publications


CHAPTER 1

Introduction

This thesis studies *coordinate update methods*, which reduce a large problem to smaller subproblems and are useful for solving large-sized problems. These methods handle both linear and nonlinear maps, smooth and nonsmooth functions, and convex and nonconvex problems. The common special examples of these methods are the Jacobian and Gauss-Seidel algorithms for solving linear systems, and they are also commonly used for solving differential equations and optimization problems (e.g., *coordinate descent*).

After coordinate update methods were initially introduced in each topic area, their evolution had been slow until recently, when data-driven applications (e.g., in signal processing, image processing, and statistical and machine learning) impose strong demand for scalable numerical solutions; consequently, numerical methods of *small footprints*, including coordinate update methods, become increasingly popular. These methods are generally applicable to many problems involving large or high-dimensional datasets.

Coordinate update methods generate simple subproblems that update one variable, or a small block of variables, while fixing others. The variables can be updated in the *cyclic*, *random*, or *greedy* orders, which can be selected to adapt to the problem. The subproblems that perform coordinate updates also have different forms. Coordinate updates can be applied either sequentially on a single thread (or process) or concurrently on multiple threads (or processes), or even in an asynchronous parallel fashion. They have been demonstrated to give rise to
very powerful and scalable algorithms.

Clearly, the strong performance of coordinate update methods relies on solving simple subproblems. The cost of each subproblem must be proportional to how many coordinates it updates. When there are totally \( m \) coordinates, the cost of updating one coordinate should not exceed \( \frac{1}{m} \) of the cost of the full update (made to all the coordinates at once). Otherwise, coordinate update is not computationally worthy. For example, let \( f : \mathbb{R}^m \to \mathbb{R} \) be a \( C^2 \) function, and consider the Newton update \( x^{k+1} \leftarrow x^k - (\nabla^2 f(x^k))^{-1} \nabla f(x^k) \). Since updating each \( x_i \) (keeping others fixed) still requires forming the Hessian matrix \( \nabla^2 f(x) \) (at least \( O(n^2) \) operations) and factorizing it (\( O(n^3) \) operations), there is little to save in computation compared to updating all the components of \( x \) at once; hence, the Newton's method is generally not amenable to coordinate update.

The recent coordinate update literature has introduced new algorithms. However, they are primarily applied to a few, albeit important, classes of problems that arise in machine learning. For many complicated problems, it remains open whether simple subproblems can be obtained. We provide positive answers to several new classes of applications and introduce their coordinate update algorithms. Therefore, one of the focuses of this thesis is to build a set of tools for deriving simple subproblems and extending coordinate updates to new territories of applications.

We will frame each application into an equivalent fixed-point problem

\[
x = \mathcal{T} x
\]  

(1.1)

by specifying the operator \( \mathcal{T} : \mathbb{H} \to \mathbb{H} \), where \( x = (x_1, \ldots, x_m) \in \mathbb{H} \), and \( \mathbb{H} = \mathbb{H}_1 \times \cdots \times \mathbb{H}_m \) is a Hilbert space. In many cases, the operator \( \mathcal{T} \) itself represents an iteration:

\[
x^{k+1} = \mathcal{T} x^k
\]  

(1.2)

such that the limit of the sequence \( \{ x^k \} \) exists and is a fixed point of \( \mathcal{T} \), which
is also a solution to the application or from which a solution to the application can be obtained. We call the scheme (1.2) a full update, as opposed to updating one $x_i$ at a time. The scheme (1.2) has a number of interesting special cases including methods of gradient descent, gradient projection, proximal gradient, operator splitting, and many others.

We study the structures of $\mathcal{T}$ that make the following coordinate update algorithm computationally worthy

$$x_i^{k+1} = x_i^k - \eta_k (x^k - \mathcal{T} x^k)_i,$$

(1.3)

where $\eta_k$ is a step size and $i \in [m] := \{1, \ldots, m\}$ is arbitrary. Specifically, the cost of performing (1.3) is roughly $\frac{1}{m}$, or lower, of that of performing (1.2). We call such $\mathcal{T}$ a Coordinate Friendly (CF) operator, which we will formally define.

This thesis will explore a variety of CF operators. Single CF operators include linear maps, projections to certain simple sets, proximal maps and gradients of (nearly) separable functions, as well as gradients of sparsely supported functions. There are many more composite CF operators, which are built from single CF and non-CF operators under a set of rules. The fact that some of these operators are CF is not obvious.

The developed coordinate update algorithms are easy to parallelize. In addition, the work in this thesis gives rise to parallel and asynchronous extensions to existing algorithms including the Alternating Direction Method of Multipliers (ADMM), primal-dual splitting algorithms, and others. The asynchronous extensions will be discussed in Chapter 3.

Throughout this thesis, all functions $f, g, h$ are proper closed and can take the extended value $\infty$, and all sets $X, Y, Z$ are nonempty closed sets. The indicator function $\iota_X(x)$ returns 0 if $x \in X$, and $\infty$ elsewhere. For a positive integer $m$, we let $[m] := \{1, \ldots, m\}$. 

3
1.1 Coordinate Update Algorithmic Frameworks

This subsection reviews the sequential and parallel algorithmic frameworks for coordinate updates, as well as the relevant literature.

The general framework of coordinate update is the Algorithm 1. Next we review the index rules and the methods to update $x_i$.

**Algorithm 1:** Framework of coordinate update

**Input:** $k \leftarrow 0$ and initialize $x^0 = H = H_1 \times \cdots \times H_m$

**while not converged do**

- select an index $i_k \in [m]$;
- update $x_i^{k+1}$ for $i = i_k$ while keeping $x_i^{k+1} = x_i^k$, $\forall i \neq i_k$;
- $k \leftarrow k + 1$;

1.1.1 Sequential Update

In this framework, there is a sequence of coordinate indices $i_1, i_2, \ldots$ chosen according to one of the following rules: cyclic, cyclic permutation, random, and greedy rules. At iteration $k$, only the $i_k$th coordinate is updated:

$$\begin{cases} 
  x_i^{k+1} = x_i^k - \eta_k (x^k - T x^k)_i, & i = i_k, \\
  x_i^{k+1} = x_i^k, & \text{for all } i \neq i_k.
\end{cases}$$

Sequential updates have been applied to many problems such as the Gauss-Seidel iteration for solving a linear system of equations, alternating projection [105, 7] for finding a point in the intersection of two sets, ADMM [44, 43] for solving monotropic programs, and Douglas-Rachford Splitting (DRS) [34] for finding a zero to the sum of two operators.

In optimization, coordinate descent algorithms, at each iteration, minimize the function $f(x_1, \ldots, x_m)$ by fixing all but one variable $x_i$. Let

$$x_{i-} := (x_1, \ldots, x_{i-1}), \quad x_{i+} = (x_{i+1}, \ldots, x_m)$$
collect all but the $i$th coordinate of $x$. Coordinate descent solves one of the following subproblems:

$$\mathcal{T} x^k_i = \arg \min_{x_i} f(x_{i-}, x_i, x_{i+}^k), \quad (1.4a)$$

$$\mathcal{T} x^k_i = \arg \min_{x_i} f(x_{i-}, x_i, x_{i+}^k) + \frac{1}{2\eta_k} \|x_i - x_i^k\|^2, \quad (1.4b)$$

$$\mathcal{T} x^k_i = \arg \min_{x_i} \langle \nabla_i f(x^k), x_i \rangle + \frac{1}{2\eta_k} \|x_i - x_i^k\|^2, \quad (1.4c)$$

$$\mathcal{T} x^k_i = \arg \min_{x_i} \langle \nabla_i f_{\text{diff}}(x^k), x_i \rangle + f_{\text{prox}}^i(x_i) + \frac{1}{2\eta_k} \|x_i - x_i^k\|^2, \quad (1.4d)$$

which are called direct update, proximal update, gradient update, and prox-gradient update, respectively. The last update applies to the function

$$f(x) = f_{\text{diff}}(x) + \sum_{i=1}^m f_{\text{prox}}^i(x_i),$$

where $f_{\text{diff}}$ is differentiable and each $f_{\text{prox}}^i$ is proximable (its proximal map takes $O\left(\dim(x_i)\text{polylog}(\dim(x_i))\right)$ operations to compute).

**Sequential-update literature.** Coordinate descent algorithms date back to the 1950s [49], when the cyclic index rule was used. Its convergence has been established under a variety of cases, for both convex and nonconvex objective functions; see [107, 118, 83, 46, 65, 97, 45, 98, 84, 11, 51, 109]. Proximal updates are studied in [45, 3] and developed into prox-gradient updates in [104, 103, 18] and mixed updates in [112].

The random index rule first appeared in [69] and then [87, 64]. Recently, [113, 111] compared the convergence speeds of cyclic and stochastic update-orders. The gradient update has been relaxed to stochastic gradient update for large-scale problems in [29, 114].

The greedy index rule leads to fewer iterations but is often impractical since it requires a lot of effort to calculate scores for all the coordinates. However, there are cases where calculating the scores is inexpensive [14, 60, 110] and the
save in the total number of iterations significantly outweighs the extra calculation
[104, 32, 79, 70].

A simple example. We present the coordinate update algorithms under
different index rules for solving a simple least squares problem:

\[
\min_x f(x) := \frac{1}{2} \|Ax - b\|^2,
\]

where \( A \in \mathbb{R}^{p \times m} \) and \( b \in \mathbb{R}^p \) are Gaussian random. The goal is to numerically
demonstrate the advantages of coordinate updates over the full update of gradient
descent:

\[
x^{k+1} = x^k - \eta_k A^T (Ax^k - b).
\]

The four tested index rules are: cyclic, cyclic permutation, random, and greedy
under the Gauss-Southwell\(^1\) rule. Note that because this example is very special,
the comparisons of different index rules are far from conclusive.

In the full update, the step size \( \eta_k \) is set to the theoretical upper bound \( \frac{2}{\|A\|_2^2} \),
where \( \|A\|_2 \) denotes the matrix operator norm and equals the largest singular
value of \( A \). For each coordinate update to \( x_i \), the step size \( \eta_k \) is set to \( \frac{1}{(A^T A)_{ii}} \).
All of the full and coordinate updates have the same per-epoch complexity, so we
plot the objective errors in Figure 1.1.

1.1.2 Parallel Update

As one of their main advantages, coordinate update algorithms are easy to parallelize. In this subsection, we discuss both synchronous (sync) and asynchronous (async) parallel updates.

Sync-parallel (Jacobi) update specifies a sequence of index subsets \( \mathbb{I}_1, \mathbb{I}_2, \ldots \subseteq [m] \), and at each iteration \( k \), the coordinates in \( \mathbb{I}_k \) are updated in parallel by multi-

\(^1\)it selects \( i_k = \arg \max_i \| \nabla_i f(x^k) \| \).
Figure 1.1: Gradient descent: the coordinate updates are faster than the full update since the former can take larger steps at each step.

tles agents:

\[
\begin{align*}
    x_i^{k+1} &= x_i^k - \eta_k (x^k - \mathcal{T} x^k)_i, & i \in \mathbb{I}_k, \\
    x_i^{k+1} &= x_i^k, & i \notin \mathbb{I}_k.
\end{align*}
\]

Synchronization across all agents ensures that all \(x_i\) in \(\mathbb{I}_k\) are updated and also written to the memory before the next iteration starts. Note that, if \(\mathbb{I}_k = [m]\) for all \(k\), then all the coordinates are updated and, thus, each iteration reduces to the full update: \(x^{k+1} = x^k - \eta_k (x^k - \mathcal{T} x^k)\).

Async-parallel update. In this setting, a set of agents still perform parallel updates, but synchronization is eliminated or weakened. Hence, each agent continuously applies (1.5), which reads \(x\) from and writes \(x_i\) back to the shared memory (or through communicating with other agents without shared memory):

\[
\begin{align*}
    x_i^{k+1} &= x_i^k - \eta_k ( (\mathbb{I} - \mathcal{T}) x^{k-d_k} )_i, & i = i_k, \\
    x_i^{k+1} &= x_i^k, & \text{for all } i \neq i_k.
\end{align*}
\]

Unlike before, \(k\) increases whenever any agent completes an update.

The lack of synchronization often results in computation with out-of-date information. During the computation of the \(k\)th update, other agents make \(d_k\) updates to \(x\) in the shared memory; when the \(k\)th update is written, its input is
already $d_k$ iterations out of date. This number is referred to as the asynchronous delay. In (1.5), the agent reads $x^{k-d_k}$ and commits the update to $x^{k}_{ik}$. Here we have assumed consistent reading, i.e., $x^{k-d_k}$ lying in the set $\{x^j\}_{j=1}^k$. This requires implementing a memory lock. Removing the lock can lead to inconsistent reading, which still has convergence guarantees.

Synchronization across all agents means that all agents will wait for the last (slowest) agent to complete. Async-parallel updates eliminate such idle time, spread out memory access and communication, and thus often run much faster. However, async-parallel is more difficult to analyze because of the asynchronous delay. Chapter 3 provides more details on the convergence results.


For fixed-point problems, async-parallel methods date back to [6] in 1978. In the pre-2010 methods [5, 13, 10, 35] and the review [41], each agent updates its own subset of coordinates. Convergence is established under the $P$-contraction condition and its variants [13]. Papers [10, 9] show convergence for async-parallel iterations with simultaneous reading and writing to the same set of components.
Unbounded but stochastic delays are considered in [95].


1.2 Some Key Concepts of Operators

In this section, we go over a few key concepts in monotone operator theory and operator splitting theory.

Definition 1 (monotone operator) A set-valued operator $\mathcal{T} : \mathbb{H} \Rightarrow \mathbb{H}$ is monotone if $\langle x - y, u - v \rangle \geq 0$, $\forall x, y \in \mathbb{H}$, $u \in \mathcal{T}x$, $v \in \mathcal{T}y$. Furthermore, $\mathcal{T}$ is maximally monotone if its graph $\text{Grph}(\mathcal{T}) = \{(x, u) \in \mathbb{H} \times \mathbb{H} : u \in \mathcal{T}x\}$ is not strictly contained in the graph of any other monotone operator.

Example 1 An important maximally monotone operator is the subdifferential $\partial f$ of a closed proper convex function $f$.

Definition 2 (nonexpansive operator) An operator $\mathcal{T} : \mathbb{H} \to \mathbb{H}$ is nonexpansive if $\|\mathcal{T}x - \mathcal{T}y\| \leq \|x - y\|$, $\forall x, y \in \mathbb{H}$. We say $\mathcal{T}$ is averaged, or $\alpha$-averaged, if there is one nonexpansive operator $\mathcal{R}$ such that $\mathcal{T} = (1 - \alpha)I + \alpha\mathcal{R}$ for some $0 < \alpha < 1$. A $\frac{1}{2}$-averaged operator $\mathcal{T}$ is also called firmly-nonexpansive.

By definition, a nonexpansive operator is single-valued. Let $\mathcal{T}$ be averaged. If $\mathcal{T}$ has a fixed point, the iteration (1.2) converges to a fixed point; otherwise, the iteration diverges unboundedly. Now let $\mathcal{T}$ be nonexpansive. The convergence is guaranteed [55] after damping: $x^{k+1} = x^k - \eta(x^k - \mathcal{T}x^k)$, for any $0 < \eta < 1$. 9
Example 2 A common firmly-nonexpansive operator is the resolvent of a maxi-
mally monotone map $T$, written as
\[ J_T := (I + T)^{-1}. \] (1.6)

Given $x \in \mathbb{H}$, $J_T(x) = \{ y : x \in y + Ty \}$. (By monotonicity of $T$, $J_T$ is a
singleton, and by maximality of $T$, $J_T(x)$ is well defined for all $x \in \mathbb{H}$. ) A
reflective resolvent is
\[ R_J := 2J_T - I. \] (1.7)

Definition 3 (proximal map) The proximal map for function $f$ is a special
resolvent and defined as:
\[ \text{prox}_{\gamma f}(y) = \arg \min_x \{ f(x) + \frac{1}{2\gamma} \| x - y \|^2 \}, \] (1.8)
where $\gamma > 0$. The first-order variational condition of the minimization yields
\[ \text{prox}_{\gamma f} = (I + \gamma \partial f)^{-1}; \] hence, \text{prox}_{\gamma f} is firmly-nonexpansive. When $x \in \mathbb{R}^m$ and
\[ \text{prox}_{\gamma f} \] can be computed in $O(m)$ or $O(m \log m)$, we call $f$ proximable.

Examples of proximal functions include $\ell_1, \ell_2, \ell_\infty$-norms, several matrix norms,
(piec-wise) linear functions, certain quadratic functions, and many more.

Example 3 A special proximal map is the projection map. Let $X$ be a nonempty
closed convex set, and $\iota_X$ be its indicator function. Minimizing $\iota_X(x)$ enforces
$x \in X$, so $\text{prox}_{\gamma \iota_X}$ reduces to the projection map $\text{proj}_X$ for any $\gamma > 0$. Therefore,$\text{proj}_X$ is also firmly nonexpansive.

Definition 4 ($\beta$-cocoercive operator) An operator $T : \mathbb{H} \to \mathbb{H}$ is $\beta$-cocoercive
if $\langle x - y, Tx - Ty \rangle \geq \beta \| Tx - Ty \|^2$, $\forall x, y \in \mathbb{H}$.

Example 4 A special example of cocoercive operator is the gradient of a smooth
function. Let $f$ be a differentiable function. Then $\nabla f$ is $\beta$-Lipschitz continuous
if and only if $\nabla f$ is $\frac{1}{\beta}$-cocoercive [8, Corollary 18.16].
Definition 5 ($\mu$-strongly monotone) Consider an operator $T : \mathbb{H} \to \mathbb{H}$. $T$ is $\mu$-strongly monotone, where $\mu > 0$, if it satisfies $\langle x - y, Tx - Ty \rangle \geq \mu \|x - y\|^2$, $\forall x, y \in \mathbb{H}$. When the inequality holds for $\mu = 0$, $T$ is monotone. $T$ is quasi-$\mu$-strongly monotone, where $\mu > 0$, if it satisfies $\langle x - y, Tx \rangle \geq \mu \|x - y\|^2$, $\forall x \in \mathbb{H}, y \in \text{zer} T := \{y \in \mathbb{H} \mid Ty = 0\}$. When the inequality holds for $\mu = 0$, $T$ is quasi-monotone.

Averaged operators are nonexpansive. By the Cauchy-Schwarz inequality, a $\beta$-cocoercive operator is $\frac{1}{\beta}$-Lipschitz; the converse is generally untrue, but true for the gradients of convex differentiable functions.

Definition 6 (Demicompact operator) $T$ is demicompact [81] at $x \in \mathbb{H}$ if for every bounded sequence $(x^k)_{k \geq 0}$ in $\mathbb{H}$ such that $Tx^k - x^k \to x$, there exists a strongly convergent subsequence.

1.3 Overview of Multi-core Computer Systems

A multi-core computer system uses two or more cores. The system can have multiple chips or multiple cores on a single chip. There are two major types of multi-core systems: shared-memory systems and distributed memory systems. In this thesis, we will discuss the realizations of CU methods on both shared memory systems and distributed memory systems.

1.3.1 Shared Memory Systems

In shared memory systems, multiple processors are connected to a memory system via an interconnection network, and each processor can access each memory location. These processors usually communicate with each other implicitly through shared data structures. Depending on the type of interconnect, shared memory systems can be classified into two types: uniform memory access (UMA) system
and nonuniform memory access (NUMA) system. UMA system connects all the processors directly to main memory. All processors in NUMA system have a direct connect to a block of main memory and they can access each others’ blocks memory through hardware built into the processors. Figure 1.3 shows the architectures for the two types of shared memory systems.

![Figure 1.3: Architectures for UMA and NUMA systems.](image)

1.3.2 Distributed Memory Systems

Distributed memory system consists of a collection of commodity processors which are connected by a interconnection network. Each processor has its own private memory and communicates others by explicitly sending messages through an interconnection network. Figure 1.4 demonstrates the architecture for distributed memory systems.

1.4 Outline

The rest of the thesis is organized as follows.

Chapter 2 discusses the CF properties found in both single and composite operators underlying many interesting applications. We introduce approaches to recognize CF operators and develop coordinate-update algorithms based on them.

\footnote{The figure is adapted from Figure 2.5 and Figure 2.6 in [74].}
Our analysis also provides guidance to the implementation of coordinate-update algorithms by specifying how to compute certain operators and maintain certain quantities in memory [77].

Chapter 3 introduces the asyc-parallel framework of coordinate update methods (ARock) for finding a fixed point to a nonexpansive operator. By introducing a new metric and establishing stochastic Fejér monotonicity, we show that, with probability one, ARock converges to a point in the solution set; linear convergence is obtained for quasi-strongly monotone operators [78].

Chapter 4 considers the applications of CF operators and ARock. We will discuss a variety of applications including, but not limited to linear equations, linear and second-order cone programming, empirical risk minimization, variational image processing, portfolio optimization, distributed computing and nonnegative matrix factorization.

Chapter 5 presents the software architecture of ARock - the abstract framework for implementing asynchronous algorithms on shared memory platform. A multilevel approach is developed to reduce the gap between expert to low-level multi-threading programming and novice-level programming.

Chapter 6 details the numerical results of several applications. We show the faster convergence of sequential coordinate update compared to its counterpart, i.e., full update method. Better scalability of ARock is demonstrated by com-
paring it against its sync-parallel counterpart. A distributed greedy coordinate update method is compared against the state-of-art distributed algorithms.
CHAPTER 2

Coordinate Friendly Operators

For convenience, we do not distinguish a coordinate from a block of coordinates. We assume our variable $x$ consists of $m$ coordinates:

$$x = (x_1, \ldots, x_m) \in \mathbb{H} := \mathbb{H}_1 \times \cdots \times \mathbb{H}_m \quad \text{and} \quad x_i \in \mathbb{H}_i, \ i = 1, \ldots, m.$$  

For simplicity, we assume that $\mathbb{H}_1, \ldots, \mathbb{H}_m$ are finite-dimensional real Hilbert spaces, though most results hold for general Hilbert spaces. A function maps from $\mathbb{H}$ to $\mathbb{R}$, the set of real numbers, and an operator maps from $\mathbb{H}$ to $\mathbb{G}$, where the definition of $\mathbb{G}$ depends on the context.

Our discussion often involves two points $x, x^+ \in \mathbb{H}$ that differ over one coordinate: there exists an index $i \in [m]$ and a point $\delta_i \in \mathbb{H}_i$ that is supported on $\mathbb{H}_i$, such that

$$x^+ = x + \delta_i.$$  

Note that $x_j^+ = x_j$ for all $j \neq i$.

**Definition 7 (number of operations)** We let $\mathcal{M}[a \mapsto b]$ denote the number of basic operations that it takes to compute the quantity $b$ from the input $a$.

For example, $\mathcal{M}[x \mapsto (\mathcal{T}x)_i]$ denotes the number of operations to compute the $i$th component of $\mathcal{T}x$ given $x$. We explore the possibility to compute $(\mathcal{T}x)_i$ with much fewer operations than what is needed to first compute $\mathcal{T}x$ and then take its $i$th component.
2.1 Single Coordinate Friendly Operators

This subsection studies a few classes of CF operators and then formally defines the CF operator. We motivate the first class through an example.

In the example below, we let $A_{i,:}$ and $A_{:j}$ be the $i$th row and $j$th column of a matrix $A$, respectively. Let $A^\top$ be the transpose of $A$ and $A^\top_{i,:}$ be $(A^\top)_{i,:}$, i.e., the $i$th row of the transpose of $A$.

**Example 5 (least squares)** Consider the least squares problem

$$\min_x f(x) := \frac{1}{2} \|Ax - b\|^2; \quad (2.2)$$

where $A \in \mathbb{R}^{p \times m}$ and $b \in \mathbb{R}^p$. In this example, assume that $m = \Theta(p)$, namely, $m$ and $p$ are of the same order. We compare the full update of gradient descent to its coordinate update.$^1$ The full update is referred to as the iteration $x^{k+1} = T x^k$ where $T$ is given by

$$T x := x - \eta \nabla f(x) = x - \eta A^\top A x + \eta A^\top b. \quad (2.3)$$

Assuming that $A^\top A$ and $A^\top b$ are already computed, we have $\mathcal{M}[x \mapsto T x] = O(m^2)$. The coordinate update at the $k$th iteration performs

$$x^{k+1}_{i_k} = (T x^k)_{i_k} = x^k_{i_k} - \eta \nabla_{i_k} f(x^k),$$

and $x^{k+1}_j = x^k_j$, for $j \neq i_k$, where $i_k$ is some selected coordinate.

Since for all $i$, $\nabla_i f(x^k) = (A^\top (Ax - b))_i = (A^\top A)_{i,:} \cdot x - (A^\top b)_i$, we have $\mathcal{M}[x \mapsto (T x)_i] = O(m)$ and thus $\mathcal{M}[x \mapsto (T x)_i] = O(\frac{1}{m} \mathcal{M}[x \mapsto T x])$. Therefore, the coordinate gradient descent is computationally worthy.

The operator $T$ in the above example is a special Type-I CF operator.

$^1$Although gradient descent is seldom used to solve least squares, it often appears as a part in first-order algorithms for problems involving a least squares term.
**Definition 8 (Type-I CF)** For an operator $\mathcal{T} : \mathbb{H} \to \mathbb{H}$, let $M[x \mapsto (\mathcal{T}x)_i]$ be the number of operations for computing the $i$th coordinate of $\mathcal{T}x$ given $x$ and $M[x \mapsto \mathcal{T}x]$ the number of operations for computing $\mathcal{T}x$ given $x$. We say $\mathcal{T}$ is Type-I CF (denoted as $\mathcal{F}_1$) if for any $x \in \mathbb{H}$ and $i \in [m]$, it holds

$$M[x \mapsto (\mathcal{T}x)_i] = O\left(\frac{1}{m} M[x \mapsto \mathcal{T}x]\right).$$

**Example 6 (least squares II)** We can implement the coordinate update in Example 5 in a different manner by maintaining $\mathcal{T}x_k$ in the memory. This approach works when $m = \Theta(p)$ or $p \gg m$. The full update (2.3) is unchanged. At each coordinate update, from the maintained quantity $\mathcal{T}x_k$, we immediately obtain $x_{ik}^{k+1} = (\mathcal{T}x_k)_{ik}$. But we need to update $\mathcal{T}x_k$ to $\mathcal{T}x_{k+1}$. Since $x_{k+1}$ and $x_k$ differ only over the coordinate $i_k$, this update can be computed as

$$\mathcal{T}x_{k+1} = \mathcal{T}x_k + (x_{ik}^{k+1} - x_{ik}^k)(A^\top A)_{:,i_k},$$

which is a scalar-vector multiplication followed by a vector addition, taking only $O(m)$ operations. Computing $\mathcal{T}x_{k+1}$ from scratch involves a matrix-vector multiplication, taking $O(M[x \mapsto \mathcal{T}(x)]) = O(m^2)$ operations. Therefore,

$$M[\{x^k, \mathcal{T}x^k, x^{k+1}\} \mapsto \mathcal{T}x^{k+1}] = O\left(\frac{1}{m} M[\{x^{k+1} \mapsto \mathcal{T}x^{k+1}\}]\right).$$

The operator $\mathcal{T}$ in the above example is a special Type-II CF operator.

**Definition 9 (Type-II CF)** An operator $\mathcal{T}$ is called Type-II CF (denoted as $\mathcal{F}_2$) if, for any $x, i, \delta, x^+$ satisfying (2.1), the following holds

$$M[\{x, \mathcal{T}x, x^{+}\} \mapsto \mathcal{T}x^{+}] = O\left(\frac{1}{m} M[\{x^+ \mapsto \mathcal{T}x^{+}\}]\right).$$

(2.4)

The next example illustrates an efficient coordinate update by maintaining certain quantity other than $\mathcal{T}x$. 

17
Example 7 (least squares III) For the case $p \ll m$, we should avoid pre-computing the relative large matrix $A^\top A$, and it is cheaper to compute $A^\top (Ax)$ than $(A^\top A)x$. Therefore, we change the implementations of both the full and coordinate updates in Example 5. In particular, the full update

$$x^{k+1} = T x^k = x^k - \eta \nabla f(x^k) = x^k - \eta A^\top (Ax^k - b),$$

pre-multiplies $x^k$ by $A$ and then $A^\top$. Hence, $\mathcal{M} \left[ x^k \mapsto T(x^k) \right] = O(mp)$.

We change the coordinate update to maintain the intermediate quantity $Ax^k$. In the first step, the coordinate update computes

$$(T x^k)_{i_k} = x^k_{i_k} - \eta (A^\top (Ax^k) - A^\top b)_{i_k},$$

by pre-multiplying $Ax^k$ by $A_{i_k,:}^\top$. Then, the second step updates $Ax^k$ to $Ax^{k+1}$ by adding $(x^{k+1}_{i_k} - x^k_{i_k}) A_{:,i_k}$ to $Ax^k$. Both steps take $O(p)$ operations, so

$$\mathcal{M} \left[ \{x^k, Ax^k \} \mapsto \{x^{k+1}, Ax^{k+1} \} \right] = O(p) = O \left( \frac{1}{m} \mathcal{M} \left[ x^k \mapsto T x^k \right] \right).$$

Combining Type-I and Type-II CF operators with the last example, we arrive at the following CF definition.

