RANDOMIZED LINEAR ALGEBRA FOR LARGE-SCALE DATA APPLICATIONS

A DISSERTATION
SUBMITTED TO THE INSTITUTE FOR
COMPUTATIONAL AND MATHEMATICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

JIYAN YANG
AUGUST 2016

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Michael Saunders, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Chris Re

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Michael Mahoney

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.

Dedicated to my parents

ABSTRACT

As the ability to generate datasets with billions of records in relatively automated ways continues to grow, new challenges are posed to modern large-scale data analysis from several points of view such as scalability, feasibility, and interpretability. Thus, improved algorithms on large-scale data platforms are of interest. Recent years have seen great interest in Randomized Linear Algebra (RLA) algorithms. RLA is an interdisciplinary research area that exploits randomization as a computational resource for the development of improved algorithms for common matrix problems such as least-squares approximation, low-rank matrix approximation, and Laplacian-based linear equation solvers. In this thesis, our focus is on the underlying theory and practical implementation of RLA algorithms. In particular:

CHAPTER 3 describes a novel sampling algorithm for large-scale over-determined quantile regression problems whose running time is roughly proportional to the number of non-zero elements in the matrix plus a term that depends on the low dimension only.

CHAPTER 4 describes a hybrid algorithm named pwSGD — precondition weighted stochastic gradient descent that combines RLA and SGD. We prove that pwSGD inherits faster convergence rates that only depend on the lower dimension of the linear system, while maintaining low computational complexity.

CHAPTER 5 describes a class of randomized Newton-type algorithms that exploit non-uniform sub-sampling as well as inexact updates for a class of constrained optimization problems. We show that our methods are more robust to ill-conditioned problems than other similar previous approaches.

CHAPTER 6 presents results of implementing RLA algorithms for least-squares problems, quantile regression, and CX decomposition problems in parallel and distributed environments using Apache Spark, and discuss various tradeoffs in the implementations. In particular, we demonstrate that least-squares problems with up to terabyte-sized data can be solved to low, medium, or high precision on existing distributed systems.

CHAPTER 7 demonstrates that applying CX/CUR decomposition to large-scale Mass Spectrometry Imaging (MSI) datasets returns interpretable results as it successfully identifies important ions and locations in complex biological samples.

All in all, we show that RLA is a powerful tool for deriving scalable algorithms for large-scale regression and optimization problems, and we demonstrate that RLA algorithms are amenable to distributed computing platforms and are useful in scientific applications.

ACKNOWLEDGMENTS

My experience at Stanford has been a very rewarding and influential chapter of my life. I developed lifelong friendships and countless memories during my study here.

This body of work would not be possible without, first and foremost, my advisors, Michael Saunders and Michael Mahoney. It has been a great pleasure working with them. To me, they are not only my role models, but also my friends. MS, throughout my graduate career, I was continuously inspired by your hard work, optimism, and good nature. Thank you for always being available to discuss, read drafts, and support me, and giving me advice and understanding on what comes after graduate school.

MM, thank you for opening my eyes to this wonderful research area. Your great ideas and insights have provided me with sound technical guidance. Your door was always open and you were always willing to serve as a sounding board for my many random ideas. Thank you!

Besides my advisors I would like to thank Chris Ré, whom I have been closely working with over the past three years. Chris, thank you for your encouragements and the hard questions that motivate me to widen my research from various perspectives.

My other committee members are Art Owen and John Duchi. I would also like to thank them for their insightful comments. They are exemplary researchers and people.

I was extremely fortunate to collaborate with many brilliant and talented researchers, next to whom I have become a better researcher: Xiangrui, Ofer, Vikas, Haim, Alex, Yinlam, Junjie, Ben, Prabhat, Yuekai, Peng, and Fred. I learned far more than I could have imagined from them. Their knowledge and insights have broadened my eyesights and research interests.

I am extremely grateful for the help of many amazing people at ICME. To begin, I wish to thank Margot for turning ICME into a better and better place. I truly enjoyed the resources I have received here. I also wish to thank Indira for saving me from missing deadlines again and again, and Brian for keeping `icme-sharedmem.stanford.edu` and `icme-hadoop1.stanford.edu` in good conditions, and many other ICME staffs for keeping ICME running smoothly: Chris, Antoinette, Claudine, Emily, and Matt. Last but not least, I wish to thank my most caring and supporting friends at Stanford for their encouragements along the way.

I also need to acknowledge my family, without whose support I would not have gotten this far. My parents have been unwavering in their encouragement of me. Their sacrifices allowed me to get the education that laid the foundation for coming to Stanford. They are the kindest, most supportive people you could hope to have as parents.

Finally, I would like to acknowledge the support of ARO, NSF, and DARPA for making this work possible.

CONTENTS

1 INTRODUCTION

Matrix algorithms lie at the heart of modern data analysis. The essential reason is that matrices provide a convenient mathematical structure with which to model data arising in a broad range of applications: an $n \times d$ real-valued matrix A provides a natural structure for encoding information about n objects, each of which is described by d features. For example, in many internet applications, term-document matrices can be constructed with A_{ij} indicating the frequency of the j-th term in the i-th document. In genetics, DNA SNP (Single Nucleotide Polymorphism) or DNA microarray expression data can be represented in such a framework, with A_{ij} representing the expression level of the i-th gene or SNP in the j-th experimental condition or individual.

With this handy structure, many data applications can be formulated as matrix problems. One prototypical example is the least-squares problem. That is, given $A \in \mathbb{R}^{n \times d}$ and $b \in \mathbb{R}^n$, solve

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

The solution is the best fit in the least-squares sense: it minimizes the sum of squared residuals. It can be used to depict the relationship between the response variable, i.e., vector b and the observed covariates, i.e., columns of A. Another fundamental matrix problem is low-rank matrix factorization. Given an $m \times n$ data matrix A,[1] find two smaller matrices whose product is a good approximation to A. That is, they aim to find matrices Y and Z such that

$$\underset{m \times n}{A} \approx \underset{m \times k}{Y} \times \underset{k \times n}{Z}$$

is a rank-k approximation to the original matrix A ($k \leq \min(m, n)$). Low-rank factorizations are often useful in data compression, as smaller matrices can be stored more efficiently. Also, as it is difficult to visualize data sets containing a massive number of rows or columns, low-rank approximation methods express every data point in a low-dimensional space defined by only a few features.

However, as the ability to generate datasets with billions of records in relatively automated ways continues to grow, new challenges are posed to modern large-scale data analysis. There are three main reasons. First, many traditional matrix algorithms don't scale well with the size of the dataset. For example, as a direct solver requires $\mathcal{O}(nd^2)$ time to solve the above least-squares problem, it might take three days to solve a large-scale least-squares problem, e.g., $n = 10^9$ and $d = 10^3$. Second, with large datasets, it is often the case that the data cannot fit into the memory on a single machine, then secondary storage will be necessary, which incurs enormous I/O costs during each pass. However, most traditional matrix algorithms are designed to run on a single machine and are not amenable to parallel and distributed computing environments, which are easily accessible nowadays. Third, as the size of the datasets grows, practitioners are often interested in matrix decomposition methods that use a few *actual* columns/rows to reconstruct the matrix. It is then much easier to interpret the results for domain experts. Thus, improved algorithms on large-scale data platforms are of interest.

1 Here when talking about matrix factorization, we assume A is an $m \times n$ matrix rather than $n \times d$.

Recent years have seen great interest in Randomized Linear Algebra (RLA) algorithms. RLA is an interdisciplinary research area that exploits randomization as a computational resource for the development of improved algorithms for common matrix problems such as least-squares approximation, low-rank matrix approximation, and Laplacian-based linear equation solvers.

In this work, our main focus is on the underlying theory and practical implementation of RLA algorithms for various matrix problems such as ℓ_p regression and CX/CUR decomposition. The contributions can be summarized as follows. In terms of the theory,

CHAPTER 3 describes a novel sampling algorithm for large-scale overdetermined quantile regression problems that runs in time that is nearly linear in the number of nonzeros in the input data. Quantile regression is a method to estimate the quantiles of the conditional distribution of a response variable, expressed as functions of observed covariates, which permits a much more accurate portrayal of the relationship between the response variable and observed covariates. The core of our algorithm is constructing a subspace-preserving embedding for the loss function of quantile regression using RLA. The material in this chapter appears in Yang, Meng, and Mahoney [YMM13; YMM14].

CHAPTER 4 describes a hybrid algorithm named pwSGD — precondition weighted stochastic gradient descent that uses RLA techniques to construct a randomized preconditioner and an importance sampling distribution, and then performs an SGD-like iterative process with weighted sampling on the preconditioned system for ℓ_p regression. Here the randomized preconditioner is constructed using subspace embeddings in RLA. We prove that pwSGD inherits faster convergence rates that only depend on the lower dimension of the linear system, while maintaining low computational complexity. The material in this chapter appears in Yang et al. [Yan+16a; Yan+16b].

CHAPTER 5 describes a class of randomized Newton-type algorithms that exploit *non-uniform* sub-sampling as well as inexact updates to reduce the computational complexity for optimization problems where the objective can be written as $F(w) = \sum_{i=1}^{n} f_i(w) + R(w)$. Two RLA based data-aware non-uniform sampling techniques, namely leverage scores sampling and row norm squares sampling, are considered. We show that these techniques drastically drive down the complexity of Newton's algorithm while maintaining a fast convergence rate. We also demonstrate that they are more robust to the ill-conditioning of the problem both theoretically and empirically. The material in this chapter appears in Xu et al. [Xu+16].

As for the practical side,

CHAPTER 6 presents results of implementing RLA algorithms for overdetermined least-squares problems, overdetermined quantile regression, and CX/CUR decomposition problems in parallel and distributed environments using Apache Spark. These problems are ubiquitous in many fields to which machine learning and data analysis methods are routinely applied, such as geophysical applications and high-frequency trading. We show that these algorithms are amenable to distributed computing. In particular, we demonstrate that least squares problems with up to terabyte-sized data can be solved to low, medium, or high precision on existing distributed systems. For CX/CUR decomposition, in order to assess the relative performance on various hardware, we consider three contemporary platforms. We report results on

these platforms and their implications on the hardware and software issues arising in supporting data-centric workloads. The material in this chapter appears in Yang, Meng, and Mahoney [YMM16; YMM14] and Gittens et al. [Git+16a]

CHAPTER 7 demonstrates that applying CX/CUR decomposition to large-scale Mass Spectrometry Imaging (MSI) datasets returns interpretable results as it successfully identifies biologically important ions and locations in complex biological samples. This demonstrates the usefulness of RLA algorithms in large-scale scientific applications. The material in this chapter appears in Yang et al. [Yan+15].

In addition, we present two novel works regarding large-scale kernel learning, which is a useful tool for exploiting nonlinear patterns in the dataset.

CHAPTER 8 describes Quasi-Monte Carlo feature maps for shift-invariant kernels to accelerate training and testing speed of kernel methods on large datasets. This uses Quasi-Monte Carlo techniques to approximate integral representations of shift-invariant kernel functions. We derive a new discrepancy measure called *box discrepancy* based on theoretical characterizations of the integration error with respect to a given sequence. Our theoretical analyses are complemented with empirical results that demonstrate the effectiveness of classical and adaptive QMC techniques for this problem. The material in this chapter appears in Yang et al. [Yan+14a] and Avron et al. [Avr+16].

CHAPTER 9 develops a new randomized technique called random Laplace features to approximate a family of kernel functions adapted to the semigroup structure of $\mathbb{R}_+^d$. This is the natural algebraic structure on the set of histograms and other nonnegative data representations and thus it finds applications in computer vision. We provide theoretical results on the uniform convergence of random Laplace features. Empirical analyses on image classification and surveillance event detection tasks demonstrate the attractiveness of using random Laplace features. The material in this chapter appears in Yang et al. [Yan+14b].

The rest of this thesis is organized as follows. In Chapter 2 we provide background and preliminaries in RLA that are frequently used in the thesis. Following that are the corresponding chapters that contain details of works mentioned above. For better presentation, we defer all proofs to the appendices.

Part I

THEORETICAL RESULTS

2 RANDOMIZED LINEAR ALGEBRA BACKGROUND

In this chapter, we first define the notation used. In section 2.2 we introduce two important notions in RLA—conditioning and leveraging scores. Three core technical tools, namely ellipsoidal rounding, low-distortion subspace embedding, and preconditioning, are described in Section 2.3. Finally, in Section 2.4 we discuss the application of RLA to two classical problems—ℓ_p regression and low-rank factorization. Overviews of the general RLA area have been provided in [Mah11; DM16].

2.1 NOTATION

- Uppercase letters denote matrices and constants: e.g., A, R, C, etc.
- Lowercase letters denote vectors and scalars: e.g., x, b, p, m, n, etc.
- For any matrix A, $A_{(i)}$, $A^{(j)}$, and A_{ij} denote the i-th row, the j-th column of A, and the (i,j)-th element of A; and $\mathcal{A}$ denotes the column space of A.
- As long as it is clear in the text, we use x_i to denote either the i-th coordinate of x or the i-th vector in a sequence of vectors $\{x_i\}_{i=1}^n$.
- We use $\|\cdot\|_p$ to denote the ℓ_p norm of a vector, $\|\cdot\|_2$ the spectral norm of a matrix, $\|\cdot\|_F$ the Frobenius norm of a matrix, and $|\cdot|_p$ the element-wise ℓ_p norm of a matrix.
- We use $[n]$ to denote the set $\{1, 2, \ldots, n\}$.
- We use $\texttt{nnz}(A)$ to denote the number of nonzero elements in A.
- We use $\texttt{poly}(d)$ to denote a polynomial in d of low degree, e.g., 7.
- We use $A^\dagger$ to denote the Moore-Penrose pseudoinverse of A.

2.2 CONDITIONING AND LEVERAGE SCORES

Leveraging and conditioning refer to two types of problem-specific complexity measures, i.e., quantities that can be computed for any problem instance that characterize how difficult the problem instance is for a particular class of algorithms. Understanding these, as well as different uses of randomization in algorithm design, is important for designing RLA algorithms, in theory and in practice.

2.2.1 *Leverage scores*

If we let $H = A(A^TA)^\dagger A^T$ be the projection matrix onto the column span of A, then leverage scores of A are defined as the diagonal elements of H. Formally, we have the following definition.

Definition 2.1 (Leverage scores). *Given $A \in \mathbb{R}^{n\times d}$, then for $i = 1,\dots,n$, the i-th statistical leverage score of A is defined as*

$$\ell_i = A_{(i)}^T (A^T A)^\dagger A_{(i)}.$$

Since H can alternatively be expressed as $H = UU^T$, where U is any orthogonal basis for the column space of A, e.g., the Q matrix from a QR decomposition or the matrix of left singular vectors from the thin SVD, the leverage of the i-th observation can also be expressed as

$$h_{ii} = \sum_{j=1}^{n} U_{ij}^2 = \|U_{(i)}\|^2. \tag{2.1}$$

Leverage scores provide a notion of "coherence" or "outlierness," in that they measure how well-correlated the singular vectors are with the canonical basis [MD09; Dri+12; CR12] as well as which rows/constraints have largest "influence" on the least-squares fit [HW78; CH86; VW81; CH88]. Computing the leverage scores $\{h_{ii}\}_{i=1}^n$ *exactly* is generally as hard as solving the original LS problem, but $1\pm\epsilon$ approximations to them can be computed more quickly, for arbitrary input matrices [Dri+12].

Leverage scores are important from an algorithm design perspective because they define the key non-uniformity structure needed to control the complexity of high-quality random sampling algorithms. In particular, naïve uniform random sampling algorithms perform poorly when the leverage scores are very non-uniform, while randomly sampling in a manner that depends on the leverage scores leads to high-quality solutions. Thus, in designing RLA algorithms, whether in RAM or in parallel-distributed environments, one must either quickly compute approximations to the leverage scores or quickly preprocess the input matrix so they are nearly uniformized—in which case uniform random sampling on the preprocessed matrix performs well.

Informally, the leverage scores characterize where in the high-dimensional Euclidean space the (singular value) information in A is being sent, i.e., how the quadratic well (with aspect ratio $\kappa(A)$ that is implicitly defined by the matrix A) "sits" with respect to the canonical axes of the high-dimensional Euclidean space. If one is interested in obtaining *low-precision solutions* for least-squares problems, e.g., $\epsilon = 10^{-1}$, that can be obtained by an algorithm that provides $1\pm\epsilon$ relative-error approximations for a fixed value of ϵ but whose ϵ dependence is polynomial in $1/\epsilon$, then the key quantities that must be dealt with are statistical leverage scores of the input data.