Definition 10 (CF operator) We say that an operator $T : \mathbb{H} \to \mathbb{H}$ is CF if, for any $x, i, \delta, x^+$ satisfying (2.1), the following holds

$$\mathcal{M} \left[ \{x, \mathcal{M}(x) \} \mapsto \{x^+, \mathcal{M}(x^+) \} \right] = O \left( \frac{1}{m} \mathcal{M} \left[ x \mapsto T x \right] \right), \quad (2.5)$$

where $\mathcal{M}(x)$ is some quantity maintained in the memory to facilitate each coordinate update and refreshed to $\mathcal{M}(x^+)$. $\mathcal{M}(x)$ can be empty, i.e., besides $x$, no other quantity is maintained.

The left-hand side of (2.5) measures the cost of performing one coordinate update (including the cost of updating $\mathcal{M}(x)$ to $\mathcal{M}(x^+)$) while the right-hand
side measures the average per-coordinate cost of updating all the coordinates together. When (2.5) holds, \( \mathcal{T} \) is amenable to coordinate updates.

By definition, a Type-I CF operator \( \mathcal{T} \) is CF without maintaining any quantity, i.e., \( \mathcal{M}(x) = \emptyset \).

A Type-II CF operator \( \mathcal{T} \) satisfies (2.5) with \( \mathcal{M}(x) = \mathcal{T}x \), so it is also CF. Indeed, given any \( x \) and \( i \), we can compute \( x_i^+ \) by immediately letting \( x_i^+ = (\mathcal{T}x)_i \) (at \( O(1) \) cost) and keeping \( x_j^+ = x_j, \forall j \neq i \); then, by (2.4), we update \( \mathcal{T}x \) to \( \mathcal{T}x^+ \) at a low cost. Formally, letting \( \mathcal{M}(x) = \mathcal{T}x, \)\

\[
\mathcal{M} \left[ \{x, \mathcal{M}(x)\} \mapsto \{x^+, \mathcal{M}(x^+)\} \right] \\
\leq \mathcal{M} \left[ \{x, \mathcal{T}x\} \mapsto x^+ \right] + \mathcal{M} \left[ \{x, \mathcal{T}x, x^+\} \mapsto \mathcal{T}x^+ \right] \\
\overset{(2.4)}{=} O(1) + O \left( \frac{1}{m} \mathcal{M} [x^+ \mapsto \mathcal{T}x^+] \right) \\
= O \left( \frac{1}{m} \mathcal{M} [x \mapsto \mathcal{T}x] \right).
\]

In general, the set of CF operators is much larger than the union of Type-I and Type-II CF operators.

Another important subclass of CF operators are operators \( \mathcal{T} : \mathbb{H} \rightarrow \mathbb{H} \) where \( (\mathcal{T}x)_i \) only depends on one, or a few, entries among \( x_1, \ldots, x_m \). Based on how many input coordinates they depend on, we partition them into three subclasses.

**Definition 11 (separable operator)** Consider \( \mathfrak{S} := \{\mathcal{T} \mid \mathcal{T} : \mathbb{H} \rightarrow \mathbb{H} \} \). We have the partition \( \mathfrak{S} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3 \), where

- separable operator: \( \mathcal{T} \in \mathcal{C}_1 \) if, for any index \( i \), there exists \( \mathcal{T}_i : \mathbb{H}_i \rightarrow \mathbb{H}_i \) such that \( (\mathcal{T}x)_i = \mathcal{T}_i x_i \), that is, \( (\mathcal{T}x)_i \) only depends on \( x_i \).

- nearly-separable operator: \( \mathcal{T} \in \mathcal{C}_2 \) if, for any index \( i \), there exists \( \mathcal{T}_i \) and index set \( I_i \) such that \( (\mathcal{T}x)_i = \mathcal{T}_i(\{x_j\}_{j \in I_i}) \) with \( |I_i| \ll m \), that is, each \( (\mathcal{T}x)_i \) depends on a few coordinates of \( x \).
• non-separable operator: \( C_3 : = \mathcal{X} \setminus (C_1 \cup C_2) \). If \( T \in C_3 \), there exists some \( i \) such that \((Tx)_i\) depends on many coordinates of \( x \).

Throughout the chapter, we assume the coordinate update of a (nearly-) separable operator costs roughly the same for all coordinates. Under this assumption, separable operators are both Type-I CF and Type-II CF, and nearly-separable operators are Type-I CF.\(^2\)

### 2.1.1 Examples of CF Operators

In this subsection, we give examples of CF operators arising in different areas including linear algebra, optimization, and machine learning.

**Example 8 ((block) diagonal matrix)** Consider the diagonal matrix

\[
A = \begin{bmatrix}
a_{1,1} & 0 \\
& \ddots \\
0 & a_{m,m}
\end{bmatrix} \in \mathbb{R}^{m \times m}.
\]

Clearly, \( T : x \mapsto Ax \) is separable.

**Example 9 (gradient and proximal maps of a separable function)** Consider a separable function

\[
f(x) = \sum_{i=1}^{m} f_i(x_i).
\]

Then, both \( \nabla f \) and \( \prox_{\gamma f} \) are separable, in particular,

\[
(\nabla f(x))_i = \nabla f_i(x_i) \quad \text{and} \quad (\prox_{\gamma f}(x))_i = \prox_{\gamma f_i}(x_i).
\]

Here, \( \prox_{\gamma f}(x) \) (\( \gamma > 0 \)) is the proximal operator that we define in Definition 3.

\(^2\)Not all nearly-separable operators are Type-II CF. Indeed, consider a sparse matrix \( A \in \mathbb{R}^{m \times m} \) whose non-zero entries are only located in the last column. Let \( Tx = Ax \) and \( x^+ = x + \delta_m \). As \( x^+ \) and \( x \) differ over the last entry, \( \mathcal{M}[x, Tx, x^+] \rightarrow \mathcal{M}[T x^+] = O(m) \). Since \( T x^+ = x^+ A_{:,m} \) takes \( m \) operations, we also have \( \mathcal{M}[x^+ \rightarrow T x^+] = O(m) \). Therefore, (2.4) is violated, and there is no benefit from maintaining \( Tx \).
Example 10 (projection to box constraints) Consider the “box” set \( B := \{ x : a_i \leq x_i \leq b_i, \ i \in [m] \} \subset \mathbb{R}^m. \) Then, the projection operator \( \text{proj}_B \) is separable. Indeed,
\[
(\text{proj}_B(x))_i = \max(b_i, \min(a_i, x_i)).
\]

Example 11 (sparse matrices) If every row of the matrix \( A \in \mathbb{R}^{m \times m} \) is sparse, \( T : x \mapsto Ax \) is nearly-separable.

Examples of sparse matrices arise from various finite difference schemes for differential equations, problems defined on sparse graphs. When most pairs of a set of random variables are conditionally independent, their inverse covariance matrix is sparse.

Example 12 (sum of sparsely supported functions) Let \( E \) be a class of index sets and every \( e \in E \) be a small subset of \([m]\), \(|e| \ll m\). In addition \(#\{e : i \in e\} \ll \#\{e\}\) for all \( i \in [m] \). Let \( x_e := (x_i)_{i \in e} \), and
\[
f(x) = \sum_{e \in E} f_e(x_e).
\]
The gradient map \( \nabla f \) is nearly-separable.

An application of this example arises in wireless communication over a graph of \( m \) nodes. Let each \( x_i \) be the spectrum assignment to node \( i \), each \( e \) be a neighborhood of nodes, and each \( f_e \) be a utility function. The input of \( f_e \) is \( x_e \) since the utility depends on the spectra assignments in the neighborhood.

In machine learning, if each observation only involves a few features, then each function of the optimization objective will depend on a small number of components of \( x \). This is the case when graphical models are used [92, 12].

Example 13 (squared hinge loss function) Consider for \( a, x \in \mathbb{R}^m \),
\[
f(x) := \frac{1}{2} \left( \max(0, 1 - \beta a^\top x) \right)^2,
\]
which is known as the squared hinge loss function. Consider the operator

$$\mathcal{T} x := \nabla f(x) = -\beta \max(0, 1 - \beta a^\top x)a. \quad (2.6)$$

Let us maintain \( \mathcal{M}(x) = a^\top x. \) For arbitrary \( x \) and \( i \), let

$$x_i^+ := (\mathcal{T} x)_i = -\beta \max(0, 1 - \beta a^\top x)a_i$$

and \( x_j^+ := x_j, \forall j \neq i. \) Then, computing \( x_i^+ \) from \( x \) and \( a^\top x \) takes \( O(1) \) (as \( a^\top x \) is maintained), and computing \( a^\top x^+ \) from \( x_i^+ - x_i \) and \( a^\top x \) costs \( O(1) \). Formally, we have

$$\mathcal{M} [\{x, a^\top x\} \mapsto \{x^+, a^\top x^+\}]$$

$$= \mathcal{M} [\{x, a^\top x\} \mapsto x^+] + \mathcal{M} [\{a^\top x, x_i^+ - x_i\} \mapsto a^\top x^+]$$

$$= O(1) + O(1) = O(1).$$

On the other hand, \( \mathcal{M} [x \mapsto \mathcal{T} x] = O(m). \) Therefore, \((2.5) \) holds, and \( \mathcal{T} \) defined in \((2.6) \) is CF.

### 2.2 Composite Coordinate Friendly Operators

Compositions of two or more operators arise in algorithms for problems that have composite functions, as well as algorithms that are derived from operator splitting methods. To update the variable \( x^k \) to \( x^{k+1} \), two or more operators are sequentially applied, and therefore the structures of all operators determine whether the update is CF. This is where CF structures become less trivial but more interesting. This section studies composite CF operators. The exposition leads to the recovery of existing algorithms, as well as powerful new algorithms.

#### 2.2.1 Combinations of Operators

We start by an example with numerous applications. It is a generalization of Example 13.
Example 14 (scalar map pre-composing affine function) Let $a_j \in \mathbb{R}^m, b_j \in \mathbb{R},$ and $\phi_j : \mathbb{R} \rightarrow \mathbb{R}$ be differentiable functions, $j \in [p]$. Let

$$f(x) = \sum_{j=1}^{p} \phi_j(a_j^\top x + b_j).$$

Assume that evaluating $\phi'_j$ costs $O(1)$ for each $j$. Then, $\nabla f$ is CF. Indeed, let

$$T_1 y := A^\top y, \quad T_2 y := \text{Diag}(\phi'_1(y_1), \ldots, \phi'_p(y_p)), \quad T_3 x := Ax + b,$$

where $A = [a_1^\top; a_2^\top; \ldots; a_p^\top] \in \mathbb{R}^{p \times m}$ and $b = [b_1; b_2; \ldots; b_p] \in \mathbb{R}^{p \times 1}$. Then we have $\nabla f(x) = T_1 \circ T_2 \circ T_3 x$. For any $x$ and $i \in [m]$, let $x_i^+ = \nabla_i f(x)$ and $x_j^+ = x_j, \forall j \neq i$, and let $M(x) := T_3 x$. We can first compute $T_2 \circ T_3 x$ from $T_3 x$ for $O(p)$ operations, then compute $\nabla_i f(x)$ and thus $x^+$ from $\{x, T_2 \circ T_3 x\}$ for $O(p)$ operations, and finally update the maintained $T_3 x$ to $T_3 x^+$ from $\{x, x^+, T_3 x\}$ for another $O(p)$ operations. Formally,

$$M[\{x, T_3 x\} \mapsto \{x^+, T_3 x^+\}] = M[\{T_3 x\} \mapsto \{T_2 \circ T_3 x\}] + M[\{x, T_2 \circ T_3 x\} \mapsto x^+] + M[\{x, T_3 x, x^+\} \mapsto \{T_3 x^+\}] = O(p) + O(p) + O(p) = O(p).$$

Since $M[\{x \mapsto \nabla f(x)\}] = O(pm)$, therefore $\nabla f = T_1 \circ T_2 \circ T_3$ is CF.

If $p = m$, $T_1, T_2, T_3$ all map from $\mathbb{R}^m$ to $\mathbb{R}^m$. Then, it is easy to check that $T_1$ is Type-I CF, $T_2$ is separable, and $T_3$ is Type-II CF. The last one is crucial since not maintaining $T_3 x$ would disqualify $T$ from CF. Indeed, to obtain $(T x)_i$, we must multiply $A_i^\top$ to all the entries of $T_2 \circ T_3 x$, which in turn needs all the entries of $T_3 x$, computing which from scratch would cost $O(pm)$.

There are general rules to preserve Type-I and Type-II CF. For example, $T_1 \circ T_2$ is still Type-I CF, and $T_2 \circ T_3$ is still CF, but there are counter examples where $T_2 \circ T_3$ can be neither Type-I nor Type-II CF. Such properties are important for developing efficient coordinate update algorithms for complicated problems; we will formalize them in the following.
The operators $T_2$ and $T_3$ in the above example are prototypes of cheap and easy-to-maintain operators from $H$ to $G$ that arise in operator compositions.

**Definition 12 (cheap operator)** For a composite operator $T = T_1 \circ \cdots \circ T_p$, an operator $T_i : H \rightarrow G$ is cheap if $M [x \mapsto T_i x]$ is less than or equal to the number of remaining coordinate-update operations, in order of magnitude.

**Definition 13 (easy-to-maintain operator)** For a composite operator $T = T_1 \circ \cdots \circ T_p$, an operator $T_j : H \rightarrow G$ is easy-to-maintain, if for any $x, i, \delta_i, x^+$ satisfying (2.1), $M [(x, T_j x, x^+ \mapsto T_j x^+)]$ is less than or equal to the number of remaining coordinate-update operations, in order of magnitude, or belongs to $O(\frac{1}{\dim G} M [x^+ \mapsto T x^+])$.

The splitting schemes in §2.2.2 below will be based on $T_1 + T_2$ or $T_1 \circ T_2$, as well as a sequence of such combinations. If $T_1$ and $T_2$ are both CF, $T_1 + T_2$ remains CF, but $T_1 \circ T_2$ is not necessarily so. This subsection discusses how $T_1 \circ T_2$ inherits the properties from $T_1$ and $T_2$. Our results are summarized in Tables 2.1 and 2.2 and explained in detail below. The combination $T_1 \circ T_2$ generally inherits the weaker property from $T_1$ and $T_2$.

The separability ($C_1$) property is preserved by composition. If $T_1, \ldots, T_n$ are separable, then $T_1 \circ \cdots \circ T_n$ is separable. However, combining nearly-separable ($C_2$) operators may not yield a nearly-separable operator since composition introduces more dependence among the input entries. Therefore, composition of nearly-separable operators can be either nearly-separable or non-separable.

Next, we discuss how $T_1 \circ T_2$ inherits the CF properties from $T_1$ and $T_2$. For simplicity, we only use matrix-vector multiplication as examples to illustrate the ideas; more interesting examples will be given later.

- If $T_1$ is separable or nearly-separable ($C_1 \cup C_2$), then as long as $T_2$ is CF ($F$), $T_1 \circ T_2$ remains CF. In addition, if $T_2$ is Type-I CF ($F_1$), so is $T_1 \circ T_2$. 
Table 2.1: $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits the weaker separability property from those of $\mathcal{T}_1$ and $\mathcal{T}_2$. 

<table>
<thead>
<tr>
<th>Case</th>
<th>$\mathcal{T}_1 \in$</th>
<th>$\mathcal{T}_2 \in$</th>
<th>$(\mathcal{T}_1 \circ \mathcal{T}_2) \in$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$C_1$ (separable)</td>
<td>$C_1, C_2, C_3$</td>
<td>$C_1, C_2, C_3$, respectively</td>
</tr>
<tr>
<td>2</td>
<td>$C_2$ (nearly-sep.)</td>
<td>$C_1, C_3$</td>
<td>$C_2, C_3$, resp.</td>
</tr>
<tr>
<td>3</td>
<td>$C_2$</td>
<td>$C_2$</td>
<td>$C_2$ or $C_3$, case by case</td>
</tr>
<tr>
<td>4</td>
<td>$C_3$ (non-sep.)</td>
<td>$C_1 \cup C_2 \cup C_3$</td>
<td>$C_3$</td>
</tr>
</tbody>
</table>

Table 2.2: Summary of how $\mathcal{T}_1 \circ \mathcal{T}_2$ inherits CF properties from those of $\mathcal{T}_1$ and $\mathcal{T}_2$. 

<table>
<thead>
<tr>
<th>Case</th>
<th>$\mathcal{T}_1 \in$</th>
<th>$\mathcal{T}_2 \in$</th>
<th>$(\mathcal{T}_1 \circ \mathcal{T}_2) \in$</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>$\mathcal{C}_1 \cup \mathcal{C}_2$</td>
<td>$\mathcal{F}, \mathcal{F}_1$</td>
<td>$\mathcal{F}, \mathcal{F}_1$, resp.</td>
<td>Examples 15 and 17</td>
</tr>
<tr>
<td>6</td>
<td>$\mathcal{F}, \mathcal{F}_2$</td>
<td>$\mathcal{C}_1$</td>
<td>$\mathcal{F}, \mathcal{F}_2$, resp.</td>
<td>Example 14</td>
</tr>
<tr>
<td>7</td>
<td>$\mathcal{F}_1$</td>
<td>$\mathcal{F}_2$</td>
<td>$\mathcal{F}$</td>
<td>Example 16</td>
</tr>
<tr>
<td>8</td>
<td>cheap</td>
<td>$\mathcal{F}_2$</td>
<td>$\mathcal{F}$</td>
<td>Example 17</td>
</tr>
<tr>
<td>9</td>
<td>$\mathcal{F}_1$</td>
<td>cheap</td>
<td>$\mathcal{F}_1$</td>
<td>Examples 14 and 17</td>
</tr>
</tbody>
</table>
**Example 15** Let $A \in \mathbb{R}^{m \times m}$ be sparse and $B \in \mathbb{R}^{m \times m}$ be dense. Then $T_1 x = Ax$ is nearly-separable and $T_2 x = Bx$ is Type-I CF\(^3\). For any $i$, let $\mathbb{I}_i$ denote the set of nonzeros indexes on the $i$th row of $A$. We first compute $(Bx)_{\mathbb{I}_i}$, which costs $O(|\mathbb{I}_i|m)$, and then $a_{i,\mathbb{I}_i}(Bx)_{\mathbb{I}_i}$, which costs $O(|\mathbb{I}_i|)$, where $a_{i,\mathbb{I}_i}$ is formed by the nonzero entries on the $i$th row of $A$. Assume $O(|\mathbb{I}_i|) = O(1), \forall i$. We have, from the above discussion, that $M[x \mapsto (T_1 \circ T_2 x)_{\mathbb{I}_i}] = O(m)$, while $M[x \mapsto T_1 \circ T_2 x] = O(m^2)$. Hence, $T_1 \circ T_2$ is Type-I CF.

• Assume that $T_2$ is separable ($\mathcal{C}_1$). It is easy to see that if $T_1$ is CF ($\mathcal{F}$), then $T_1 \circ T_2$ remains CF. In addition if $T_1$ is Type-II CF ($\mathcal{F}_2$), so is $T_1 \circ T_2$; see Example 14.

Note that, if $T_2$ is nearly-separable, we do not always have CF properties for $T_1 \circ T_2$. This is because $T_2 x$ and $T_2 x^+$ can be totally different (so updating $T_2 x$ is expensive) even if $x$ and $x^+$ only differ over one coordinate; see the footnote 2 on Page 20.

• Assume that $T_1$ is Type-I CF ($\mathcal{F}_1$). If $T_2$ is Type-II CF ($\mathcal{F}_2$), then $T_1 \circ T_2$ is CF ($\mathcal{F}$).

**Example 16** Let $A, B \in \mathbb{R}^{m \times m}$ be dense. Then $T_1 x = Ax$ is Type-I CF and $T_2 x = Bx$ Type-II CF (by maintaining $Bx$; see Example 6). For any $x$ and $i$, let $x^+$ satisfy (2.1). Maintaining $T_2 x$, we can compute $(T_1 \circ T_2 x)_j$ for $O(m)$ operations for any $j$ and update $T_2 x^+$ for $O(m)$ operations. On the other hand, computing $T_1 \circ T_2 x^+$ without maintaining $T_2 x$ takes $O(m^2)$ operations.

• Assume that one of $T_1$ and $T_2$ is cheap. If $T_2$ is cheap, then as long as $T_1$ is Type-I CF ($\mathcal{F}_1$), $T_1 \circ T_2$ is Type-I CF. If $T_1$ is cheap, then as long as $T_2$ is Type-II CF ($\mathcal{F}_2$), $T_1 \circ T_2$ is CF ($\mathcal{F}$); see Example 17.

\(^3\)For this example, one can of course pre-compute $AB$ and claim that $(T_1 \circ T_2)$ is Type-I CF. Our arguments keep $A$ and $B$ separate and only use the nearly-separability of $T_1$ and Type-I CF property of $T_2$, so our result holds for any such composition even when $T_1$ and $T_2$ are nonlinear.
We will see more examples of the above cases in the rest of the chapter.

2.2.2 Operator Splitting Schemes

We will apply our discussions above to operator splitting and obtain new algorithms. But first, we review several major operator splitting schemes and discuss their CF properties. We will encounter important concepts such as \textit{(maximum) monotonicity} and \textit{cocoercivity}, which are given in Section 1.2. For a monotone operator $A$, the \textit{resolvent operator} $J_A$ and the \textit{reflective-resolvent operator} $R_A$ are also defined in (1.6) and (1.7), respectively.

Consider the following problem: given three operators $A, B, C$, possibly set-valued, find $x \in H$ such that $0 \in Ax + Bx + Cx$, \hfill (2.7)

where “+” is the Minkowski sum. This is a high-level abstraction of many problems or their optimality conditions. The study began in the 1960s, followed by a large number of algorithms and applications over the last fifty years. Next, we review a few basic methods for solving (2.7).

When $A, B$ are maximally monotone (think it as the subdifferential $\partial f$ of a proper convex function $f$) and $C$ is $\beta$-cocoercive (think it as the gradient $\nabla f$ of a $1/\beta$-Lipschitz differentiable function $f$), a solution can be found by the iteration (1.2) with $T = T_{3S}$, introduced recently in [31], where

$$T_{3S} := I - J_{\gamma B} + J_{\gamma A} \circ (2J_{\gamma B} - I - \gamma C \circ J_{\gamma B}).$$ \hfill (2.8)

Indeed, by setting $\gamma \in (0, 2\beta)$, $T_{3S}$ is $\frac{2\beta}{3\beta - \gamma}$-averaged. Following the standard convergence result (cf. textbook [8]), provided that $T$ has a fixed point, the sequence from (1.2) converges to a fixed-point $x^*$ of $T$. Note that, instead of $x^*$, $J_{\gamma B}(x^*)$ is a solution to (2.7).

Following §2.2.1, $T_{3S}$ is CF if $J_{\gamma A}$ is separable ($C_1$), $J_{\gamma B}$ is Type-II CF ($F_2$), and $C$ is Type-I CF ($F_1$).
We give a few special cases of $T_{3S}$ below, which have much longer history. They all converge to a fixed point $x^*$ whenever a solution exists and $\gamma$ is properly chosen. If $B \neq 0$, then $J_{\gamma B}(x^*)$, instead of $x^*$, is a solution to (2.7).

**Forward-Backward Splitting (FBS).** Letting $B = 0$ yields $J_{\gamma B} = I$. Then, $T_{3S}$ reduces to FBS [75]:

$$T_{\text{FBS}} := J_{\gamma A} \circ (I - \gamma C) \quad (2.9)$$

for solving the problem $0 \in A x + C x$.

**Backward-Forward Splitting (BFS).** Letting $A = 0$ yields $J_{\gamma A} = I$. Then, $T_{3S}$ reduces to BFS:

$$T_{\text{BFS}} := (I - \gamma C) \circ J_{\gamma B} \quad (2.10)$$

for solving the problem $0 \in B x + C x$. When $A = B$, $T_{\text{FBS}}$ and $T_{\text{BFS}}$ apply the same pair of operators in the opposite orders, and they solve the same problem. Iterations based on $T_{\text{BFS}}$ are rarely used in the literature because they need an extra application of $J_{\gamma B}$ to return the solution, so $T_{\text{BFS}}$ is seemingly an unnecessary variant of $T_{\text{FBS}}$. However, they become different for coordinate update; in particular, $T_{\text{BFS}}$ is CF (but $T_{\text{FBS}}$ is generally not) when $J_{\gamma B}$ is Type-II CF ($\mathcal{F}_2$) and $C$ is Type-I CF ($\mathcal{F}_1$). Therefore, $T_{\text{BFS}}$ is worth discussing alone.

**Douglas-Rachford Splitting (DRS).** Letting $C = 0$, $T_{3S}$ reduces to

$$T_{\text{DRS}} := I - J_{\gamma B} + J_{\gamma A} \circ (2J_{\gamma B} - I) = \frac{1}{2}(I + R_{\gamma A} \circ R_{\gamma B}) \quad (2.11)$$

introduced in [34] for solving the problem $0 \in A x + B x$. A more general splitting is the Relaxed Peaceman-Rachford Splitting (RPRS) with $\lambda \in [0, 1]$:

$$T_{\text{RPRS}} = (1 - \lambda)I + \lambda R_{\gamma A} \circ R_{\gamma B}, \quad (2.12)$$

which recovers $T_{\text{DRS}}$ by setting $\lambda = \frac{1}{2}$ and Peaceman-Rachford Splitting (PRS) [76] by letting $\lambda = 1$.

**Forward-Douglas-Rachford Splitting (FDRS).** Let $V$ be a linear subspace, and $\mathcal{N}_V$ and $\mathcal{P}_V$ be its normal cone and projection operator, respectively.
The FDRS \[21\]

\[ \mathcal{T}_{\text{FDRS}} = \mathcal{I} - \mathcal{P}_V + \mathcal{J}_{\gamma A} \circ (2\mathcal{P}_V - \mathcal{I} - \gamma \mathcal{P}_V \circ \tilde{\mathcal{C}} \circ \mathcal{P}_V), \]

aims at finding a point \( x \) such that \( 0 \in \mathcal{A} x + \tilde{\mathcal{C}} x + \mathcal{N}_V x \). If an optimal \( x \) exists, we have \( x \in V \) and \( \mathcal{N}_V x \) is the orthogonal complement of \( V \). Therefore, the problem is equivalent to finding \( x \) such that \( 0 \in \mathcal{A} x + \mathcal{P}_V \circ \tilde{\mathcal{C}} \circ \mathcal{P}_V x + \mathcal{N}_V x \). Thus, \( \mathcal{T}_{3S} \) recovers \( \mathcal{T}_{\text{FDRS}} \) by letting \( \mathcal{B} = \mathcal{N}_V \) and \( \mathcal{C} = \mathcal{P}_V \circ \tilde{\mathcal{C}} \circ \mathcal{P}_V \).

**Forward-Backward-Forward Splitting (FBFS).** Composing \( \mathcal{T}_{\text{FBS}} \) with one more forward step gives \( \mathcal{T}_{\text{FBFS}} \) introduced in \[101\]:

\[ \mathcal{T}_{\text{FBFS}} = -\gamma \mathcal{C} + (\mathcal{I} - \gamma \mathcal{C})\mathcal{J}_{\gamma A}(\mathcal{I} - \gamma \mathcal{C}). \] (2.13)

\( \mathcal{T}_{\text{FBFS}} \) is not a special case of \( \mathcal{T}_{3S} \). At the expense of one more application of \((\mathcal{I} - \gamma \mathcal{C})\), \( \mathcal{T}_{\text{FBFS}} \) relaxes the convergence condition of \( \mathcal{T}_{\text{FBS}} \) from the cocoercivity of \( \mathcal{C} \) to its monotonicity. (For example, a nonzero skew symmetric matrix is monotonic but not cocoercive.) From Table 2.2, we know that \( \mathcal{T}_{\text{FBFS}} \) is CF if both \( \mathcal{C} \) and \( \mathcal{J}_{\gamma A} \) are separable.

### 2.2.3 Examples in Optimization

Consider the optimization problem

\[ \min_{x \in X} f(x) + g(x), \] (2.14)

where \( X \) is the feasible set and \( f \) and \( g \) are objective functions. We present examples of operator splitting methods discussed above.

**Example 17 (proximal gradient method)** Let \( X = \mathbb{R}^m \), \( f \) be differentiable, and \( g \) be proximable in (2.14). Setting \( \mathcal{A} = \partial g \) and \( \mathcal{C} = \nabla f \) in (2.9) gives \( \mathcal{J}_{\gamma A} = \text{prox}_{\gamma g} \) and reduces \( x^{k+1} = \mathcal{T}_{\text{FBS}}(x^k) \) to prox-gradient iteration:

\[ x^{k+1} = \text{prox}_{\gamma g}(x^k - \gamma \nabla f(x^k)). \] (2.15)
A special case of (2.15) with $g = \iota_X$ is the projected gradient iteration:

$$x^{k+1} = \mathcal{P}_X(x^k - \gamma \nabla f(x^k)).$$  \hspace{1cm} (2.16)

If $\nabla f$ is CF and $\text{prox}_{\gamma g}$ is (nearly-)separable (e.g., $g(x) = \|x\|_1$ or the indicator function of a box constraint) or if $\nabla f$ is Type-II CF and $\text{prox}_{\gamma g}$ is cheap (e.g., $\nabla f(x) = Ax - b$ and $g = \|x\|_2$), then the FBS iteration (2.15) is CF. In the latter case, we can also apply the BFS iteration (2.10) (i.e, compute $\text{prox}_{\gamma g}$ and then perform the gradient update), which is also CF.

**Example 18 (ADMM)** Setting $X = \mathbb{R}^m$ simplifies (2.14) to

$$\min_{x,y} f(x) + g(y), \quad \text{subject to } x - y = 0.$$ \hspace{1cm} (2.17)

The ADMM method iterates:

$$x^{k+1} = \text{prox}_{\gamma f}(y^k - \gamma s^k),$$ \hspace{1cm} (2.18a)

$$y^{k+1} = \text{prox}_{\gamma g}(x^{k+1} + \gamma s^k),$$ \hspace{1cm} (2.18b)

$$s^{k+1} = s^k + \frac{1}{\gamma}(x^{k+1} - y^{k+1}).$$ \hspace{1cm} (2.18c)

(The iteration can be generalized to handle the constraint $Ax - By = b$.) The dual problem of (2.17) is $\min_s f^*(-s) + g^*(s)$, where $f^*$ is the convex conjugate of $f$, i.e., $f^*(x) = \sup_y \{x^\top y - f(y)\}$. Letting $A = -\partial f^*(-x)$ and $B = \partial g^*(x)$ in (2.11) recovers the iteration (2.18) through (see the derivation in Appendix 8.1)

$$t^{k+1} = \mathcal{T}_{\text{DRS}}(t^k) = t^k - J_{\gamma B}(t^k) + J_{\gamma A} \circ (2J_{\gamma B} - I)(t^k).$$

From the results in Section 2.2.1, a sufficient condition for the above iteration to be CF is that $J_{\gamma A}$ is (nearly-)separable and $J_{\gamma B}$ being CF.

The above abstract operators and their CF properties will be applied in Chapter 4 to give interesting algorithms for several applications.
2.3 Primal-dual Coordinate Friendly Operators

We study how to solve the problem

\[
\min_{x \in H} f(x) + g(x) + h(Ax),
\]  

(2.19)

with primal-dual splitting algorithms, as well as their coordinate update versions. Here, \( f \) is differentiable and \( A \) is a “\( p \)-by-\( m \)” linear operator from \( H = \mathbb{H}_1 \times \cdots \times \mathbb{H}_m \) to \( G = G_1 \times \cdots \times G_p \). Problem (2.19) abstracts many applications in image processing and machine learning.

**Example 19 (image deblurring/denoising)** Let \( u^0 \) be an image, where \( u^0_i \in [0, 255] \), and \( B \) be the blurring linear operator. Let \( \|\nabla u\|_1 \) be the anisotropic\(^4\) total variation of \( u \) (see (4.16) for definition). Suppose that \( b \) is a noisy observation of \( Bu^0 \). Then, we can try to recover \( u^0 \) by solving

\[
\min_u \frac{1}{2}\|Bu - b\|^2 + \iota_{[0,255]}(u) + \lambda\|\nabla u\|_1,
\]  

(2.20)

which can be written in the form of (2.19) with \( f = \frac{1}{2}\|B \cdot - b\|^2 \), \( g = \iota_{[0,255]} \), \( A = \nabla \), and \( h = \lambda\|\cdot\|_1 \).

More examples with the formulation (2.19) will be given in Section 2.3.2. In general, primal-dual methods are capable of solving complicated problems involving constraints and the compositions of proximable and linear maps like \( \|\nabla u\|_1 \).

In many applications, although \( h \) is proximable, \( h \circ A \) is generally non-proximable and non-differentiable. To avoid using slow subgradient methods, we can consider the primal-dual splitting approaches to separate \( h \) and \( A \) so that \( \text{prox}_h \) can be applied. We derive that the equivalent form (for convex cases) of (2.19) is to find \( x \) such that

\[
0 \in (\nabla f + \partial g + A^\top \circ \partial h \circ A)(x).
\]  

(2.21)

\(^4\)Generalization to the isotropic case is straightforward by grouping variables properly.
Introducing the dual variable \( s \in \mathcal{G} \) and applying the biconjugation property:
\( s \in \partial h(Ax) \iff Ax \in \partial h^*(s) \), yields the equivalent condition
\[
0 \in \begin{bmatrix}
\nabla f & 0 \\
0 & 0
\end{bmatrix} + \begin{bmatrix}
\partial g & 0 \\
\partial h^* & -A
\end{bmatrix} \begin{bmatrix}
x \\
s
\end{bmatrix},
\]
(2.22)

which we shorten as \( 0 \in Ax + Bz \), with \( z \in \mathbb{H} \times \mathcal{G} =: \mathbb{F} \).

Problem (2.22) can be solved by the Condat-Vũ algorithm [28, 106]:
\[
\begin{aligned}
s^{k+1} &= \text{prox}_{\gamma h^*}(s^k + \gamma Ax^k), \\
x^{k+1} &= \text{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^T(2s^{k+1} - s^k))),
\end{aligned}
\]
(2.23)

which explicitly applies \( A \) and \( A^T \) and updates \( s, x \) in a Gauss-Seidel style\(^5\). We introduce an operator \( \mathcal{T}_{CV} : \mathbb{F} \to \mathbb{F} \) and write

\[
\text{iteration (2.23)} \iff z^{k+1} = \mathcal{T}_{CV}(z^k).
\]

Switching the orders of \( x \) and \( s \) yields the following algorithm:
\[
\begin{aligned}
x^{k+1} &= \text{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^T s^k)), \\
s^{k+1} &= \text{prox}_{\gamma h^*}(s^k + \gamma A(2x^{k+1} - x^k)),
\end{aligned}
\]
as \( z^{k+1} = \mathcal{T}'_{CV}z^k \).
(2.24)

It is known from [27, 30] that both (2.23) and (2.24) reduce to iterations of non-expansive operators (under a special metric), i.e., \( \mathcal{T}_{CV} \) is nonexpansive; see Appendix 8.2 for the reasoning.

**Remark 1** Similar primal-dual algorithms can be used to solve other problems such as saddle point problems [59, 66, 22] and variational inequalities [99]. Our coordinate update algorithms below apply to these problems as well.

---

\(^5\)By the Moreau identity: \( \text{prox}_{\gamma h^*} = I - \gamma \text{prox}_{\frac{1}{\gamma} h}(\cdot) \), one can compute \( \text{prox}_{\frac{1}{\gamma} h} \) instead of \( \text{prox}_{\gamma h^*} \), which inherits the same separability properties from \( \text{prox}_{\frac{1}{\gamma} h} \).
2.3.1 Primal-dual Coordinate Update Algorithms

In this subsection, we make the following assumption.

**Assumption 1** Functions $g$ and $h^*$ in the problem (2.19) are separable and proximable. Specifically,

$$g(x) = \sum_{i=1}^{m} g_i(x_i) \quad \text{and} \quad h^*(y) = \sum_{j=1}^{p} h^*_i(y_i).$$

Furthermore, $\nabla f$ is CF.

**Proposition 1** Under Assumption 1, the followings hold:

(a) when $p = O(m)$, the Condat-Vu operator $T_{CV}$ in (2.23) is CF, more specifically,

$$M \left[ \{z^k, Ax\} \mapsto \{z^+, Ax^+\} \right] = O \left( \frac{1}{m+p} M \left[ z^k \mapsto T_{CV} z^k \right] \right);$$

(b) when $m \ll p$ and $M [x \mapsto \nabla f(x)] = O(m)$, the Condat-Vu operator $T'_{CV}$ in (2.24) is CF, more specifically,

$$M \left[ \{z^k, A^\top s\} \mapsto \{z^+, A^\top s^+\} \right] = O \left( \frac{1}{m+p} M \left[ z^k \mapsto T'_{CV} z^k \right] \right).$$

**Proof 1** Computing $z^{k+1} = T_{CV} z^k$ involves evaluating $\nabla f$, $\text{prox}_g$, and $\text{prox}_{h^*}$, applying $A$ and $A^\top$, and adding vectors. It is easy to see $M \left[ z^k \mapsto T_{CV} z^k \right] = O(mp + m + p) + M [x \mapsto \nabla f(x)]$, and $M \left[ z^k \mapsto T'_{CV} z^k \right]$ is the same.

(a) We assume $\nabla f \in F_1$ for simplicity, and other cases are similar.

1. If $(T_{CV} z^k)_j = s_j^{k+1}$, computing it involves adding $s_j^k$ and $\gamma(Ax^k)_i$, and evaluating $\text{prox}_{\gamma h^*_i}$. In this case $M \left[ \{z^k, Ax\} \mapsto \{z^+, Ax^+\} \right] = O(1)$.

2. If $(T_{CV} z^k)_j = x_j^{k+1}$, computing it involves evaluating the entire $s^{k+1}$ for $O(p)$ operations, $(A^\top (2s^{k+1} - s^k))_i$ for $O(p)$ operations, $\text{prox}_{\eta g}$, for $O(1)$
operations, $\nabla f(x^k)$ for $O(\frac{1}{m}M [x \mapsto \nabla f(x)])$ operations, as well as updating $Ax^+$ for $O(p)$ operations. In this case

$$M \left[ \{z^k, Ax\} \mapsto \{z^+, Ax^+\} \right] = O(p + \frac{1}{m}M [x \mapsto \nabla f(x)]).$$

Therefore, $M \left[ \{z^k, Ax\} \mapsto \{z^+, Ax^+\} \right] = O\left(\frac{1}{m+p}M [z^k \mapsto T_{CV}z^k]\right)$.

(b) When $m \ll p$ and $M [x \mapsto \nabla f(x)] = O(m)$, following arguments similar to the above, we have

$$M \left[ \{z^k, A^t s\} \mapsto \{z^+, A^t s^+\} \right] = O(1) + M [x \mapsto \nabla_i f(x)] \text{ if } (T'_{CV}z^k)_j = x^{k+1}_i; \text{ and}$$

$$M \left[ \{z^k, A^t s\} \mapsto \{z^+, A^t s^+\} \right] = O(m) + M [x \mapsto \nabla f(x)] \text{ if } (T'_{CV}z^k)_j = s^{k+1}_i.$$

In both cases $M \left[ \{z^k, A^t s\} \mapsto \{z^+, A^t s^+\} \right] = O\left(\frac{1}{m+p}M [z^k \mapsto T_{CV}z^k]\right)$.

### 2.3.2 Extended Monotropic Programming

We develop a primal-dual coordinate update algorithm for the extended monotropic program:

$$\begin{align*}
\text{minimize} \quad & g_1(x_1) + g_2(x_2) + \cdots + g_m(x_m) + f(x), \\
\text{subject to} \quad & A_1x_1 + A_2x_2 + \cdots + A_mx_m = b,
\end{align*}$$

(2.25)

where $x = (x_1, \ldots, x_m) \in \mathbb{H} = \mathbb{H}_1 \times \ldots \times \mathbb{H}_m$ with $\mathbb{H}_i$ being Euclidean spaces. It generalizes linear, quadratic, second-order cone, semi-definite programs by allowing extended-valued objective functions $g_i$ and $f$. It is a special case of (2.19) by letting $g(x) = \sum_{i=1}^m g_i(x_i)$, $A = [A_1, \ldots, A_m]$ and $h = \iota_\{b\}$.

**Example 20 (quadratic programming)** Consider the quadratic program

$$\begin{align*}
\text{minimize} \quad & \frac{1}{2}x^T U x + c^T x, \quad \text{subject to} \quad Ax = b, \quad x \in X,
\end{align*}$$

(2.26)

where $U$ is a symmetric positive semidefinite matrix and $X = \{x : x_i \geq 0 \ \forall i\}$. Then, (2.26) is a special case of (2.25) with $g_i(x_i) = \iota_{\geq 0}(x_i)$, $f(x) = \frac{1}{2}x^T U x + c^T x$ and $h = \iota_\{b\}$. 

34
Example 21 (Second Order Cone Programming (SOCP)) \textit{The SOCP}

\[
\begin{align*}
\text{minimize} & \quad c^\top x, \quad \text{subject to} \quad Ax = b, \\
& \quad x \in X = Q_1 \times \cdots \times Q_n,
\end{align*}
\]

where \(Q_i, \forall i \in [n]\) are second order cones. \textit{The SOCP can be written in the form of (2.25): minimize}_{x \in \mathbb{R}^m} \iota_X(x) + c^\top x + \iota_{\{b\}}(Ax).

Applying iteration (2.23) to problem (2.25) and eliminating \(s^{k+1}\) from the second row yield the Jacobi-style update (denoted as \(T_{\text{emp}}\)):

\[
\begin{align*}
\begin{cases}
\quad s^{k+1} = s^k + \gamma(Ax^k - b), \\
\quad x^{k+1} = \text{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^\top s^k + 2\gamma A^\top Ax^k - 2\gamma A^\top b)).
\end{cases}
\end{align*}
\]

To the best of our knowledge, this update is never found in the literature. Note that \(x^{k+1}\) no longer depends on \(s^{k+1}\), making it more convenient to perform coordinate updates.

\textbf{Remark 2} In general, when the \(s\) update is affine, we can decouple \(s^{k+1}\) and \(x^{k+1}\) by plugging the \(s\) update into the \(x\) update. It is the case when \(h\) is affine or quadratic in problem (2.19).

A sufficient condition for \(T_{\text{emp}}\) to be CF is \(\text{prox}_g \in \mathcal{C}_1\) i.e., separable. Indeed, we have \(T_{\text{emp}} = T_1 \circ T_2\), where

\[
T_1 = \begin{bmatrix}
I & 0 \\
0 & \text{prox}_{\eta g}
\end{bmatrix}, \quad T_2 = \begin{bmatrix}
s & s + \gamma(Ax - b) \\
x & x - \eta(\nabla f(x) + A^\top s + 2\gamma A^\top Ax - 2\gamma A^\top b)
\end{bmatrix}.
\]

Following Case 5 of Table 2.2, \(T_{\text{emp}}\) is CF. When \(m = \Theta(p)\), the separability condition on \(\text{prox}_g\) can be relaxed to \(\text{prox}_g \in \mathcal{F}_1\) since in this case \(T_2 \in \mathcal{F}_2\), and we can apply Case 7 of Table 2.2 by maintaining \(\nabla f(x), A^\top s, Ax\) and \(A^\top Ax\).
2.3.3 Overlapping-Block Coordinate Updates

In the coordinate update scheme based on (2.23), if we select $x_i$ to update, then we must first compute $s^{k+1}$, because the variables $x_i$'s and $s_j$'s are coupled through the matrix $A$. However, once $x^{k+1}_i$ is obtained, $s^{k+1}$ is discarded. It is not used to update $s$ or cached for further use. This subsection introduces ways to utilize the otherwise wasted computation.

We define, for each $i$, $J(i) \subset [p]$ as the set of indices $j$ such that $A_{ij}^\top \neq 0$, and, for each $j$, $I(j) \subset [m]$ as the set of indices of $i$ such that $A_{ij}^\top \neq 0$. We also let $m_j := |I(j)|$, and assume $m_j \neq 0, \forall j \in [p]$ without loss of generality.

We arrange the coordinates of $z = [x; s]$ into $m$ overlapping blocks. The $i$th block consists of the coordinate $x_i$ and all $s_j$'s for $j \in J(i)$. This way, each $s_j$ may appear in more than one block. We propose a block coordinate update scheme based on (2.23). Because the blocks overlap, each $s_j$ may be updated in multiple blocks, so the $s_j$ update is relaxed with parameters $\rho_{i,j} \geq 0$ (see (2.28) below) that satisfy $\sum_{i \in I(j)} \rho_{i,j} = 1, \forall j \in [p]$. The aggregated effect is to update $s_j$ without scaling. (Following the KM iteration [55], we can also assign a relaxation parameter $\eta_k$ for the $x_i$ update; then, the $s_j$ update should be relaxed with $\rho_{i,j}\eta_k$.)

We propose the following update scheme:

$$
\begin{cases}
\text{select } i \in [m], \text{ and then compute} \\
\quad \tilde{s}_j^{k+1} = \text{prox}_{\gamma \eta_j}(s_j^k + \gamma(Ax^k)_j), \text{ for all } j \in J(i), \\
\quad \tilde{x}_i^{k+1} = \text{prox}_{\eta \eta_i}(x_i^k - \eta(\nabla_i f(x^k) + \sum_{j \in J(i)} A_{ij}^\top (2\tilde{s}_j^{k+1} - s_j^k))), \\
\text{update } x_i^{k+1} = x_i^k + (\tilde{x}_i^{k+1} - x_i^k), \\
\text{update } s_j^{k+1} = s_j^k + \rho_{i,j}(\tilde{s}_j^{k+1} - s_j^k), \text{ for all } j \in J(i).
\end{cases}
$$

(2.28)

**Remark 3** The use of relaxation parameters $\rho_{i,j}$ makes our scheme different from that in [80].
Following the assumptions and arguments in §2.3.1, if we maintain $Ax$, the cost for each block coordinate update is $O(p) + \mathfrak{M}[x \mapsto \nabla_i f(x)]$, which is $O(\frac{1}{m} \mathfrak{M}[z \mapsto \mathcal{T}_{CV} z])$. Therefore the coordinate update scheme (2.28) is computationally worthy.

Typical choices of $\rho_{i,j}$ include: (1) one of the $\rho_{i,j}$’s is 1 for each $j$, others all equal to 0. This can be viewed as assigning the update of $s_j$ fully to a block containing $x_i$. (2) $\rho_{i,j} = \frac{1}{m_j}$ for all $i \in \mathbb{I}(j)$. This approach averages the updates of $s_j$ over all related blocks.

**Remark 4** The recent paper [39] proposes a different primal-dual coordinate update algorithm. The authors produce a new matrix $\bar{A}$ based on $A$, with only one nonzero entry in each row, i.e. $m_j = 1$ for each $j$. They also modify $h$ to $\bar{h}$ so that the problem

$$\min_{x \in \mathbb{H}} f(x) + g(x) + \bar{h}(\bar{A}x)$$

has the same solution as (2.19). Then they solve (2.29) by the scheme (2.28). Because they have $m_j = 1$, every dual variable coordinate is only associated with one primal variable coordinate. They create non-overlapping blocks of $z$ by duplicating each dual variable coordinate $s_j$ multiple times. The computation cost for each block coordinate update of their algorithm is the same as (2.28), but more memory is needed for the duplicated copies of each $s_j$. 
CHAPTER 3

ARock: Asynchronous Parallel Coordinate Update Framework

3.1 Introduction

This chapter introduces a new approach to asynchronous parallel computing with convergence guarantees. In a synchronous (sync) parallel iterative algorithm, the agents must wait for the slowest agent to finish an iteration before they can all proceed to the next one (Figure 3.1a). Hence, the slowest agent may cripple the system. In contrast, the agents in an asynchronous (async) parallel iterative algorithm run continuously with little idling (Figure 3.1b). However, the iterations are disordered, and an agent may carry out an iteration without the newest information from other agents.

Asynchrony has other advantages [16]: the system is more tolerant to computing faults and communication glitches; it is also easy to incorporate new agents.

On the other hand, it is more difficult to analyze asynchronous algorithms and ensure their convergence. It becomes impossible to find a sequence of iterates
that one completely determines the next. Nonetheless, we let any update be a new iteration and propose an async-parallel algorithm (ARock) for the generic fixed-point iteration. It converges if the fixed-point operator is nonexpansive (Def. 2) and has a fixed point.

Let $\mathbb{H}_1, \ldots, \mathbb{H}_m$ be Hilbert spaces and $\mathbb{H} := \mathbb{H}_1 \times \cdots \times \mathbb{H}_m$ be their Cartesian product. For a nonexpansive operator $T : \mathbb{H} \rightarrow \mathbb{H}$, our problem is to

\[
\text{find } x^* \in \mathbb{H} \quad \text{such that} \quad x^* = Tx^*.
\]

Finding a fixed point to $T$ is equivalent to finding a zero of $S \equiv I - T$, denoted by $x^*$ such that $0 = Sx^*$. Hereafter, we will use both $S$ and $T$ for convenience.

Problem (3.1) is widely applicable in linear and nonlinear equations, statistical regression, machine learning, convex optimization, and optimal control. A generic framework for problem (3.1) is the Krasnosel’ski–Mann (KM) iteration [55]:

\[
x^{k+1} = x^k + \alpha (Tx^k - x^k), \quad \text{or equivalently,} \quad x^{k+1} = x^k - \alpha Sx^k;
\]

where $\alpha \in (0, 1)$ is the step size. If $\text{Fix } T$ — the set of fixed points of $T$ (zeros of $S$) — is nonempty, then the sequence $(x^k)_{k \geq 0}$ converges weakly to a point in $\text{Fix } T$ and $(Tx^k - x^k)_{k \geq 0}$ converges strongly to 0. The KM iteration generalizes algorithms in convex optimization, linear algebra, differential equations, and monotone inclusions. Its special cases include the following iterations: alternating projection, gradient descent, projected gradient descent, proximal-point algorithm, Forward-Backward Splitting (FBS) [75], Douglas-Rachford Splitting (DRS) [61], a three-operator splitting [31], and the Alternating Direction Method of Multipliers (ADMM) [61, 44].

In ARock, a set of $p$ agents, $p \geq 1$, solve problem (3.1) by updating the coordinates $x_i \in \mathbb{H}_i$, $i \in [m]$, in a random and asynchronous fashion. Algorithm 2 describes the framework. Its special forms for several applications are given in
Algorithm 2: ARock: a framework for async-parallel coordinate updates

**Input**: \( x^0 \in \mathbb{H}, K > 0, \) a distribution \((p_1, \ldots, p_m) > 0\) with \( \sum_{i=1}^{m} p_i = 1; \)

global iteration counter \( k \leftarrow 0; \)

**while** \( k < K, \) *every agent asynchronously and continuously* **do**

- select \( i_k \in [m] \) with \( \text{Prob}(i_k = i) = p_i; \)
- perform an update to \( x_{i_k} \) according to (3.3);
- update the global counter \( k \leftarrow k + 1; \)

Whenever an agent updates a coordinate, the global iteration counter \( k \) increases by one. The \( k \)th update is applied to \( x_{i_k} \in \mathbb{H}_{i_k}, \) where \( i_k \in [m] \) is an independent random variable. Each coordinate update has the form

\[
x^{k+1} = x^k - \frac{\eta_k}{\text{mp}_{i_k}} \mathcal{S}_{i_k} \hat{x}^k, \tag{3.3}
\]

where \( \eta_k > 0 \) is the step size, \( \mathcal{S}_{i_k} x := (0, \ldots, 0, (\mathcal{S}x)_{i_k}, 0, \ldots, 0), \) and \( \text{mp}_{i_k} \) is used to normalize nonuniform selection probabilities. In the uniform case, namely, \( p_i \equiv \frac{1}{m} \) for all \( i, \) we have \( \text{mp}_{i_k} \equiv 1, \) which simplifies the update (3.3) to

\[
x^{k+1} = x^k - \eta_k \mathcal{S}_{i_k} \hat{x}^k. \tag{3.4}
\]

Here, the point \( \hat{x}^k \) is what an agent reads from global memory to its local cache and to which \( \mathcal{S}_{i_k} \) is applied, and \( x^k \) denotes the state of \( x \) in global memory just before the update (3.3) is applied. In a sync-parallel algorithm, we have \( \hat{x}^k = x^k, \) but in ARock, due to possible updates to \( x \) by other agents, \( \hat{x}^k \) can be different from \( x^k. \) This is a key difference between sync-parallel and async-parallel algorithms. In Section 3.1.2 below, we will establish the relationship between \( \hat{x}^k \) and \( x^k \) as

\[
\hat{x}^k = x^k + \sum_{d \in J(k)} (x^d - x^{d+1}), \tag{3.5}
\]

where \( J(k) \subseteq \{k - 1, \ldots, k - \tau\} \) and \( \tau \in \mathbb{Z}^+ \) is the maximum number of other updates to \( x \) during the computation of (3.3). Equation (3.5) has appeared in [62].
The update (3.3) is only computationally worthy if $S_\xi x$ is much cheaper to compute than $Sx$. Otherwise, it is more preferable to apply the full KM update (3.2). Chapter 2 studies coordinate friendly structures more thoroughly. In Chapter 4, we will present several applications that have the favorable structures for ARock.

To establish the convergence, we need to assume that $\hat{x}^k$ is independent of the coordinate choice. This is true if the computation costs of different coordinate update are the same. The convergence of ARock (Algorithm 2) is stated in Theorems 3 and 4. Here we include a shortened version, leaving detailed bounds to the full theorems:

**Theorem 1 (Global and linear convergence)** Let $T : H \rightarrow H$ be a nonexpansive operator that has a fixed point. Let $(x^k)_{k \geq 0}$ be the sequence generated by Algorithm 2 with properly bounded step sizes $\eta_k$. Then, with probability one, $(x^k)_{k \geq 0}$ converges weakly to a fixed point of $T$. This convergence becomes strong if $H$ has a finite dimension.

In addition, if $T$ is demicompact (see Definition 5 below), then with probability one, $(x^k)_{k \geq 0}$ converges strongly to a fixed point of $T$.

Furthermore, if $S \equiv I - T$ is quasi-strongly monotone (see Definition 5), then $T$ has a unique fixed-point $x^*$, $(x^k)_{k \geq 0}$ converges strongly to $x^*$ with probability one, and $E\|x^k - x^*\|^2$ converges to 0 at a linear rate.

In the theorem, the weak convergence result only requires $T$ to be nonexpansive and has a fixed point. In addition, the computation requires: (a) bounded step sizes; (b) random coordinate selection; and (c) a finite maximal delay $\tau$. Assumption (a) is standard, and we will see the bound can be $O(1)$. Assumption (b) is essential to both the analysis and the numerical performance of our algorithms. Assumption (c) is not essential; an infinite delay with a light tail is allowed. The strong convergence result applies to all the examples in Chapter 4, and the lin-
ear convergence result applies to Examples 4.2 and 4.3.1 when the corresponding operator $S$ is quasi-strongly monotone. Step sizes $\eta_k$ are discussed in Remarks 6 and 8.

3.1.1 On Random Coordinate Selection

ARock employs random coordinate selection. This section discusses its advantages and disadvantages.

Its main disadvantage is that an agent cannot caching the data associated with a coordinate. The variable $x$ and its related data must be either stored in global memory or passed through communication. A secondary disadvantage is that pseudo-random number generation takes time, which becomes relatively significant if each coordinate update is cheap. (The network optimization examples in Sections 4.5.1 and 4.5.3 are exceptions, where data are naturally stored in a distributed fashion and random coordinate assignments are the results of Poisson processes.)

There are several advantages of random coordinate selection. It realizes the user-specified update frequency $p_i$ for every component $x_i$, $i = 1,\ldots,m$, even when different agents have different computing powers and different coordinate updates cost different amounts of computation. Therefore, random assignment ensures load balance. The algorithm is also fault tolerant in the sense that if one or more agents fail, it will still converge to a fixed-point of $T$. In addition, it has been observed numerically on certain problems [23] that random coordinate selection accelerates convergence.

3.1.2 Uncoordinated Memory Access

In ARock, since multiple agents simultaneously read and update $x$ in global memory, $\hat{x}^k$ — the result of $x$ that is read from global memory by an agent to its local
cache for computation — may not equal \( x^j \) for any \( j \leq k \), that is, \( \hat{x}^k \) may never be consistent with a state of \( x \) in global memory. This is known as inconsistent read. In contrast, consistent read means that \( \hat{x}^k = x^j \) for some \( j \leq k \), i.e., \( \hat{x}^k \) is consistent with a state of \( x \) that existed in global memory.

We illustrate inconsistent read and consistent read in the following example, which is depicted in Figure 3.2. Consider \( x = [x_1, x_2, x_3, x_4]^T \in \mathbb{R}^4 \) and \( x^0 = [0, 0, 0, 0]^T \) initially, at time \( t_0 \). Suppose at time \( t_1 \), agent 2 updates \( x_1 \) from 0 to 1, yielding \( x^1 = [1, 0, 0, 0]^T \); then, at time \( t_2 \), agent 3 updates \( x_4 \) from 0 to 2, further yielding \( x^2 = [1, 0, 0, 2]^T \). Suppose that agent 1 starts reading \( x \) from the first component \( x_1 \) at \( t_0 \). For consistent read (Figure 3.2a), agent 1 acquires a memory lock and only releases the lock after finishing reading all of \( x_1, x_2, x_3, \) and \( x_4 \). Therefore, agent 1 will read in \( [0, 0, 0, 0]^T \). Inconsistent read, however, allows agent 1 to proceed without a memory lock: agent 1 starts reading \( x_1 \) at \( t_0 \) (Figure 3.2b) and reaches the last component, \( x_4 \), after \( t_2 \); since \( x_4 \) is updated by agent 3 prior to it is read by agent 1, agent 1 has read \( [0, 0, 0, 2]^T \), which is different from any of \( x^0, x^1, \) and \( x^2 \).