2.2.2 ℓ_p-norm condition number

If we let $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ denote the largest and smallest nonzero singular values of A, respectively, then $\kappa(A) = \sigma_{\max}(A)/\sigma_{\min}^+(A)$ is the ℓ_2-norm condition number of A which is formally defined in Definition 2.2. Computing $\kappa(A)$ *exactly* is generally as hard as solving a least-square problem. The condition number $\kappa(A)$ is important from an algorithm design perspective because $\kappa(A)$ defines the key non-uniformity structure needed to control the complexity of high-precision iterative algorithms, i.e., it bounds the number of iterations needed for iterative methods to converge. In particular, for ill-conditioned problems, e.g., if $\kappa(A) \approx 10^6 \gg 1$, then the convergence speed of iterative methods is very slow, while if $\kappa \gtrsim 1$ then iterative algorithms converge very quickly. Informally, $\kappa(A)$ defines the aspect ratio of the quadratic well implicitly defined by A in the high-dimensional Euclidean space. If one is interested in obtaining *high-precision solutions* for least-squares problems, e.g., $\epsilon = 10^{-10}$, that can be obtained by iterating

a low-precision solution to high precision with an iterative algorithm that converges as $\log(1/\epsilon)$, then the key quantity that must be dealt with is the condition number of the input data.

For linear systems and least-squares problems, the ℓ_2-norm condition number is already a well-established term.

Definition 2.2 (ℓ_2-norm condition number)**.** *Given a matrix $A \in \mathbb{R}^{n\times d}$ with full column rank, let $\sigma_2^{\max}(A)$ be the largest singular value and $\sigma_2^{\min}(A)$ be the smallest singular value of A. The* ℓ_2-norm condition number *of A is defined as $\kappa_2(A) = \sigma_2^{\max}(A)/\sigma_2^{\min}(A)$. For simplicity, we use κ_2 (or κ), $\sigma_2^{\min}$, and $\sigma_2^{\max}$ when the underlying matrix is clear from context.*

For general ℓ_p norm and general ℓ_p regression problems, here we state two related notions of condition number and then a lemma that characterizes the relationship between them.

Definition 2.3 (ℓ_p-norm condition number [Cla+13])**.** *Given a matrix $A \in \mathbb{R}^{n\times d}$ and $p \in [1,\infty]$, let*

$$\sigma_p^{\max}(A) = \max_{\|x\|_2=1} \|Ax\|_p \text{ and } \sigma_p^{\min}(A) = \min_{\|x\|_2=1} \|Ax\|_p.$$

Then, we denote by $\kappa_p(A)$ the ℓ_p-norm condition number of A, defined to be

$$\kappa_p(A) = \sigma_p^{\max}(A)/\sigma_p^{\min}(A).$$

For simplicity, we use κ_p, $\sigma_p^{\min}$, and $\sigma_p^{\max}$ when the underlying matrix is clear.

Definition 2.4 ((α,β,p)-conditioning [Das+09])**.** *Given a matrix $A \in \mathbb{R}^{n\times d}$ and $p \in [1,\infty]$, let $\|\cdot\|_q$ be the dual norm of $\|\cdot\|_p$. Then A is* (α,β,p)-conditioned *if (1) $|A|_p \le \alpha$, and (2) for all $z \in \mathbb{R}^n$, $\|z\|_q \le \beta\|Az\|_p$. Define $\bar{\kappa}_p(A)$, the (α,β,p)-condition number of A, as the minimum value of $\alpha\beta$ such that A is (α,β,p)-conditioned. We use $\bar{\kappa}_p$ for simplicity if the underlying matrix is clear.*

Lemma 2.5 (Equivalence of κ_p and $\bar{\kappa}_p$ [Cla+13])**.** *Given a matrix $A \in \mathbb{R}^{n\times d}$ and $p \in [1,\infty]$, we always have*

$$d^{-|1/2-1/p|}\kappa_p(A) \le \bar{\kappa}_p(A) \le d^{\max\{1/2,1/p\}}\kappa_p(A).$$

That is, by Lemma 2.5, if $n \gg d$, then the notions of condition number provided by Definition 2.3 and Definition 2.4 are equivalent, up to low-dimensional factors. These low-dimensional factors typically do not matter in theoretical formulations of the problem, but they can matter in practical implementations.

The ℓ_p-norm condition number of a matrix can be arbitrarily large. Given the equivalence established by Lemma 2.5, we say that a matrix A is *well-conditioned in the ℓ_p norm* if κ_p or $\bar{\kappa}_p = \mathcal{O}(\texttt{poly}(d))$, independent of the high dimension n. We see in the following sections that the condition number plays a very important part in the design of RLA algorithms.

2.3 ROUNDING, EMBEDDING AND PRECONDITIONING

Ellipsoidal rounding, low-distortion subspace embedding and preconditioning are three core technical tools underlying RLA algorithms. In this section, we describe in detail how these methods are used. For $p = 2$, a preconditioner R can be computed in $O(nd^2)$ time as the "R" matrix from a QR decomposition, although it is of interest to compute other such preconditioners R that are nearly as good more quickly; and for $p = 1$ and other values of

p, it is of interest to compute a preconditioner R in time that is linear in n and low-degree polynomial in d. In this section, we discuss these and related issues. The algorithms fall into two general families: ellipsoidal rounding (Section 2.3.1) and subspace embedding (Section 2.3.2). We discuss practical tradeoffs in Section 6.1.

Throughout the rest of this section, we assume $n \geq d$. Many algorithms are only meaningful when $n \geq \texttt{poly}(d)$. Here, the degree of $\texttt{poly}(d)$ depends on the underlying algorithm, and may range from $\mathcal{O}(d)$ to $\mathcal{O}(d^7)$.

2.3.1 *Ellipsoidal rounding and fast ellipsoid rounding*

In this subsection, we describe *ellipsoidal rounding* methods. In particular, we are interested in the ellipsoidal rounding of a centrally symmetric convex set and its application to ℓ_p-norm preconditioning. We start with a definition.

Definition 2.6 (Ellipsoidal rounding)**.** *Let $\mathcal{C} \subseteq \mathbb{R}^d$ be a convex set that is full-dimensional, closed, bounded, and centrally symmetric with respect to the origin. An ellipsoid $\mathcal{E}(0, E) = \{x \in \mathbb{R}^d \mid \|Ex\|_2 \leq 1\}$ is a κ-*rounding *of $\mathcal{C}$ if it satisfies $\mathcal{E}/\kappa \subseteq \mathcal{C} \subseteq \mathcal{E}$, for some $\kappa \geq 1$, where $\mathcal{E}/\kappa$ means shrinking $\mathcal{E}$ by a factor of $1/\kappa$.*

The ℓ_p-norm condition number κ_p naturally connects to ellipsoidal rounding. To see this, let $\mathcal{C} = \{x \in \mathbb{R}^d \mid \|Ax\|_p \leq 1\}$ and assume that we have a κ-rounding of $\mathcal{C}$: $\mathcal{E} = \{x \mid \|Rx\|_2 \leq 1\}$. This implies

$$\|Rx\|_2 \leq \|Ax\|_p \leq \kappa \|Rx\|_2, \quad \forall x \in \mathbb{R}^d.$$

If we let $y = Rx$, then we get

$$\|y\|_2 \leq \|AR^{-1}y\|_p \leq \kappa \|y\|_2, \quad \forall y \in \mathbb{R}^d.$$

Therefore, we have $\kappa_p(AR^{-1}) \leq \kappa$. So a κ-rounding of $\mathcal{C}$ leads to a κ-preconditioning of A.

Recall the well-known result due to John [Joh48] that for a centrally symmetric convex set $\mathcal{C}$ there exists a $d^{1/2}$-rounding. It is known that this result is sharp and that such rounding is given by the Löwner-John (LJ) ellipsoid of $\mathcal{C}$, i.e., the minimal-volume ellipsoid containing $\mathcal{C}$. Unfortunately, finding a $d^{1/2}$-rounding is a hard problem. No constant-factor approximation in polynomial time is known for general centrally symmetric convex sets, and hardness results have been shown [Lov86].

To state algorithmic results, suppose that $\mathcal{C}$ is described by a separation oracle and that we are provided an ellipsoid $\mathcal{E}_0$ that gives an L-rounding for some $L \geq 1$. In this case, we can find a $(d(d+1))^{1/2}$-rounding in polynomial time, in particular, in $\mathcal{O}(d^4 \log L)$ calls to the oracle; see Lovasz [Lov86, Theorem 2.4.1]. This algorithmic result was used by Clarkson [Cla05] and then by Dasgupta et al. [Das+09] for ℓ_p regression. Note that, in the works mentioned above, only $\mathcal{O}(d)$-rounding is actually needed, instead of $(d(d+1))^{1/2}$-rounding.

Recent work has focused on constructing ellipsoidal rounding methods that are much faster than these more classical techniques but that lead to only slight degradation in preconditioning quality. See Table 1 for a summary of these results. In particular, Clarkson et al. [Cla+13] followed the same construction as in the proof of Lovasz [Lov86] but show that it is much faster ($\mathcal{O}(d^2 \log L)$ calls to the oracle) to find a (slightly worse) $2d$-rounding

Table 1: Summary of several ellipsoidal rounding for ℓ_p conditioning. Above, the $*$ superscript denotes that the oracles are described and called through a smaller matrix with size n/d by d.

	κ	TIME	# PASSES	# CALLS TO ORACLE
ER [Cla05; Das+09]	$(d(d+1))^{1/2}$	$\mathcal{O}(nd^5 \log n)$	$d^3 \log n$	$\mathcal{O}(d^4 \log n)$
Fast ER [Cla+13]	$2d$	$\mathcal{O}(nd^3 \log n)$	$d \log n$	$\mathcal{O}(d^2 \log n)$
Single-pass ER [MM13b]	$2d^{\lvert 2/p-1\rvert+1}$	$\mathcal{O}(nd^2 \log n)$	1	$\mathcal{O}(d^2 \log n)^*$

of a centrally symmetric convex set in $\mathbb{R}^d$ that is described by a separation oracle. When it is applied to ℓ_p conditioning, the result as summarized below.

Lemma 2.7 (Fast ellipsoidal rounding [Cla+13]). *Given a matrix $A \in \mathbb{R}^{n\times d}$ with full column rank, it takes at most $\mathcal{O}(nd^3 \log n)$ time to find a matrix $R \in \mathbb{R}^{d\times d}$ such that $\kappa_p(AR^{-1}) \leq 2d$.*

Unfortunately, even this improvement for computing a $2d$-conditioning is not immediately applicable to very large matrices. The reason is that such matrices are usually distributively stored on secondary storage and each call to the oracle requires a pass through the data. We could group d calls together within a single pass, but this would still need $\mathcal{O}(d \log n)$ passes. Instead, Meng and Mahoney [MM13b] presented a deterministic single-pass conditioning algorithm that balances the cost-performance trade-off to provide a $2d^{|2/p-1|+1}$-conditioning of A [MM13b]. This algorithm essentially invokes the fast ellipsoidal rounding method (Lemma 2.7) on a smaller problem that is constructed via a single-pass on the original dataset. Their main algorithm is stated in Algorithm 1, and the main result for Algorithm 1 is the following.

Lemma 2.8 (One-pass conditioning [MM13b]). *Algorithm 1 is a $2d^{|2/p-1|+1}$-conditioning algorithm, and it runs in $\mathcal{O}((nd^2 + d^4) \log n)$ time. It needs to compute a $2d$-rounding on a problem of size n/d by d, which needs $\mathcal{O}(d^2 \log n)$ calls to the separation oracle on the smaller problem.*

2.3.2 *Low-distortion subspace embedding and subspace-preserving embedding*

In this subsection, we describe in detail *subspace embedding* methods. Subspace embedding methods were first used in RLA by Drineas, Mahoney, and Muthukrishnan [DMM06] in their relative-error approximation algorithm for ℓ_2 regression; they were first used in a data-oblivious manner in RLA by Sarlós [Sar06]; and an overview of data-oblivious subspace embedding methods as used in RLA has been provided by Woodruff [Woo14]. Before getting into the details of these methods, we first provide some background and definitions.

Let us denote by $\mathcal{A} \subset \mathbb{R}^n$ the subspace spanned by the columns of A. A subspace embedding of $\mathcal{A}$ into $\mathbb{R}^s$ with $s > 0$ is a structure-preserving mapping $\phi : \mathcal{A} \hookrightarrow \mathbb{R}^s$, where the meaning of "structure-preserving" varies depending on the application. Here, we are interested in low-distortion linear embeddings of the normed vector space $\mathcal{A}_p = (\mathcal{A}, \|\cdot\|_p)$, the subspace $\mathcal{A}$ paired with the ℓ_p norm $\|\cdot\|_p$. We start with the following definition.

Algorithm 1 Single-pass conditioning algorithm

1: **Input:** $A \in \mathbb{R}^{n\times d}$ with full column rank and $p \in [1,\infty]$.
2: **Output:** A nonsingular matrix $E \in \mathbb{R}^{d\times d}$ such that

$$\|y\|_2 \le \|AEy\|_p \le 2d^{|2/p-1|+1}\|y\|_2,\ \forall y \in \mathbb{R}^d.$$

3: Partition A along its rows into sub-matrices of size $d^2 \times d$, denoted by $A_1,\dots,A_M$.
4: For each A_i, compute its economy-sized SVD: $A_i = U_i\Sigma_i V_i^T$.
5: Let $\tilde{A}_i = \Sigma_i V_i^T$ for $i = 1,\dots,M$,

$$\tilde{\mathcal{C}} = \left\{ x \in \mathbb{R}^d \;\middle|\; \left(\sum_{i=1}^{M} \|\tilde{A}_i x\|_2^p\right)^{1/p} \le 1 \right\},\ \text{and}\ \tilde{A} = \begin{pmatrix} \tilde{A}_1 \\ \vdots \\ \tilde{A}_M \end{pmatrix}.$$

6: Compute the SVD $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$.
7: Let $\mathcal{E}_0 = \mathcal{E}(0, E_0)$, where $E_0 = d^{\max\{1/p-1/2,0\}}\tilde{V}\tilde{\Sigma}^{-1}$.
8: Compute an ellipsoid $\mathcal{E} = \mathcal{E}(0,E)$ that gives a $2d$-rounding of $\tilde{\mathcal{C}}$ starting from $\mathcal{E}_0$ that gives an $(Md^2)^{|1/p-1/2|}$-rounding of $\tilde{\mathcal{C}}$.
9: **Return** $d^{\min\{1/p-1/2,0\}}E$.

Definition 2.9 (Low-distortion ℓ_p subspace embedding)**.** *Given a matrix $A \in \mathbb{R}^{n\times d}$ and $p \in [1,\infty]$, $\Phi \in \mathbb{R}^{s\times n}$ is an embedding of $\mathcal{A}_p$ if $s = \mathcal{O}(\texttt{poly}(d))$, independent of n, and there exist $\sigma_\Phi > 0$ and $\kappa_\Phi > 0$ such that*

$$\sigma_\Phi \cdot \|y\|_p \le \|\Phi y\|_p \le \kappa_\Phi\sigma_\Phi \cdot \|y\|_p, \quad \forall y \in \mathcal{A}_p. \tag{2.2}$$

We call Φ a low-distortion subspace embedding of $\mathcal{A}_p$ if the distortion of the embedding $\kappa_\Phi = \mathcal{O}(\texttt{poly}(d))$, independent of n.

Given a low-distortion embedding matrix Φ of $\mathcal{A}_p$ with distortion κ_Φ, let R be the "R" matrix from the QR decomposition of ΦA. Then, the matrix AR^{-1} is well-conditioned in the ℓ_p norm. To see this, note that

$$\begin{aligned}\|AR^{-1}x\|_p &\le \|\Phi AR^{-1}x\|_p/\sigma_\Phi \le s^{\max\{0,1/p-1/2\}} \cdot \|\Phi AR^{-1}\|_2 \cdot \|x\|_2/\sigma_\Phi \\ &= s^{\max\{0,1/p-1/2\}} \cdot \|x\|_2/\sigma_\Phi, \quad \forall x \in \mathbb{R}^d,\end{aligned}$$

where the first inequality is due to low distortion and the second inequality is due to the equivalence of vector norms. By similar arguments, we can show that

$$\begin{aligned}\|AR^{-1}x\|_p &\ge \|\Phi AR^{-1}\|_p/(\sigma_\Phi\kappa_\Phi) \ge s^{\min\{0,1/p-1/2\}} \cdot \|\Phi AR^{-1}x\|_2/(\sigma_\Phi\kappa_\Phi) \\ &= \sigma_\Phi s^{\min\{0,1/p-1/2\}} \cdot \|x\|_2/(\sigma_\Phi\kappa_\Phi), \quad \forall x \in \mathbb{R}^d.\end{aligned}$$

By combining these results, we have

$$\kappa_p(AR^{-1}) \le \kappa_\Phi s^{|1/p-1/2|} = \mathcal{O}(\texttt{poly}(d)), \tag{2.3}$$

Table 2: Summary of ℓ_1 and ℓ_2 norm conditioning methods. QR and ER refer to methods based on the QR factorization and Ellipsoid Rounding, as discussed in the text. `QR_small` and `ER_small` denote the running time for applying QR factorization and Ellipsoid Rounding, respectively, on a small matrix with size independent of n.

NORM	NAME	κ_p	RUNNING TIME	TYPE	REFERENCE
ℓ_1	Ellipsoidal Rounding	$(d(d+1))^{1/2}$	$\mathcal{O}(nd^5 \log n)$	ER	[Cla05]
	Fast ER	$2d$	$\mathcal{O}(nd^3 \log n)$	ER	[Cla+13]
	Single-pass ER	$2d^2$	$\mathcal{O}(nd^2 \log n)$	ER	[MM13b]
	Dense Cauchy	$\mathcal{O}(d^{3/2} \log^{3/2} d)$	$\mathcal{O}(nd^2 \log d + d^3 \log d)$	QR	[SW11]
	Fast Cauchy	$\mathcal{O}(d^{9/2} \log^{9/2} d)$	$\mathcal{O}(nd \log d + d^3 \log d)$	QR	[Cla+13]
	Sparse Cauchy	$\mathcal{O}(d^{11/2} \log^{11/2} d)$	$\mathcal{O}(\mathrm{nnz}(A) + d^7 \log^5 d)$	QR	[MM13b]
	SPCT$_2$	$6d$	$\mathcal{O}(\mathrm{nnz}(A) \log n + d^7 \log^5 d) + \mathtt{ER\_small}$	QR+ER	[MM13b]
	SPCT$_3$	$\mathcal{O}(d^{15/4} \log^{11/4} d)$	$\mathcal{O}(\mathrm{nnz}(A) \log n + d^7 \log^5 d) + \mathtt{QR\_small}$	QR+QR	[YMM14]
	Reciprocal exponential	$\mathcal{O}(d^{5/2} \log^{5/2} d)$	$\mathcal{O}(\mathrm{nnz}(A) + d^3 \log d)$	QR	[WZ13]
	Lewis weights	$\mathcal{O}(d^{\frac{1}{2}} \log^{\frac{1}{2}} d)$	$\mathcal{O}(\mathrm{nnz}(A) \cdot \log n + d^3 \log d)$	QR	[CP15]
ℓ_2	Gaussian	$\mathcal{O}(1)$	$\mathcal{O}(nd^2)$	QR	[DS01]
	SRHT	$\mathcal{O}(1)$	$\mathcal{O}(nd \log n + d^3 \log n \log d)$	QR	[Tro11]
	Sparse embedding	$\mathcal{O}(1)$	$\mathcal{O}(\mathrm{nnz}(A) + d^4)$	QR	[CW13a]
	Improved sparse embedding	$\mathcal{O}(1)$	$\mathcal{O}(\mathrm{nnz}(A) \log d + d^3 \log d)$	QR	[Coh16]
	Refinement sampling	$\mathcal{O}(1)$	$\mathcal{O}(\mathrm{nnz}(A) \log(n/d) \log d + d^3 \log(n/d) \log d)$	QR	[Coh+15]

i.e., the matrix AR^{-1} is well-conditioned in the ℓ_p norm. *We call a conditioning method that is obtained via the QR factorization of a low-distortion embedding a QR-type method; and we call a conditioning method that is obtained via an ellipsoid rounding of a low-distortion embedding an ER-type method.*

Furthermore, one can construct a well-conditioned basis by combining QR-like and ER-like methods. To see this, let R be the matrix obtained by applying the fast ellipsoidal rounding method (Lemma 2.7) to ΦA. We have

$$\|AR^{-1}x\|_p \leq \|\Phi AR^{-1}x\|_p/\sigma_\Phi \leq 2d|x\|_2/\sigma_\Phi, \quad \forall x \in \mathbb{R}^d,$$

where the second inequality is due to the ellipsoidal rounding result, and

$$\|AR^{-1}x\|_p \geq \|\Phi AR^{-1}x\|_p/(\sigma_\Phi \kappa_\Phi) \geq \|x\|_2/(\sigma_\Phi \kappa_\Phi), \quad \forall x \in \mathbb{R}^d.$$

Hence

$$\kappa_p(AR^{-1}) \leq 2d\kappa_\Phi = \mathcal{O}(\mathtt{poly}(d))$$

and AR^{-1} is well-conditioned. *Following our previous conventions, we call this combined type of conditioning method a QR+ER-type method.*

In Table 2, we summarize several different types of conditioning methods for ℓ_1 and ℓ_2 conditioning. Comparing the QR-type approach and the ER-type approach to obtaining the preconditioner matrix R, we see there are trade-offs between running times and conditioning quality. Performing the QR decomposition takes $\mathcal{O}(sd^2)$ time [GVL96], which is faster than fast ellipsoidal rounding, which takes $\mathcal{O}(sd^3 \log s)$ time. However, the latter approach might provide a better conditioning quality when $2d < s^{|1/p-1/2|}$. We note that those trade-offs are not important in most theoretical formulations, as long as both take $\mathcal{O}(\mathtt{poly}(d))$ time and provide $\mathcal{O}(\mathtt{poly}(d))$ conditioning, independent of n, but they certainly do affect the performance in practice.

A special family of low-distortion subspace embedding that has very low distortion factor is called subspace-preserving embedding.

Definition 2.10 (Subspace-preserving embedding). *Given a matrix $A \in \mathbb{R}^{n\times d}$, $p \in [1,\infty]$ and $\epsilon \in (0,1)$, $\Phi \in \mathbb{R}^{s\times n}$ is a subspace-preserving embedding of $\mathcal{A}_p$ if $s = \mathcal{O}(\texttt{poly}(d))$, independent of n, and*

$$(1-\epsilon)\cdot\|y\|_p \le \|\Phi y\|_p \le (1+\epsilon)\cdot\|y\|_p, \quad \forall y \in \mathcal{A}_p.$$

Based on the properties of the subspace embedding methods, we present them in the following four categories; namely, data-oblivious subspace embedding methods for ℓ_2 and ℓ_1 norms, respectively; and data-aware subspace embedding methods for ℓ_2 and ℓ_1 norms, respectively.

2.3.2.1 *Data-oblivious low-distortion ℓ_2 subspace embeddings*

An ℓ_2 subspace embedding is distinct from but closely related to the Johnson-Lindenstrauss transform (JLT) provided by the following lemma.

Lemma 2.11 (Johnson-Lindenstrauss lemma [JL84]). *Given $\epsilon \in (0,1)$, a point set $\mathcal{X}$ of N points in $\mathbb{R}^d$, there is a linear map $\phi : \mathbb{R}^d \hookrightarrow \mathbb{R}^s$ with $s = C\log N/\epsilon^2$, where $C > 0$ is a global constant, such that*

$$(1-\epsilon)\|x-y\|^2 \le \|\phi(x)-\phi(y)\|^2 \le (1+\epsilon)\|x-y\|^2, \quad \forall x,y \in \mathcal{X}.$$

We say a mapping has the J-L property if it satisfies the above condition with a constant probability.

Indyk and Motwani [IM98] showed that a matrix whose entries are independent random variables drawn from the standard normal distribution scaled by $s^{-1/2}$ also satisfies the J-L property. Later, Achlioptas [Acho1] showed that the random normal variables can be replaced by random signs, and moreover, we can zero out approximately $2/3$ of the entries with proper scaling, while still maintaining the J-L property. The latter approach allows faster construction and projection with less storage, although still at the same order as the random normal projection. By using an ϵ-net argument and triangle inequality, Sarlós [Saro6] showed that a J-L transform can also aafvectors in $\mathbb{R}^n$, with embedding dimension $\mathcal{O}(d\log(d/\epsilon)/\epsilon^2)$.

It is important to note, however, that for some J-L transforms we are able to obtain more refined results. In particular, these can be obtained by bounding the spectral norm of $(\Phi U)^T(\Phi U) - I$, where U is an orthonormal basis of $\mathcal{A}_2$. If $\|(\Phi U)^T(\Phi U) - I\| \le \epsilon$, for any $x \in \mathcal{A}_2$, we have

$$|\|\Phi x\|_2^2 - \|x\|_2^2| = |z^T((\Phi U)^T(\Phi U) - I)z| \le \epsilon\|z\|_2^2 = \epsilon\|x\|_2^2,$$

where $z = Ux$ with $\|z\|_2 = \|x\|_2$.

We show some results following this approach. First consider the a random normal matrix. Using concentration result on its extreme singular values by Davidson and Szarek [DSo1], we have the following result.

Lemma 2.12. *Given a d-dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^n$ and $\epsilon, \delta \in (0,1)$, let $G \in \mathbb{R}^{s\times n}$ be a random matrix whose entries are independently drawn from the standard normal distribution. There exist $s = \mathcal{O}((\sqrt{d} + \log(1/\delta))^2/\epsilon^2)$ such that, with probability at least $1-\delta$, we have*

$$(1-\epsilon)\|x\|_2 \le \|s^{-1/2}Gx\|_2 \le (1+\epsilon)\|x\|_2, \quad \forall x \in \mathcal{A}_2.$$

Dense J-L transforms, e.g., a random normal projection and its variants, are "slow" in the sense that they use a matrix-vector multiplication for the embedding. Given a matrix

$A \in \mathbb{R}^{n\times d}$, computing $\tilde{A} = \Phi A$ takes $\mathcal{O}(\mathtt{nnz}(A)\cdot s)$ time when Φ is a dense matrix of size $s \times n$ and $\mathtt{nnz}(A)$ is the number of non-zero elements in A. There is also a line of research work on "fast" J-L transforms (FJLT) that started with [AC06; AC09]. These approaches use FFT-like algorithms for the embedding, and thus they lead to $\mathcal{O}(n \log n)$ time for each projection. Hence, computing $\tilde{A} = \Phi A$ takes $\mathcal{O}(nd \log n)$ time when Φ is a fast J-L transform. Their construction was further simplified by Ailon and Liberty [AL09; AL11].

A subsequently-refined FJLT was analyzed by Tropp [Tro11], and it is named the subsampled randomized Hadamard transform (SRHT). An SRHT is a product of three matrices $\Phi = \sqrt{n/s}RHD \in \mathbb{R}^{s\times n}$, where $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix whose entries are independent random signs; $H \in \mathbb{R}^{n\times n}$ is a Walsh-Hadamard matrix scaled by $n^{-1/2}$; and $R \in \mathbb{R}^{s\times n}$ restricts an n-dimensional vector to s coordinates, chosen uniformly at random. As with other FJLT methods, the SRHT preserves the geometry of an entire ℓ_2 subspace of vectors by using a matrix Chernoff inequality to bound $\|(\Phi U)^T(\Phi U) - I\|_2$. Below we present the main results for SRHT from [Dri+12] because it has a better characterization of the subspace-preserving properties. We note that its proof is essentially a combination of the results in [Tro11; Dri+11].

Lemma 2.13 (Subsampled randomized Hadamard transform (SRHT) [Tro11; Dri+11])**.** *Given an d-dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^n$ and $\epsilon, \delta \in (0,1)$, let $\Phi \in \mathbb{R}^{s\times n}$ be a random SRHT with embedding dimension $s \geq \frac{14^2 d \ln(40nd)}{\epsilon^2} \ln\left(\frac{30^2 d \ln(40nd)}{\epsilon^2}\right)$. Then, with probability at least 0.9, we have*

$$(1-\epsilon)\|x\|_2 \leq \|\Phi x\|_2 \leq (1+\epsilon)\|x\|_2, \quad \forall x \in \mathcal{A}_2.$$

An important point to keep in mind (in particular, for parallel and distributed applications) is that, although called "fast," a fast transform might be slower than a dense transform: when $\mathtt{nnz}(A) = \mathcal{O}(n)$ (since machines are optimized for matrix-vector multiplies); when A's columns are distributively stored (since this slows down FFT-like algorithms, due to communication issues); or for other machine-related issues.

More recently, Clarkson and Woodruff [CW13a] developed an algorithm for the ℓ_2 subspace embedding that runs in so-called *input-sparsity* time, i.e., in $\mathcal{O}(\mathtt{nnz}(A))$ time, plus lower-order terms that depend polynomially on the low dimension of the input. Their construction is exactly the CountSketch matrix in the data stream literature [CCFC02], which is an extremely simple and sparse matrix. It can be written as the product of two matrices $\Phi = SD \in \mathbb{R}^{s\times n}$, where $S \in \mathbb{R}^{s\times n}$ has each column chosen independently and uniformly from the s standard basis vectors of $\mathbb{R}^s$ and $D \in \mathbb{R}^{n\times n}$ is a diagonal matrix with diagonal entries chosen independently and uniformly from ± 1. Improved bounds and simpler proofs (that have much more linear algebraic flavor) were subsequently provided by Meng and Mahoney [MM13a] and Nelson and Nguyen [NN13]. Moreover, Cohen [Coh16] considered a more general class of embedding by controlling the sparsity of the embedding matrix and obtained better distortion quality. Below, we present the main results from [CW13a; MM13a; NN13].

Lemma 2.14 (Sparse embedding for ℓ_2 [CW13a; MM13a; NN13])**.** *Given an d-dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^n$ and any $\delta \in (0,1)$, let $s = (d^2+d)/(\epsilon^2\delta)$. Then, with probability at least $1-\delta$,*

$$(1-\epsilon)\|x\|_2 \leq \|\Phi x\|_2 \leq (1+\epsilon)\|x\|_2, \quad \forall x \in \mathcal{A}_2,$$

where $\Phi \in \mathbb{R}^{s\times n}$ is the CountSketch matrix described above.

To summarize, in Table 3 we provide a summary of the basic properties of several data-oblivious ℓ_2 subspace embeddings discussed here (as well as of several data-aware ℓ_2 subspace-preserving embeddings discussed below).

Table 3: Summary of data-oblivious and data-aware ℓ_2 embeddings. Above, s denotes the embedding dimension. By running time, we mean the time needed to compute ΦA. For each method, we set the failure rate to be a constant. Moreover, "Exact lev. scores sampling" means sampling algorithm based on using the exact leverage scores (as importance sampling probabilities); and "Appr. lev. scores sampling" is the sampling algorithm based on approximate leverage scores estimated by using sparse embedding (using the algorithm of [Dri+12]) as the underlying random projection.

NAME	RUNNING TIME	s	κ_Φ	REFERENCE
Gaussian	$\mathcal{O}(nds)$	$\mathcal{O}(d/\epsilon^2)$	$1+\epsilon$	[DS01]
SRHT	$\mathcal{O}(nd\log s)$	$\mathcal{O}(d\log(nd)\log(d/\epsilon^2)/\epsilon^2)$	$1+\epsilon$	[Tro11]
Sparse embedding	$\mathcal{O}(\mathrm{nnz}(A))$	$(d^2+d)/\epsilon^2$	$1+\epsilon$	[CW13a]
Improved sparse embedding	$\mathcal{O}(\mathrm{nnz}(A)\log d)$	$\mathcal{O}(d\log d/\epsilon^2)$	$1+\epsilon$	[Coh16]
Exact lev. scores sampling	$\mathcal{O}(nd^2)$	$\mathcal{O}(d\log d/\epsilon^2)$	$1+\epsilon$	[DMM06]
Appr. lev. scores sampling	$\mathcal{O}(\mathrm{nnz}(A)\log n+d^4)$	$\mathcal{O}(d\log d/\epsilon^2)$	$1+\epsilon$	[CW13a; Dri+12]
Refinement sampling	$\mathcal{O}(\mathrm{nnz}(A)\log(n/d)\log d+d^3\log(n/d)\log d)$	$\mathcal{O}(d\log d)$	$\mathcal{O}(1)$	[Coh+15]

2.3.2.2 *Data-oblivious low-distortion ℓ_1 subspace embeddings*

General ℓ_p subspace embedding and even ℓ_1 subspace embedding is quite different from ℓ_2 subspace embedding. Here, we briefly introduce some existing results on ℓ_1 subspace embedding; for more general ℓ_p subspace embedding, see Meng and Mahoney [MM13a] and Clarkson and Woodruff [CW13b].

Table 4: Summary of data-oblivious and data-aware ℓ_1 embeddings. Above, s denotes the embedding dimension. By running time, we mean the time needed to compute ΦA. For each method, we set the failure rate to be a constant. Moreover, "Lev. scores sampling" denotes the ℓ_1 sampling algorithms obtained by using reciprocal exponential transform as the underlying preconditioning method.