![Figure 3.2](image_url)

(a) Consistent read. While agent 1 reads \( x \) in memory, it acquires a global lock. (b) Inconsistent read. Agent 1 reads \( (0, 0, 0, 2)^T \), a non-existing state of \( x \).

Figure 3.2: Consistent read versus inconsistent read: A demonstration.

Even with inconsistent read, each component is consistent under the atomic
coordinate update assumption, which will be defined below. Therefore, we can express what has been read in terms of the changes of individual coordinates. In the above example, the first change is $x_1^1 - x_1^0 = 1$, which is added to $x_1$ just before time $t_1$ by agent 2, and the second change is $x_4^2 - x_4^1 = 2$, added to $x_4$ just before time $t_2$ by agent 3. The inconsistent read by agent 1, which gives the result $[0, 0, 0, 2]^T$, equals $x^0 + 0 \times (x^1 - x^0) + 1 \times (x^2 - x^1)$.

We have demonstrated that $\hat{x}^k$ can be inconsistent, but each of its coordinates is consistent, that is, for each $i$, $\hat{x}^k_i$ is an ever-existed state of $x_i$ among $x_i^k, \ldots, x_i^{k-\tau}$. Suppose that $\hat{x}^k_i = x^d_i$, where $d \in \{k, k-1, \ldots, k-\tau\}$. Therefore, $\hat{x}^k_i$ can be related to $x_i^k$ through the interim changes applied to $x_i$. Let $J_i(k) \subset \{k-1, \ldots, k - \tau\}$ be the index set of these interim changes. If $J_i(k) \neq \emptyset$, then $d = \min\{d \in J_i(k)\}$; otherwise, $d = k$. In addition, we have $\hat{x}^k_i = x^d_i = x^k_i + \sum_{d \in J_i(k)}(x^d_i - x^{d+1}_i)$. Since the global counter $k$ is increased after each coordinate update, updates to $x_i$ and $x_j, i \neq j$, must occur at different $k$’s and thus $J_i(k) \cap J_j(k) = \emptyset, \forall i \neq j$. Therefore, by letting $J(k) := \cup_i J_i(k) \subset \{k-1, \ldots, k - \tau\}$ and noticing $(x^d_i - x^{d+1}_i) = 0$ for $d \in J_j(k)$ where $i \neq j$, we have $\hat{x}^k_i = x^k_i + \sum_{d \in J(k)}(x^d_i - x^{d+1}_i), \forall i = 1, \ldots, m$, which is equivalent to (3.5). Here, we have made two assumptions:

- **atomic coordinate update**: a coordinate is not further broken to smaller components during an update; they are all updated at once.

- **bounded maximal delay $\tau$**: during any update cycle of an agent, $x$ in global memory is updated at most $\tau$ times by other agents.

When each coordinate is a single scalar, updating the scalar is a single atomic instruction on most modern hardware, so the first assumption naturally holds, and our algorithm is lock-free. The case where a coordinate is a block that includes multiple scalars is discussed in the next Section.
3.1.3 Block Coordinate

In the “block coordinate” case (updating a block of several coordinates each time), the atomic coordinate update assumption can be met by either employing a per-coordinate memory lock or taking the following dual-memory approach: Store two copies of each coordinate \(x_i \in \mathbb{H}_i\) in global memory, denoting them as \(x_i^{(0)}\) and \(x_i^{(1)}\); let a bit \(\alpha_i \in \{0, 1\}\) point to the active copy; an agent will only read \(x_i\) from the active copy \(x_i^{(\alpha_i)}\); before an agent updates the components of \(x_i\), it obtains a memory lock to the inactive copy \(x_i^{(1-\alpha_i)}\) to prevent other agents from simultaneously updating it; then after it finishes updating \(x_i^{(1-\alpha_i)}\), flip the bit \(\alpha_i\) so that other agents will begin reading from the updated copy. This approach never blocks any read of \(x_i\), yet it eliminates inconsistency.

3.1.4 Straightforward Generalization

Our async-parallel coordinate update scheme (3.3) can be generalized to (overlapping) block coordinate updates after a change to the step size. Specifically, the scheme (3.3) can be generalized to

\[
x^{k+1} = x^k - \frac{\eta}{n p_{i_k}} (\mathcal{U}_{i_k} \circ S)x^k,
\]

where \(\mathcal{U}_{i_k}\) is randomly drawn from a set of operators \(\{\mathcal{U}_1, \ldots, \mathcal{U}_n\}\) (\(n \leq m\)), \(\mathcal{U}_i : \mathbb{H} \to \mathbb{H}\), following the probability \(P(i_k = i) = p_i, i = 1, \ldots, n (p_i > 0,\) and \(\sum_{i=1}^n p_i = 1\)). The operators must satisfy \(\sum_{i=1}^n \mathcal{U}_i = I_{\mathbb{H}}\) and \(\sum_{i=1}^n \|\mathcal{U}_i x\|^2 \leq C\|x\|^2\) for some \(C > 0\).

Let \(\mathcal{U}_i : x \mapsto (0, \ldots, 0, x_i, 0, \ldots, 0), i = 1, \ldots, m\), which has \(C = 1\); then (3.6) reduces to (3.3). If \(\mathbb{H}\) is endowed with a metric \(M\) such that \(\rho_1\|x\|^2 \leq \|x\|_M^2 \leq \rho_2\|x\|^2\) (e.g., the metric in the Condat-Vũ primal-dual splitting [28, 106]), then we have

\[
\sum_{i=1}^m \|\mathcal{U}_i x\|_M^2 \leq \rho_2 \sum_{i=1}^m \|\mathcal{U}_i x\|^2 = \rho_2 \|x\|^2 \leq \frac{\rho_2}{\rho_1} \|x\|_M^2.
\]
In general, multiple coordinates can be updated in (3.6). Consider linear $U_i: x \mapsto (a_{i1}x_1, \ldots, a_{im}x_m)$, $i = 1, \ldots, m$, where $\sum_{i=1}^n a_{ij} = 1$ for each $j$. Then, for $C := \max \{\sum_{i=1}^n a_{ij}^2, \ldots, \sum_{i=1}^n a_{im}^2\}$, we have
\[
\sum_{i=1}^n \|U_i x\|^2 = \sum_{i=1}^n \sum_{j=1}^m a_{ij}^2 \|x_j\|^2 = \sum_{j=1}^m \sum_{i=1}^n a_{ij}^2 \|x_j\|^2 \leq C \|x\|^2.
\]

3.1.5 Special Cases

If there is only one agent ($p = 1$), ARock (Algorithm 2) reduces to randomized coordinate update, which includes the special case of randomized coordinate descent [69] for convex optimization. Sync-parallel coordinate update is another special case of ARock corresponding to $\hat{x}^k \equiv x^k$. In both cases, there is no delay, i.e., $\tau = 0$ and $J(k) = \emptyset$. In addition, the step size $\eta_k$ can be more relaxed. In particular, if $p_i = \frac{1}{m}$, $\forall i$, then we can let $\eta_k = \eta$, $\forall k$, for any $\eta < 1$, or $\eta < 1/\alpha$ when $T$ is $\alpha$-averaged (see Definition 5 for the definition of an $\alpha$-averaged operator).

3.1.6 Related work

Chazan and Miranker [25] proposed the first async-parallel method in 1969. The method was designed for solving linear systems. Later, async-parallel methods have been successful applied in many fields, e.g., linear systems [4, 17, 40, 91], nonlinear problems [5, 6], differential equations [1, 2, 24, 33], consensus problems [57, 38], and optimization [52, 62, 63, 96, 119]. We review the theory for async-parallel fixed-point iteration and its applications.

**General fixed point problems.** Totally async-parallel\footnote{\textsuperscript{1}“Totally asynchronous” means no upper bound on the delays; however, other conditions are required, for example: each coordinate must be updated infinitely many times. By default, “asynchronous” in this chapter assumes a finite maximum delay.} iterative methods for a fixed-point problem go back as early as to Baudet [6], where the operator was assumed to be $P$-contraction.\footnote{An operator $T : \mathbb{R}^n \to \mathbb{R}^n$ is $P$-contraction if $|T(x) - T(y)| \leq P|x - y|$, component-wise,} Later, Bertsekas [13] generalized the $P$-contraction

46
assumption and showed convergence. Frommer and Szyld [41] reviewed the theory and applications of totally async-parallel iterations prior to 2000. This review summarized convergence results under the conditions in [13]. However, ARock can be applied to solve many more problems since our nonexpansive assumption, though not strictly weaker than P-contraction, is more pervasive. As opposed to totally asynchronous methods, Tseng, Bertsekas, and Tsitsiklis [15, 102] assumed quasi-nonexpansiveness\(^3\) and proposed an async-parallel method, converging under an additional assumption, which is difficult to justify in general but can be established for problems such as linear systems and strictly convex network flow problems [15, 102].

The above works assign coordinates in a deterministic manner. Different from them, ARock is stochastic, works for nonexpansive operators, and is more applicable.

**Linear, nonlinear, and differential equations.** The first async-parallel method for solving linear equations was introduced by Chazan and Miranker in [25]. They proved that on solving linear systems, P-contraction was necessary and sufficient for convergence. The performance of the algorithm was studied by Iain et al. [17, 91] on different High Performance Computing (HPC) architectures. Recently, Avron et al. [4] revisited the async-parallel coordinate update and showed its linear convergence for solving positive-definite linear systems. Tarazi and Nabih [37] extended the pioneering work [25] to solving nonlinear equations, and the async-parallel methods have also been applied for solving differential equations, e.g., in [1, 2, 24, 33]. Except for [4], all these methods are totally async-parallel with the P-contraction condition or its variants. On solving a positive-definite linear system, [4] made assumptions similar to ours, and it obtained better linear convergence rate on that special problem.

---

where \(|x|\) denotes the vector with components \(|x_i|, i = 1, ..., n\), and \(P \in \mathbb{R}^{n \times n}\) is a nonnegative matrix with a spectral radius strictly less than 1.

\(^3\)An operator \(T : H \to H\) is quasi-nonexpansive if \(\|Tx - x^*\| \leq \|x - x^*\|, \forall x \in H, x^* \in \text{Fix}T\).
Optimization. The first async-parallel coordinate update gradient-projection method was due to Bertsekas and Tsitsiklis [15]. The method solves constrained optimization problems with a smooth objective and simple constraints. It was shown that the objective gradient sequence converges to zero. Tseng [100] further analyzed the convergence rate and obtained local linear convergence based on the assumptions of isocost surface separation and a local Lipschitz error bound. Recently, Liu et al. [63] developed an async-parallel stochastic coordinate descent algorithm for minimizing convex smooth functions. Later, Liu and Wright [62] suggested an async-parallel stochastic proximal coordinate descent algorithm for minimizing convex composite objective functions. They established the convergence of the expected objective-error sequence for convex functions. Hsieh et al. [52] proposed an async-parallel dual coordinate descent method for solving $\ell_2$ regularized empirical risk minimization problems. Other async-parallel approaches include asynchronous ADMM [50, 108, 119, 53]. Among them, [108, 53] use an asynchronous clock, and [50, 119] use a central node to update the dual variable; they do not deal with delay or inconsistency. Async-parallel stochastic gradient descent methods have also been considered in [67, 85].

Our framework differs from the recent surge of the aforementioned sync-parallel and async-parallel coordinate descent algorithms (e.g., [79, 56, 63, 62, 52, 88]). While they apply to convex function minimization, ARock covers more cases (such as ADMM, primal-dual, and decentralized methods) and also provides sequence convergence. In Chapter 4, we will show that some of the existing async-parallel coordinate descent algorithms are special cases of ARock, through relating their optimality conditions to nonexpansive operators. Another difference is that the convergence of ARock only requires a nonexpansive operator with a fixed point, whereas properties such as strong convexity, bounded feasible set, and bounded sequence, which are seen in some of the recent literature for async-parallel convex minimization, are unnecessary.
Others. Besides solving equations and optimization problems, there are also applications of async-parallel algorithms to optimal control [57], network flow [36], and consensus problems of multi-agent systems [38].

3.1.7 Notation

Throughout this chapter, $\mathbb{H}$ denotes a separable Hilbert space equipped with the inner product $\langle \cdot, \cdot \rangle$ and norm $\| \cdot \|$, and $(\Omega, \mathcal{F}, P)$ denotes the underlying probability space, where $\Omega$, $\mathcal{F}$, and $P$ are the sample space, $\sigma$-algebra, and probability measure, respectively. The map $x : (\Omega, \mathcal{F}) \to (\mathbb{H}, \mathcal{B})$, where $\mathcal{B}$ is the Borel $\sigma$-algebra, is an $\mathbb{H}$-valued random variable. Let $(x^k)_{k \geq 0}$ denote either a sequence of deterministic points in $\mathbb{H}$ or a sequence of $\mathbb{H}$-valued random variables, which will be clear from the context, and let $x_i \in \mathbb{H}_i$ denote the $i$th coordinate of $x$.

In addition, we let $\mathcal{X}^k := \sigma(x^0, x^1, ..., x^k)$ denote the smallest $\sigma$-algebra generated by $x^0, x^1, ..., x^k$. “Almost surely” is abbreviated as “a.s.”, and the $n$ product space of $\mathbb{H}$ is denoted by $\mathbb{H}^n$. We use $\to$ and $\rightharpoonup$ for strong convergence and weak convergence, respectively.

We define $\text{Fix } \mathcal{T} := \{ x \in \mathbb{H} \mid \mathcal{T}x = x \}$ as the set of fixed points of operator $\mathcal{T}$, and, in the product space, we let $X^* := \{(x^*, x^*, ..., x^*) \mid x^* \in \text{Fix } \mathcal{T}\} \subseteq \mathbb{H}^{r+1}$.

3.2 Convergence

We establish weak and strong convergence in Section 3.2.1 and linear convergence in Section 3.2.2. Step size selection is also discussed.

3.2.1 Almost Sure Convergence

Throughout the analysis, we let $p_{\min} = \min_i p_i > 0$ and $|J(k)|$ be the number of elements in $J(k)$ (see Section 3.1.2). Only for the purpose of analysis, we define
the (never computed) full update at \( k \)th iteration:

\[
\bar{x}^{k+1} := x^k - \eta_k \hat{S} \hat{x}^k. \tag{3.7}
\]

Lemma 1 below shows that \( \mathcal{T} \) is nonexpansive if and only if \( S \) is \( 1/2 \)-cocoercive.

**Lemma 1** \( \)\( \) Operator \( \mathcal{T} : \mathbb{H} \to \mathbb{H} \) is nonexpansive if and only if \( S = I - \mathcal{T} \) is \( 1/2 \)-cocoercive, i.e., \( \langle x - y, Sx - Sy \rangle \geq \frac{1}{2} \| Sx - Sy \|^2 \), \( \forall \ x, y \in \mathbb{H} \).

**Proof 2** See textbook [8, Proposition 4.33] for the proof of the “if” part, and the “only if” part, though missing there, follows by just reversing the proof.

The lemma below develops an an upper bound for the expected distance between \( x^{k+1} \) and any \( x^* \in \text{Fix} \mathcal{T} \).

**Lemma 2** \( \) Let \( (x^k)_{k \geq 0} \) be the sequence generated by Algorithm 2. Then for any \( x^* \in \text{Fix} \mathcal{T} \) and \( \gamma > 0 \) (to be optimized later), we have

\[
\mathbb{E}(\|x^{k+1} - x^*\|^2 | \mathcal{X}^k) \leq \|x^k - x^*\|^2 + \frac{\gamma}{m} \sum_{d \in J(k)} \| x^d - x^{d+1} \|^2 \\
+ \frac{1}{m} \left( \frac{|J(k)|}{\gamma} + \frac{1}{mp_{\text{min}}} - \frac{1}{\eta_k} \right) \| x^k - \bar{x}^{k+1} \|^2. \tag{3.8}
\]

**Proof 3** \( \) Recall \( \text{Prob}(i_k = i) = p_i. \) Then we have

\[
\mathbb{E} \left( \| x^{k+1} - x^* \|^2 \mid \mathcal{X}^k \right) = \mathbb{E} \left( \| x^k - \frac{\eta_k}{mp_k} \hat{S} \hat{x}^k - x^* \|^2 \mid \mathcal{X}^k \right)
= \| x^k - x^* \|^2 + \mathbb{E} \left( \frac{2\eta_k}{mp_k} \langle \hat{S} \hat{x}^k, x^* - x^k \rangle + \frac{\eta_k^2}{m^2 p_k^2} \| \hat{S} \hat{x}^k \|^2 \mid \mathcal{X}^k \right) \tag{3.9}
= \| x^k - x^* \|^2 + \frac{2\eta_k}{m} \sum_{i = 1}^m \langle S_i \hat{x}^k, x^* - x^k \rangle + \frac{\eta_k^2}{m^2} \sum_{i = 1}^m \frac{1}{p_i} \| S_i \hat{x}^k \|^2
= \| x^k - x^* \|^2 + \frac{2\eta_k}{m} \sum_{i = 1}^m \langle S_i \hat{x}^k, x^* - x^k \rangle + \frac{\eta_k^2}{m^2} \sum_{i = 1}^m \frac{1}{p_i} \| S_i \hat{x}^k \|^2.
\]

Note that

\[
\sum_{i = 1}^m \frac{1}{p_i} \| S_i \hat{x}^k \|^2 \leq \frac{1}{p_{\text{min}}} \sum_{i = 1}^m \| S_i \hat{x}^k \|^2 = \frac{1}{p_{\text{min}}} \| S \hat{x}^k \|^2 \overset{(3.7)}{=} \frac{1}{\eta_k p_{\text{min}}} \| x^k - \bar{x}^{k+1} \|^2, \tag{3.10}
\]
sequences of
Lemma 3 ([89, Theorem 1])

\[ (3.11) \]
\[ (3.10) \]
\[ (3.7) \]
\[ (3.5) \]
\[ (3.9) \]
\[ (3.8) \]
\[ (3.7) \]
\[ (3.10) \]

where the first inequality follows from the Young's inequality. Plugging (3.10) and (3.11) into (3.9) gives the desired result.

We need the following lemma on nonnegative almost supermartingales [89].

Lemma 3 ([89, Theorem 1]) Let \( \mathcal{F} = (\mathcal{F}^k)_{k \geq 0} \) be a sequence of sub-sigma algebras of \( \mathcal{F} \) such that \( \forall k \geq 0, \mathcal{F}^k \subset \mathcal{F}^{k+1} \). Define \( \ell_+(\mathcal{F}) \) as the set of sequences of \( [0, +\infty) \)-valued random variables \( (\xi_k)_{k \geq 0} \), where \( \xi_k \) is \( \mathcal{F}^k \) measurable, and \( \ell_1(\mathcal{F}) := \{ (\xi_k)_{k \geq 0} \in \ell_+(\mathcal{F}) | \sum_k \xi_k < +\infty \text{ a.s.} \} \). Let \( (\alpha_k)_{k \geq 0}, (v_k)_{k \geq 0} \in \ell_+(\mathcal{F}), \) and \( (\eta_k)_{k \geq 0}, (\xi_k)_{k \geq 0} \in \ell_1(\mathcal{F}) \) be such that

\[
\mathbb{E}(\alpha_{k+1}|\mathcal{F}^k) + v_k \leq (1 + \xi_k)\alpha_k + \eta_k.
\]

Then \( (v_k)_{k \geq 0} \in \ell_1(\mathcal{F}) \) and \( \alpha_k \) converges to a \( [0, +\infty) \)-valued random variable a.s.

Let \( \mathbb{H}^{\tau+1} = \prod_{i=0}^{\tau} \mathbb{H} \) be a product space and \( \langle \cdot | \cdot \rangle \) be the induced inner product:

\[
\langle \langle z^0, \ldots, z^\tau \rangle | \langle y^0, \ldots, y^\tau \rangle \rangle = \sum_{i=0}^{\tau} \langle z^i | y^i \rangle, \quad \forall (z^0, \ldots, z^\tau), (y^0, \ldots, y^\tau) \in \mathbb{H}^{\tau+1}.
\]

Let \( M' \) be a symmetric \( (\tau+1) \times (\tau+1) \) tri-diagonal matrix with its main diagonal as

\[
\sqrt{P_{\min}}\left[ \frac{1}{\sqrt{P_{\min}}} + \tau, 2\tau-1, 2\tau-3, \ldots, 1 \right]
\]

and first off-diagonal as \( -\sqrt{P_{\min}}[\tau, \tau-1, \ldots, 1] \), and let \( M = M' \otimes I_H \). Here \( \otimes \) represents the Kronecker product. For a given
\((y^0, \cdots, y^\tau) \in \mathbb{H}^{\tau+1}, (z^0, \cdots, z^\tau) = M(y^0, \cdots, y^\tau)\) is given by:

\[
\begin{align*}
    z^0 &= y^0 + \sqrt{p_{\min}}(y^0 - y^1), \\
    z^i &= \sqrt{p_{\min}}((i - \tau - 1)y^{i-1} + (2\tau - 2i + 1)y^i + (i - \tau)y^{i+1}), \text{ if } 1 \leq i \leq \tau - 1, \\
    z^\tau &= \sqrt{p_{\min}}(y^\tau - y^{\tau-1}).
\end{align*}
\]

Then \(M\) is a self-adjoint and positive definite linear operator since \(M'\) is symmetric and positive definite, and we define \(\langle \cdot | \cdot \rangle_M = \langle \cdot | M \cdot \rangle\) as the \(M\)-weighted inner product and \(\| \cdot \|_M\) the induced norm. Let

\[
x^k = (x^k, x^{k-1}, \ldots, x^{k-\tau}) \in \mathbb{H}^{\tau+1}, \quad k \geq 0, \quad \text{and} \quad x^* = (x^*, x^*, \ldots, x^*) \in X^* \subseteq \mathbb{H}^{\tau+1},
\]

where we set \(x^k = x^0\) for \(k < 0\). With

\[
\xi_k(x^*) := \|x^k - x^*\|_M^2 = \|x^k - x^*\|_M^2 + \sqrt{p_{\min}} \sum_{i=k-\tau}^{k-1} (i - (k - \tau) + 1) \|x^i - x^{i+1}\|_M^2,
\]

we have the following fundamental inequality:

**Theorem 2 (Fundamental inequality)** Let \((x^k)_{k \geq 0}\) be the sequence generated by ARock. Then for any \(x^* \in X^*\), it holds that

\[
\mathbb{E} \left( \xi_{k+1}(x^*) \bigg| X^k \right) + \frac{1}{m} \left( \frac{1}{\eta_k} - \frac{2\tau}{m\sqrt{p_{\min}}} - \frac{1}{mp_{\min}} \right) \|\bar{x}^{k+1} - x^k\|_M^2 \leq \xi_k(x^*). \tag{3.13}
\]

**Proof 4** Let \(\gamma = m\sqrt{p_{\min}}\). Since \(J(k) \subset \{k - 1, \cdots, k - \tau\}\), then (3.8) indicates

\[
\mathbb{E}(\|x^{k+1} - x^*\|_M^2 | X^k) \leq \|x^k - x^*\|_M^2 + \frac{1}{\sqrt{p_{\min}}} \sum_{i=k-\tau}^{k-1} \|x^i - x^{i+1}\|_M^2 \\
+ \frac{1}{m} \left( \frac{\tau}{m\sqrt{p_{\min}}} + \frac{1}{mp_{\min}} - \frac{1}{\eta_k} \right) \|x^k - \bar{x}^{k+1}\|_M^2. \tag{3.14}
\]

From (3.3) and (3.7), it is easy to have \(\mathbb{E}(\|x^k - x^{k+1}\|_M^2 | X^k) \leq \frac{1}{mp_{\min}} \|x^k - \bar{x}^{k+1}\|_M^2\), which together with (3.14) implies (3.13) by using the definition of \(\xi_k(x^*)\).

**Remark 5 (Stochastic Fejér monotonicity)** From (3.13), if \(0 < \eta_k \leq \frac{mp_{\min}}{2\tau \sqrt{p_{\min} + 1}}\), then we have \(\mathbb{E}(\|x^{k+1} - x^*\|_M^2 | X^k) \leq \|x^k - x^*\|_M^2, \forall x^* \in X^*\).
Remark 6 Let us check our step size bound \( \frac{mp_{\min}}{2r\sqrt{p_{\min}+1}} \). Consider the uniform case: \( p_{\min} \equiv p_i \equiv \frac{1}{m} \). Then, the bound simplifies to \( \frac{1}{1+2r/\sqrt{m}} \). If the max delay is no more than the square root of coordinates, i.e., \( \tau = O(\sqrt{m}) \), then the bound is \( O(1) \). In general, \( \tau \) depends on several factors such as problem structure, system architecture, load balance, etc. If all updates and agents are identical, then \( \tau \) is proportional to \( p \), the number of agents. Hence, ARock takes an \( O(1) \) step size for solving a problem with \( m \) coordinates by \( p = \sqrt{m} \) agents under balanced loads.

The next lemma is a direct consequence of the invertibility of the metric \( M \).

Lemma 4 A sequence \((z^k)_{k \geq 0} \subset \mathbb{H}^{r+1}\) (weakly) converges to \( z \in \mathbb{H}^{r+1} \) under the metric \( \langle \cdot | \cdot \rangle \) if and only if it does so under the metric \( \langle \cdot | \cdot \rangle_M \).

In light of Lemma 4, the metric of the inner product for weak convergence in the next lemma is not specified. The lemma and its proof are adapted from [26].

Lemma 5 Let \((x^k)_{k \geq 0} \subset \mathbb{H} \) be the sequence generated by ARock with \( \eta_k \in [\eta_{\min}, \frac{c^mp_{\min}}{2r\sqrt{p_{\min}+1}}] \) for any \( \eta_{\min} > 0 \) and \( 0 < c < 1 \). Then we have:

(i) \( \sum_{k=0}^{\infty} \|x^k - \tilde{x}^{k+1}\|^2 < \infty \) a.s..

(ii) \( x^k - x^{k+1} \rightarrow 0 \) a.s. and \( \tilde{x}^k - x^{k+1} \rightarrow 0 \) a.s.

(iii) The sequence \((x^k)_{k \geq 0} \subset \mathbb{H}^{r+1}\) is bounded a.s..

(iv) There exists \( \hat{\Omega} \in \mathcal{F} \) such that \( P(\hat{\Omega}) = 1 \) and, for every \( \omega \in \hat{\Omega} \) and every \( x^* \in X^* \), \( (\|x^k(\omega) - x^*\|_M)_{k \geq 0} \) converges.

(v) Let \( \mathcal{Z}(x^k) \) be the set of weakly convergent cluster points of \((x^k)_{k \geq 0}\). Then, \( \mathcal{Z}(x^k) \subseteq X^* \) a.s..

Proof 5 (i): Applying Lemma 3 with \( \xi_k = \eta_k = 0 \) and \( \alpha_k = \xi_k(x^*) \), \( \forall k \) to (3.13) and noting \( \inf_k \left( \frac{1}{\eta_k} - \frac{2r}{m\sqrt{p_{\min}}} - \frac{1}{mp_{\min}} \right) > 0 \) gives this result directly.
(ii) From (i), we have \(x^k - \hat{x}^{k+1} \to 0\) a.s. Since \(\|x^k - x^{k+1}\| \leq \frac{1}{m_{\min}}\|x^k - \hat{x}^{k+1}\|\), we have \(x^k - x^{k+1} \to 0\) a.s. Then from (3.5), we have \(\hat{x}_k^k - x^k \to 0\) a.s.

(iii): From Lemma 3, we have that \((\|x^k - x^*\|_M^2)_{k \geq 1}\) converges a.s. and so does \((\|x^k - x^*\|_M)_{k \geq 1}\), i.e., \(\lim_{k \to \infty} \|x^k - x^*\|_M = \gamma\) a.s., where \(\gamma\) is a \([0, +\infty)\)-valued random variable. Hence, \((\|x^k - x^*\|_M)_{k \geq 1}\) must be bounded a.s. and so is \((x^k)_{k \geq 1}\).

(iv): The proof follows directly from \([26, Proposition 2.3 (iii)]\). It is worth noting that \(\tilde{\Omega}\) in the statement works for all \(x^* \in X^*\), namely, \(\tilde{\Omega}\) does not depend on \(x^*\).