NAME	RUNNING TIME	s	κ_Φ	REFERENCE
Cauchy	$\mathcal{O}(nd^2\log d)$	$\mathcal{O}(d\log d)$	$\mathcal{O}(d\log d)$	[SW11]
Fast Cauchy	$\mathcal{O}(nd\log d)$	$\mathcal{O}(d\log d)$	$\mathcal{O}(d^4\log^4 d)$	[Cla+13]
Sparse Cauchy	$\mathrm{nnz}(A)$	$\mathcal{O}(d^5\log^5 d)$	$\mathcal{O}(d^3\log^3 d)$	[MM13a]
Reciprocal exponential	$\mathrm{nnz}(A)$	$\mathcal{O}(d\log d)$	$\mathcal{O}(d^2\log^2 d)$	[WZ13]
Lev. scores sampling	$\mathcal{O}(\mathrm{nnz}(A)\log n+d^3\log d)$	$\mathcal{O}(d^{9/2}\log^{5/2} d\log(1/\epsilon)/\epsilon^2)$	$1+\epsilon$	[WZ13; Das+09]
Lewis weights	$\mathcal{O}(\mathrm{nnz}(A)\log n)$	$\mathcal{O}(d\log d/\epsilon^2)$	$1+\epsilon$	[CP15]

For ℓ_1, Sohler and Woodruff [SW11] gave the first linear oblivious embedding of a d-dimensional subspace of ℓ_1^n into $\ell_1^{\mathcal{O}(d\log d)}$ with distortion $\mathcal{O}(d\log d)$, where both the embedding dimension and the distortion are independent of n. In particular, they prove the following quality bounds.

Lemma 2.15 (Cauchy transform (CT) [SW11])**.** *Let $\mathcal{A}_1$ be an arbitrary d-dimensional linear subspace of $\mathbb{R}^n$. Then there is an $s_0 = s_0(n) = \mathcal{O}(d\log d)$ and a sufficiently large constant $C_0 > 0$, such that for any s with $s_0 \le s \le d^{\mathcal{O}(1)}$, and any constant $C \ge C_0$, if $\Phi \in \mathbb{R}^{s\times n}$ is a random matrix whose entries are chosen independently from the standard Cauchy distribution and are scaled by C/s, then with probability at least 0.99,*

$$\|x\|_1 \le \|\Phi x\|_1 \le \mathcal{O}(d\log d)\cdot\|x\|_1, \quad \forall x \in \mathcal{A}_1.$$

As CT is the ℓ_1 counterpart of the dense Gaussian transform, the Fast Cauchy Transform (FCT) proposed by Clarkson et al. [Cla+13] is the ℓ_1 counterpart of FJLT. This FCT construction first preprocesses by a deterministic low-coherence matrix, then rescales by Cauchy random variables, and finally samples linear combinations of the rows. Indeed, there are several parameters associated with the construction. Below, we present the theoretical result of FCT by choosing the specific values for those parameters that best balance the computation time and embedding quality.

Lemma 2.16 (Fast Cauchy transform (FCT) [Cla+13])**.** *There is a distribution over matrices $\Phi \in \mathbb{R}^{s\times n}$, with $s = \mathcal{O}(d \log d)$, such that for an arbitrary (but fixed) $A \in \mathbb{R}^{n\times d}$, the inequality*

$$\|x\|_1 \leq \|\Phi x\|_1 \leq \mathcal{O}(d^4 \log^4 d) \cdot \|x\|_1, \quad \forall x \in \mathcal{A}_1$$

holds with high probability. Further, for any $y \in \mathbb{R}^n$, the product Φy can be computed in $\mathcal{O}(n \log s)$ time.

That is, while faster in terms of FLOPS than the CT, the FCT leads to worse embedding/preconditioning quality. Importantly, this result is different from how FJLT compares to dense Gaussian transform: FJLT is faster than the dense Gaussian transform, while both provide the same order of distortion; but FCT becomes faster than the dense Cauchy transform as d grows, at the cost of somewhat worse distortion quality.

Similar to [CW13a; MM13a; NN13] for computing an ℓ_2 subspace embedding, Meng and Mahoney [MM13a] developed an algorithm for computing an ℓ_1 subspace embedding matrix in input-sparsity time, i.e., in $\mathcal{O}(\mathtt{nnz}(A))$ time. They used a CountSketch-like matrix which can be written as the product of two matrices $\Phi = SC \in \mathbb{R}^{s\times n}$, where $S \in \mathbb{R}^{s\times n}$ has each column chosen independently and uniformly from the s standard basis vectors of $\mathbb{R}^s$ and $C \in \mathbb{R}^{n\times n}$ is a diagonal matrix with diagonal entries chosen independently from the standard Cauchy distribution. We summarize the main theoretical results in the following lemma.

Lemma 2.17 (Sparse Cauchy transform (SPCT) [MM13a])**.** *Given an d-dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^n$ and $\epsilon \in (0, 1)$, there is $s = \mathcal{O}(d^5 \log^5 d)$ such that with a constant probability,*

$$1/\,\mathcal{O}(d^2 \log^2 d)\|x\|_1 \leq \|\Phi x\|_1 \leq \mathcal{O}(d \log d)\|x\|_1, \quad \forall x \in \mathcal{A}_1,$$

where Φ is the sparse Cauchy transform described above.

More recently, Woodruff and Zhang [WZ13] proposed another algorithm that computes an ℓ_1 subspace embedding matrix in input-sparsity time. Its construction is similar to that of sparse Cauchy transform. That is, $\Phi = SD$ where D is a diagonal matrix with diagonal entries $1/u_1, 1/u_2, \ldots, 1/u_n$, where u_i are exponential variables. Compared to sparse Cauchy transform, the embedding dimension and embedding quality have been improved. We summarize the main results in the following lemma.

Lemma 2.18 (Reciprocal exponential transform (RET) [WZ13])**.** *Given an d-dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^n$ and $\epsilon \in (0, 1)$, there is $s = \mathcal{O}(d \log d)$ such that with a constant probability,*

$$1/\,\mathcal{O}(d \log d)\|x\|_1 \leq \|\Phi x\|_1 \leq \mathcal{O}(d \log d)\|x\|_1, \quad \forall x \in \mathcal{A}_1,$$

where Φ is the sparse transform using reciprocal exponential variables described above.

To summarize, in Table 4 we provide a summary of the basic properties of several data-oblivious ℓ_1 subspace embeddings discussed here (as well as of several data-aware ℓ_1 subspace-preserving embeddings that will be discussed below).

2.3.2.3 *Data-aware low-distortion ℓ_2 subspace embeddings*

All of the linear subspace embedding algorithms mentioned in previous subsections are oblivious, i.e., independent of the input subspace. That has obvious algorithmic advantages, e.g., one can construct the embedding matrix without even looking at the data. One might wonder if data-aware embeddings could yield better conditioning performance. The answer was given by Drineas, Mahoney, and Muthukrishnan [DMM06] in which they developed a sampling algorithm for solving ℓ_2 regression by constructing a $(1 \pm \epsilon)$-distortion ℓ_2 subspace-preserving sampling matrix. The underlying sampling distribution is defined based on the statistical leverage scores (Definition 2.1) of the design matrix, which can be viewed as the "influence" of that row on the least-squares fit. That is, the sampling distribution is a distribution $\{p_i\}_{i=1}^n$ satisfying

$$p_i \geq \beta \cdot \frac{\ell_i}{\sum_j \ell_j}, \quad i = 1, \ldots, n, \tag{2.4}$$

where $\{\ell_i\}_{i=1}^n$ are the leverage scores of A and $\beta \in (0, 1]$. When $\beta = 1$ and $\beta < 1$, (2.4) implies we define $\{p_i\}_{i=1}^m$ according to the exact and estimated leverage scores, respectively.

More importantly, theoretical results indicate that, given a target desired accuracy, the required sampling complexity is independent of the higher dimension of the matrix. Similar construction of the sampling matrix appeared in several subsequent works, e.g., [DMM06; Dri+11; Dri+12], with improved analysis of the sampling complexity. For completeness, we include the main theoretical result regarding the subspace-preserving quality below, stated here for ℓ_2.

Theorem 2.19 (ℓ_2 subspace-preserving sampling [DMM06; Dri+11; Dri+12])**.** *Given a d-dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^n$ represented by a matrix $A \in \mathbb{R}^{n \times d}$ and $\epsilon \in (0, 1)$, choose $s = \mathcal{O}(d \log d \log(1/\delta)/\beta\epsilon^2)$ and construct a sampling matrix $S \in \mathbb{R}^{n \times n}$ with diagonals*

$$s_{ii} = \begin{cases} 1/\sqrt{q_i} & \text{with probability } q_i, \\ 0 & \text{otherwise,} \end{cases} \quad i = 1, \ldots, n,$$

where

$$q_i \geq \min\{1, s \cdot p_i\}, \quad i = 1, \ldots, n,$$

and $\{p_i\}_{i=1}^n$ satisfies (2.4). Then, with probability at least 0.7,

$$(1 - \epsilon)\|y\|_2 \leq \|Sy\|_2 \leq (1 + \epsilon)\|y\|_2, \quad \forall y \in \mathcal{A}_2.$$

An obvious (but surmountable) challenge to applying this result is that computing the leverage scores *exactly* involves forming an orthonormal basis for A first. Normally, this step will take $\mathcal{O}(nd^2)$ time, which is undesirable for large-scale applications.

On the other hand, the algorithm of [Dri+12], computes the leverage scores *approximately* in essentially the time it takes to perform a random projection. In particular, Drineas et al. [Dri+12] suggested that one can estimate the leverage scores by replacing A with a "similar" matrix in the computation of the pseudo-inverse (which is the main computational bottleneck in the exact computation of the leverage scores). To be more specific, by notic-

ing that the leverage scores can be expressed as the row norms of $AA^\dagger$, we can use ℓ_2 subspace embeddings to estimate them. The high-level idea is,

$$\ell_i = \|e_i^T AA^\dagger\|_2 \approx \|e_i^T A(\Phi_1 A)^\dagger\|_2 \approx \|e_i^T A(\Phi_1 A)^\dagger \Phi_2\|_2 = \tilde{\ell}_i,$$

where e_i is a vector with zeros but 1 in the i-th coordinate, $\Phi_1 \in \mathbb{R}^{r_1 \times n}$ is a FJLT, and $\Phi_2 \in \mathbb{R}^{d \times r_2}$ is a JLT that preserves the ℓ_2 norms of certain set of points. If the estimation of the leverage scores $\tilde{\ell}_i$ satisfies

$$(1-\gamma)\ell_i \leq \tilde{\ell}_i \leq (1+\gamma)\ell_i, \quad i = 1, \ldots, n,$$

it is not hard to show that a sampling distribution $\{p_i\}_{i=1}^n$ defined according to $p_i = \frac{\tilde{\ell}_i}{\sum_j \tilde{\ell}_j}$ satisfies (2.4) with $\beta = \frac{1-\gamma}{1+\gamma}$. When γ is constant, say 0.5, from Theorem 2.19 the required sampling complexity will only need to be increased by a constant factor $1/\beta = 3$. This is less expensive, compared to the gain in the computation cost.

Suppose, we now use sparse embedding (Lemma 2.14) method as the underlying FJLT, i.e., Φ_1, and use Gaussian transform as the underlying JLT, i.e., Φ_2 in the approximation of the leverage scores. Then, combining the theory suggested in [Dri+12] and Theorem 2.19, we have the following lemma.

Lemma 2.20 (Fast ℓ_2 subspace-preserving sampling [Dri+12; CW13a])**.** *Given a d-dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^n$ represented by a matrix $A \in \mathbb{R}^{n \times d}$ and $\epsilon \in (0,1)$, it takes $\mathcal{O}(nnz(A) \cdot \log n)$ time to compute a sampling matrix $S \in \mathbb{R}^{s' \times n}$ (with only one nonzero element per row) with $s' = \mathcal{O}(d \log d/\epsilon^2)$ such that with constant probability*

$$(1-\epsilon)\|y\|_2 \leq \|Sy\|_2 \leq (1+\epsilon)\|y\|_2, \quad \forall y \in \mathcal{A}_2.$$

More recently, Cohen et al. [Coh+15] considered approximating leverage scores in an iterative manner. Given A, starting with a uniform sampling distribution as an initial estimation, the algorithm iteratively refine the estimation based on A', an approximation of A obtained by sampling rows according to the current leverage scores estimation. The result is summarized below.

Lemma 2.21 (Refinement sampling [Coh+15])**.** *Given a d-dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^n$ represented by a matrix $A \in \mathbb{R}^{n \times d}$, it takes $\mathcal{O}(nnz(A)\log(n/d)\log d + d^3 \log(n/d)\log d)$ time to compute a sampling matrix $S \in \mathbb{R}^{s' \times n}$ (with only one nonzero element per row) with $s' = \mathcal{O}(d \log d)$ such that with constant probability*

$$\|y\|_2 \leq \|Sy\|_2 \leq 2\|y\|_2, \quad \forall y \in \mathcal{A}_2.$$

Finally, recall that a summary of both data-oblivious and data-aware subspace embedding for ℓ_2 norm can be found in Table 3.

2.3.2.4 *Data-aware low-distortion ℓ_1 subspace embeddings*

In the same way as we can use data-aware embeddings for ℓ_2 regression, we can use data-aware embeddings for ℓ_1 regression. Indeed, the idea of using data-aware sampling to obtain $(1 \pm \epsilon)$-distortion subspace embeddings for ℓ_1 regression was first used in [Clao5], where it was shown that an ℓ_1 subspace embedding can be done by weighted sampling after preprocessing the matrix, including preconditioning, using ellipsoidal rounding. Sampling probabilities depend on the ℓ_1 norms of the rows of the preconditioned matrix.

Moreover, the resulting sample has each coordinate weighted by the reciprocal of its sampling probability.

Recall that the ℓ_2 leverage scores used in the ℓ_2 sampling algorithm described in Theorem 2.19 are the squared row norms of an orthonormal basis of $\mathcal{A}_2$ that can be viewed as a "nice" basis for the subspace of interest. Dasgupta et al. [Das+09] generalized this method to the general ℓ_p case; in particular, they proposed to sample rows according to the ℓ_p row norms of AR^{-1}, where AR^{-1} is a well-conditioned basis for $\mathcal{A}_p$ (in the ℓ_p sense of well-conditioning). Different from ℓ_1 sampling algorithm [Cla05] described above, computing such a matrix R is usually sufficient, meaning it is not necessary to preprocess A and form the basis AR^{-1} explicitly.

Theorem 2.22 (ℓ_p subspace-preserving sampling [Das+09]). *Given a d-dimensional subspace $\mathcal{A}_p \subset \mathbb{R}^n$ represented by a matrix $A \in \mathbb{R}^{n\times d}$ and a matrix $R \in \mathbb{R}^{d\times d}$ such that AR^{-1} is well-conditioned, $p \in [1,\infty)$, $\epsilon \in (0,1/7)$, and $\delta \in (0,1)$, choose*

$$s \geq 16(2^p+2)\bar{\kappa}_p^p(AR^{-1})(d\log(12/\epsilon)+\log(2/\delta))/(p^2\epsilon^2)$$

and construct a sampling matrix $S \in \mathbb{R}^{n\times n}$ with diagonals

$$s_{ii} = \begin{cases} 1/p_i^{1/p} & \text{with probability } p_i, \\ 0 & \text{otherwise,} \end{cases} \quad i = 1,\dots,n,$$

where

$$p_i \geq \min\left\{1, s\cdot \|A_{(i)}R^{-1}\|_p^p/|AR^{-1}|_p^p\right\}, \quad i = 1,\dots,n.$$

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

$$(1-\epsilon)\|y\|_p \leq \|Sy\|_p \leq (1+\epsilon)\|y\|_p, \quad \forall y \in \mathcal{A}_p.$$

In fact, Theorem 2.22 holds for any choice of R. When $R = I$, it implies sampling according to the ℓ_p row norms of A and the sampling complexity relies on $\bar{\kappa}_p^p(A)$. However, it is worth mentioning that a large condition number for A will lead to a large sampling size, which in turn affects the running time of the subsequent operations. Therefore, preconditioning is typically necessary. One must find a matrix $R \in \mathbb{R}^{d\times d}$ such that $\bar{\kappa}_p(AR^{-1}) = \mathcal{O}(\texttt{poly}(d))$, which could be done by the preconditioning algorithms introduced in the previous sections.

Given R such that AR^{-1} is well-conditioned, computing the row norms of AR^{-1} takes $\mathcal{O}(\texttt{nnz}(A)\cdot d)$ time. Clarkson et al. [Cla+13] improve this running time by estimating the row norms of AR^{-1} instead of computing them exactly. The central idea is to post-multiply a random projection matrix $\Phi_2 \in \mathbb{R}^{n\times r}$ with $r = \mathcal{O}(\log n)$, which takes only $\mathcal{O}(\texttt{nnz}(A)\cdot\log n)$ time.

If one uses the reciprocal exponential transform (Lemma 2.18) to compute a matrix R such that AR^{-1} is well-conditioned and then uses the above idea to estimate quickly the ℓ_1 row norms of AR^{-1} to define the sampling distribution, then by combining with Theorem 2.22, we have the following result.

Lemma 2.23 (Fast ℓ_1 subspace-preserving sampling [WZ13; Das+09]). *Given an d-dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^n$ represented by a matrix $A \in \mathbb{R}^{n\times d}$ and $\epsilon \in (0,1)$, it takes $\mathcal{O}(\texttt{nnz}(A)\log n +$*

$d^3 \log d)$ *time to compute a sampling matrix* $S \in \mathbb{R}^{s' \times n}$ *(with only one nonzero element per row) with* $s' = \mathcal{O}(d^{\frac{9}{2}} \log^{\frac{5}{2}} d \log(1/\epsilon)/\epsilon^2)$ *such that with a constant probability,*

$$(1-\epsilon)\|x\|_1 \leq \|Sx\|_1 \leq (1+\epsilon)\|x\|_1, \quad \forall x \in \mathcal{A}_1.$$

More recently, Cohen and Peng [CP15] provided an iterative algorithm for approximating the"Lewis weights", based on which sampling $\mathcal{O}(d \log d/\epsilon^2)$ rows from A yields an ℓ_1 subspace-preserving embedding with high probability. This algorithm requires $\mathcal{O}(\log \log n)$ calls to computing approximate leverage scores of matrices of the form WA where W is a non-negative diagonal matrix. Result of [CP15] can be found in Table 4.

Finally, recall that a summary of both data-oblivious and data-aware subspace embeddings for ℓ_1 norm can be found in Table 4.

2.4 APPLICATION OF RLA TO MATRIX PROBLEMS

In this section, we discuss the application of randomized linear algebra to several matrix problems. Specifically, in Section 2.4.1 we discuss how a low-precision or a high-precision solution of ℓ_p regression problems can be obtained using RLA. Next in Section 2.4.2 we show how RLA can be used to derive improved algorithms for matrix factorization problems.

2.4.1 ℓ_p *regression*

A parameterized family of linear regression problems that is often of interest is ℓ_p *regression* problem, defined as follows.

Definition 2.24 (ℓ_p regression)**.** *Given a matrix* $A \in \mathbb{R}^{n \times d}$*, a vector* $b \in \mathbb{R}^n$*, and* $p \geq 1$*, the* ℓ_p regression problem *specified by* A*,* b*, and* p *is the following optimization problem:*

$$\text{minimize}_{x \in \mathbb{R}^d} \ \|Ax - b\|_p. \tag{2.5}$$

We call the problem strongly overdetermined *if* $n \gg d$*.*

Important special cases include the ℓ_2 regression problem, also known as least-squares (LS), and the ℓ_1 regression problem, also known as least absolute deviations (LAD) or least absolute errors (LAE). The former is ubiquitous; and the latter is of particular interest as a robust regression technique, in that it is less sensitive to the presence of outliers than the former.

For general $p \in [1, \infty)$, denote by $\mathcal{X}^*$ the set of optimal solutions to (2.5). Let $x^* \in \mathcal{X}^*$ be an arbitrary optimal solution, and let $f^* = \|Ax^* - b\|_p$ be the optimal objective value. We are particularly interested in finding a *relative-error approximation*, in terms of the objective value, to the general ℓ_p regression problem (2.5).

Definition 2.25 (Relative-error approximation)**.** *Given an error parameter* $\epsilon > 0$*,* $\hat{x} \in \mathbb{R}^d$ *is a* $(1+\epsilon)$-approximate solution *to the* ℓ_p *regression problem* (2.5) *if and only if*

$$\hat{f} = \|A\hat{x} - b\|_p \leq (1+\epsilon) f^*.$$

In order to make our theory and our algorithms for general ℓ_p regression simpler and more concise, we can use an equivalent formulation of (2.5) in our discussion:

$$\begin{array}{ll} \text{minimize}_{x\in\mathbb{R}^d} & \|\bar{A}x\|_p \\ \text{subject to} & c^T x = 1 \end{array} \tag{2.6}$$

where $\bar{A} = \begin{pmatrix} A & -b \end{pmatrix}$ and c is the last column of I_{n+1}. This formulation of ℓ_p regression, which consists of a homogeneous objective and an affine constraint, is equivalent to the formulation of (2.5).

In the following we describe how the ellipsoidal rounding and subspace embedding methods described in the previous subsections can be applied to solve ℓ_2 and ℓ_1 regression problems. We elaborate on how we can use the methods described previously to construct low-precision solvers and high-precision solvers for solving ℓ_p regression problems. As a reference, see Table 5 and Table 6 for a summary of several representative RLA algorithms for solving ℓ_2 and ℓ_1 regression problems, respectively.

Table 5: Summary of RLA-based ℓ_2 regression solvers; PC stands for preconditioning.

TYPE	PRECISION	EXAMPLE	REFERENCE
embedding + solving subproblem	low	CW + (FJLT+SVD)	[CW13a]
		appr. lev. samp. (SRHT) + SVD	[Dri+12]
direct solver	high	SVD or QR	[GVL96]
PC + direct solver	high	PC (Gaussian) + normal equation	[CRT11]
PC + iterative alg.	high	PC (FJLT) + LSQR	[AMT10; RT08]

Table 6: Summary of RLA-based ℓ_1 regression solvers; PC stands for preconditioning.

TYPE	PRECISION	EXAMPLE	REFERENCE
(PC + sampling) + solving subproblem	low	(ER/fast ER + sampling) + IPM	[Cla05; Das+09]
		(SCT/FCT + sampling) + IPM	[SW11; Cla+13]
second-order	high	IPM	[NN94]
PC + first-order	high	ER + accelerated gradient descent	[Nes08]

2.4.1.1 *Low-precision solvers*

The most straightforward use of these methods (and the one for which most of the theory has been developed) is to construct a subspace-preserving embedding matrix and then solve the resulting reduced-sized problem exactly, thereby obtaining an approximate solution to the original problem. In somewhat more detail, this algorithmic approach performs the following two steps.

1. Construct a subspace-preserving embedding matrix Φ with distortion $1 \pm \frac{\epsilon}{4}$.
2. Using a black-box solver, solve the reduced-sized problem exactly:

$$\hat{x} = \min_{x\in\mathbb{R}^d} \|\Phi Ax - \Phi b\|_p.$$

(We refer to this approach as *low-precision* because the running time complexity with respect to the error parameter ϵ is $\text{poly}(1/\epsilon)$. Thus, while the approach can be analyzed for

a fixed ϵ, this dependence means that as a practical matter the approach cannot achieve high-precision solutions.)

To see why this approach gives us a $(1+\epsilon)$-approximate solution to the original problem, recall that a subspace-preserving embedding matrix Φ with distortion factor $(1 \pm \frac{\epsilon}{4})$ satisfies

$$(1-\epsilon/4) \cdot \|\bar{A}x\|_p \leq \|\Phi\bar{A}x\|_p \leq (1+\epsilon/4) \cdot \|\bar{A}x\|_p, \quad \forall x \in \mathbb{R}^d.$$

Therefore, the following simple reasoning shows that $\hat{x}$ is indeed a $(1+\epsilon)$-approximation solution:

$$\|\bar{A}\hat{x}\|_p \leq \frac{1}{1-\epsilon/4}\|\Phi\bar{A}\hat{x}\|_p \leq \frac{1}{1-\epsilon/4}\|\Phi\bar{A}x^*\|_p \leq \frac{1+\epsilon/4}{1-\epsilon/4}\|\bar{A}x^*\|_p < (1+\epsilon)\|\bar{A}x^*\|_p.$$

The following lemma states this result more precisely.

Lemma 2.26. *Given an ℓ_p regression problem specified by $A \in \mathbb{R}^{n\times d}$, $b \in \mathbb{R}^n$ and $p \in [1,\infty)$ using the constrained formulation (2.6), let Φ be a $(1 \pm \epsilon/4)$-distortion embedding of $\bar{A}_p$, and $\hat{x}$ be an optimal solution to the reduced-sized problem $\min_{c^T x=1} \|\Phi Ax\|_p$. Then $\hat{x}$ is a $(1+\epsilon)$-approximate solution to the original problem.*

A great deal of work has followed this general approach; see, e.g., [Mah11] and references therein. Here we simply cite several of the most immediately relevant for our subsequent discussion.

- Sampling for ℓ_2 regression. One could use the original algorithm of [DMM06; DMM08], which performs a data-aware random sampling and solves the subproblem in $\mathcal{O}(nd^2)$ time to obtain an approximate solution. With the algorithm of [Dri+12], the running time of this method was improved to roughly $\mathcal{O}(nd\log(d))$ time, and by combining the algorithm of [Dri+12] with the algorithm of [CW13a], the running time was still further improved to input-sparsity time.

- Projections for ℓ_2 regression. Alternatively, one could use the algorithm of [Sar06; Dri+11], which performs a data-oblivious Hadamard-based random projection (SRHT) and solves the subproblem in roughly $\mathcal{O}(nd\log(d))$ time, or one could use the algorithm of [CW13a], which runs in input-sparsity time. We remark with an improved analysis, Drineas et al. [Dri+11] showed that the required embedding dimension needed for SRHT to obtain a low-precision solution for least-squares problems depends on $\mathcal{O}(1/\epsilon)$ only as opposed to $\mathcal{O}(1/\epsilon^2)$ in obtaining a subspace-preserving embedding (Lemma 2.13).

- Sampling and projections for ℓ_1 and ℓ_p regression: see [Cla05; SW11; Cla+13] and also [Das+09; MM13a; CW13b] and references therein for both data-oblivious and data-aware methods.

To summarize these and other results, depending on whether the idealization that $n \gg d$ holds, either the Hadamard-based projections for ℓ_2 regression (e.g., the projection algorithm of [Dri+11] or the sampling algorithm of [DMM06] combined with the algorithm of [Dri+12]) and ℓ_1 regression (e.g., the algorithm of [Cla+13]) or the input-sparsity time algorithms for ℓ_1 regression (e.g., the algorithms of [CW13b] and [MM13a]) lead to the best worst-case asymptotic performance. There are, however, practical tradeoffs, both in RAM and in parallel-distributed environments, and the most appropriate method to use in any particular situation is still a matter of ongoing research.

2.4.1.2 *High-precision solvers*

A more refined use of these methods (and the one that has been used most in implementations) is to construct a subspace-preserving embedding matrix and use that to construct a preconditioner for the original ℓ_p regression problem, thereby obtaining an approximate solution to the original problem. In somewhat more detail, this algorithmic approach performs the following two steps:

1. Construct a randomized preconditioner for A, called N.
2. Invoke an iterative algorithm whose convergence rate depends on the condition number of the problem being solved (a linear system for ℓ_2 regression, and a linear or convex program for ℓ_1 regression) on the preconditioned system AN.

(We refer to this approach as *high-precision* because the running time complexity with respect to the error parameter ϵ is $\log(1/\epsilon)$. Among other things, this means that, given a moderately good solution—e.g., the one obtained from the embedding that could be used in a low-precision solver—one can easily obtain a very high precision solution.)

Most of the work for high-precision RLA solvers for ℓ_p regression has been for ℓ_2 regression (although we mention a few solvers for ℓ_1 regression for completeness and comparison).

ℓ_2 REGRESSION Recall that theoretical (and empirical) results suggest that the required number of iterations in many iterative solvers such as LSQR [PS82] depends strongly on the condition number of the system. Thus, a natural idea is first to compute a randomized preconditioner and then to apply one of these iterative solvers on the preconditioned system. For example, if we use SRHT (Lemma 2.13) to create a preconditioned system with condition number bounded by a small constant and then use LSQR to solve the preconditioned problem iteratively, the total running time would be $\mathcal{O}(nd\log(n/\epsilon) + d^3\log d)$, where $\mathcal{O}(nd\log(n))$ comes from SRHT, $\mathcal{O}(d^3\log d)$ from computing the preconditioner matrix, and $\mathcal{O}(nd\log(1/\epsilon))$ from LSQR iterations. Avron, Maymounkov, and Toledo [AMT10] and Rokhlin and Tygert [RT08] developed algorithms that use FJLT for preconditioning and LSQR as an iterative solver. In [MSM14], the authors developed a randomized solver LSRN for ℓ_2 regression using Gaussian transform and LSQR or the Chebyshev semi-iterative method.

As with the low-precision solvers, note that if we use the input-sparsity time algorithm of [CW13a] for embedding and then an (SRHT + LSQR) approach above to solve the reduced-sized problem, then under the assumption that $n \geq \texttt{poly}(d)$ and ϵ is fixed, this particular combination would become the best approach proposed. However, there are various trade-offs among those approaches. For instance, there are trade-offs between running time and preconditioning quality for computing the subspace-preserving sampling matrix, and there are trade-offs between embedding dimension/sample size and failure rate in embedding/sampling. Some of the practical trade-offs on different problem types and computing platforms are discussed in Section 6.1 below.

ℓ_1 REGRESSION While most of the work in RLA for high-precision solvers has been for ℓ_2 regression, we should point out related work for ℓ_1 regression. In particular, Nesterov [Nes08] proposed an algorithm that employs a combination of ellipsoid rounding and accelerated gradient descent; and second-order methods from [NN94]

use interior point techniques more generally; Meng and Mahoney [MM13b] coupled these ideas with RLA ideas to develop an iterative medium-precision algorithm for ℓ_1 regression. More recently, Yang et al. [Yan+16a] proposed a weighted stochastic gradient descent algorithm for ℓ_1 regression with randomized preconditioning; see Chapter 4 for more details.

2.4.2 *Low-rank matrix decompositions*

Given an $m \times n$ data matrix A,[1] low-rank matrix factorization methods aim to find two smaller matrices whose product is a good approximation to A. That is, they aim to find matrices Y and Z such that

$$\underset{m\times n}{A} \approx \underset{m\times k}{Y} \times \underset{k\times n}{Z} \tag{2.7}$$

is a rank-k approximation to the original matrix A. Low-rank matrix factorization is an important topic in linear algebra and numerical analysis, and it finds use in a variety of scientific fields and scientific computing including machine learning and data analysis applications such as pattern recognition and personalized recommendation.

Depending on the application, various low-rank factorization techniques are of interest. Popular choices include the singular value decomposition [GVL96], principal component analysis [Jol86], rank-revealing QR factorization [GE96], nonnegative matrix factorization [LS01], and CUR/CX decompositions [MD09]. Here, we are interested in using the SVD and CX decompositions for scalable and interpretable data analysis. In the remainder of this section, we briefly describe these decompositions and discuss the usage of RLA in improving these algorithms.

2.4.2.1 *SVD and PCA*

The singular value decomposition (SVD) is the factorization of $A \in \mathbb{R}^{m\times n}$ into the product of three matrices $U\Sigma V^T$ where $U \in \mathbb{R}^{m\times r}$ and $V \in \mathbb{R}^{n\times r}$ have orthonormal columns, $\Sigma \in \mathbb{R}^{r\times r}$ is a diagonal matrix with positive real entries, and $r = \operatorname{rank}(A) \leq \min\{m, n\}$. The columns of U and V are called left and right singular vectors and the diagonal entries of Σ are called singular values. For notation convenience, we assume the singular values are sorted such that $\sigma_1 \geq \cdots \geq \sigma_r > 0$, and this means that the columns of U and V are sorted by the order given by the singular values.

The SVD is of central interest because it provides the best low-rank matrix approximation with respect to any unitarily invariant matrix norm. In particular, for any target rank $k \leq r$, the SVD provides the minimizer of the optimization problem

$$\min_{\operatorname{rank}(\tilde{A})=k} \|A - \tilde{A}\|_F, \tag{2.8}$$

where the Frobenius norm $\|\cdot\|_F$ is defined as $\|X\|_F^2 = \sum_{i=1}^m \sum_{j=1}^n X_{ij}^2$. Specifically, the solution to (2.8) is given by the truncated SVD, i.e., $A_k = U_k \Sigma_k V_k^T$, where the columns of U_k and V_k are the top k singular vectors, i.e., the first k columns of U and V, and Σ_k is a diagonal matrix containing the top-k singular values.

Principal component analysis (PCA) and SVD are closely related. PCA aims to convert the original features into a set of orthogonal directions called *principal components* that capture most of the variance in the data points. The PCA decomposition of A is given by the

1 Here when talking about matrix factorization, we assume A is $m \times n$ rather than $n \times d$.

Algorithm 2 RANDOMIZEDSVD

1: **Input:** $A \in \mathbb{R}^{m \times n}$, number of iterations $q \geq 1$, target rank $k > 0$, slack $\ell \geq 0$, and let $r = k + \ell$.
2: **Output:** $U\Sigma V^T \approx \text{THINSVD}(A, r)$.
3: Initialize $B \in \mathbb{R}^{n \times r}$ by sampling $B_{ij} \sim \mathcal{N}(0, 1)$.
4: **for** q times **do**
5: $\quad B \leftarrow \text{MULTIPLYGRAMIAN}(A, B)$
6: $\quad (B, \_) \leftarrow \text{THINQR}(B)$
7: **end for**
8: Let Q be the first k columns of B.
9: Let $C = AQ$.
10: Compute $(U, \Sigma, \tilde{V}^T) = \text{THINSVD}(C)$.
11: Let $V = Q\tilde{V}$.
12: **Return** U, Σ, V.

SVD of the matrix formed by centering each column of A (i.e., removing the mean of each column). When low-rank methods are appropriate, the number of principal components needed to preserve most of the information in A is far less than the number of original features, and thus the goal of dimension reduction is achieved.