(v): By (ii), there exists \(\hat{\Omega} \in \mathcal{F}\) such that \(P(\hat{\Omega}) = 1\) and

\[
x^k(w) - x^{k+1}(w) \to 0, \quad \forall w \in \hat{\Omega}.
\] (3.15)

For any \(\omega \in \hat{\Omega}\), let \((x^{k_n}(\omega))_{n \geq 1}\) be a weakly convergent subsequence of \((x^k(\omega))_{k \geq 1}\), i.e., \(x^{k_n}(\omega) \rightharpoonup x\), where \(x^{k_n}(\omega) = (x^{k_n}(\omega), x^{k_n-1}(\omega), \ldots, x^{k_n-\tau}(\omega))\) and \(x = (u^0, \ldots, u^\tau)\). Note that \(x^{k_n}(\omega) \rightharpoonup x\) implies \(x^{k_n-i}(\omega) \rightharpoonup u^i, \forall j\). Therefore, \(u^i = u^j\), for any \(i, j \in \{0, \ldots, \tau\}\) because \(x^{k_n-i}(\omega) - x^{k_n-j}(\omega) \to 0\).

Furthermore, observing \(\eta_k \geq \eta_{\min} > 0\), we have

\[
\lim_{n \to \infty} \hat{x}^{k_n}(\omega) - T\hat{x}^{k_n}(\omega) = \lim_{n \to \infty} S\hat{x}^{k_n}(\omega) = \lim_{n \to \infty} \frac{1}{\eta_{k_n}}(x^{k_n}(\omega) - \hat{x}^{k_n+1}(\omega)) = 0. \quad (3.16)
\]

From the triangle inequality and the nonexpansiveness of \(T\), it follows that

\[
\|x^{k_n}(\omega) - Tx^{k_n}(\omega)\|
\]

\[
= \|x^{k_n}(\omega) - \hat{x}^{k_n}(\omega) + \hat{x}^{k_n}(\omega) - T\hat{x}^{k_n}(\omega) + T\hat{x}^{k_n}(\omega) - Tx^{k_n}(\omega)\|
\]

\[
\leq \|x^{k_n}(\omega) - \hat{x}^{k_n}(\omega)\| + \|\hat{x}^{k_n}(\omega) - T\hat{x}^{k_n}(\omega)\| + \|T\hat{x}^{k_n}(\omega) - Tx^{k_n}(\omega)\|
\]

\[
\leq 2 \|x^{k_n}(\omega) - \hat{x}^{k_n}(\omega)\| + \|\hat{x}^{k_n}(\omega) - T\hat{x}^{k_n}(\omega)\|
\]

\[
\leq 2 \sum_{d \in J(k_n)} \|x^d(\omega) - x^{d+1}(\omega)\| + \|\hat{x}^{k_n}(\omega) - T\hat{x}^{k_n}(\omega)\|.
\]

From (3.15), (3.16), and the above inequality, it follows \(\lim_{n \to \infty} x^{k_n}(\omega) - Tx^{k_n}(\omega) = 0\). Finally, the demiclosedness principle \([8, Theorem 4.17]\) implies \(u^0 \in \text{Fix } T\).
Theorem 3 Under the assumptions of Lemma 5, the sequence $(x^k)_{k \geq 0}$ weakly converges to an $X^\ast$-valued random variable a.s.. In addition, if $T$ is demicompact at 0, $(x^k)_{k \geq 0}$ strongly converges to an $X^\ast$-valued random variable a.s..

Proof 6 The proof for a.s. weak convergence follows from Opial’s Lemma [72, 82] and Lemma 5 (iv)-(v). Next we assume that $T$ is demicompact at 0. From the proof of Lemma 5 (v), there is $\hat{\Omega} \in \mathcal{F}$ such that $P(\hat{\Omega}) = 1$ and, for any $w \in \hat{\Omega}$ and any weakly convergent subsequence of $(x^{k_n}(w))_{n \geq 0}$, $\lim_{n \to \infty} x^{k_n}(w) - T x^{k_n}(w) = 0$. Since $T$ is demicompact, $(x^{k_n}(w))_{n \geq 0}$ has a strongly convergent subsequence, for which we still use $(x^{k_n}(w))_{n \geq 0}$.

Then by Lemma 5 (iv), there is $\tilde{\Omega} \in \mathcal{F}$ such that $P(\tilde{\Omega}) = 1$ and, for every $w \in \tilde{\Omega}$ and every $x^* \in X^\ast$, $(\|x^k(w) - x^*\|_M)_{k \geq 0}$ converges. Thus, for any $w \in \hat{\Omega} \cap \tilde{\Omega}$, we have $\lim_{k \to \infty} \|x^k(w) - \tilde{x}(w)\|_M = 0$. Because $P(\hat{\Omega} \cap \tilde{\Omega}) = 1$, we conclude that $(x^k)_{k \geq 0}$ strongly converges to an $X^\ast$-valued random variable a.s..

Remark 7 For the generalization in Section 3.1.4, we need to replace (3.10) by

$$\sum_{i=1}^{m} \frac{1}{p_i} \|U_i \circ \mathcal{S} \tilde{x}^k\|^2 \leq \frac{1}{p_{\min}} \sum_{i=1}^{m} \|U_i \circ \mathcal{S} \tilde{x}^k\|^2 \leq \frac{C}{p_{\min}} \|\mathcal{S} \tilde{x}^k\|^2 = \frac{C}{\eta_{\min} p_{\min}} \|x^k - \tilde{x}^{k+1}\|^2,$$

and update the step size condition to $\eta_k \in [\eta_{\min}, \frac{c(p_{\min})}{2r\sqrt{p_{\min}+C}}]$. Then the proofs of Theorem 3 and Lemma 5 will go through and yield the same convergence result.

3.2.2 Linear Convergence

In this section, we establish linear convergence under the assumption that $S$ is quasi-strongly monotone. We first present a key lemma.

Lemma 6 Assume that the step size is fixed, i.e., $\eta_k = \eta$, and satisfies

$$0 < \eta \leq \eta_1 := (1 - \frac{1}{\rho}) \frac{m \sqrt{p_{\min}}}{8} \frac{\rho^{1/2-1}}{\rho^{r+1/2-1}}$$  (3.17)

55
for some $\rho > 1$. Then we have, for all $k \geq 1$,

$$
E\|\bar{x}^k - x^{k-1}\|^2 \leq \rho E\|\bar{x}^{k+1} - x^k\|^2.
$$

(3.18)

**Proof 7** We prove (3.18) by induction. First, based on the inequality $\|a\|^2 - \|b\|^2 \leq 2\|a\|\|b - a\|$ we observe that, for any $k \geq 1$,

$$
\|\bar{x}^k - x^{k-1}\|^2 - \|\bar{x}^{k+1} - x^k\|^2 \leq 2\|\bar{x}^k - x^{k-1}\|\|\bar{x}^{k+1} - x^k - \bar{x}^k + x^{k-1}\|
$$

\[
= 2\|\bar{x}^k - x^{k-1}\|\eta S(\bar{x}^k) - \eta S(\bar{x}^{k-1})
\]

\[
\leq 4\eta\|\bar{x}^k - x^{k-1}\||\bar{x}^k - \bar{x}^{k-1}|.
\]

(3.19)

Applying the triangle inequality and (3.5) yields

$$
\|\bar{x}^k - x^{k-1}\| \leq \|\bar{x}^k - \bar{x}^k\| + \|\bar{x}^k - x^{k-1}\| + \|x^{k-1} - \bar{x}^{k-1}\|
$$

\[
\leq \sum_{d \in J(k)} \|x^d - x^{d+1}\| + \|x^{k-1} - x^{k-1}\| + \sum_{d \in J(k-1)} \|x^d - x^{d+1}\|
\]

\[
\leq 2 \sum_{t=0}^T \|x^{k-t} - x^{k-t-1}\|.
\]

(3.20)

For the basic case, we have $\bar{x}^0 = x^0$, $\bar{x}^1 \in \{x^0, x^1\}$. Letting $k = 1$ in (3.19) gets us

$$
E\|\bar{x}^1 - x^0\|^2 - E\|\bar{x}^2 - x^1\|^2 \leq 4\eta E\|\bar{x}^1 - x^0\||x^1 - x^0|
$$

\[
\leq 2\eta \left( \frac{1}{m\sqrt{p_{\text{min}}}} E\|\bar{x}^1 - x^0\|^2 + m\sqrt{p_{\text{min}}} E\|x^1 - x^0\|^2 \right)
\]

\[
= 2\eta \left( \frac{1}{m\sqrt{p_{\text{min}}}} E\|\bar{x}^1 - x^0\|^2 + m\sqrt{p_{\text{min}}} \sum_{i=1}^m p_i \frac{\eta^2}{m}\|S_i x^0\|^2 \right)
\]

\[
\leq 2\eta \left( \frac{1}{m\sqrt{p_{\text{min}}}} E\|\bar{x}^1 - x^0\|^2 + \frac{1}{m\sqrt{p_{\text{min}}}} E\|\bar{x}^1 - x^0\|^2 \right)
\]

\[
= \frac{4\eta}{m\sqrt{p_{\text{min}}}} E\|\bar{x}^1 - x^0\|^2.
\]

Rearranging the above inequality yields

$$
E\|\bar{x}^1 - x^0\|^2 \leq \frac{1}{1 - \frac{4\eta}{m\sqrt{p_{\text{min}}}}} E\|\bar{x}^2 - x^1\|^2.
$$

By (3.17) and $\rho > 1$, it holds that $0 < \eta \leq (1 - \frac{1}{\rho}) \frac{m\sqrt{p_{\text{min}}}}{8} \frac{\rho^{1/2} - 1}{\rho^{(r+1)/2} - 1} \leq (1 - \frac{1}{\rho}) \frac{m\sqrt{p_{\text{min}}}}{4}$.

Hence, $E\|\bar{x}^1 - x^0\|^2 \leq \rho E\|\bar{x}^2 - x^1\|^2$.
For the induction step, applying Young’s inequality gives us
\[
\mathbb{E} \| \bar{x}^{k} - x^{k-1} \| \| x^{k-t} - x^{k-t-1} \| \leq \frac{1}{2} \mathbb{E} \left\{ a \| x^{k-t} - x^{k-t-1} \|^2 + \frac{1}{a} \| x^{k} - x^{k-1} \|^2 \right\}
\[
\leq \frac{1}{2} \mathbb{E} \left\{ \frac{m^2 p_{\min}}{a} \| x^{k-t} - x^{k-t-1} \|^2 + \frac{1}{a} \| x^{k} - x^{k-1} \|^2 \right\}
\[
\leq \frac{1}{2} \left\{ \frac{\rho \tau}{m^2 p_{\min}} + \frac{1}{a} \right\} \mathbb{E} \| x^{k} - x^{k-1} \|^2 
\[
= \frac{\rho^{t/2}}{m \sqrt{p_{\min}}} \mathbb{E} \| \bar{x}^{k} - x^{k-1} \|^2. \quad \text{(letting } a = m \sqrt{p_{\min}} \rho^{-t/2})
\]

Taking the expectation on (3.20) and combining it with (3.19) yield
\[
\mathbb{E} \| \bar{x}^{k} - x^{k-1} \|^2 - \mathbb{E} \| \bar{x}^{k+1} - x^{k} \|^2 \leq 8 \eta \sum_{t=0}^{\tau} \mathbb{E} \| \bar{x}^{k} - x^{k-1} \| \| x^{k-t} - x^{k-t-1} \|
\[
\leq \frac{8 \eta}{m \sqrt{p_{\min}}} \sum_{t=0}^{\tau} \rho^{t/2} \mathbb{E} \| \bar{x}^{k} - x^{k-1} \|^2 \leq \frac{8 \eta}{m \sqrt{p_{\min}}} \frac{1 - \rho^{(\tau+1)/2}}{1 - \rho^{1/2}} \mathbb{E} \| \bar{x}^{k} - x^{k-1} \|^2.
\]

Finally, rearranging the above inequality and using (3.17) lead to \( \mathbb{E} \| \bar{x}^{k} - x^{k-1} \|^2 \leq \rho \mathbb{E} \| \bar{x}^{k+1} - x^{k} \|^2. \) This completes the proof.

With this lemma, we are ready to derive the linear convergence rate of ARock.

**Theorem 4 (Linear convergence)** Assume that \( S \) is quasi-\( \mu \)-strongly monotone with \( \mu > 0 \). Let \( \beta \in (0, 1) \) and \( (x^{k})_{k \geq 0} \) be the sequence generated by ARock with a constant stepsize \( \eta \in (0, \min\{\eta_1, \eta_2\}) \), where \( \eta_1 \) is given in (3.17) and
\[
\eta_2 = \frac{-b + \sqrt{b^2 + 4(1-\beta)a}}{2a}, \quad a = \frac{2 \beta \mu \tau}{m^2 p_{\min} \rho (\rho^\tau - 1)}, \quad b = \frac{1}{m p_{\min}} + \frac{2}{m} \sqrt{\frac{\rho (\rho^\tau - 1) \tau}{(\rho - 1) p_{\min}}}. \quad (3.21)
\]
Then
\[
\mathbb{E} \left( \| x^k - x^* \|^2 \right) \leq \left( 1 - \frac{\beta \mu}{m} \right)^k \| x^0 - x^* \|^2. \quad (3.22)
\]
\[\mathbb{E}(\|x^{k+1} - x^*\|^2 | \mathcal{X}^k) \leq (1 - \frac{\beta \mu}{m}) \|x^k - x^*\|^2 + \frac{1}{m} (2\beta \eta \mu \tau + \gamma) \sum_{d=k-\tau}^{k-1} \|x^d - x^{d+1}\|^2 \]
\[+ \frac{1}{m} \left( \frac{\tau}{\gamma} + \frac{1}{m \eta \mu} - \frac{1 - \beta}{\eta} \right) \|x^k - \bar{x}^{k+1}\|^2.\]

Taking expectation over both sides of the above inequality, noting \(\mathbb{E}\|x^d - x^{d+1}\|^2 \leq \frac{1}{m \eta \mu} \mathbb{E}\|x^d - \bar{x}^{d+1}\|^2\), and using Lemma 6, we have...
$$\mathbb{E}(\|x^{k+1} - x^*\|^2)$$
\[
\leq (1 - \frac{\beta \mu \eta}{m}) \mathbb{E}\|x^k - x^*\|^2 + \frac{1}{m p_{\text{min}}} \left(2\beta \eta \mu \tau + \gamma\right) \sum_{d=1}^\tau \rho^d \mathbb{E}\|x^k - x^{k+1}\|^2 \\
+ \frac{1}{m} \left(\frac{\tau}{\gamma} + \frac{1}{m p_{\text{min}}} - \frac{1-\beta}{\eta}\right) \mathbb{E}\|x^k - \bar{x}^{k+1}\|^2 \\
= (1 - \frac{\beta \mu \eta}{m}) \mathbb{E}\|x^k - x^*\|^2 + \frac{1}{m p_{\text{min}}} \left(2\beta \eta \mu \tau + \gamma\right) \frac{\rho(\rho^\tau - 1)}{\rho - 1} \mathbb{E}\|x^k - \bar{x}^{k+1}\|^2 \\
+ \frac{1}{m} \left(\frac{\tau}{\gamma} + \frac{1}{m p_{\text{min}}} - \frac{1-\beta}{\eta}\right) \mathbb{E}\|x^k - \bar{x}^{k+1}\|^2 \\
= (1 - \frac{\beta \mu \eta}{m}) \mathbb{E}\|x^k - x^*\|^2 \\
+ \frac{1}{m} \left(\frac{2\beta \mu \tau \rho(\rho^\tau - 1)}{m p_{\text{min}} \rho - 1} + \frac{2}{m} \sqrt{\frac{\rho(\rho^\tau - 1)\tau}{(\rho - 1)p_{\text{min}}} + \frac{1}{m p_{\text{min}}} - \frac{1-\beta}{\eta}}\right) \mathbb{E}\|x^k - \bar{x}^{k+1}\|^2 \\
\leq (1 - \frac{\beta \mu \eta}{m}) \mathbb{E}\|x^k - x^*\|^2 ,
\]

where we have let $\gamma = m \sqrt{\frac{\tau(\rho^\tau - 1)p_{\text{min}}}{\rho(\rho^\tau - 1)}}$ in the second equality, and the last inequality holds because of the choice of $\eta$. Therefore, (3.22) holds.

**Remark 8** Assume $i_k$ is chosen uniformly at random, so $p_{\text{min}} = \frac{1}{m}$. We consider the case when $m$ and $\tau$ are large. Let $\sqrt{\rho} = 1 + \frac{1}{\tau}$. Then from the fact that $(1 + \frac{1}{\tau})^k$ increasingly converges to the natural number $e$, we have from (3.17) that $\eta_1 = O(\sqrt{m})$. In addition, note from (3.21) that $a = O(b^2) = O(\frac{\tau^2}{m})$, and thus $\eta_2 = O(\sqrt{m})$. Therefore, if $\tau = O(m^{\frac{1}{4}})$, then the stepsize in Theorem 4 can be $\eta = O(1)$. Hence, linear speedup can be achieved.
CHAPTER 4

Applications

In this chapter, we provide examples to illustrate how to develop coordinate update algorithms based on CF operators and AROck. The applications are categorized into six different areas. The first section provides the application of AROck for solving linear equations. The following sections discuss applications in various optimization areas, including convex smooth optimization, convex nonsmooth optimization, constrained optimization, distributed and decentralized optimization and nonconvex optimization.

For each problem, we describe the operator $\mathcal{T}$ and how to efficiently calculate $(\mathcal{T}x)_i$. The final algorithm is obtained after plugging the update in a coordinate update framework in Section 1.1 along with parameter initialization, an index selection rule, as well as some termination criteria. The extension of the algorithms to async-parallel setting under the AROck framework will also be discussed.

4.1 Linear Equations

Consider the linear system $Ax = b$, where $A \in \mathbb{R}^{m \times m}$ is a nonsingular matrix with nonzero diagonal entries. Let $A = D + R$, where $D$ and $R$ are the diagonal and off-diagonal parts of $A$, respectively. Let $M := -D^{-1}R$ and $\mathcal{T}x := Mx + D^{-1}b$. Then the system $Ax = b$ is equivalent to the fixed-point problem $x = D^{-1}(b - Rx) =: \mathcal{T}x$, where $\mathcal{T}$ is nonexpansive if the spectral norm $\|M\|_2$ satisfies $\|M\|_2 \leq 1$. The iteration $x^{k+1} = \mathcal{T}x^k$ is widely known as the Jacobi algorithm. Let $\mathcal{S} = I - \mathcal{T}$.
Each update $S_{ik}\hat{x}^k$ involves multiplying just the $i_k$th row of $M$ to $x$ and adding the $i_k$th entry of $D^{-1}b$, so we arrive at the following algorithm.

**Algorithm 3: ARock for linear equations**

**Input**: $x^0 \in \mathbb{R}^n$, $K > 0$.

1. set the global iteration counter $k = 0$;
2. while $k < K$, every agent asynchronously and continuously do
   - select $i_k \in [m]$ uniformly at random;
   - subtract $\frac{\eta_k}{a_{ik}}(\sum_j a_{ikj}\hat{x}_j^k - b_{ik})$ from the component $x_{ik}$ of the variable $x$;
   - update the global counter $k \leftarrow k + 1$;

**Proposition 2** [8, Example 22.5] Suppose that $T$ is $c$-Lipschitz continuous with $c \in [0,1)$. Then, $I - T$ is $(1 - c)$-strongly monotone.

Suppose $\|M\|_2 < 1$. Since $T$ is $\|M\|_2$-Lipschitz continuous, by Proposition 2, $S$ is $(1 - \|M\|_2)$-strongly monotone. By Theorem 4, Algorithm 3 converges linearly.

### 4.2 Convex Smooth Optimization

Consider the optimization problem

$$
\minimize_{x \in \mathcal{H}} f(x),
$$

where $f$ is a closed proper convex differentiable function and $\nabla f$ is $L$-Lipschitz continuous, $L > 0$. Let $S := \frac{2}{L}\nabla f$. As $f$ is convex and differentiable, $x$ is a minimizer of $f$ if and only if $x$ is a zero of $S$. Note that $S$ is $\frac{1}{2}$-cocoercive. By Lemma 1, $T \equiv I - S$ is nonexpansive and CF if $\nabla f$ is a CF operator. Applying ARock, we have the following iteration:

$$
x^{k+1} = x^k - \eta_k S_{ik}\hat{x}^k,
$$

where $S_{ik}x = \frac{2}{L}(0, ..., 0, \nabla_{ik} f(x), 0, ..., 0)^T$. Note that $\nabla f$ needs a structure that makes it cheap to compute $\nabla_{ik} f(\hat{x}^k)$. Let us give two such examples: (i) quadratic
programming: \( f(x) = \frac{1}{2}x^T Ax - b^T x \), where \( \nabla f(x) = Ax - b \) and \( \nabla_i f(\hat{x}^k) \) only depends on a part of \( A \) and \( b \); (ii) sum of sparsely supported functions: \( f = \sum_{j=1}^N f_j \) and \( \nabla f = \sum_{j=1}^N \nabla f_j \), where each \( f_j \) depends on just a few variables.

Theorem 3 below guarantees the convergence of \((x^k)_{k\geq 0}\) if \( \eta_k \in [\eta_{\min}, \frac{1}{2\tau/\sqrt{m+1}}] \).

In addition, if \( f(x) \) is restricted strongly convex, namely, for any \( x \in H \) and \( x^* \in X^* \), where \( X^* \) is the solution set to (4.1), we have \( \langle x - x^*, \nabla f(x) \rangle \geq \mu \|x - x^*\|^2 \) for some \( \mu > 0 \), then \( S \) is quasi-strongly monotone with modulus \( \mu \). According to Theorem 4, iteration (4.2) converges at a linear rate if the step size meets the condition therein.

Our convergence and rates are given in term of the distance to the solution set \( X^* \). In comparison, the results in the work [63] are given in terms of objective error under the assumption of a uniformly bounded \((x^k)_{k\geq 0}\). In addition, their step size decays like \( O\left(\frac{1}{\tau \rho^k}\right) \) for some \( \rho > 1 \) depending on \( \tau \), and our \( O\left(\frac{1}{\tau}\right) \) is better. Under similar assumptions, Bertsekas and Tsitsiklis [15, Section 7.5] also describes an algorithm for (4.1) and proves only subsequence convergence [15, Proposition 5.3] in \( \mathbb{R}^n \).

### 4.3 Convex Nonsmooth Optimization

#### 4.3.1 Minimize Smooth + Nonsmooth Functions

Consider the problem

\[
\min_{x \in H} f(x) + g(x),
\]

where \( f \) is closed proper convex and \( g \) is convex and \( L \)-Lipschitz differentiable with \( L > 0 \). Problems in the form of (4.3) arise in statistical regression, machine learning, and signal processing and include well-known problems such as the support vector machine, regularized least-squares, and regularized logistic regression.

For any \( x \in H \) and scalar \( \gamma \in (0, \frac{2}{L}) \), define \( A = \text{prox}_{\gamma f} \) and \( B = (I - \gamma \nabla g) \), in
the forward-backward operator (2.9), then we have $T_{\text{FBS}} := \text{prox}_{\gamma f} \circ (I - \gamma \nabla g)$. Because $\text{prox}_{\gamma f}$ is $\frac{1}{2}$-averaged and $(I - \gamma \nabla g)$ is $\frac{\gamma L}{2}$-averaged, $T_{\text{FBS}}$ is $\alpha$-averaged for $\alpha \in [\frac{2}{3}, 1]$ [8, Propositions 4.32 and 4.33]. Define $S := I - T_{\text{FBS}} = I - \text{prox}_{\gamma f} \circ (I - \gamma \nabla g)$. When we apply Algorithm 2 to $T_{\text{FBS}}$ to solve (4.3), and assume $f$ is separable in all coordinates, that is, $f(x) = \sum_{i=1}^{m} f_i(x_i)$, the update for the $i_k$th selected coordinate is

$$x_{i_k}^{k+1} = x_{i_k}^{k} - \eta_k \left( \hat{x}_{i_k}^{k} - \text{prox}_{\gamma f_{i_k}}(\hat{x}_{i_k}^{k} - \gamma \nabla_{i_k} g(\hat{x}_{i_k}^{k})) \right),$$ (4.4)

Examples of separable functions include $\ell_1$ norm, $\ell_2$ norm square, the Huber function, and the indicator function of box constraints, i.e., $\{x|a_i \leq x_i \leq b_i, \forall i\}$. They all have simple $\text{prox}$ maps. If $\eta_k \in [\eta_{\text{min}}, \frac{1}{2\gamma \sqrt{m+1}})$, then the convergence is guaranteed by Theorem 3. To show linear convergence, we need to assume that $g(x)$ is strongly convex. Then, Proposition 3 below shows that $\text{prox}_{\gamma f} \circ (I - \gamma \nabla g)$ is a quasi-contractive operator, and by Proposition 2, operator $I - \text{prox}_{\gamma f} \circ (I - \gamma \nabla g)$ is quasi-strongly monotone. Finally, linear convergence and its rate follow from Theorem 4.

**Proposition 3** Assume that $f$ is a closed proper convex function, and $g$ is $L$-Lipschitz differentiable and strongly convex with modulus $\mu > 0$. Let $\gamma \in (0, \frac{2}{L})$.

Then, both $I - \gamma \nabla g$ and $\text{prox}_{\gamma f} \circ (I - \gamma \nabla g)$ are quasi-contractive operators.

**Proof 9** We first show that $I - \gamma \nabla g$ is a quasi-contractive operator. Note

$$\|(x - \gamma \nabla g(x)) - (x^* - \gamma \nabla g(x^*))\|^2$$

$$= \|x - x^*\|^2 - 2\gamma \langle x - x^*, \nabla g(x) - \nabla g(x^*) \rangle + \gamma^2 \|\nabla g(x) - \nabla g(x^*)\|^2$$

$$\leq \|x - x^*\|^2 - \gamma(2 - \gamma L) \langle x - x^*, \nabla g(x) - \nabla g(x^*) \rangle$$

$$\leq (1 - 2\gamma \mu + \mu \gamma^2 L) \|x - x^*\|^2,$$

where the first inequality follows from the Baillon-Haddad theorem\(^1\) and the second one from the strong convexity of $g$. Hence, $I - \gamma \nabla g$ is quasi-contractive if $0 < \gamma < \frac{1}{L} - \text{cocoercive.}$

---

\(^1\)Let $g$ be a convex differentiable function. Then, $\nabla g$ is $L$-Lipschitz if and only if it is $\frac{1}{L}$-cocoercive.
Since \( f \) is convex, \( \text{prox}_{\gamma f} \) is firmly nonexpansive, and thus we immediately have the quasi-contractiveness of \( \text{prox}_{\gamma f} \circ (I - \gamma \nabla g) \) from that of \( I - \gamma \nabla g \).

### 4.3.2 Minimize Nonsmooth + Nonsmooth Functions

Consider

\[
\min_{x \in \mathbb{H}} f(x) + g(x),
\]

where both \( f(x) \) and \( g(x) \) are closed proper convex and their \( \text{prox} \) maps are easy to compute. Define the Peaceman-Rachford [61] operator:

\[
\mathcal{T}_{\text{PRS}} := \text{refl}_{\gamma f} \circ \text{refl}_{\gamma g}.
\]

Since both \( \text{refl}_{\gamma f} \) and \( \text{refl}_{\gamma g} \) are nonexpansive, their composition \( \mathcal{T}_{\text{PRS}} \) is also nonexpansive. Let \( S := I - \mathcal{T}_{\text{PRS}} \). When applying ARock to \( \mathcal{T} = \mathcal{T}_{\text{PRS}} \) to solve problem (4.5), the update (3.6) reduces to:

\[
z^{k+1} = z^k - \eta_k U_{i_k} \circ (I - \text{refl}_{\gamma f} \circ \text{refl}_{\gamma g}) \hat{z}^k,
\]

where we use \( z \) instead of \( x \) since the limit \( z^* \) of \( (z^k)_{k \geq 0} \) is not a solution to (4.5); instead, a solution must be recovered via \( x^* = \text{prox}_{\gamma g} z^* \). The convergence follows from Theorem 3 and that \( \mathcal{T}_{\text{PRS}} \) is nonexpansive. If either \( f \) or \( g \) is strongly convex, then \( \mathcal{T}_{\text{PRS}} \) is contractive and thus by Theorem 4, ARock converges linearly. A naive implementation of (4.6) is

\[
\hat{x}^k = \text{prox}_{\gamma g}(\hat{z}^k),
\]

\[
\hat{y}^k = \text{prox}_{\gamma f}(2\hat{x}^k - \hat{z}^k),
\]

\[
z^{k+1} = z^k + 2\eta_k U_{i_k} (\hat{y}^k - \hat{x}^k),
\]

where \( \hat{x}^k \) and \( \hat{y}^k \) are intermediate variables. Note that the order in which the proximal operators are applied to \( f \) and \( g \) affects both \( z^k \) [115] and whether coordinate-wise updates can be efficiently computed. Next, we present two special cases of (4.5) in Subsections 4.3.6 and 4.4.4 and discuss how to efficiently implement the update (4.7).
4.3.3 Group Lasso

The group Lasso regression problem [117] is

$$\min_{x \in \mathbb{R}^n} f(x) + \sum_{i=1}^{m} \lambda_i \|x_i\|_2, \quad (4.8)$$

where $f$ is a differentiable convex function, often bearing the form $\frac{1}{2}\|Ax - b\|_2^2$, and $x_i \in \mathbb{R}^{n_i}$ is a subvector of $x \in \mathbb{R}^n$ supported on $I_i \subset [n]$, and $\cup I_i = [n]$. If $I_i \cap I_j = \emptyset$, $\forall i \neq j$, it is called non-overlapping group Lasso, and if there are two different groups $I_i$ and $I_j$ with a non-empty intersection, it is called overlapping group Lasso. The model finds a coefficient vector $x$ that minimizes the fitting (or loss) function $f(x)$ and that is group sparse: all but a few $x_i$’s are zero.