However, the computation of the SVD (and thus of PCA for a data matrix A) is expensive [GVL96]. For example, to compute the truncated SVD with rank k using traditional deterministic methods, the running time complexity is $\mathcal{O}(mnk)$, and $\mathcal{O}(k)$ passes over the dataset are needed. This becomes prohibitively expensive with datasets of even moderately-large size, e.g., $m = 10^6$, $n = 10^4$, and $k = 20$. To address these and related issues, recent work in RLA has focused on using randomized approximation to perform scalable linear algebra computations for large-scale data problems. For a review of RLA methods for low-rank matrix approximation, see [HMT11; Mar16].

Here, we describe an algorithm introduced in [MRT11] that uses a random Gaussian projection to construct a rank-k approximation to A that approximates A nearly as well as A_k does. Importantly, the algorithm runs in $\mathcal{O}(mn \log k)$ time, and the algorithm needs only a constant number of passes over the data matrix. These properties becomes extremely desirable in many large-scale data analytics. This algorithm, which we refer to as RANDOMIZEDSVD, is summarized in Algorithm 2. (Algorithm 2 calls MULTIPLYGRAMIAN, which is summarized in Algorithm 3, as well as two algorithms, THINQR and THINSVD, which are standard in numerical linear algebra [GVL96].) The running-time cost for RANDOMIZEDSVD is dominated by the matrix-matrix multiplication in Step 5 and Step 9 of Algorithm 2, which involves passing over the entire data matrix. These steps can be parallelized, and hence RANDOMIZEDSVD is amenable to distributed computing. We refer to [HMT11; Mar16] for more details including theoretical guarantees. It is worth mentioning that a variant of this algorithm that based on sub-Gaussian random matrices is also available [AA16].

2.4.2.2 *CX/CUR decompositions*

In addition to developing improved algorithms for PCA/SVD and related problems, work in RLA has focused on so-called CX/CUR decompositions [DMM08; MD09]. As a motivation, observe that singular vectors are eigenvectors of the Gram matrix $A^T A$, and thus

Algorithm 3 MULTIPLYGRAMIAN

Input: $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times k}$.
Output: $X = A^T AB$.
Initialize $X = 0$.
for each row a in A **do**
    $X \leftarrow X + aa^T B$.
end for
Return X.

they are linear combinations of up to all of the original features. A natural question arises: can we reconstruct the matrix using a small number of actual columns of A?

CX/CUR decompositions affirmatively answer this question. That is, these are low-rank matrix decompositions that are expressed in terms of a small number of actual columns/rows. As such, they have found applicability in scientific applications where coupling analytical techniques with domain knowledge is at a premium, including genetics [Pas+07], astronomy [Yip+14], and mass spectrometry imaging [Yan+15].

In more detail, CX decomposition factorizes an $m \times n$ matrix A into two matrices C and X, where C is an $m \times c$ matrix that consists of c actual columns of A, and X is a $c \times n$ matrix such that $A \approx CX$. (CUR decompositions further choose $X = UR$, where R is a small number of actual rows of A [DMM08; MD09].) For CX, using the same optimality criterion defined in (2.8), we seek matrices C and X such that the residual error $\|A - CX\|_F$ is small.

The algorithm of [DMM08] computes a $1 \pm \epsilon$ relative-error low-rank CX matrix approximation and consists of three basic steps:

- Compute (exactly or approximately) the *statistical leverage scores* of the columns of A associated with rank k;
- Use those scores as a sampling distribution to select c columns from A and form C;
- Compute the optimal matrix X with rank-k that minimizes $\|A - CX\|_F$; see [DMM08] for detailed construction.

Below, we give more details regarding the motivation of choosing such an importance sampling distribution $\{p_i\}_{i=1}^n$ based on statistical leverage scores. We shall point out that the statistical leverage scores used here are defined based on a low-dimensional subspace, different from the ones define in Definition 2.1.

Let $A = U\Sigma V^T$ be the SVD of A. Given a target rank parameter k, for $i = 1, \ldots, n$, the i-th leverage score is defined as

$$\ell_i = \sum_{j=1}^{k} V_{ij}^2. \tag{2.9}$$

These scores quantify the amount of "leverage" each column of A exerts on the best rank-k approximation to A. For each column of A, we have

$$A^{(i)} = \sum_{j=1}^{r} (\sigma_j U^{(j)}) V_{ij} \approx \sum_{j=1}^{k} (\sigma_j U^{(j)}) V_{ij}.$$

Algorithm 4 CXDECOMPOSITION

1: **Input:** $A \in \mathbb{R}^{m \times n}$, rank parameter $k \leq \text{rank}(A)$, number of power iterations q.
2: **Output:** C.
3: Compute an approximation of the top-k right singular vectors of A denoted by $\tilde{V}_k$, using RANDOMIZEDSVD with q power iterations.
4: Let $\ell_i = \sum_{j=1}^{k} \tilde{V}_{ij}^2$, where $\tilde{V}_{ij}^2$ is the (i,j)-th element of $\tilde{V}_k$, for $i = 1, \ldots, n$.
5: Define $p_i = \ell_i / \sum_{j=1}^{n} \ell_j$ for $i = 1, \ldots, n$.
6: Randomly sample c columns from A in i.i.d. trials, using the importance sampling distribution $\{p_i\}_{i=1}^{n}$, to form C.
7: Compute the optimal matrix X with rank-k that minimizes $\|A - CX\|_F$; see [DMM08] for detailed construction.
8: **Return** C and X.

That is, the i-th column of A can be expressed as a linear combination of the basis of the dominant k-dimensional left singular space with v_{ij} as the coefficients. For $i = 1, \ldots, n$, we define the *normalized leverage scores* as

$$p_i = \frac{\ell_i}{\sum_{j=1}^{n} \ell_j}, \tag{2.10}$$

where ℓ_i is defined in (2.9), and choose columns from A according to those normalized leverage scores. It has been shown by Drineas, Mahoney, and Muthukrishnan [DMM08] that the selected columns are able to reconstruct the matrix A nearly as well as A_k does. To be more specific, if sampling size $c = \mathcal{O}(k \log k / \epsilon^2)$, then with probability at least 0.99, the matrices C and X will satisfy

$$\|A - CX\|_F \leq (1 + \epsilon)\|A - A_k\|_F, \tag{2.11}$$

where A_k is the best rank-k approximation to A.

As can be seen above, the running time for CXDECOMPOSITION is determined by the computation of the importance sampling distribution. To compute the leverage scores based on (2.9), one needs to compute the top k right-singular vectors V_k. This can be prohibitive on large matrices. However, we can use RANDOMIZEDSVD to compute *approximate* leverage scores. This approach, originally proposed by Drineas et al. [Dri+12], runs in "random projection time," so requires fewer FLOPS and fewer passes over the data matrix than deterministic algorithms that compute the leverage scores exactly. In Algorithm 4, we summarize the steps of the algorithm described above, which is a variant of the algorithm of [DMM08] that uses approximate leverage scores provided by RANDOMIZEDSVD.

Finally, results of implementing RANDOMIZEDSVD and CXDECOMPOSITION on terabyte-sized real datasets on clusters with up to 1600 nodes are presented in Section 6.3. Results of applying CX decompositions to bioimaging for more interpretable analysis are presented in Chapter 7.

3 FAST ALGORITHMS FOR LARGE-SCALE QUANTILE REGRESSION

Quantile regression is a method to estimate the quantiles of the conditional distribution of a response variable, expressed as functions of observed covariates [KB78], in a manner analogous to the way in which least-squares regression estimates the conditional mean. The least absolute deviations regression (i.e., ℓ_1 regression) is a special case of quantile regression that involves computing the median of the conditional distribution. In contrast with ℓ_1 regression and the more popular ℓ_2 or least-squares regression, quantile regression involves minimizing asymmetrically-weighted absolute residuals. Doing so, however, permits a much more accurate portrayal of the relationship between the response variable and observed covariates, and it is more appropriate in certain non-Gaussian settings. For these reasons, quantile regression has found applications in many areas, e.g., survival analysis and economics [Buc94; KH01; Buh05]. As with ℓ_1 regression, the quantile regression problem can be formulated as a linear programming problem, and thus simplex or interior-point methods can be applied [KD93; PK97; Por97]. Most of these methods are efficient only for problems of small to moderate size, and thus to solve very large-scale quantile regression problems more reliably and efficiently, we need new computational techniques.

In this chapter, we provide a fast algorithm to compute a $(1+\epsilon)$ relative-error approximate solution to the over-constrained quantile regression problem using randomized linear algebra. Our algorithm constructs a subspace-preserving embedding in terms of the loss of quantile regression, and runs in time that is nearly linear in the number of nonzeros in the input data. In Section 3.1 we provide background on quantile regression. The main algorithm and theoretical results are presented in Section 3.2. Finally, empirical results on medium-scale and large-scale quantile regression problems are provided in Section 3.3 and Section 6.2, respectively. The material in this chapter appears in Yang, Meng, and Mahoney [YMM13; YMM14].

3.1 BACKGROUND ON QUANTILE REGRESSION

Recall that a quantile regression problem can be specified by a (design) matrix $A \in \mathbb{R}^{n\times d}$, a (response) vector $b \in \mathbb{R}^n$, and a parameter $\tau \in (0,1)$, in which case the quantile regression problem can be solved via the optimization problem

$$\text{minimize}_{x\in\mathbb{R}^d} \quad \rho_\tau(b - Ax), \tag{3.1}$$

where $\rho_\tau(y) = \sum_{i=1}^d \rho_\tau(y_i)$, for $y \in \mathbb{R}^d$, where

$$\rho_\tau(z) = \begin{cases} \tau z, & z \geq 0; \\ (\tau - 1)z, & z < 0, \end{cases} \tag{3.2}$$

for $z \in \mathbb{R}$, is the corresponding loss function. In the remainder of this paper, we will use A to denote the augmented matrix $\begin{pmatrix} b & -A \end{pmatrix}$, and we will consider $A \in \mathbb{R}^{n\times d}$. With

Algorithm 5 SPCT2: QR + ER type method with sparse Cauchy transform

1: **Input:** $A \in \mathbb{R}^{n\times d}$ with full column rank.
2: **Output:** $R \in \mathbb{R}^{d\times d}$ such that AR^{-1} is a well-conditioned basis with $\bar{\kappa}_1 \leq 6d^2$.
3: Construct a low-distortion embedding matrix $\Pi_1 \in \mathbb{R}^{r_1\times n}$ of $(\mathcal{A}, \|\cdot\|_1)$ via SPCT (Lemma 2.17).
4: Construct $\tilde{R} \in \mathbb{R}^{d\times d}$ such that $A\tilde{R}^{-1}$ is a well-conditioned basis for $\mathcal{A}$ via QR factorization of $\Pi_1 A$.
5: Compute a $(1 \pm 1/2)$-distortion sampling matrix $\tilde{S} \in \mathbb{R}^{\mathtt{poly}(d)\times n}$ of $(\mathcal{A}, \|\cdot\|_1)$ via Theorem 2.22.
6: Compute $R \in \mathbb{R}^{d\times d}$ by ellipsoid rounding for $\tilde{S}A$ via Lemma 2.7.
7: **Return** R.

this notation, the quantile regression problem of (3.1) can equivalently be expressed as a constrained optimization problem with a single linear constraint,

$$\text{minimize}_{x\in\mathcal{C}} \quad \rho_\tau(Ax), \tag{3.3}$$

where $\mathcal{C} = \{x \in \mathbb{R}^d \mid c^T x = 1\}$ and c is a unit vector with the first coordinate set to be 1. We will focus on very over-constrained problems with size $n \gg d$.

For simplicity, we assume A has full column rank; and we always assume that $\tau \geq \frac{1}{2}$. All the results hold for $\tau < \frac{1}{2}$ by simply switching the positions of τ and $1-\tau$.

3.2 MAIN ALGORITHM AND THEORETICAL RESULTS

In this section, we first give an overview of nearly input-sparsity time ℓ_1 conditioning methods in which we propose a new hybrid scheme and then we present our main theoretical results on $(1\pm\epsilon)$-distortion subspace-preserving embeddings and our fast randomized algorithm for quantile regression. Throughout the rest of this chapter, we assume that $n \geq \mathtt{poly}(d)$, where the degree of $\mathtt{poly}(d)$ depends on the terms in d in the running time computation of SPCT, SPCT2, and SPCT3 (see below). Proofs of all the results can be found in Appendix A.

3.2.1 *Nearly input-sparsity time ℓ_1 conditioning methods*

Recall that Section 2.3 provides a summary of ℓ_1 conditioning methods. In this chapter we will make use of the conditioning methods based on the sparse Cauchy transform (SPCT) [MM13a] described in Lemma 2.17. We name them as SPCT, SPC2 and SPC3 where SPCT, SPCT2 are proposed in [MM13a] and SPCT3 is new. According to the discussion in Section 2.3.2, SPCT is a QR method, SPCT2 is a QR+ER method and SPCT3 is a QR+QR method, which is presented in Algorithm 6. The main result for Algorithm 6 is given in Lemma 3.2. For completeness, we present SPCT2 in Algorithm 5 and state its main result in Lemma 3.1.

Lemma 3.1 ([MM13a]). *Given $A \in \mathbb{R}^{n\times d}$ with full rank, Algorithm 5 takes $\mathcal{O}(nnz(A)\cdot\log n)$ time to compute a matrix $R \in \mathbb{R}^{d\times d}$ such that with a constant probability, AR^{-1} is a well-conditioned basis for $\mathcal{A}$ with $\bar{\kappa}_1 \leq 6d^2$.*

Algorithm 6 SPCT3: QR + QR type method with sparse Cauchy transform

1: **Input:** $A \in \mathbb{R}^{n\times d}$ with full column rank.
2: **Output:** $R \in \mathbb{R}^{d\times d}$ such that AR^{-1} is a well-conditioned basis with $\bar{\kappa}_1 \leq \mathcal{O}(d^{\frac{19}{4}} \log^{\frac{11}{4}} d)$.
3: Construct a low-distortion embedding matrix $\Pi_1 \in \mathbb{R}^{r_1\times n}$ of $(\mathcal{A}, \|\cdot\|_1)$ via SPCT (Lemma 2.17).
4: Construct $\tilde{R} \in \mathbb{R}^{d\times d}$ such that $A\tilde{R}^{-1}$ is a well-conditioned basis for $\mathcal{A}$ via QR factorization of $\Pi_1 A$.
5: Compute a $(1 \pm 1/2)$-distortion sampling matrix $\tilde{S} \in \mathbb{R}^{\mathtt{poly}(d)\times n}$ of $(\mathcal{A}, \|\cdot\|_1)$ via Theorem 2.22.
6: Compute $R \in \mathbb{R}^{d\times d}$ via the QR factorization of $\tilde{S}A$.
7: **Return** R.

Lemma 3.2. *Given $A \in \mathbb{R}^{n\times d}$ with full rank, Algorithm 6 takes $\mathcal{O}(\mathtt{nnz}(A) \cdot \log n)$ time to compute a matrix $R \in \mathbb{R}^{d\times d}$ such that with a constant probability, AR^{-1} is a well-conditioned basis for $\mathcal{A}$ with $\bar{\kappa}_1 \leq \mathcal{O}(d^{\frac{19}{4}} \log^{\frac{11}{4}} d)$.*

Both SPCT2 and SPCT3 have additional steps (Steps 3 & 4), when compared with SPCT, and this leads to some improvements, at the cost of additional computation time. For example, in Algorithm 6 (SPCT3), we obtain a well-conditioned basis with smaller κ when comparing to SPCT. As for the running time, it will be still $\mathcal{O}(\mathtt{nnz}(A) \cdot \log n)$, since the additional time is for constructing sampling matrix and solving a QR factorization of a matrix whose dimensions are determined by d. Note that when we summarize these results in Table 7, we explicitly list the additional running time for SPCT2 and SPCT3, in order to show the tradeoff between these SPCT-derived methods.

Table 7: Summary of running time, condition number, and type of three sparse Cauchy transform based conditioning methods. QR and ER refer, respectively, to methods based on the QR factorization and methods based on Ellipsoid Rounding, as discussed in the text. `QR_small` and `ER_small` denote the running time for applying QR factorization and Ellipsoid Rounding, respectively, on a small matrix with size independent of n.

NAME	RUNNING TIME	$\bar{\kappa}_1$	TYPE
SPCT [MM13a]	$\mathcal{O}(\mathrm{nnz}(A))$	$\mathcal{O}(d^{\frac{13}{2}} \log^{\frac{11}{2}} d)$	QR
SPCT2 [MM13a]	$\mathcal{O}(\mathrm{nnz}(A) \cdot \log n)$ + `ER_small`	$6d^2$	QR+ER
SPCT3 (proposed in this work)	$\mathcal{O}(\mathrm{nnz}(A) \cdot \log n)$ + `QR_small`	$\mathcal{O}(d^{\frac{19}{4}} \log^{\frac{11}{4}} d)$	QR+QR

Although our new algorithm, SPCT3, is *not* uniformly better than either of the two previous algorithms with respect to either condition number or the running time, it is only slightly worse than the better of the two previous algorithms with respect to each of those two measures. Because of the trade-offs involved in implementing quantile regression algorithms in practical settings, our empirical results (Section 3.3) show that by using a conditioning algorithm that is only slightly worse than the best previous conditioning algorithms for each of these two criteria, our new conditioning algorithm can lead to better results than either of the previous algorithms that was superior by only one of those criteria.