Let $U_i$ be formed by the columns of the identity matrix $I$ corresponding to the indices in $I_i$, and let $U = [U_1^T; \ldots; U_m^T] \in \mathbb{R}^{(\sum I_i) \times n}$. Then, $x_i = U_i^T x$. Let $h_i(y_i) = \lambda_i \|y_i\|_2$, $y_i \in \mathbb{R}^{n_i}$ for $i \in [m]$, and $h(y) = \sum_{i=1}^{m} h_i(y_i)$ for $y = [y_1; \ldots; y_m] \in \mathbb{R}^{\sum I_i}$. In this way, (4.8) becomes

$$\min_{x} f(x) + h(Ux). \quad (4.9)$$

Non-overlapping case [117]

In this case, we have $I_i \cap I_j = \emptyset$, $\forall i \neq j$, and can apply the FBS scheme (2.9) to (4.9). Specifically, let $\mathcal{T}_1 = \partial(h \circ U)$ and $\mathcal{T}_2 = \nabla f$. The FBS full update is

$$x^{k+1} = J_{\gamma \mathcal{T}_1} \circ (I - \gamma \mathcal{T}_2)(x^k).$$

The corresponding coordinate update is the following

$$\begin{cases} \text{if } i \in [m] \text{ is chosen, then compute} \\
\quad x_i^{k+1} = \arg \min_{x_i} \frac{1}{2}\|x_i - x_i^k + \gamma_i \nabla_i f(x^k)\|_2^2 + \gamma_i h_i(x_i), \quad (4.10)\end{cases}$$

where the step size can be taken to be $\gamma_i = \frac{1}{\|A_i\|}$. When $\nabla f$ is either cheap or easy-to-maintain, the coordinate update in (4.10) is inexpensive.

65
Overlapping case [54]

This case allows $I_i \cap I_j \neq \emptyset$ for some $i \neq j$, causing the evaluation of $\mathcal{J}_{\gamma \tau}$ to be generally difficult. However, we can apply the primal-dual update (2.23) to this problem as

\[
s^{k+1}_i = \text{prox}_{\gamma h^*}(s^k + \gamma Ux^k),
\]

\[
x^{k+1} = x^k - \eta(\nabla f(x^k) + U^T(2s^{k+1} - s^k)),
\]

where $s$ is the dual variable. Note that

\[
h^*(s) = \begin{cases} 
0, & \text{if } ||s_i||_2 \leq \lambda_i, \forall i, \\
+\infty, & \text{otherwise},
\end{cases}
\]

is cheap. Hence, the corresponding coordinate update of (4.11) is

\[
\begin{cases} 
\text{if } s_i \text{ is chosen for some } i \in [m], \text{ then compute } \\
\quad s^{k+1}_i = \text{proj}_{B_{\lambda}}(s^k_i + \gamma x^{k}_i) \\
\text{if } x_i \text{ is chosen for some } i \in [m], \text{ then compute } \\
\quad x^{k+1}_i = x^k_i - \eta \left( U_i^T \nabla f(x^k) + U_i^T \sum_{j, U_j \neq 0} U_j \left(2 \text{proj}_{B_{\lambda_j}}(s^k_j + \gamma x^{k}_j) - s^k_j) \right) \right),
\end{cases}
\]

(4.12)

where $B_{\lambda}$ is the Euclidean ball of radius $\lambda$. When $\nabla f$ is easy-to-maintain, the coordinate update in (4.12) is inexpensive. To the best of our knowledge, the coordinate update method (4.12) is new. The update (4.10) and (4.12) can also be extended to the async-parallel setting under the ARock framework.

4.3.4 Empirical Risk Minimization (ERM)

We consider the following regularized empirical risk minimization problem

\[
\min_{x \in \mathbb{R}^m} \frac{1}{p} \sum_{j=1}^{p} \phi_j(a_j^T x) + f(x) + g(x),
\]

(4.13)

where $a_j$’s are sample vectors, $\phi_j$’s are loss functions, and $f$ and $g$ are regularization functions. We assume that $f$ is differentiable and $g$ is proximable. Examples
of (4.13) include linear SVM, regularized logistic regression, ridge regression, and Lasso. Further information on ERM can be found in [47]. The need for coordinate update algorithms arises in many applications of (4.13) where the number of samples or the dimension of $x$ is large.

We define $A = [a_1^T; a_2^T; \ldots; a_p^T]$ and $h(y) := \frac{1}{p} \sum_{j=1}^p \phi_j(y_j)$. Hence, $h(Ax) = \frac{1}{p} \sum_{j=1}^p \phi_j(a_j^T x)$, and problem (4.13) reduces to form (2.19). We can apply the primal-dual update scheme to solve this problem, for which we introduce the dual variable $s = (s_1, ..., s_p)^T$. We use $p + 1$ coordinates, where the 0th coordinate is $x \in \mathbb{R}^m$ and the $j$th coordinate is $s_j$, $j \in [p]$. The operator $T$ is given in (2.24). At each iteration, a coordinate is updated:

\[
\begin{cases}
\text{if } x \text{ is chosen (the index 0), then compute} \\
\quad x^{k+1} = \text{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^T s^k)), \\
\text{if } s_j \text{ is chosen (an index } j \in [p]), \text{ then compute} \\
\quad \tilde{x}^{k+1} = \text{prox}_{\eta g}(x^k - \eta(\nabla f(x^k) + A^T s^k)), \\
\quad \text{and} \\
\quad s_j^{k+1} = \frac{1}{p} \text{prox}_{p \gamma \phi_j^2}(p s_j^k + p \gamma a_j^T (2 \tilde{x}^{k+1} - x^k)).
\end{cases}
\] (4.14)

We maintain $A^T s \in \mathbb{R}^m$ in the memory. Depending on the structure of $\nabla f$, we can compute it each time or maintain it. When $\text{prox}_g \in \mathcal{F}_1$, we can consider breaking $x$ into coordinates $x_i$’s and also select an index $i$ to update $x_i$ at each time.

### 4.3.5 DRS for Image Processing in the Primal-dual Form

Many convex image processing problems have the general form

\[
\min_x f(x) + g(Ax),
\]

where $A$ is a matrix such as a dictionary, sampling operator, or finite difference operator. We can reduce the problem to the system: $0 \in A(z) + \mathcal{B}(z)$, where
\( z = [x; s], \)

\[
A(z) := \begin{bmatrix}
\partial f(x) \\
\partial g^*(s)
\end{bmatrix}, \quad \text{and} \quad B(z) := \begin{bmatrix}
0 & A^T \\
-A & 0
\end{bmatrix}
\begin{bmatrix}
x \\
s
\end{bmatrix}.
\]

(see Appendix 8.2 for the reduction.) The work [71] gives their resolvents

\[
J_{\gamma A} = \begin{bmatrix}
\text{prox}_{\gamma f} & 0 \\
0 & \text{prox}_{\gamma g^*}
\end{bmatrix},
\]

\[
J_{\gamma B} = (I + \gamma B)^{-1} = \begin{bmatrix}
0 & 0 \\
0 & I
\end{bmatrix} + \begin{bmatrix}
I \\
\gamma A
\end{bmatrix}(I + \gamma^2 A^T A)^{-1} \begin{bmatrix}
I \\
-\gamma A
\end{bmatrix}^T,
\]

where \( J_{\gamma A} \) is often cheap or separable and we can explicitly form \( J_{\gamma B} \) as a matrix or implement it based on a fast transform. With the defined \( J_{\gamma A} \) and \( J_{\gamma B} \), we can apply the RPRS method as \( z^{k+1} = T_{\text{RPRS}} z^k \). The resulting RPRS operator is CF when \( J_{\gamma B} \) is CF. Hence, we can derive a new RPRS coordinate update algorithm. We leave the derivation to the readers. Derivations of coordinate update algorithms for more specific image processing problems are shown in the following subsections.

### 4.3.5.1 Total Variation Image Processing

We consider the following Total Variation (TV) image processing model

\[
\minimize_x \lambda \|x\|_{TV} + \frac{1}{2} \|Ax - b\|^2,
\]

where \( x \in \mathbb{R}^n \) is the vector representation of the unknown image, \( A \) is an \( m \times n \) matrix describing the transformation from the image to the measurements \( b \). Common \( A \) includes sampling matrices in MRI, CT, denoising, deblurring, etc. Let \((\nabla^h_i, \nabla^v_i)\) be the discrete gradient at pixel \( i \) and \( \nabla x = (\nabla^h_1 x, \nabla^v_1 x, \ldots, \nabla^h_n x, \nabla^v_n x)^\top \). Then the TV semi-norm \( \| \cdot \|_{TV} \) in the isotropic and anisotropic fashions are, re-
respectively,
\[
\|x\|_{TV} = \sum_i \sqrt{(\nabla^h_i x)^2 + (\nabla^v_i x)^2},
\]
\[
\|x\|_{TV} = \|\nabla x\|_1 = \sum_i (|\nabla^h_i x| + |\nabla^v_i x|).
\]

For simplicity, we use the anisotropic TV for analysis and in the numerical experiment in § 6.1.2. It is slightly more complicated for the isotropic TV. Introducing the following notation
\[
B = \begin{pmatrix} \nabla \\ A \end{pmatrix}, \quad h(p,q) = \lambda \|p\|_1 + \frac{1}{2} \|q - b\|^2,
\]
we can reformulate (4.15) as
\[
\min_{x} h(Bx) = h(\nabla x, Ax),
\]
which reduces to the form of (2.19) with \( f = g = 0 \). Based on its definition, the convex conjugate of \( h(p,q) \) and its proximal operator are, respectively,
\[
h^*(s,t) = \iota_{\|\cdot\|_{\infty} \leq \lambda}(s) + \frac{1}{2} \|t + b\|^2 - \frac{1}{2} \|b\|^2,
\]
\[
\prox_{\gamma h^*}(s,t) = \prox_{\|\cdot\|_{\infty} \leq \lambda}(s) + \frac{1}{1 + \gamma}(t - \gamma b).
\]
Let \( s, t \) be the dual variables corresponding to \( \nabla x \) and \( Ax \) respectively, then using (4.18) and applying (2.24) give the following full update:
\[
x^{k+1} = x^k - \eta (\nabla^\top s^k + A^\top t^k),
\]
\[
s^{k+1} = \prox_{\|\cdot\|_{\infty} \leq \lambda} \left( s^k + \gamma \nabla (x^k - 2\eta (\nabla^\top s^k + A^\top t^k)) \right),
\]
\[
t^{k+1} = \frac{1}{1 + \gamma} \left( t^k + \gamma A(x^k - 2\eta (\nabla^\top s^k + A^\top t^k)) - \gamma b \right).
\]
To perform the coordinate updates as described in §2.3, we can maintain \( \nabla^\top s^k \) and \( A^\top t^k \). Whenever a coordinate of \( (s,t) \) is updated, the corresponding \( \nabla^\top s^k \) (or \( A^\top t^k \)) should also be updated. Specifically, we have the following coordinate
update algorithm
\[
\begin{align*}
\text{if } x_i & \text{ is chosen for some } i \in [n], \text{ then compute } \\
x_i^{k+1} & = x_i^k - \eta (\nabla^\top s^k + A^\top t^k)_i; \\
\text{if } s_i & \text{ is chosen for some } i \in [2n], \text{ then compute } \\
s_i^{k+1} & = \text{proj}_{\|\cdot\| \leq \lambda} (s_i^k + \gamma \nabla_i(x^k - 2\eta (\nabla^\top s^k + A^\top t^k))) \\
& \quad \text{and update } \nabla^\top s^k \text{ to } \nabla^\top s^{k+1}; \\
\text{if } t_i & \text{ is chosen for some } i \in [m], \text{ then compute } \\
t_i^{k+1} & = \frac{1}{1+\gamma} (t_i^k + \gamma A_{i,:}(x^k - 2\eta (\nabla^\top s^k + A^\top t^k)) - \gamma b_i) \\
& \quad \text{and update } A^\top t^k \text{ to } A^\top t^{k+1}.
\end{align*}
\] (4.20)

4.3.6 Feasibility Problem

Suppose that $C_1, \ldots, C_m$ are closed convex subsets of $\mathbb{H}$ with a nonempty intersection. The problem is to find a point in the intersection. Let $I_{C_i}$ be the indicator function of the set $C_i$, that is, $I_{C_i}(x) = 0$ if $x \in C_i$ and $\infty$ otherwise. The feasibility problem can be formulated as the following

\[
\minimize_{x = (x_1, \ldots, x_m) \in \mathbb{H}^m} \sum_{i=1}^{m} I_{C_i}(x_i) + I_{\{x_1 = \cdots = x_m\}}(x).
\]

Let $z^k = (z^k_1, \ldots, z^k_m) \in \mathbb{H}^m$, $\hat{z}^k = (\hat{z}^k_1, \ldots, \hat{z}^k_m) \in \mathbb{H}^m$, and $\hat{\bar{z}}^k \in \mathbb{H}$. We can implement (4.7) as follows:

\[
\begin{align*}
\hat{z}^k & = \frac{1}{m} \sum_{i=1}^{m} z^k_i, \\
\hat{y}_{ik} & = \text{Proj}_{C_{ik}} (2\hat{z}^k - \hat{z}_{ik}^k), \\
\hat{z}_{ik}^{k+1} & = z_{ik}^k + 2\eta_k (\hat{y}_{ik} - \hat{z}^k).
\end{align*}
\] (4.21)

The update (4.21) can be implemented as follows. Let global memory hold $z_1, \ldots, z_m$, as well as $\bar{z} = \frac{1}{m} \sum_{i=1}^{m} z_i$. At the $k$th update, an agent independently generates a random number $i_k \in \{1, \ldots, m\}$, then reads $z_{ik}$ as $\hat{z}_{ik}^k$ and $\bar{z}$ as $\hat{\bar{z}}^k$, and finally computes $\hat{y}_{ik}$ and updates $z_{ik}$ in global memory according to (4.21). Since $\bar{z}$ is maintained in global memory, the agent updates $\bar{z}$ according to $\hat{z}_{ik}^{k+1} = z_{ik}^k + \frac{1}{m} (z_{ik}^{k+1} - z_{ik}^k)$. This implementation saves each agent from
computing (4.21a) or reading all $z_1, \ldots, z_m$. Each agent only reads $z_{i_k}$ and $\bar{z}$, executes (4.21b), and updates $z_{i_k}$ (4.21c) and $\bar{z}$.

### 4.3.7 3D Mesh Denoising

Following an example in [86], we consider a 3D mesh described by their nodes $ar{x}_i = (\bar{x}_i^X, \bar{x}_i^Y, \bar{x}_i^Z), i \in [n]$, and the adjacency matrix $A \in \mathbb{R}^{n \times n}$, where $A_{ij} = 1$ if nodes $i$ and $j$ are adjacent, otherwise $A_{ij} = 0$. We let $\mathcal{V}_i$ be the set of neighbours of node $i$. Noisy mesh nodes $z_i, i \in [n]$, are observed. We try to recover the original mesh nodes by solving the following optimization problem [86]:

$$
\min_x \sum_{i=1}^n f_i(x_i) + \sum_{i=1}^n g_i(x_i) + \sum_{i \in \mathcal{V}_i} \sum_{j \in \mathcal{V}_i} h_{i,j}(x_i - x_j),
$$

(4.22)

where $f_i$’s are differentiable data fidelity terms, $g_i$’s are box constraints, and $\sum_{i \in \mathcal{V}_i} h_{i,j}(x_i - x_j)$ is the total variation on the mesh.

We introduce a dual variable $s$ with coordinates $s_{i,j}$, for all ordered pairs of adjacent nodes $(i, j)$, and, based on the overlapping-block coordinate updating scheme (2.28), perform coordinate update:

- select $i$ from $[n]$, then compute
  
  $$
  s_{i,j}^{k+1} = \text{prox}_{\gamma h_{i,j}^k} (s_{i,j}^k + \gamma x_i^k - \gamma x_j^k), \forall j \in \mathcal{V}_i,
  $$
  $$
  s_{j,i}^{k+1} = \text{prox}_{\gamma h_{j,i}^k} (s_{j,i}^k + \gamma x_j^k - \gamma x_i^k), \forall j \in \mathcal{V}_i,
  $$

and update

$$
\begin{align*}
  x_i^{k+1} &= \text{prox}_{\eta g_i} (x_i^k - \eta (\nabla f_i(x_i^k) + \sum_{j \in \mathcal{V}_i} (2s_{i,j}^{k+1} - 2s_{j,i}^{k+1} - s_{i,j}^k + s_{j,i}^k))), \\
  s_{i,j}^{k+1} &= s_{i,j}^k + \frac{1}{2}(s_{i,j}^{k+1} - s_{i,j}^k), \forall j \in \mathcal{V}_i, \\
  s_{j,i}^{k+1} &= s_{j,i}^k + \frac{1}{2}(s_{j,i}^{k+1} - s_{j,i}^k), \forall j \in \mathcal{V}_i.
\end{align*}
$$
4.4 Constrained Optimization

4.4.1 Support Vector Machine

Given the training data \( \{(a_i, \beta_i)\}_{i=1}^{m} \) with \( \beta_i \in \{+1, -1\} \), \( \forall i \), the kernel support vector machine \(^93\) is

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} \|x\|_2^2 + C \sum_{i=1}^{m} \xi_i, \\
\text{s.t.} & \quad \beta_i (x^T \phi(a_i) - y) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \in [m],
\end{align*}
\]

(4.23)

where \( \phi \) is a vector-to-vector map, mapping each data \( a_i \) to a point in a (possibly) higher-dimensional space. If \( \phi(a) = a \), then (4.23) reduces to the linear support vector machine. The model (4.23) can be interpreted as finding a hyperplane \( \{w : x^T w - y = 0\} \) to separate two sets of points \( \{\phi(a_i) : \beta_i = 1\} \) and \( \{\phi(a_i) : \beta_i = -1\} \).

The dual problem of (4.23) is

\[
\begin{align*}
\text{minimize} & \quad \frac{1}{2} s^T Q s - e^T s, \quad \text{subject to } 0 \leq s_i \leq C, \quad \forall i, \quad \sum_i \beta_i s_i = 0, \\
\end{align*}
\]

(4.24)

where \( Q_{ij} = \beta_i \beta_j k(a_i, a_j) \), \( k(\cdot, \cdot) \) is a so-called kernel function, and \( e = (1, ..., 1)^T \). If \( \phi(a) = a \), then \( k(a_i, a_j) = a_i^T a_j \).

Unbiased case

If \( y = 0 \) is enforced in (4.23), then the solution hyperplane \( \{w : x^T w = 0\} \) passes through the origin and is called unbiased. Consequently, the dual problem (4.24) will no longer have the linear constraint \( \sum_i \beta_i s_i = 0 \), leaving it with the coordinate-wise separable box constraints \( 0 \leq s_i \leq C \). To solve (4.24), we can apply the FBS operator \( T \) defined by (2.9). Let \( d(s) := \frac{1}{2} s^T Q s - e^T s \), \( A = \text{prox}_{[0,C]} \), and \( C = \nabla d \). The coordinate update based on FBS is

\[
s_{i}^{k+1} = \text{proj}_{[0,C]}(s_{i}^{k} - \gamma_i \nabla_{i} d(s_{i}^{k})),
\]

where we can take \( \gamma_i = \frac{1}{Q_{ii}} \).
Biased (general) case

In this case, the mode (4.23) has \( y \in \mathbb{R} \), so the hyperplane \( \{ w : \mathbf{x}^\top w - y = 0 \} \) may not pass the origin and is called biased. Then, the dual problem (4.24) retains the linear constraint \( \sum_i \beta_i s_i = 0 \). In this case, we apply the primal-dual splitting scheme (2.23) or the three-operator splitting scheme (2.8).

The coordinate update based on the full primal-dual splitting scheme (2.23) is:

\[
\begin{align*}
t^{k+1} &= t^k + \gamma \sum_{i=1}^m \beta_i s^k_i, \quad (4.25a) \\
\hat{s}^{k+1}_i &= \text{proj}_{[0,C]} \left( s^k_i - \eta \left( \nabla_i d(s^k) + \beta_i (2t^{k+1} - t^k) \right) \right), \quad (4.25b)
\end{align*}
\]

where \( t, s \) are the primal and dual variables, respectively. Note that we can let \( w := \sum_{i=1}^m \beta_i s_i \) and maintain it. With variable \( w \) and substituting (4.25a) into (4.25b), we can equivalently write (4.25) into

\[
\begin{align*}
\begin{cases}
\text{if } t \text{ is chosen (the index 0), then compute} & \quad t^{k+1} = t^k + \gamma w^k, \\
\text{if } s_i \text{ is chosen (an index } i \in [m] \text{), then compute} & \quad \hat{s}^{k+1}_i = \text{proj}_{[0,C]} \left( s^k_i - \eta \left( \nabla_i d(s^k) + \beta_i (2\gamma w^k + t^k) \right) \right), \\
& \quad w^{k+1} = w^k + \beta_i (s^{k+1}_i - s^k_i). 
\end{cases} \quad (4.26)
\end{align*}
\]

We can also apply the three-operator splitting (2.8) as follows. Let \( D_1 := [0,C]^m \) and \( D_2 := \{ s : \sum_{i=1}^m \beta_i s_i = 0 \} \). Let \( A = \text{proj}_{D_2}, B = \text{proj}_{D_1}, \) and \( C(x) = Qx - e \), The full update corresponding to \( T = (I - \eta_k)I + \eta_k T_3 \) is

\[
\begin{align*}
\begin{cases}
\text{if } s \text{ is just an intermediate variable. Let } \tilde{\beta} := \frac{\beta}{\|\beta\|_2} \text{ and } w := \tilde{\beta}^\top u. \text{ Then } \text{proj}_{D_2}(u) &= (I - \tilde{\beta}\tilde{\beta}^\top)u. \text{ Hence, } s^{k+1} = u^k - w^k \tilde{\beta}. \text{ Plugging it into (4.27b) yields} \\
\text{proj}_{D_2}(u^k) &= \text{proj}_{D_2}(2s^{k+1} - u^k - \gamma (Qs^{k+1} - e) - s^{k+1}), \\
u^{k+1} &= u^k + \eta_k \left( \text{proj}_{D_1}(2s^{k+1} - u^k - \gamma (Qs^{k+1} - e) - s^{k+1}) \right), \quad (4.27a)
\end{cases} \quad (4.27b)
\end{align*}
\]
the following coordinate update scheme:

\[
\begin{cases}
    \text{if } i \in [m] \text{ is chosen, then compute} \\
    s_i^{k+1} = u_i^k - w^k \beta_i, \\
    u_i^{k+1} = u_i^k + \eta_k \left( \text{proj}_{[0,C]} \left( 2s_i^{k+1} - u_i^k - \gamma(q_i^\top u_i^k - w^k (q_i^\top \tilde{\beta}) - 1) \right) - s_i^{k+1} \right), \\
    w^{k+1} = w^k + \tilde{\beta}_i (u_i^{k+1} - u_i^k),
\end{cases}
\]

where \( w^k \) is the maintained variable and \( s^k \) is the intermediate variable.

### 4.4.2 Second-Order Cone Programming (SOCP)

SOCP extends LP by incorporating second-order cones. A second-order cone in \( \mathbb{R}^n \) is

\[
Q = \{(x_1, x_2, \ldots, x_n) \in \mathbb{R}^n : \| (x_2, \ldots, x_n) \|_2 \leq x_1 \}.
\]

Given a point \( v \in \mathbb{R}^n \), let \( \rho_1^v := \| (v_2, \ldots, v_n) \|_2 \) and \( \rho_2^v := \frac{1}{2} (v_1 + \rho_1^v) \). Then, the projection of \( v \) to \( Q \) returns 0 if \( v_1 < -\rho_1^v \), returns \( v \) if \( v_1 \geq \rho_1^v \), and returns \((\rho_2^v, \frac{\rho_2^v}{\rho_1^v}, (v_2, \ldots, v_n))\) otherwise. Therefore, if we define the scalar couple:

\[
(\xi_1^v, \xi_2^v) = \begin{cases}
(0, 0), & v_1 < -\rho_1^v, \\
(1, 1), & v_1 \geq \rho_1^v, \\
(\rho_2^v, \frac{\rho_2^v}{\rho_1^v}), & \text{otherwise},
\end{cases}
\]

then we have \( u = \text{proj}_Q(v) = (\xi_1^v v_1, \xi_2^v \cdot (v_2, \ldots, v_n)) \). Based on this, we have

**Proposition 4**

1. Let \( v \in \mathbb{R}^n \) and \( v^+ := v + \nu e_i \) for any \( \nu \in \mathbb{R} \). Then, given \( \rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v \) defined above, it takes \( O(1) \) operations to obtain \( \rho_1^{v^+}, \rho_2^{v^+}, \xi_1^{v^+}, \xi_2^{v^+} \).

2. Let \( v \in \mathbb{R}^n \) and \( A = [a_1 \ A_2] \in \mathbb{R}^{m \times n} \), where \( a_1 \in \mathbb{R}^m, A_2 \in \mathbb{R}^{m \times (n-1)} \). Given \( \rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v \), we have

\[
A(2 \cdot \text{proj}_Q(v) - v) = ((2\xi_1^v - 1) v_1) \cdot a_1 + (2\xi_2^v - 1) \cdot A_2 (v_2, \ldots, v_n)^\top.
\]

By the proposition, if \( \mathcal{T}_1 \) is an affine operator, then in the composition \( \mathcal{T}_1 \circ \text{proj}_Q \), the computation of \( \text{proj}_Q \) is cheap as long as we maintain \( \rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v \).
Given \( x, c \in \mathbb{R}^n, b \in \mathbb{R}^m, \) and \( A \in \mathbb{R}^{m \times n}, \) the standard form of SOCP is

\[
\begin{align*}
\text{minimize} & \quad c^\top x, \quad \text{subject to} \quad Ax = b, \\
& \quad x \in X = Q_1 \times \cdots \times Q_{\bar{n}},
\end{align*}
\]

(4.28a)

where each \( Q_i \) is a second-order cone, and \( \bar{n} \neq n \) in general. The problem (4.28) is equivalent to

\[
\begin{align*}
\text{minimize} & \quad (c^\top x + \iota_{A=b}(x)) + \iota_X(x), \\
\text{subject to} & \quad Ay = b.
\end{align*}
\]

Assume that the matrix \( A \) has full row-rank (otherwise, \( Ax = b \) has either redundant rows or no solution). Then, in (2.11), we have \( \mathcal{R}_{\gamma A}(x) = Bx + d, \) where \( B := I - 2A^\top (AA^\top)^{-1}A \) and \( d := 2A^\top (AA^\top)^{-1}(b + \gamma Ac) - 2\gamma c. \)

It is easy to apply coordinate updates to \( z^{k+1} = \mathcal{T}_{\text{DRS}}(z^k) \) following Proposition 4. Specifically, by maintaining the scalars \( \rho_1^v, \rho_2^v, \xi_1^v, \xi_2^v \) for each \( v = x_i \in Q_i \) during coordinate updates, the computation of the projection can be completely avoided.

We pre-compute \( (AA^\top)^{-1} \) and cache the matrix \( B \) and vector \( d. \) Then, \( \mathcal{T}_{\text{DRS}} \) is CF, and we have the following coordinate update method

\[
\begin{cases}
\text{select } i \in [\bar{n}], \text{ then compute} \\
y_i^{k+1} = B_i x^k + d_i \\
x_i^{k+1} = \text{proj}_{Q_i}(y_i^{k+1}) + \frac{1}{2}(x_i^k - y_i^{k+1}),
\end{cases}
\]

(4.29)

where \( B_i \in \mathbb{R}^{n_i \times n} \) is the \( i \)th row block submatrix of \( B, \) and \( y_i^{k+1} \) is the intermediate variable.

It is trivial to extend this method for SOCPs with a quadratic objective:

\[
\begin{align*}
\text{minimize} & \quad c^\top x + \frac{1}{2} x^\top Cx, \quad \text{subject to} \quad Ax = b, \quad x \in X = Q_1 \times \cdots \times Q_{\bar{n}},
\end{align*}
\]

75
because \( J_2 \) is still linear. Clearly, this method applies to linear programs as they are special SOCPs.

Note that many LPs and SOCPs have sparse matrices \( A \), which deserve further investigation. In particular, we may prefer not to form \((AA^\top)^{-1}\) and use the results in Subsection 2.3.2 instead.

### 4.4.3 Portfolio Optimization

Assume that we have one unit of capital and \( m \) assets to invest on. The \( i \)th asset has an expected return rate \( \xi_i \geq 0 \). Our goal is to find a portfolio with the minimal risk such that the expected return is no less than \( c \). This problem can be formulated as

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} x^\top Q x, \\
\text{subject to} \quad & x \geq 0, \sum_{i=1}^{m} x_i \leq 1, \sum_{i=1}^{m} \xi_i x_i \geq c,
\end{align*}
\]

where the objective function is a measure of risk, and the last constraint imposes that the expected return is at least \( c \). Let \( a_1 = e/\sqrt{m} \), \( b_1 = 1/\sqrt{m} \), \( a_2 = \xi/\|\xi\|_2 \), and \( b_2 = c/\|\xi\|_2 \), where \( e = (1, \ldots, 1)^\top \), \( \xi = (\xi_1, \ldots, \xi_m)^\top \). The above problem is rewritten as

\[
\begin{align*}
\text{minimize} \quad & \frac{1}{2} x^\top Q x, \text{ subject to } x \geq 0, a_1^\top x \leq b_1, a_2^\top x \geq b_2. 
\end{align*}
\]

We apply the three-operator splitting scheme (2.8) to (4.30). Let \( f(x) = \frac{1}{2} x^\top Q x \), \( D_1 = \{ x : x \geq 0 \} \), \( D_2 = \{ x : a_1^\top x \leq b_1, a_2^\top x \geq b_2 \} \), \( D_{21} = \{ x : a_1^\top x = b_1 \} \), and \( D_{22} = \{ x : a_2^\top x = b_2 \} \). Based on (2.8), the full update is

\[
\begin{align*}
y^{k+1} &= \text{proj}_{D_{21}}(x^k), \quad & (4.31a) \\
x^{k+1} &= x^k + \eta_k \left( \text{proj}_{D_2}(2y^{k+1} - x^k - \gamma \nabla f(y^{k+1}) - y^{k+1}) \right), \quad & (4.31b)
\end{align*}
\]

where \( y \) is an intermediate variable. As the projection to \( D_1 \) is simple, we discuss how to evaluate the projection to \( D_2 \). Assume that \( a_1 \) and \( a_2 \) are neither
perpendicular nor co-linear, i.e., \( a_1^\top a_2 \neq 0 \) and \( a_1 \neq \lambda a_2 \) for any scalar \( \lambda \). In addition, assume \( a_1^\top a_2 > 0 \) for simplicity. Let \( a_3 = a_2 - \frac{1}{a_1^\top a_2} a_1 \), \( b_3 = b_2 - \frac{1}{a_1^\top a_2} b_1 \), \( a_4 = a_1 - \frac{1}{a_1^\top a_2} a_2 \), and \( b_4 = b_1 - \frac{1}{a_1^\top a_2} b_2 \). Then we can partition the whole space into four areas by the four hyperplanes \( a_i^\top x = b_i \), \( i = 1, \ldots, 4 \). Let \( P_i = \{ x : a_i^\top x \leq b_i, a_{i+1}^\top x \geq b_{i+1} \} \), \( i = 1, 2, 3 \) and \( P_4 = \{ x : a_4^\top x \leq b_1, a_1^\top x \geq b_1 \} \). Then

\[
\text{proj}_{D_1}(x) = \begin{cases} 
  x, & \text{if } x \in P_1, \\
  \text{proj}_{D_2}(x), & \text{if } x \in P_2, \\
  \text{proj}_{D_2 \cap D_2}(x), & \text{if } x \in P_3, \\
  \text{proj}_{D_2}(x), & \text{if } x \in P_4.
\end{cases}
\]

Let \( w_i = a_i^\top x - b_i, i = 1, 2 \), and maintain \( w_1, w_2 \). Let \( \tilde{a}_2 = \frac{a_2-a_1(a_1^\top a_2)}{1-(a_1^\top a_2)^2}, \tilde{a}_1 = \frac{a_1-a_2(a_1^\top a_2)}{1-(a_1^\top a_2)^2} \). Then

\[
\begin{align*}
\text{proj}_{D_2}(x) &= x - w_1 a_1, \\
\text{proj}_{D_2}(x) &= x - w_2 a_2, \\
\text{proj}_{D_2 \cap D_2}(x) &= x - w_1 \tilde{a}_1 - w_2 \tilde{a}_2,
\end{align*}
\]

Hence, the coordinate update of (4.31) is

\[
\begin{align*}
x^k &\in P_1 : \quad x_i^{k+1} = (1 - \eta_k)x_i^k + \eta_k \max(0, x_i^k - \gamma q_i^\top x^k), \\
x^k &\in P_2 : \quad x_i^{k+1} = (1 - \eta_k)x_i^k + \eta_k w_k^a(a_2)_i + \eta_k \max(0, \\
&\quad x_i^k - \gamma q_i^\top x^k - w_k^b(2(a_2)_i - \gamma q_i^\top a_2)), \\
x^k &\in P_3 : \quad x_i^{k+1} = (1 - \eta_k)x_i^k + \eta_k (w_k^a(\tilde{a}_1)_i + w_k^b(\tilde{a}_2)_i) + \eta_k \max(0, \\
&\quad x_i^k - \gamma q_i^\top x^k - w_k^a(2(\tilde{a}_1)_i - \gamma q_i^\top a_1) - w_k^b(2(\tilde{a}_2)_i - \gamma q_i^\top a_2)), \\
x^k &\in P_4 : \quad x_i^{k+1} = (1 - \eta_k)x_i^k + \eta_k w_k^1(a_1)_i + \eta_k \max(0, \\
&\quad x_i^k - \gamma q_i^\top x^k - w_k^1(2(a_1)_i - \gamma q_i^\top a_1)),
\end{align*}
\]

where \( q_i \) is the \( i \)th column of \( Q \). At each iteration, we select \( i \in [m] \), and perform an update to \( x_i \) according to (4.32) based on where \( x^k \) is. We then renew \( w_j^{k+1} = \ldots \)
\[ w_j^k + a_{ij}(x_i^{k+1} - x_i^k), \quad j = 1, 2. \] Note that checking \( x^k \) in some \( P_j \) requires only \( O(1) \) operations by using \( w_1 \) and \( w_2 \), so the coordinate update in (4.32) is inexpensive.

### 4.4.4 Async-parallel ADMM for Linear Constrained Optimization

This is another application of (4.7). Consider

\[
\begin{align*}
\text{minimize} & \quad f(x) + g(y) \quad \text{subject to} \quad Ax + By = b, \quad (4.33) \\
\text{where} & \quad \mathbb{H}_1 \text{ and } \mathbb{H}_2 \text{ are Hilbert spaces, } A \text{ and } B \text{ are bounded linear operators. We apply the update (4.7) to the Lagrange dual of (4.33) (see [42] for the derivation):}
\end{align*}
\]

\[
\begin{align*}
\text{minimize} & \quad d_f(w) + d_g(w), \quad (4.34) \\
\text{where} & \quad d_f(w) := f^*(A^*w), \quad d_g(w) := g^*(B^*w) - \langle w, b \rangle, \text{ and } f^* \text{ and } g^* \text{ denote the convex conjugates of } f \text{ and } g, \text{ respectively. The proximal maps induced by } d_f \text{ and } d_g \text{ can be computed via solving subproblems that involve only the original terms in (4.33): } z^+ = \text{prox}_{\gamma d_f}(z) \text{ can be computed by (see Appendix 8.1 for the derivation)}
\end{align*}
\]

\[
\begin{align*}
\begin{cases}
  x^+ \in \arg \min_x f(x) - \langle z, Ax \rangle + \frac{\gamma}{2} \|Ax\|^2, \\
  z^+ = z - \gamma Ax^+,
\end{cases} \quad (4.35)
\end{align*}
\]

and \( z^+ = \text{prox}_{\gamma d_g}(z) \) by

\[
\begin{align*}
\begin{cases}
  y^+ \in \arg \min_y g(y) - \langle z, By - b \rangle + \frac{\gamma}{2} \|By - b\|^2, \\
  z^+ = z - \gamma (By^+ - b).
\end{cases} \quad (4.36)
\end{align*}
\]

Plugging (4.35) and (4.36) into (4.7) yields the following naive implementation

\[
\begin{align*}
\hat{y}^k & \in \arg \min_y g(y) - \langle \hat{z}^k, By - b \rangle + \frac{\gamma}{2} \|By - b\|^2, \quad (4.37a) \\
\hat{w}_g^k & = \hat{z}^k - \gamma (B\hat{y}^k - b), \quad (4.37b) \\
\hat{x}^k & \in \arg \min_x f(x) - \langle 2\hat{w}_g^k - \hat{z}^k, Ax \rangle + \frac{\gamma}{2} \|Ax\|^2, \quad (4.37c) \\
\hat{w}_f^k & = 2\hat{w}_g^k - \hat{z}^k - \gamma A\hat{x}^k, \quad (4.37d) \\
z_{i_k}^{k+1} & = z_{i_k}^k + \eta_k (\hat{w}_{f,i_k}^k - \hat{w}_{g,i_k}^k). \quad (4.37e)
\end{align*}
\]
Note that $2\eta_k$ in (4.7c) becomes $\eta_k$ in (4.37e) because ADMM is equivalent to the Douglas-Rachford operator, which is the average of the Peaceman-Rachford operator and the identity operator [61]. Under favorable structures, (4.37) can be implemented efficiently. For instance, when $A$ and $B$ are block diagonal matrices and $f, g$ are corresponding block separable functions, steps (4.37a)-(4.37d) reduce to independent computation for each $i$. Since only $\hat{w}_{f,ik}^k$ and $\hat{w}_{g,ik}^k$ are needed to update the main variable $z^k$, we only need to compute (4.37a)-(4.37d) for the $i_k$th block. This is exploited in distributed and decentralized ADMM in the next section.

4.5 Distributed and Decentralized Optimization

In this section, we discuss the application of ARock to distributed and decentralized optimization problems under three different settings.

4.5.1 Decentralized Consensus Optimization

Consider that $m$ agents in a connected network solve the consensus problem of minimizing $\sum_{i=1}^m f_i(x)$, where $x \in \mathbb{R}^d$ is the shared variable and the convex differentiable function $f_i$ is held privately by agent $i$. We assume that $\nabla f_i$ is $L_i$-Lipschitz continuous for all $i$. A decentralized gradient descent algorithm [68] can be developed based on the equivalent formulation

$$\min_{x_1, \ldots, x_m \in \mathbb{R}^d} f(x) := \sum_{i=1}^m f_i(x_i), \quad \text{subject to } Wx = x,$$

where $x = (x_1, \ldots, x_m)^T \in \mathbb{R}^{m \times d}$ and $W \in \mathbb{R}^{m \times m}$ is the so-called mixing matrix satisfying: $Wx = x$ if and only if $x_1 = \cdots = x_m$. For $i \neq j$, if $w_{i,j} \neq 0$, then agent $i$ can communicate with agent $j$; otherwise they cannot. We assume that $W$ is symmetric and doubly stochastic. Then, the decentralized consensus algorithm [68] can be expressed as $x^{k+1} = Wx^k - \gamma \nabla f(x^k) = x^k - \gamma(\nabla f(x^k) +$.
\( \frac{1}{\gamma}(I-W)x^k \), where \( \nabla f(x) \in \mathbb{R}^{m \times d} \) is a matrix with its \( i \)th row equal to \( (\nabla f_i(x_i))^T \); see [116]. The computation of \( Wx^k \) involves communication between agents, and \( \nabla f_i(x_i) \) is independently computed by each agent \( i \). The iteration is equivalent to the gradient descent iteration applied to \( \min_x \sum_{i=1}^m f_i(x_i) + \frac{1}{2\gamma}x^T(I-W)x \). To apply our algorithm, we let \( S := \frac{2}{L} \nabla F = \frac{2}{L}(\nabla f + \frac{1}{\gamma}(I-W)) \) with \( L = \max_i L_i + (1 - \lambda_{\text{min}}(W))/\gamma \), where \( \lambda_{\text{min}}(A) \) is the smallest eigenvalue of \( W \). Computing \( S_i \tilde{x}^k \) reduces to computing \( \nabla f_i(\tilde{x}^k_i) \) and the \( i \)th entry of \( W \tilde{x}^k \) or \( \sum_j w_{i,j} \tilde{x}^k_j \), which involves only \( \tilde{x}^k_i \) and \( \tilde{x}^k_j \) from the neighbors of agent \( i \). Note that since each agent \( i \) can store its own \( x_i \) locally, we have \( \tilde{x}^k_i \equiv x^k_i \).

If the agents are \( p \) independent Poisson processes and that each agent \( i \) has activation rate \( \lambda_i \), then the probability that agent \( i \) activates before other agents is equal to \( \frac{\lambda_i}{\sum_{i=1}^p \lambda_i} \) [58] and therefore our random sample scheme holds and ARock applies naturally. The algorithm is summarized as follows:

**Algorithm 4: ARock for decentralized optimization (4.38)**

**Input**: Each agent \( i \) sets \( x^0_i \in \mathbb{R}^d \), \( K > 0 \).

**while** \( k < K \) **do**

- when an agent \( i \) is activated,
  
  \[
  x^k_{i+1} = x^k_i - \frac{\eta_k}{L} (\nabla f_i(x^k_i) + \frac{1}{\gamma}(x^k_i - \sum_j w_{i,j} \tilde{x}^k_j));
  \]

- increase the global counter \( k \leftarrow k + 1 \);

**4.5.2 Async-parallel ADMM for consensus optimization**

Consider the consensus optimization problem:

\[
\min_{x_i, y \in \mathcal{H}} \sum_{i=1}^m f_i(x_i) \quad \text{subject to } x_i - y = 0, \quad \forall i \in [m], \quad (4.39)
\]
where $f_i(x_i)$ are proper close convex functions. Rewrite (4.39) to the ADMM form:

$$\min_{x_i, y} \sum_{i=1}^m f_i(x_i) + g(y)$$

subject to

$$\begin{bmatrix}
I_H & 0 & \cdots & 0 \\
0 & I_H & \cdots & 0 \\
0 & 0 & \cdots & I_H \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_m \\
\end{bmatrix}
- \begin{bmatrix}
I_H \\
I_H \\
I_H \\
\end{bmatrix}
y = 0,$$

where $g = 0$. Now apply the async-parallel ADMM (4.37) to (4.40) with dual variables $z_1, \ldots, z_m \in \mathbb{H}$. In particular, the update (4.37a), (4.37b), (4.37c), (4.37d) reduce to

$$\begin{align*}
\hat{y}^k &= \arg\min_y \left\{ \sum_{i=1}^m \langle \hat{z}_i^k, y \rangle + \frac{2m}{2} \|y\|^2 \right\} = -\frac{1}{\gamma m} \sum_{i=1}^m \hat{z}_i^k \\
(\hat{\omega}_d^k)_i &= \hat{z}_i^k + \gamma \hat{y}^k \\
\hat{x}_i^k &= \arg\min_{x_i} \left\{ f_i(x_i) - \langle 2(\hat{\omega}_d^k)_i - \hat{z}_i^k, x_i \rangle + \frac{\gamma}{2} \|x_i\|^2 \right\}, \\
(\hat{\omega}_d^k)_i &= 2(\hat{\omega}_d^k)_i - \hat{z}_i^k - \gamma \hat{x}_i^k
\end{align*}$$

(4.41)

Therefore, we obtain the following async-parallel ADMM algorithm for the problem (4.39). This algorithm applies to all the distributed applications in [19].

**Algorithm 5: ARock for consensus optimization**

**Input**: set shared variables $y^0, z_i^0, \forall i$, and $K > 0$.  

**while** $k < K$ **every agent asynchronously and continuously** **do**

choose $i_k$ from $[m]$ with equal probability; 

evaluate $(\hat{\omega}_d^k)_{i_k}, \hat{x}_i^k$, and $(\hat{\omega}_d^k)_{i_k}$ following (4.41);  

update $z_{i_k}^{k+1} = z_{i_k}^k + \eta_k ((\hat{\omega}_d^k)_{i_k} - (\hat{\omega}_d^k)_{i_k});$

update $y^{k+1} = y^k + \frac{1}{\gamma m} (z_{i_k}^k - z_{i_k}^{k+1});$

update the global counter $k \leftarrow k + 1;$
4.5.3 Async-parallel ADMM for decentralized optimization

Let \( V = \{1, ..., m\} \) be a set of agents and \( E = \{(i, j) \mid \text{if agent } i \text{ connects to agent } j, i < j\} \) be the set of undirected links between the agents. Consider the following decentralized consensus optimization problem on the graph \( G = (V, E) \):

\[
\min_{x_1, ..., x_m \in \mathbb{R}^d} f(x_1, \ldots, x_m) := \sum_{i=1}^{m} f_i(x_i), \quad \text{subject to } x_i = x_j, \; \forall (i, j) \in E,
\]

(4.42)

where \( x_1, ..., x_m \in \mathbb{R}^d \) are the local variables and each agent can only communicate with its neighbors in \( G \). By introducing the auxiliary variable \( y_{ij} \) associated with each edge \((i, j) \in E\), the problem (4.42) can be reformulated as:

\[
\min_{x_i, y_{ij}} \sum_{i=1}^{m} f_i(x_i), \quad \text{subject to } x_i = y_{ij}, \; x_j = y_{ij}, \; \forall (i, j) \in E.
\]

(4.43)

Define \( x = (x_1, ..., x_m)^T \) and \( y = (y_{ij})_{(i, j) \in E} \in \mathbb{R}^{|E|d} \) to rewrite (4.43) as

\[
\min_{x, y} \sum_{i=1}^{m} f_i(x_i), \quad \text{subject to } Ax + By = 0,
\]

(4.44)

for proper matrices \( A \) and \( B \). Applying the async-parallel ADMM (4.37) to (4.44) gives rise to the following simplified update: Let \( E(i) \) be the set of edges connected with agent \( i \) and \( |E(i)| \) be its cardinality. Let \( L(i) = \{j \mid (j, i) \in E(i), j < i\} \) and \( R(i) = \{j \mid (i, j) \in E(i), j > i\} \). To every pair of constraints \( x_i = y_{ij} \) and \( x_j = y_{ij} \), \((i, j) \in E\), we associate the dual variables \( z_{ij,i} \) and \( z_{ij,j} \), respectively. Whenever some agent \( i \) is activated, it calculates

\[
\hat{x}_i^k = \arg \min_{x_i} f_i(x_i) + (\sum_{l \in L(i)} \hat{z}_{li,i}^k + \sum_{r \in R(i)} \hat{z}_{ir,i}^k) x_i + \frac{\gamma}{2} |E(i)| \cdot \|x_i\|^2,
\]

(4.45a)

\[
\hat{z}_{li,i}^{k+1} = \hat{z}_{li,i}^k - \eta_k((\hat{z}_{li,i}^k + \hat{z}_{li,l})/2 + \gamma \hat{x}_i^k), \; \forall l \in L(i),
\]

(4.45b)

\[
\hat{z}_{ir,i}^{k+1} = \hat{z}_{ir,i}^k - \eta_k((\hat{z}_{ir,i}^k + \hat{z}_{ir,r})/2 + \gamma \hat{x}_i^k), \; \forall r \in R(i).
\]

(4.45c)
We present the algorithm based on (4.45) for problem (4.42) in Algorithm 6.

**Algorithm 6: ARock for the decentralized problem (4.43)**

**Input** : Each agent $i$ sets the dual variables $z_{e,i}^0 = 0$ for $e \in E(i)$, $K > 0$.

While $k < K$, any activated agent $i$ do

- (previously received $\hat{z}_{l,i}^k$ from neighbors $l \in L(i)$ and $\hat{z}_{r,i}^k$ from $r \in R(i)$);
- update $\hat{x}_{i}^k$ according to (4.45a);
- update $z_{l,i}^{k+1}$ and $z_{r,i}^{k+1}$ according to (4.45b) and (4.45c), respectively;
- send $z_{l,i}^{k+1}$ to neighbors $l \in L(i)$ and $z_{r,i}^{k+1}$ to neighbors $r \in R(i)$;

Algorithm 6 activates one agent at each iteration and updates all the dual variables associated with the agent. In this case, only one-sided communication is needed, for sending the updated dual variables in the last step. We allow this communication to be delayed in the sense that agent $i$’s neighbors may be activated and start their computation before receiving the latest dual variables from agent $i$.

Our algorithm is different from the asynchronous ADMM algorithm by Wei and Ozdaglar [108]. Their algorithm activates an edge and its two associated agents at each iteration and thus requires two-sided communication at each activation. We can recover their algorithm as a special case by activating an edge $(i, j) \in E$ and its associated agents $i$ and $j$ at each iteration, updating the dual variables $z_{ij,i}$ and $z_{ij,j}$ associated with the edge, as well as computing the intermediate variables $x_i$, $x_j$, and $y_{ij}$. The updates are derived from (4.44) with the orders of $x$ and $y$ swapped. Note that [108] does not consider the situation that adjacent edges are activated in a short period of time, which may cause overlapped computation and delay communication. Indeed, their algorithm corresponds to $\tau = 0$ and the corresponding stepsize $\eta_k \equiv 1$. 

83
4.6 Nonconvex Problems

4.6.1 Nonnegative Matrix Factorization

Nonnegative matrix factorization (NMF) is an important dimension reduction method for nonnegative data. It was proposed by Paatero and his coworkers in [73]. Given a nonnegative matrix $A \in \mathbb{R}^{p \times n}_+$, NMF aims at finding two nonnegative matrices $W \in \mathbb{R}^{p \times r}_+$ and $H \in \mathbb{R}^{n \times r}_+$ such that $WH^T \approx A$, where $r$ is user-specified depending on the applications, and usually $r \ll \min(p,n)$. A widely used model is

$$
\minimize_{W,H} F(W, H) := \frac{1}{2} \| WH^T - A \|^2_F, 
$$

subject to $W \in \mathbb{R}^{p \times r}_+$, $H \in \mathbb{R}^{n \times r}_+$.

Applying the projected gradient method (2.16) to (4.46), we have

$$
W^{k+1} = \max \left( 0, W^k - \eta_k \nabla_w F(W^k, H^k) \right),
$$

$$
H^{k+1} = \max \left( 0, H^k - \eta_k \nabla_h F(W^k, H^k) \right).
$$

In general, we do not know the Lipschitz constant of $\nabla F$, so we have to choose $\eta_k$ by line search such that the Armijo condition is satisfied.

Partitioning the variables into $2r$ block coordinates: $(w_1, \ldots, w_r, h_1, \ldots, h_r)$ where $w_i$ and $h_i$ are the $i$th columns of $W$ and $H$, respectively, we can apply the coordinate update based on the projected-gradient method:

$$
\begin{align*}
\text{if } w_{i_k} \text{ is chosen for some } i_k \in [r], & \text{ then compute} \\
& w_{i_k}^{k+1} = \max \left( 0, w_{i_k}^k - \eta_k \nabla_{w_{i_k}} F(W^k, H^k) \right) \\
\text{if } h_{i_k-r} \text{ is chosen for some } i_k \in \{r+1, \ldots, 2r\}, & \text{ then compute} \\
& h_{i_k-r}^{k+1} = \max \left( 0, h_{i_k-r}^k - \eta_k \nabla_{h_{i_k-r}} F(W^k, H^k) \right)
\end{align*}
$$

It is easy to see that $\nabla_{w_i} F(W^k, H^k)$ and $\nabla_{h_i} F(W^k, H^k)$ are both Lipschitz continuous with constants $\|h_i^k\|_2^2$ and $\|w_i^k\|_2^2$ respectively. Hence, we can set

$$
\eta_k = \begin{cases} 
\frac{1}{\|h_{i_k}^k\|_2^2}, & \text{if } 1 \leq i_k \leq r, \\
\frac{1}{\|w_{i_k-r}^k\|_2^2}, & \text{if } r+1 \leq i_k \leq 2r.
\end{cases}
$$
However, it is possible to have $w_i^k = 0$ or $h_i^k = 0$ for some $i$ and $k$, and thus the setting in the above formula may have trouble of being divided by zero. To overcome this problem, one can first modify the problem \((4.46)\) by restricting $W$ to have unit-norm columns and then apply the coordinate update method in \((4.48)\). Note that the modification does not change the optimal value since $WH^\top = (WD)(HD^{-1})^\top$ for any $r \times r$ invertible diagonal matrix $D$. We refer the readers to [113] for more details.

Note that $\nabla_W F(W, H) = (WH^\top - A)H, \nabla_H F(W, H) = (WH^\top - A)^\top W$ and $\nabla_{w_i} F(W, H) = (WH^\top - A)h_i, \nabla_{h_i} F(W, H) = (WH^\top - A)^\top w_i, \forall i$. Therefore, the coordinate updates given in \((4.48)\) are computationally worthy (by maintaining the residual $W^k(H^k)^\top - A$).
The Software Architecture

ARock is a C++ library to simplify the implementation of both sequential, sync-parallel and async-parallel coordinate update algorithms. It is a realization of the CF framework (Chapter 2) and the async-parallel framework (Chapter 3). Parallelism of ARock is empowered by the thread library from the C++11 standard. The novelty of ARock is the introduction of a multilevel approach which reduces the gap between expert to low-level programming and novice-level programming.

At the highest level, ARock supports a wide range of prebuilt applications, including, but not limited to, linear equations, $\ell_1$ and $\ell_2$ regularized logistic regression, portfolio optimization, $\ell_1$ and $\ell_2$ regularized least squares, robust regression, quadratic programming, and nonnegative matrix factorization. The solvers of these applications can be interacted through easy-to-use command-line tools.

At the lower levels, ARock includes a rich set of operators, several common operator splitting schemes, and easy-to-use library calls. User can either use the existing operators or build their own operators and assemble them as an update scheme, which can then be used to solve their applications in an async-parallel fashion. This chapter discuss the system implementation of ARock.

The ARock package is available at

https://github.com/ZhiminPeng/arock-new

Currently, ARock supports Windows, Linux and Mac systems. Users can find the installation guidelines under the doc folder.
5.1 Motivation

Prior to the ARock project, I developed several async-parallel solvers for different applications. Though each of the applications were implemented independently, they shared the same coordinate update framework as discussed in Chapter 3. It was of interest to investigate a method for leveraging their common aspects, and simplify the implementation process for any new applications. Currently, ARock has a unified interface for asynchronous parallelization. Users only need to implement the application specific coordinate update operator. Parallelization, coordinate selection, dynamic step size tuning come automatically.

5.2 Quick Start

To illustrate the practical usage of ARock, we use the $\ell_1$ regularized logistic regression

$$\min_{x \in \mathbb{R}^n} \lambda \|x\|_1 + \sum_{i=1}^{m} \log \left( 1 + \exp(-b_i \cdot a_i^T x) \right)$$

(5.1)

as an example, where $\{(a_i, b_i)\}_{i=1}^m$ is the training dataset. We set the regularization parameter $\lambda$ to 1, and set the maximum number of epochs to 10. The following is the command to train the model on the news20$^1$ dataset with 2 threads,

```
$ arock_fbs_l1_log -data news20.svm -epoch 10 -nthread 2 -lambda 1.
[other outputs skipped]
Computing time is: 4.88(s).
```

where -data, -epoch, -nthread, -lambda are the flags for the data file, maximum number of epochs, total number of threads, and regularization parameter $\lambda$ respectively. We can see that the command-line tool is easy-to-use. Beyond the simplicity, ARock is also efficient in the sense that the training time is less than 5 seconds for a problem with more than 1 million variables. Next, we show the

---

$^1$ This dataset is from [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets](http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets).
major components of the source codes for building arock_fbs_l1_log.

We solve (5.1) with the forward-backward splitting scheme

$$x^{k+1} = \text{prox}_{\lambda \|x\|_1} (x^k - \eta \nabla \sum_{i=1}^{m} \log(1 + \exp(-b_i \cdot a_i^T x^k)), \quad (5.2)$$

where the gradient step of logistic loss and the proximal operator of $\ell_1$ norm correspond to the forward step and backward step respectively. The following snippet of code (extracted from apps/arock_fbs_l1_log.cc) implements (5.2) with the ARock framework. Specifically, line 2 and line 5 define the forward operator and backward operator respectively. Line 3 and line 6 use decltype to simplify the names for the operator types. Line 7 define the FBS with the previously defined forward and backward operators. Line 8 calls AROCK on the fbs object. We can see that creating an async-parallel solver can be easily achieved through assembling appropriate operators together.

```cpp
1 // forward operator: gradient step for logistic loss
forward_grad_for_log_loss<SpMat> forward(&A, &b, &Atx, eta);
3 using F = decltype(forward); // yields the type of forward
// backward operator: proximal operator for l1 norm
prox_l1 backward(eta, lambda);
5 using B = decltype(backward); // yields the type of backward
7 ForwardBackwardSplitting<F, B> fbs(&x, forward, backward);
AROCK(fbs, params);
```

5.3 Architecture

ARock reduces each application to the fixed-point problem (3.1) with the fixed-point operator $T$ being CF. The operator $T$ can be a single stand along operator or formed by a combination of multiple CF operators. Then the ARock kernel takes the operator $T$ and runs the Algorithm 2 with a given number of workers.
Figure 5.1 gives an overview of the multilevel architecture of the ARock package. In the following subsections, we explain each of the levels in more details.