3.2.2 *Main technical ingredients*

In this subsection, we present the main technical ingredients underlying our main algorithm for quantile regression. We start with a result which says that if we sample sufficiently many (but still only $\texttt{poly}(d)$) rows according to an appropriately-defined non-uniform importance sampling distribution (of the form given in (3.4) below), then we obtain a $(1 \pm \epsilon)$-distortion embedding matrix with respect to the loss function of quantile regression. Note that the form of this lemma is based on ideas from [Das+09; Cla+13].

Lemma 3.3 (Subspace-preserving sampling lemma)**.** *Given $A \in \mathbb{R}^{n \times d}$, let $U \in \mathbb{R}^{n \times d}$ be a well-conditioned basis for $\mathcal{A}$ with ℓ_1 condition number $\bar{\kappa}_1$. For $s > 0$, define*

$$\hat{p}_i \geq \min\{1, s \cdot \|U_{(i)}\|_1 / |U|_1\}, \tag{3.4}$$

and let $S \in \mathbb{R}^{n \times n}$ be a random diagonal matrix with $S_{ii} = 1/\hat{p}_i$ with probability $\hat{p}_i$, and 0 otherwise. Then when $\epsilon < 1/2$ and

$$s \geq \frac{\tau}{1-\tau} \frac{27\bar{\kappa}_1}{\epsilon^2} \left(d \log \left(\frac{\tau}{1-\tau} \frac{18}{\epsilon} \right) + \log \left(\frac{4}{\delta} \right) \right),$$

with probability at least $1 - \delta$, for every $x \in \mathbb{R}^d$,

$$(1-\varepsilon)\rho_\tau(Ax) \leq \rho_\tau(SAx) \leq (1+\varepsilon)\rho_\tau(Ax). \tag{3.5}$$

Remark 1. *It is not hard to see that for any matrix S satisfying (3.5), the rank of A is preserved.*

In order to apply Lemma 3.3 to quantile regression, we need to compute the sampling probabilities in (3.4). This requires two steps: first, find a well-conditioned basis U; and second, compute the ℓ_1 row norms of U. For the first step, we can apply any method described in the previous subsection. Other methods are possible, but SPCT, SPCT2 and SPCT3 are of particular interest due to their nearly input-sparsity running time. We will now present an algorithm that will perform the second step of approximating the ℓ_1 row norms of U in the allotted $\mathcal{O}(\texttt{nnz}(A) \cdot \log n)$ time.

Suppose we have obtained R such that AR^{-1} is a well-conditioned basis. Consider, next, computing $\hat{p}_i$ from U (or from A and R^{-1}), and note that forming U explicitly is expensive both when A is dense and when A is sparse. In practice, however, we will not need to form U explicitly, and we will not need to compute the exact value of the ℓ_1-norm of each row of U. Indeed, it suffices to get estimates of $\|U_{(i)}\|_1$, in which case we can adjust the sampling complexity s to maintain a small approximation factor. Algorithm 7 provides a way to compute the estimates of the ℓ_1 norm of each row of U fast and construct the sampling matrix. The same algorithm was used in [Cla+13] except for the choice of desired sampling complexity s and we present the entire algorithm for completeness. Our main result for Algorithm 7 is presented in Lemma 3.4.

Lemma 3.4 (Fast construction of $(1 \pm \epsilon)$-distortion sampling matrix)**.** *Given a matrix $A \in \mathbb{R}^{n \times d}$, and a matrix $R \in \mathbb{R}^{d \times d}$ such that AR^{-1} is a well-conditioned basis for $\mathcal{A}$ with condition number κ, Algorithm 7 takes $\mathcal{O}(nnz(A) \cdot \log n)$ time to compute a sampling matrix $S \in \mathbb{R}^{\hat{s} \times n}$ (with only one nonzero per row), such that with probability at least 0.9, S is a $(1 \pm \epsilon)$-distortion sampling matrix. That is for all $x \in \mathbb{R}^d$,*

$$(1-\epsilon)\rho_\tau(Ax) \leq \rho_\tau(SAx) \leq (1+\epsilon)\rho_\tau(Ax). \tag{3.6}$$

Algorithm 7 Fast construction of $(1 \pm \epsilon)$-distortion sampling matrix of $(\mathcal{A}, \rho_\tau(\cdot))$

1: **Input:** $A \in \mathbb{R}^{n\times d}, R \in \mathbb{R}^{d\times d}$ such that AR^{-1} is well-conditioned with condition number κ, $\epsilon \in (0, 1/2)$, $\tau \in [1/2, 1)$.
2: **Output:** Sampling matrix $S \in \mathbb{R}^{n\times n}$.
3: Let $\Pi_2 \in \mathbb{R}^{d\times r_2}$ be a matrix of independent Cauchys with $r_2 = 15\log(40n)$.
4: Compute $R^{-1}\Pi_2$ and construct $\Lambda = AR^{-1}\Pi_2 \in \mathbb{R}^{n\times r_2}$.
5: For $i \in [n]$, compute $\lambda_i = \text{median}_{j\in[r_2]}|\Lambda_{ij}|$.
6: For $s = \frac{\tau}{1-\tau}\frac{81\kappa}{\epsilon^2}\left(d\log\left(\frac{\tau}{1-\tau}\frac{18}{\epsilon}\right) + \log 80\right)$ and $i \in [n]$, compute probabilities

$$\hat{p}_i = \min\left\{1, s \cdot \frac{\lambda_i}{\sum_{i\in[n]}\lambda_i}\right\}.$$

7: Let $S \in \mathbb{R}^{n\times n}$ be diagonal with independent entries

$$S_{ii} = \begin{cases} \frac{1}{\hat{p}_i}, & \text{with probability } \hat{p}_i; \\ 0, & \text{with probability } 1 - \hat{p}_i. \end{cases}$$

8: **Return** S.

Further, with probability at least $1 - o(1)$, $\hat{s} = \mathcal{O}\left(\mu\bar{\kappa}_1 d\log(\mu/\epsilon)/\epsilon^2\right)$, *where* $\mu = \frac{\tau}{1-\tau}$.

Remark 2. *Such technique can also be used to fast approximate the* ℓ_2 *row norms of a well-conditioned basis by post-multiplying a matrix consisting of Gaussian variables; see [Dri+12].*

Remark 3. *In the text before Lemma 3.4,* s *denotes an input parameter for defining the importance sampling probabilities. However, the actual sample size might be less than that. Since Lemma 3.4 is about the construction of the sampling matrix* S*, we let* $\hat{s}$ *denote the actual number of row selected. Also, as stated, the output of Algorithm 7 is an* $n \times n$ *matrix; but if we delete the all-zero rows, then the actual size of* S *is indeed* $\hat{s}$ *by* d *as described in Lemma 3.4. Throughout the following text, by sampling size* s*, we mean the desired sampling size, which is the parameter in the algorithm.*

3.2.3 *Main algorithm*

In this subsection, we state our main algorithm for computing an approximate solution to the quantile regression problem. Recall that, to compute a relative-error approximate solution, it suffices to compute a $(1 \pm \epsilon)$-distortion sampling matrix S. To construct S, we first compute a well-conditioned basis U with SPCT, SPCT2 (Algorithm 5), or SPCT3 (Algorithm 6), and then we apply Algorithm 7 to approximate the ℓ_1 norm of each row of U. These procedures are summarized in Algorithm 8. The main quality-of-approximation result for this algorithm by using Algorithm 8 is stated in Theorem 3.5.

Theorem 3.5 (Fast quantile regression)**.** *Given* $A \in \mathbb{R}^{n\times d}$ *and* $\varepsilon \in (0, 1/2)$*, if Algorithm 5 is used in Step 1, Algorithm 8 returns a vector* $\hat{x}$ *that, with probability at least 0.8, satisfies*

$$\rho_\tau(A\hat{x}) \le \left(\frac{1+\varepsilon}{1-\varepsilon}\right)\rho_\tau(Ax^*),$$

Algorithm 8 Fast randomized algorithm for quantile regression

1: **Input:** $A \in \mathbb{R}^{n \times d}$ with full column rank, $\epsilon \in (0, 1/2)$, $\tau \in [1/2, 1)$.
2: **Output:** An approximated solution $\hat{x} \in \mathbb{R}^d$ to problem minimize$_{x \in \mathcal{C}}$ $\rho_\tau(Ax)$.
3: Compute $R \in \mathbb{R}^{d \times d}$ such that AR^{-1} is a well-conditioned basis for $\mathcal{A}$ via SPCT, SPCT2 (Algorithm 5) or SPCT3 (Algorithm 6).
4: Compute a $(1 \pm \epsilon)$-distortion embedding $S \in \mathbb{R}^{n \times n}$ of $(\mathcal{A}, \rho_\tau(\cdot))$ via Algorithm 7.
5: Compute $\hat{x} \in \mathbb{R}^d$ that minimizes $\rho_\tau(SAx)$ with respect to $x \in \mathcal{C}$.
6: **Return** $\hat{x}$.

where x^* is an optimal solution to the original problem. In addition, the algorithm to construct $\hat{x}$ runs in time

$$\mathcal{O}(\text{nnz}(A) \cdot \log n) + \phi\left(\mathcal{O}(\mu d^3 \log(\mu/\epsilon)/\epsilon^2), d\right),$$

where $\mu = \frac{\tau}{1-\tau}$ and $\phi(s, d)$ is the time to solve a quantile regression problem of size $s \times d$.

Remark 4. *As stated, Theorem 3.5 uses Algorithm 5 in Step 3; we did this because it leads to the best known running-time results in worst-case analysis, but our empirical results will indicate that due to various trade-offs the situation is more complex in practice.*

Remark 5. *Our theory provides a bound on the solution quality, as measured by the objective function of the quantile regression problem, and it does not provide bounds for the difference between the exact solution vector and the solution vector returned by our algorithm. We will, however, compute this latter quantity in our empirical evaluation.*

3.3 EMPIRICAL EVALUATION

In this section and the next section, we present our main empirical results. We have evaluated an implementation of Algorithm 8 using several different conditioning methods in Step 1. We have considered both simulated data and real data, and we have considered both medium-sized data as well as terabyte-scale data. In this section, we will summarize our results for medium-sized data. The results on terabyte-scale data can be found in Chapter 6.

SIMULATED SKEWED DATA For the synthetic data, in order to increase the difficulty for sampling, we will add imbalanced measurements to each coordinates of the solution vector. A similar construction for the test data was appeared in [Cla+13]. Due to the skewed structure of the data, we will call this data set "skewed data" in the following discussion. This data set is generated in the following way.

1. Each row of the design matrix A is a canonical vector. Suppose the number of measurements on the j-th column are c_j, where $c_j = qc_{j-1}$, for $j = 2, \ldots, d$. Here $1 < q \leq 2$. A is a $n \times d$ matrix.

2. The true vector x^* with length d is a vector with independent Gaussian entries. Let $b^* = Ax^*$.

3. The noise vector ϵ is generated with independent Laplacian entries. We scale ϵ such that $\|\epsilon\|/\|b^*\| = 0.2$. The response vector is given by

$$b_i = \begin{cases} 500\epsilon_i & \text{with probability } 0.001; \\ b_i^* + \epsilon_i & \text{otherwise.} \end{cases}$$

When making the experiments, we require $c_1 \geq 161$. This implies that if we choose $s/n \geq 0.01$, and perform the uniform sampling, with probability at least 0.8, at least one row in the first block (associated with the first coordinate) will be selected, due to $1 - (1 - 0.01)^{161} \geq 0.8$. Hence, if we choose $s \geq 0.01n$, we may expect uniform sampling produce acceptable estimation.

REAL CENSUS DATA For the real data, we consider a data set consisting of a 5% sample of the U.S. 2000 Census data[1], consisting of annual salary and related features on people who reported that they worked 40 or more weeks in the previous year and worked 35 or more hours per week. The size of the design matrix is 5×10^6 by 11.

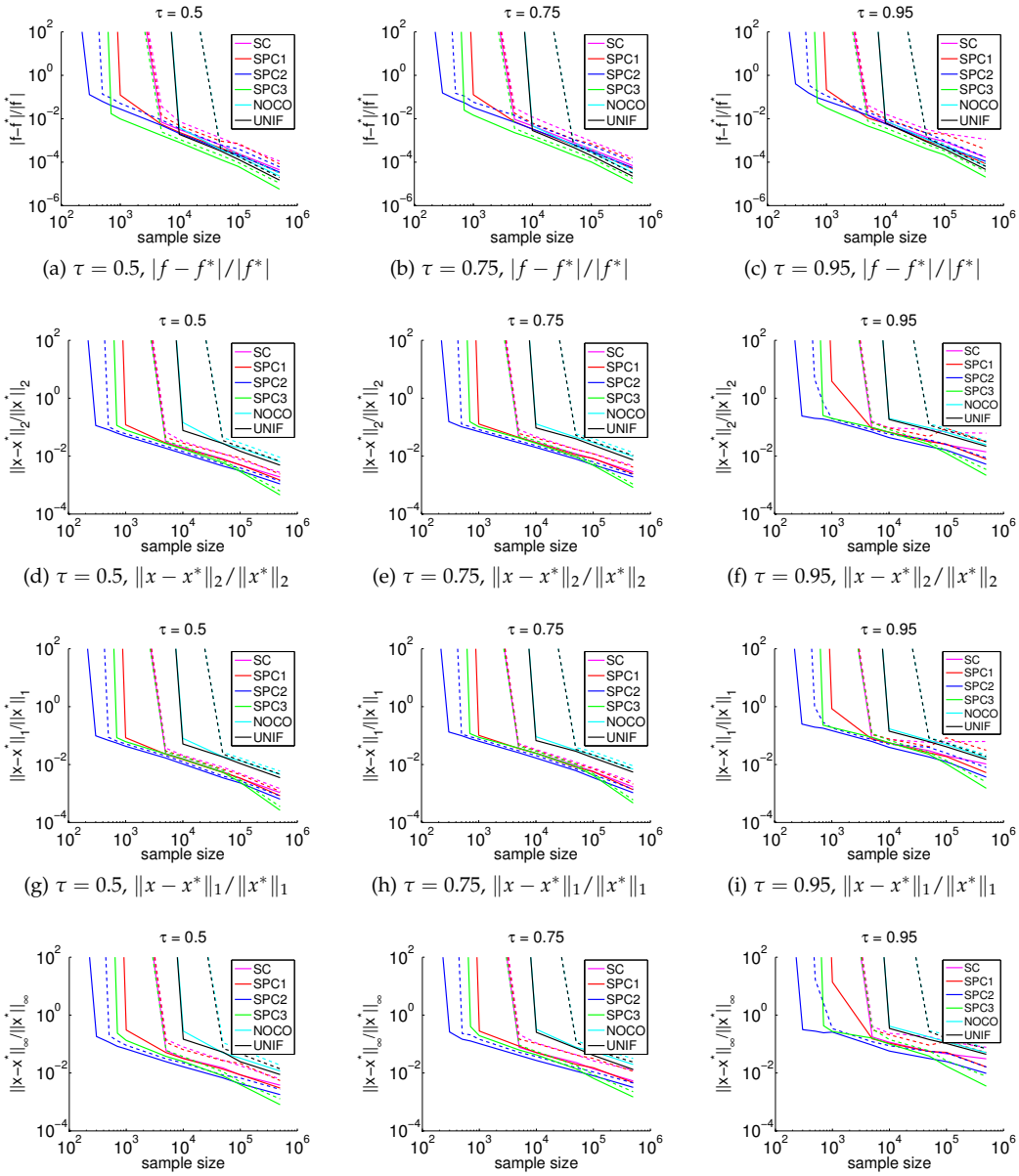
The remainder of this section will consist of six subsections, the first five of which will show the results of experiments on the skewed data, and then 3.3.6, which will show the results on census data. In more detail, 3.3.1, 3.3.2, 3.3.3, and 3.3.4 will summarize the performance of the methods in terms of solution quality as the parameters s, n, d, and τ, respectively, are varied; and 3.3.5 will show how the running time changes as s, n, and d change.