Figure 5.1: Architectures for distributed memory systems. Linear algebra serves as the foundation for ARock. Other layers are structured according to the discussions in Chapter 2 and Chapter 3.

5.3.1 Library

The library layer includes three major components: data structures and linear algebra functions, file I/O functions, and helper functions. The primary data structures are dense vector, dense matrix and sparse matrix. We implemented a few handy BLAS (Basic Linear Algebra Subprograms) wrappers. File I/O functions for matrices in the matrix market format\(^2\) and LIBSVM format\(^3\) are also implemented. Helper functions include a list of objective functions, command-line argument parsers, and error handling functions. Those functions are extensively used in the operator layer and the splitting layer.

\(^2\)http://math.nist.gov/MatrixMarket/formats.html
\(^3\)https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
5.3.2 Operators

The operator layer is a library of projection operators, proximal operators and forward operators which are useful for high-level iterative algorithms. Currently, we implemented 17 operators, which are summarized in Table 5.1, Table 5.2 and Table 5.3. These operators are modularized and implemented in the form of function object (functor).

In general, each operator struct has three overloaded parenthesis operators: one consumes a scalar and calculates the update; one takes a vector and an index then applies the coordinate update; one takes an input vector and an output vector, then performs full update to the input vector and saves the results in the output vector. Constructors and update cached variable function are also provided. If an operator involves data, pointers to the data are added as member variables. The following code is a template for implementing operators.

```cpp
struct operator_name {
    double step_size;  // operator related step size
    double weight;     // weight on the operator
    double operator() (double val);  // scalar update operator
    // coordinate update operator
    double operator() (Vector* v, int index = 0);
    // full update operator
    void operator() (Vector* v_in, Vector* v_out);
    // update cached variables
    void update_cache_vars(double old_x_i, double new_x_i, int i);
    operator_name();    // default constructor
    operator_name(argument list);  // customized constructor
};
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Proximal operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \ell_1 ) norm</td>
<td>( w|x|_1 )</td>
<td>( \text{shrink}(x, w \cdot \sigma) )</td>
</tr>
<tr>
<td>sum of squares</td>
<td>( \frac{w}{2}|x|_2^2 )</td>
<td>( \frac{1}{1+w\sigma}x )</td>
</tr>
<tr>
<td>( \ell_2 ) norm</td>
<td>( w|x|_2 )</td>
<td>( \begin{cases} 0 &amp; \text{if } |x|_2 \leq w\sigma \ (1 - \frac{w\sigma}{|x|_2}) \cdot x &amp; \text{otherwise} \end{cases} )</td>
</tr>
<tr>
<td>Huber function</td>
<td>\begin{cases} \frac{w}{2}x^2, &amp; \text{if } -\delta \leq x \leq \delta \ w\delta(</td>
<td>x</td>
</tr>
<tr>
<td>elastic net</td>
<td>( w_1|x|_1 + \frac{w_2}{2}|x|_2^2 )</td>
<td>( \frac{1}{1+w_2\sigma} \cdot \text{shrink}(x, w_1\sigma) )</td>
</tr>
<tr>
<td>log barrier</td>
<td>(-w \sum_i \log(x_i) )</td>
<td>( \frac{1}{2}(x_i + \sqrt{x_i^2 + 4w\sigma}), \forall i )</td>
</tr>
</tbody>
</table>

Table 5.1: Proximal operator.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Projection operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>positive cone</td>
<td>{ ( x \mid x \geq 0 ) }</td>
<td>\text{max}(0, x_i), \forall i</td>
</tr>
<tr>
<td>box</td>
<td>{ ( x \mid l \leq x \leq u ) }, l, u \in \mathbb{R}^n</td>
<td>\text{max}(l_i, \text{min}(x_i, u_i))</td>
</tr>
<tr>
<td>( \ell_1 ) ball</td>
<td>{ ( x \mid |x|_1 \leq r ) }</td>
<td>( O(n \log(n)) ) method [48]</td>
</tr>
<tr>
<td>( \ell_2 ) ball</td>
<td>{ ( x \mid |x|_2 \leq r ) }</td>
<td>( \begin{cases} \frac{r}{|x|} \cdot x &amp; \text{if } |x|_2 \geq r \ x &amp; \text{otherwise} \end{cases} )</td>
</tr>
<tr>
<td>hyperplane</td>
<td>{ ( x \mid a^T x = b ) }</td>
<td>( x + \frac{(b-a^T x)}{a^T a} \cdot a )</td>
</tr>
<tr>
<td>probability simplex</td>
<td>{ ( x \mid x \geq 0, \sum_i x_i = 1 ) }</td>
<td>( O(n \log(n)) ) method [48]</td>
</tr>
</tbody>
</table>

Table 5.2: Projection operator.
<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
<th>Forward gradient operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>square loss</td>
<td>$\frac{w}{2}</td>
<td></td>
</tr>
<tr>
<td>quadratic function</td>
<td>$w \left( \frac{1}{2} x^T Q x + c^T x + d \right)$</td>
<td>$x - \sigma w (Q x + c)$</td>
</tr>
<tr>
<td>logistic loss</td>
<td>$w \sum_i \log(1 + \exp(-b_i \cdot a_i^T x))$</td>
<td>$x + \sigma w \sum_i \frac{b_i}{1 + \exp(b_i \cdot a_i^T x)} \cdot a_i$</td>
</tr>
<tr>
<td>square hinge loss</td>
<td>$\frac{w}{2} \sum \max(0, b_i (1 - a_i^T x))^2$</td>
<td>$x + \sigma w \sum b_i \max(0, b_i (1 - a_i^T x)) \cdot a_i$</td>
</tr>
<tr>
<td>square huber loss</td>
<td>$w \sum \text{huber}(a_i^T x - b_i)$</td>
<td>$x - \sigma w \nabla \text{huber}(a_i^T x - b_i)$</td>
</tr>
</tbody>
</table>

Table 5.3: Forward operator.

5.3.3 Splitting Schemes

Splitting schemes are templated on operators. Based on the discussions in Section 2.2.2, we implemented several splitting schemes, including PPA, FBS, BFS and PRS. The structure of the splitting scheme has the following form.

```cpp
template <typename Op1, typename Op2, ...>

struct OperatorSplitting {
    double relaxation_step_size;
    Vector* x;
    Op1 p1; // define the operators
    // calculates the update of x at index
    double operator(int index);
    // update the operator related parameters
    void update_params(Params* params);
    // customized constructor
    OperatorSplitting(argument list);
};
```

It usually has one or more than one operators as the template arguments. It has an overloaded parenthesis operator which performs updates to the unknown variable $x$ and maintained variables. The update is carried out by calling appropriate
member functions of the templated operators. An example of the implementation of the parenthesis operator for FBS is shown below.

```cpp
double operator() (int index) {
    // Step 1: read the old x[index]
    double old_x_at_idx = (*x)[index];
    // Step 2: local calculation
    double forward_grad_at_idx = forward(x, index);
    double val = backward(forward_grad_at_idx);
    double S_i = old_x_at_idx - val;
    // Step 3: get the most recent x[index] before updating it
    old_x_at_idx = (*x)[index];
    // Step 4: update x at index
    (*x)[index] -= relaxation_step_size * S_i;
    // Step 5: update the maintained variable Aty
    forward.update_cache_vars(old_x_at_idx, (*x)[index], index);
    return S_i;
}
```

5.3.4 Worker and Controller

ARock consists of several workers and a controller. Each worker has access to the shared data, the maintained variables, the unknown parameter $x$, and other algorithm related constants. Each worker continuously updates some coordinates of $x$ until the stopping criterions are satisfied. The coordinates can be updated in a cyclic or random fashion. At the same time, workers share delay monitoring information and convergence progress information with the controller, who use the information to dynamically update the step sizes to facilitate the convergence of ARock. Figure 5.2 demonstrates the relations of workers, controller, and shared memory.
5.3.5 ARock Interface

The ARock interface simply spawns a set of threads with one thread taking the controller role and the rest of threads being the workers. The interface of ARock is the following. It takes three inputs, including a splitting scheme, a parameter structure and a controller.

```cpp
template<typename Splitting>
void AROCK(Splitting op, Params parameters, Controller<Splitting> controller);
```

5.3.6 Applications

Section 5.2 gives an example of using ARock to solve the sparse logistic regression problem. Other applications are implemented or can be implemented similarly. We have implemented several applications from statistical machine learning and scientific computing with the ARock framework. The applications can be compiled into executable files, which can then be executed through the command-line interface. For an unimplemented application, if the necessary operators and splitting schemes are defined in the ARock library, then creating an async-parallel algorithm can be as simple as calling ARock on the appropriate splitting scheme.

Figure 5.2: Relationship between workers, controller and shared memory.
CHAPTER 6

Numerical Results

We illustrate the behavior of coordinate update algorithms for solving linear equations, portfolio optimization, image processing, elastic net logistic regression, and Lasso. Our primary goal is to show the efficiency of coordinate update algorithms compared to the corresponding full update algorithms. We also illustrate that ARock is generally more scalable compared to its sync-parallel counterparts.

Our sequential experiments run on Mac OSX 10.9 with 2.4 GHz Intel Core i5 and 8 Gigabytes of RAM. The experiments were coded in Matlab.

The linear equation and elastic net logistic regression experiment runs on 1 to 20 cores on a shared memory machine with two 2.5 GHz 10-core Intel Xeon E5-2670v2 (20 cores in total) CPU and 64 Gigabytes of RAM. The experiment was coded in C++. We use the Eigen library\(^1\) and BLAS for linear algebra operations.

The distributed Lasso experiment runs on Amazon EC2. We requested 20 high-memory quadruple extra-large instances giving us a total of 160 cores and 1.2TB of memory in total. The code was implemented with C and MPI. This example is used to demonstrate the advantage of greedy coordinate method for solving sparse optimization problems.

\(^1\)http://eigen.tuxfamily.org
6.1 Sequential Experiments

In this section, we use portfolio optimization and computed tomography (CT) image reconstruction to demonstrate the efficiency of coordinate update methods.

6.1.1 Portfolio Optimization

In this subsection, we compare the performance of the 3S splitting scheme (4.31) with the corresponding coordinate update algorithm (4.32) for solving the portfolio optimization problem (4.30). In this problem, our goal is to distribute our investment resources to all the assets so that the investment risk is minimized and the expected return is greater than $c$. This test uses two datasets, which are summarized in Table 6.1. The NASDAQ dataset is collected through Yahoo! Finance. We collected one year (from 10/31/2014 to 10/31/2015) of historical closing prices for 2730 stocks.

<table>
<thead>
<tr>
<th></th>
<th>Synthetic data</th>
<th>NASDAQ data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of assets (N)</td>
<td>1000</td>
<td>2730</td>
</tr>
<tr>
<td>Expected return rate</td>
<td>0.02</td>
<td>0.02</td>
</tr>
<tr>
<td>Asset return rate</td>
<td>$3 \times \text{rand}(N, 1) - 1$</td>
<td>mean of 30 days return rate</td>
</tr>
<tr>
<td>Risk covariance matrix</td>
<td>$+ 0.01 \cdot I$</td>
<td>positive definite matrix</td>
</tr>
</tbody>
</table>

Table 6.1: Two datasets for portfolio optimization

In our numerical experiments, for comparison purposes, we first obtain a high accurate solution by solving (4.30) with an interior point solver. For both full update and coordinate update, $\eta_k$ is set to 0.8. However, we use different $\gamma$. For 3S full update, we used the step size parameter $\gamma_1 = \frac{2}{\|Q\|_2}$, and for 3S coordinate update, $\gamma_2 = \frac{2}{\max\{Q_{11}, \ldots, Q_{NN}\}}$. In general, coordinate update can benefit from more relaxed parameters. The results are reported in Figure 6.1. We can observe that
Figure 6.1: Compare the convergence of 3S full update with 3S coordinate update algorithms.

the coordinate update method converges much faster than the 3S method for the synthetic data. This is due to the fact that $\gamma_2$ is much larger than $\gamma_1$. However, for the NASDAQ dataset, $\gamma_1 \approx \gamma_2$, so 3S coordinate update is only moderately faster than 3S full update.

6.1.2 Computed Tomography Image Reconstruction

We compare the performance of algorithm (4.19) and its corresponding coordinate version on Computed Tomography (CT) image reconstruction. We generate a thorax phantom of size $284 \times 284$ to simulate spectral CT measurements. We then apply the Siddon’s algorithm [94] to form the sinogram data. There are 90 parallel beam projections, and for each projection, there are 362 measurements. Then the sinogram data is corrupted with Gaussian noise. We formulate the image reconstruction problem in the form of (4.15). The primal-dual full update corresponds to (4.19). In each iteration, a coordinate is randomly chosen and updated. The reconstruction results are shown in Figure 6.2. After 100 epochs,
the image recovered by the coordinate version is better than that by (4.19). As shown in Figure 6.2d, the coordinate version converges faster than (4.19).

![Phantom image](Image1.png)
![Recovered by PDS](Image2.png)
![Recovered by PDS coord](Image3.png)
![Objective function value](Image4.png)

Figure 6.2: CT image reconstruction.

### 6.2 Parallel Experiments

We illustrate the performance of ARock for solving the linear equations and elastic net regularized logistic regression problem. Our primary goal is to show the efficiency of the async-parallel implementation compared to the sync-parallel implementation.
The running times and speedup ratios of both sync-parallel and async-parallel algorithms are sensitive to a number of factors, such as the size of each coordinate update (granularity), sparsity of the problem data, compiler optimization flags, and operations that affect cache performance and memory access contention. In addition, since all agents in the sync-parallel implementation must wait for the last agent to finish an iteration, a large load imbalance will significantly degrade the performance. Achieving load balancing requires knowledge of the sparsity pattern of the entire dataset, which might not be available for certain optimization problems.

### 6.2.1 Linear Equations

We apply ARock presented in Subsection 4.1 to solve $Ax = b$ on three real datasets collected from the University of Florida Sparse Matrix Collection. The three selected matrices are symmetric positive definite. The sizes and the sparsity patterns are shown in Table 6.2 and Figure 6.3 respectively.

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (n)</th>
<th>Nonzeros</th>
</tr>
</thead>
<tbody>
<tr>
<td>thermal2</td>
<td>1, 228, 045</td>
<td>4, 904, 179</td>
</tr>
<tr>
<td>G3_circuit</td>
<td>1, 585, 478</td>
<td>7, 660, 826</td>
</tr>
<tr>
<td>Hook_1498</td>
<td>1, 498, 023</td>
<td>59, 374, 451</td>
</tr>
</tbody>
</table>

Table 6.2: Linear equation datasets

Throughout the experiments, $A$, $b$ and $x$ are shared variables. We set $\eta_k = 0.5$, $\forall k$. Although the theory (Theorem 3) requires a smaller step size if more cores are used (usually leading to a larger delay $\tau$), we observe numerically that the $\eta_k = 0.5$ can serve well for the algorithm on different numbers of cores. For sync-parallel algorithm (Relaxed Jacobi method), the components of $x$ are evenly

---

2https://www.cise.ufl.edu/research/sparse/matrices/
partitioned into coordinates, and each coordinate contains \( \frac{n}{\text{number of cores}} \) components of \( x \), where the number of cores varies from 1 to 20. Each core is in charge of updating one pre-assigned coordinate. For ARock, we set the block size to 1000, and each thread randomly select a block to update. We run both sync-parallel Jacobi and ARock algorithms to 100 epochs where an epoch is counted for every \( n \) updates to the components of \( x \).

Figure 6.4 depicts how the size of residual, \( \|Ax - b\| \), reduces over the wall-clock time. From the figure, we see that both sync-parallel Jacobi (full as shown on the legend) and ARock show almost-linear speedup as the number of threads increases, but ARock is approximately two times faster than the sync-parallel Jacobi. Figure 6.5 shows the speedup performance of the two methods, where the solid lines represent the average speedup of 10 different runs, and the bands represent the lower and upper speedup bounds. We see that the scaling performance of ARock is better than the sync-parallel Jacobi method.
6.2.2 Elastic Net Regularized Logistic Regression

In this subsection, we apply ARock with the update (4.4) to the elastic net regularized logistic regression problem

$$\text{minimize}_{x \in \mathbb{R}^n} \lambda_1 \|x\|_1 + \frac{\lambda_2}{2} \|x\|_2^2 + \sum_{i=1}^{N} \log \left(1 + \exp(-b_i \cdot a_i^T x)\right), \quad (6.1)$$

where \(\{(a_i, b_i)\}_{i=1}^{N}\) is the set of sample-label pairs, \(\lambda_1 = 0.001\), and \(n\) and \(N\) represent the numbers of features and samples, respectively. This test uses two libsvm datasets\(^3\): news20, and url, which are summarized in Table 6.3.

We let each coordinate hold roughly 100 features. We let each agent draw a coordinate uniformly at random at each iteration. We stop all the tests after

\(^3\)http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/
Table 6.3: Two datasets for sparse logistic regression.

<table>
<thead>
<tr>
<th>Name</th>
<th># samples</th>
<th># features</th>
<th># nonzeros in {a_1, \ldots, a_N}</th>
</tr>
</thead>
<tbody>
<tr>
<td>news20</td>
<td>19,996</td>
<td>1,355,191</td>
<td>9,097,916</td>
</tr>
<tr>
<td>url</td>
<td>2,396,130</td>
<td>3,231,961</td>
<td>277,058,644</td>
</tr>
</tbody>
</table>

Figure 6.6: The distribution of coordinate sparsity. Each dot represents the total number of nonzeros in the vectors $a_i$ that correspond to each coordinate.

50 epochs. The step size is set to $\eta_k = 0.5$, $\forall k$. Let $A = [a_1, \ldots, a_N]^T$ and $b = [b_1, \ldots, b_N]^T$. In global memory, we store $A$, $b$, and $x$. We also store the product $Ax$ in global memory so that the forward step can be efficiently computed. The step size associate with the forward operator is manually tuned. We pick the one that gives the best performance. Whenever a coordinate of $x$ gets updated, $Ax$ is immediately updated at a low cost. Note that if $Ax$ is not stored in global memory, every coordinate update will have to compute $Ax$ from scratch, which involves the entire $x$ and will be very expensive.

Figure 6.7 gives the running times of the full update (sync-parallel) and ARock (async-parallel) implementations on the two datasets. Figure 6.8 is the speedup performance comparison of the two methods. We can observe that ARock achieves approximate-linear speedup, but sync-parallel scales poorly as we explain below. One can also see that ARock converges faster due to more relaxed forward operator
step size selection.

Figure 6.7: Objective vs wall clock time.

Figure 6.8: Speedup vs number of threads.

In the sync-parallel implementation, all the running cores have to wait for the last core to finish an iteration, and therefore if a core has a large load, it slows down the iteration. Although every core is (randomly) assigned to roughly the same number of features at each iteration, their $a_i$’s have very different numbers of nonzeros (see Figure 6.6 for the distribution), and the core with the largest number of nonzeros is the slowest (Sparse matrix computation is used for both datasets, which are very large.) As more cores are used, despite that they altogether do
more work at each iteration, the per-iteration time reduces as the slowest core tends to be slower.

On the other hand, being asynchronous, ARock does not suffer from the load imbalance. Its performance grows nearly linear with the number of cores. In theory, a large load imbalance may cause a large $\tau$, and thus a small $\eta_k$. However, the uniform $\eta_k = 0.5$ works well in all the tests, possibly because the $a_i$’s are sparse. Figure 6.9 is the profile of using two threads to solve (6.1) with the news20 dataset. We can see that a large fraction of CPU time is wasted for sync-parallel due to the synchronization overhead.

Figure 6.9: Profile of sync-parallel and async-parallel: red indicates waiting time, and brown indicates computing time.
6.3 Distributed Experiments

In this section, we show a distributed implementation of a greedy coordinate update method for solving the following Lasso problem

\[
\minimize_{x} \lambda \|x\|_1 + \frac{1}{2} \|Ax - b\|^2
\] (6.2)

In this Lasso test, \(A\) is generated by \(\text{randn}(1 \times 10^5, 2 \times 10^5)\) with 20 billion nonzero entries and having a size of 170GB. The solution \(x^*\) has 4000 nonzero entries, each sampled from \(\mathcal{N}(0, 1)\) independently. We set the maximum number of iterations to 2500 and the stopping criterion as \(\frac{\|x^k - x^*\|}{\|x^*\|} \leq 10^{-5}\). On Amazon EC2, we requested 20 high-memory quadruple extra-large instances giving us a total of 160 cores and 1.2TB of memory in total. Table 6.4 compares the performance of Distributed ADMM and Accelerated FBS and greedy coordinate update (GRock). Note that the performance of D-ADMM depends on a penalty parameter. We pick it as the best out of only a few trials as we cannot afford more trials. The results show that GRock converges to a high accuracy solution much faster than the other two methods.

<table>
<thead>
<tr>
<th></th>
<th>D-ADMM</th>
<th>Accelerated FBS</th>
<th>GRock</th>
</tr>
</thead>
<tbody>
<tr>
<td>estimate stepsize (min.)</td>
<td>n/a</td>
<td>1.6</td>
<td>n/a</td>
</tr>
<tr>
<td>matrix factorization (min.)</td>
<td>51</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>iteration time (min.)</td>
<td>105</td>
<td>40</td>
<td>1.7</td>
</tr>
<tr>
<td>number of iterations</td>
<td>2500</td>
<td>2500</td>
<td>104</td>
</tr>
<tr>
<td>communication time</td>
<td>30.7</td>
<td>9.5</td>
<td>0.5</td>
</tr>
<tr>
<td>stopping relative error</td>
<td>1E-1</td>
<td>1E-3</td>
<td>1E-5</td>
</tr>
<tr>
<td>total time (min)</td>
<td>156</td>
<td>41.6</td>
<td>1.7</td>
</tr>
<tr>
<td>cost</td>
<td>$85</td>
<td>$22.6</td>
<td>$0.93</td>
</tr>
</tbody>
</table>

Table 6.4: Results for distributed Lasso test.
CHAPTER 7

Conclusion and Future Works

We have presented a coordinate update framework for fixed-point iterations, which updates one coordinate (or a few variables) at every iteration and can be applied to solve linear systems, optimization problems, saddle point problems, variational inequalities, and so on. We proposed a new concept called CF operator. When an operator is CF, its coordinate update is computationally worthy and often preferable over the full update method, in particular, in a parallel computing setting. We have also proposed ARock, an async-parallel framework, as the asynchronous extension of CF operator. We establish the almost sure weak and strong convergence, linear convergence rate and almost-linear speedup of ARock under certain assumptions. Preliminary numerical results on real data illustrate the high efficiency of the proposed framework compared to the traditional parallel (sync-parallel) algorithms.
CHAPTER 8

Appendix

8.1 Derivation of ADMM from the DRS update

We derive the ADMM update in (2.18) from the DRS update

\[ s^k = \mathcal{J}_{\eta B}(t^k), \quad (8.1a) \]
\[ t^{k+1} = \left( \frac{1}{2}(2\mathcal{J}_{\eta A} - \mathcal{I}) \circ (2\mathcal{J}_{\eta B} - \mathcal{I}) + \frac{1}{2}\mathcal{I} \right)(t^k), \quad (8.1b) \]

where \( \mathcal{A} = -\partial f^*(-\cdot) \) and \( \mathcal{B} = \partial g^* \).

Note (8.1a) is equivalent to \( t^k \in s^k + \eta \partial g^*(s^k) \), i.e., there is \( y^k \in \partial g^*(s^k) \) such that \( t^k = s^k + \eta y^k \), so

\[ t^k - \eta y^k = s^k \in \partial g(y^k). \quad (8.2) \]

In addition, (8.1b) can be written as

\[ t^{k+1} = \mathcal{J}_{\eta A}(2s^k - t^k) + t^k - s^k \]
\[ = s^k + (\mathcal{J}_{\eta A} - \mathcal{I})(2s^k - t^k) \]
\[ = s^k + (\mathcal{I} - (\mathcal{I} + \eta \partial f^*)^{-1})(t^k - 2s^k) \]
\[ = s^k + \eta(\eta \mathcal{I} + \partial f)^{-1}(t^k - 2s^k) \]
\[ = s^k + \eta(\eta \mathcal{I} + \partial f)^{-1}(\eta y^k - s^k), \quad (8.3) \]

where in the fourth equality, we have used Moreau's Identity [90]: \((\mathcal{I} + \partial h)^{-1} + (\mathcal{I} + \partial h^*)^{-1} = \mathcal{I}\) for any closed convex function \( h \). Let

\[ x^{k+1} = (\eta \mathcal{I} + \partial f)^{-1}(\eta y^k - s^k) = (\mathcal{I} + \frac{1}{\eta} \partial f)^{-1}(y^k - \frac{1}{\eta} s^k). \quad (8.4) \]
Then (8.3) becomes
\[ t^{k+1} = s^k + \eta x^{k+1}, \]
and
\[ s^{k+1} = t^{k+1} - \eta y^{k+1} = s^k + \eta x^{k+1} - \eta y^{k+1}, \] (8.5)
which together with \( s^{k+1} \in \partial g(y^{k+1}) \) gives
\[ y^{k+1} = (\eta \mathcal{I} + \partial g)^{-1}(t^k + \eta x^{k+1}) = (\mathcal{I} + \frac{1}{\eta} \partial g)^{-1}(x^{k+1} + \frac{1}{\eta} s^k). \] (8.6)
Hence, from (8.4), (8.5), and (8.6), the ADMM update in (2.18) is equivalent to the DRS update in (8.1) with \( \eta = \frac{1}{\gamma} \).

8.2 Representing the Condat-Vũ Algorithm as a Nonexpansive Operator

We show how to derive the Condat-Vũ algorithm (2.23) by applying a forward-backward operator to the optimality condition (2.22):
\[
0 \in \begin{bmatrix} \nabla f(x) \\ 0 \end{bmatrix} + \begin{bmatrix} \partial g(x) \\ \partial h^*(s) \end{bmatrix} + \begin{bmatrix} 0 & A^T \\ -A & 0 \end{bmatrix} \begin{bmatrix} x \\ s \end{bmatrix}.
\] (8.7)
It can be written as \( 0 \in Az + Bz \) after we define \( z = \begin{bmatrix} x \\ s \end{bmatrix} \). Let \( M \) be a symmetric positive definite matrix, we have
\[
0 \in Az + Bz
\]
\[ \Leftrightarrow Mz - Az \in Mz + Bz \]
\[ \Leftrightarrow z - M^{-1}Az \in z + M^{-1}Bz \]
\[ \Leftrightarrow z = (\mathcal{I} + M^{-1}B)^{-1} \circ (\mathcal{I} - M^{-1}A)z. \]
Convergence and other results can be found in [30]. The last equivalent relation is due to $M^{-1}B$ being a maximally monotone operator. We let
\[
M = \begin{bmatrix}
\frac{1}{\eta}I & A^\top \\
A & \frac{1}{\gamma}I
\end{bmatrix} > 0
\]
and iterate
\[
z^{k+1} = T z^k = (\mathcal{I} + M^{-1}B)^{-1} \circ (\mathcal{I} - M^{-1}A)z^k.
\]
We have $Mz^{k+1} + \tilde{B}z^{k+1} = Mz^k - Az^k$:
\[
\begin{cases}
\frac{1}{\eta}x^k + A^\top s^k - \nabla f(x^k) & \in \frac{1}{\eta}x^{k+1} + A^\top s^{k+1} + A^\top s^{k+1} + \nabla g(x^{k+1}), \\
\frac{1}{\gamma}s^k + \gamma A x^k & \in \frac{1}{\gamma}s^{k+1} + A x^{k+1} - A x^{k+1} + \nabla h^*(s^{k+1}),
\end{cases}
\]
which is equivalent to
\[
\begin{cases}
s^{k+1} = \text{prox}_{\gamma h^*}(s^k + \gamma Ax^k) \\
x^{k+1} = \text{prox}_{\eta g}(x^k - \frac{\eta}{\gamma}(\nabla f(x^k) + A^\top (2s^{k+1} - s^k)))
\end{cases}
\]
Now we derived the Condat-Vũ algorithm. With proper choice of $\eta$ and $\gamma$, the forward-backward operator $T = (\mathcal{I} + M^{-1}B)^{-1} \circ (\mathcal{I} - M^{-1}A)$ can be shown to be $\alpha$-averaged if we use the inner product $\langle z_1, z_2 \rangle_M = z_1^\top Mz_2$ and norm $\|z\|_M = \sqrt{z^\top Mz}$ on the space of $z = \begin{bmatrix} x \\ s \end{bmatrix}$. More details can be found in [30].

If we change the matrix $M$ to
\[
\begin{bmatrix}
\frac{1}{\eta}I & -A^\top \\
-A & \frac{1}{\gamma}I
\end{bmatrix},
\]
the other algorithm (2.24) can be derived in the same way.
REFERENCES


116