3.3.1 *Quality of approximation when the sampling size s changes*

As discussed in Section 3.2.1, we can use one of several methods for the conditioning step, i.e., for finding the well-conditioned basis $U = AR^{-1}$ in the Step 1 of Algorithm 8. Here, we will consider the empirical performance of *six* methods for doing this conditioning step, namely: SC, SPC1, SPC2, SPC3, NOCO, and UNIF. SC is the (slow) Cauchy transform (Lemma 2.15). SPC1, SPC2, SPC3 are the three nearly input-sparsity time conditioning methods, i.e., SPCT1, SPCT2 and SPCT3, described in 3.2.1; NOCO stands for "no conditioning," meaning the matrix R in the conditioning step is taken to be identity; and, UNIF stands for the uniform sampling method, which we include here for completeness. Note that, for all the methods, we compute the row norms of the well-conditioned basis exactly instead of estimating them with Algorithm 7. The reason is that this permits a cleaner evaluation of the quantile regression algorithm, as this may reduce the error due to the estimating step. We have, however, observed similar results if we approximate the row norms well.

Rather than determining the sample size from a given tolerance ϵ, we let the sample size s vary in a range as an input to the algorithm. Also, for a fixed data set, we will show the results when $\tau = 0.5, 0.75, 0.95$. In our figure, we will plot the first and the third quartiles of the relative errors of the objective value and solution measured in three different norms from 50 independent trials. We restrict the y axis in the plots to the range of $[0, 100]$ to show more details. We start with a test on skewed data with size $1e6 \times 50$. (Recall that, by $1e6 \times 50$, we mean that $n = 1 \times 10^6$ and $d = 50$.) The resulting plots are shown in Figure 1.

1 U.S. Census, http://www.census.gov/census2000/PUMS5.html



(a) $\tau = 0.5$, $|f - f^*|/|f^*|$ (b) $\tau = 0.75$, $|f - f^*|/|f^*|$ (c) $\tau = 0.95$, $|f - f^*|/|f^*|$

(d) $\tau = 0.5$, $\|x - x^*\|_2/\|x^*\|_2$ (e) $\tau = 0.75$, $\|x - x^*\|_2/\|x^*\|_2$ (f) $\tau = 0.95$, $\|x - x^*\|_2/\|x^*\|_2$

(g) $\tau = 0.5$, $\|x - x^*\|_1/\|x^*\|_1$ (h) $\tau = 0.75$, $\|x - x^*\|_1/\|x^*\|_1$ (i) $\tau = 0.95$, $\|x - x^*\|_1/\|x^*\|_1$

(j) $\tau = 0.5$, $\|x - x^*\|_\infty/\|x^*\|_\infty$ (k) $\tau = 0.75$, $\|x - x^*\|_\infty/\|x^*\|_\infty$ (l) $\tau = 0.5$, $\|x - x^*\|_\infty/\|x^*\|_\infty$

Figure 1: The first (solid lines) and the third (dashed lines) quartiles of the relative errors of the objective value (namely, $|f - f^*|/|f^*|$) and solution vector (measured in three different norms, namely, the ℓ_2, ℓ_1 and ℓ_∞ norms), by using 6 different methods, among 50 independent trials. The test is on skewed data with size $1e6$ by 50. The three different columns correspond to $\tau = 0.5, 0.75, 0.95$, respectively.

From these plots, if we look at the sampling size required for generating at least 1-digit accuracy, then SPC2 needs the fewest samples, followed by SPC3, and then SPC1. This is consistent with the order of the condition numbers of these methods. For SC, although in theory it has good condition number properties, in practice it performs worse than other methods. Not surprisingly, NOCO and UNIF are not reliable when s is very small, e.g., less than $1e4$.

When the sampling size s is large enough, the accuracy of each conditioning method is close to the others in terms of the objective value. Among these, SPC3 performs slightly better than others. When estimating the actual solution vectors, the conditioning-based methods behave substantially better than the two naive methods. SPC2 and SPC3 are the most reliable methods since they can yield the least relative error for every sample size s. NOCO is likely to sample the outliers, and UNIF cannot get accurate answer until the sampling size $s \geq 1e4$. This accords with our expectations. For example, when s is less than $1e4$, as we pointed out in the remark below the description of the skewed data, it is very likely that none of the rows in the first block corresponding to the first coordinate will be selected. Thus, poor estimation will be generated due to the imbalanced measurements in the design matrix. Note that from the plots we can see that if a method fails with some sampling complexity s, then for that value of s the relative errors will be huge e.g., larger than 100, which is clearly a trivial result). Note also that all the methods can generate at least 1-digit accuracy if s is large enough.

It is worth mentioning the performance difference among SPC1, SPC2 and SPC3. From Table 7, we show the trade-offs between running time and condition number for the three methods. As we pointed out, SPC2 always needs the least sampling complexity to generate 2-digit accuracy, followed by SPC3 and then SPC1. When s is large enough, SPC2 and SPC3 perform substantially better than SPC1. As for the running time, SPC1 is the fastest, followed by SPC3, and then SPC2. Again, all of these follow the theory about our SPCT2 methods. We will present a more detailed discussion for the running time in Section 3.3.5.

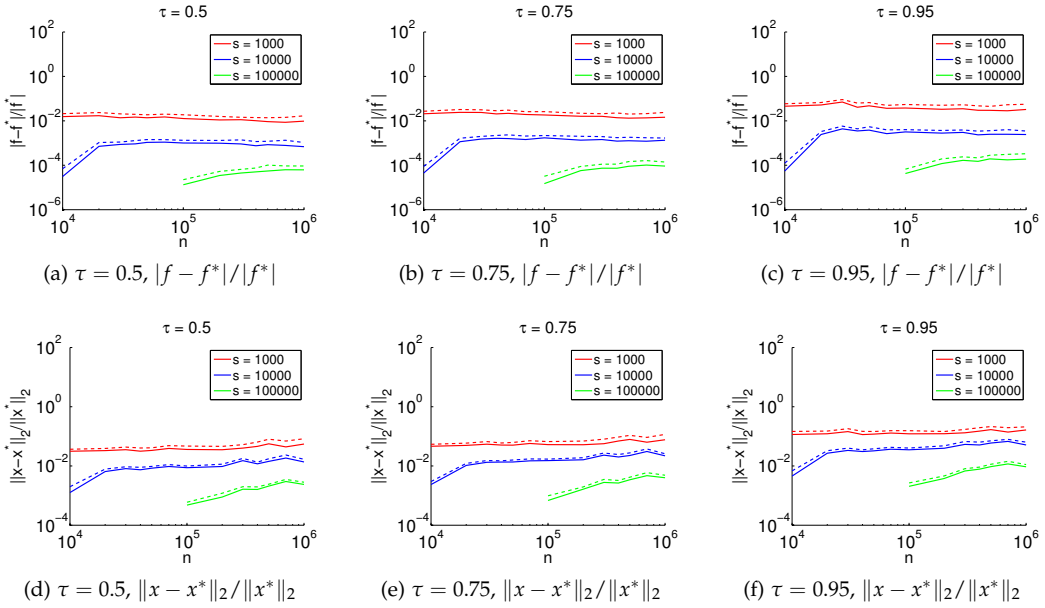
Although our theory doesn't say anything about the quality of the solution vector itself (as opposed to the value of the objective function), we evaluate this here. To measure the approximation to the solution vectors, we use three norms (the ℓ_1, ℓ_2, and ℓ_∞ norms). From Figure 1, we see that the performance among these method is qualitatively similar for each of the three norms, but the relative error is higher when measured in the ℓ_∞ norm. Not surprisingly, NOCO and UNIF are not among the reliable methods when s is small (and they get worse when s is even smaller). Note that the relative error for each method doesn't change substantially when τ takes different values.

(We note also that, for subsequent figures in subsequent subsections, we obtained similar qualitative trends for the errors in the approximate solution vectors when the errors were measured in different norms. Thus, due to this similarity and to save space, in subsequent figures, we will only show errors for ℓ_2 norm.)

3.3.2 *Quality of approximation when the higher dimension n changes*

Next, we describe how the performance of our algorithm varies when higher dimension n changes. Figure 2 shows the change of relative error on the objective value and solution vector by using SPC3 and letting n vary from $1e4$ to $1e6$ and $d = 50$ fixed. Recall, from Theorem 3.5, that for given a tolerance ϵ, the required sampling complexity s depends only on d. That is, if we fix the sampling size s and d, then the relative error should not vary much, as a function of n. If we inspect Figure 2, we see that the relative errors are

almost constant as a function of increasing n, provided that n is much larger than s. When s is very close to n, since we are sampling roughly the same number of rows as in the full data, we should expect lower errors. Also, we can see that by using SPC3, relative errors remain roughly the same in magnitude.



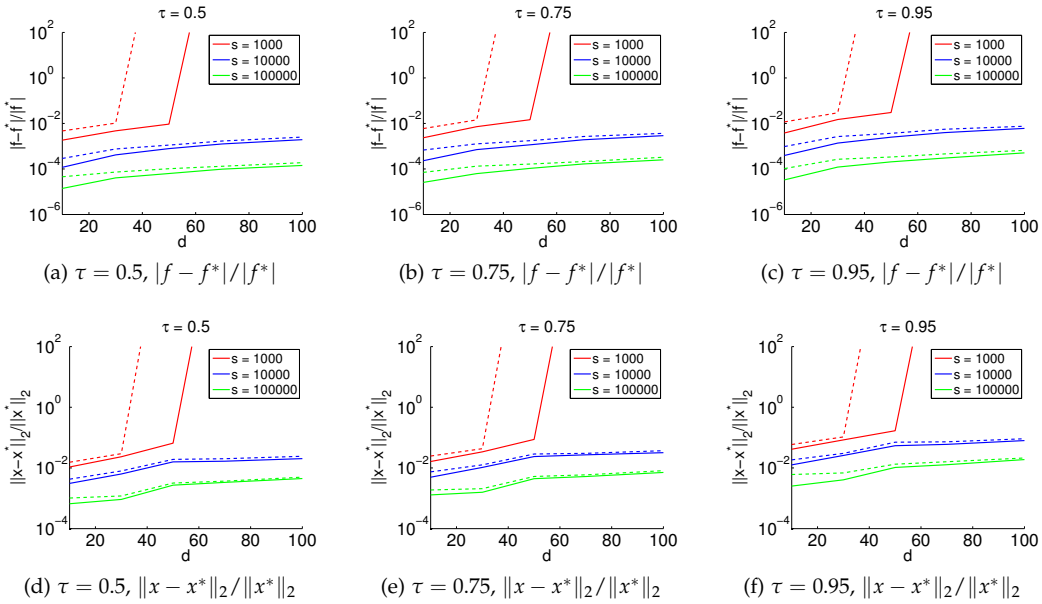
(a) $\tau = 0.5$, $|f - f^*|/|f^*|$ (b) $\tau = 0.75$, $|f - f^*|/|f^*|$ (c) $\tau = 0.95$, $|f - f^*|/|f^*|$

(d) $\tau = 0.5$, $\|x - x^*\|_2/\|x^*\|_2$ (e) $\tau = 0.75$, $\|x - x^*\|_2/\|x^*\|_2$ (f) $\tau = 0.95$, $\|x - x^*\|_2/\|x^*\|_2$

Figure 2: The first (solid lines) and the third (dashed lines) quartiles of the relative errors of the objective value (namely, $|f - f^*|/|f^*|$) and solution vector (namely, $\|x - x^*\|_2/\|x^*\|_2$), when n varying from $1e4$ to $1e6$ and $d = 50$ by using SPC3, among 50 independent trials. The test is on skewed data. The three different columns correspond to $\tau = 0.5, 0.75, 0.95$, respectively.

3.3.3 *Quality of approximation when the lower dimension d changes*

Next, we describe how the overall performance changes when the lower dimension d changes. In Figure 3, we let d take more values in the range of $[10, 100]$, and we show the relative errors by using SPC3 for different sampling sizes s and τ values. As can be seen in Figure 3, as d gets larger, the performance of the two naive methods do not vary a lot. However, this increases the difficulty for conditioning methods to yield 2-digit accuracy. When d is quite small, most methods can yield 2-digit accuracy even when s is not large. When d becomes large, SPC2 and SPC3 provide good estimation, even when $s < 1000$. The relative performance among these methods remains unchanged. For Figure 3, the relative errors are monotonically increasing for each sampling size. This is consistent with our theory that, to yield high accuracy, the required sampling size is a low-degree polynomial of d.



(a) $\tau = 0.5$, $|f - f^*|/|f^*|$ (b) $\tau = 0.75$, $|f - f^*|/|f^*|$ (c) $\tau = 0.95$, $|f - f^*|/|f^*|$

(d) $\tau = 0.5$, $\|x - x^*\|_2/\|x^*\|_2$ (e) $\tau = 0.75$, $\|x - x^*\|_2/\|x^*\|_2$ (f) $\tau = 0.95$, $\|x - x^*\|_2/\|x^*\|_2$

Figure 3: The first (solid lines) and the third (dashed lines) quartiles of the relative errors of the objective value (namely, $|f - f^*|/|f^*|$) and solution vector (namely, $\|x - x^*\|_2/\|x^*\|_2$), when d varying from 10 to 100 and $n = 1e6$ by using SPC3, among 50 independent trials The test is on skewed data. The three different columns correspond to $\tau = 0.5, 0.75, 0.95$, respectively.

3.3.4 *Quality of approximation when the quantile parameter τ changes*

Next, we will let τ change, for a fixed data set and fixed conditioning method, and we will investigate how the resulting errors behave as a function of τ. We will consider τ in the range of $[0.5, 0.9]$, equally spaced by 0.05, as well as several extreme quantiles such as 0.975 and 0.98. We consider skewed data with size $1e6 \times 50$; and our plots are shown in Figure 4.

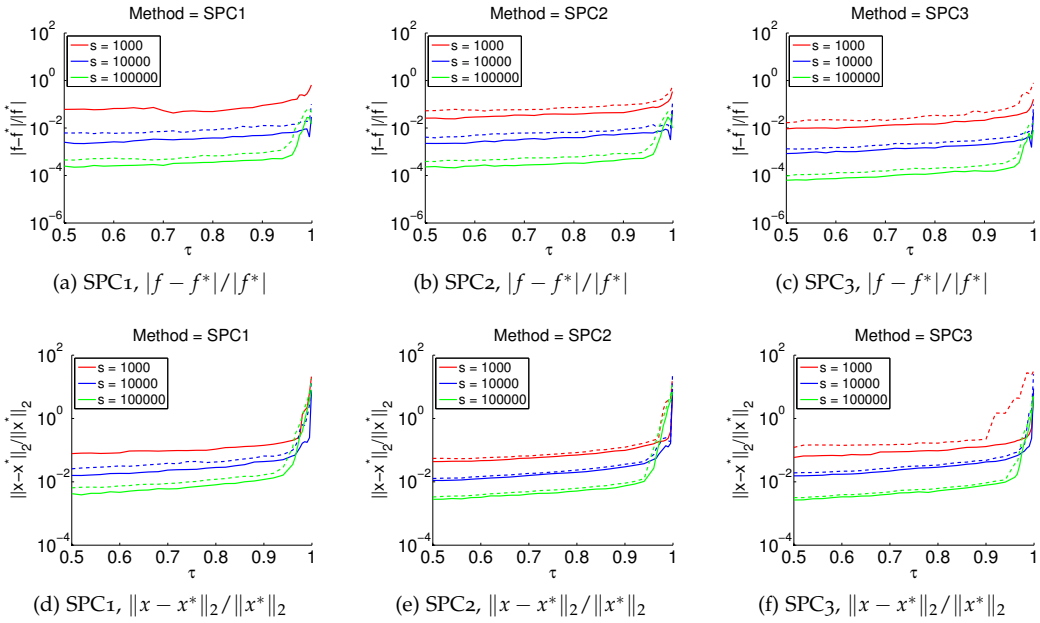
The plots in Figure 4 demonstrate that, given the same method and sampling size s, the relative errors are monotonically increasing but only very gradually, i.e., they do not change very substantially in the range of $[0.5, 0.95]$. On the other hand, all the methods generate high relative errors when τ takes extreme values very near 1 (or 0). Overall, SPC2 and SPC3 performs better than SPC1. Although for some quantiles SPC3 can yield slightly lower errors than SPC2, it too yields worst results when τ takes on extreme values.

3.3.5 *Evaluation on running time performance*

In this subsection, we will describe running time issues, with an emphasis on how the running time behaves as a function of s, d and n.

WHEN THE SAMPLING SIZE s CHANGES

To start, Figure 5 shows the running time for computing three subproblems associated with three different τ values by using six methods (namely, SC, SPC1, SPC2, SPC3, NOCO, UNIF) when the sampling size s changes. (This is simply the running time comparison



(a) SPC1, $|f - f^*|/|f^*|$ (b) SPC2, $|f - f^*|/|f^*|$ (c) SPC3, $|f - f^*|/|f^*|$

(d) SPC1, $\|x - x^*\|_2/\|x^*\|_2$ (e) SPC2, $\|x - x^*\|_2/\|x^*\|_2$ (f) SPC3, $\|x - x^*\|_2/\|x^*\|_2$

Figure 4: The first (solid lines) and the third (dashed lines) quartiles of the relative errors of the objective value, (namely $|f - f^*|/|f^*|$) and solution vector (namely, $\|x - x^*\|_2/\|x^*\|_2$), when τ varying from 0.5 to 0.999 by using SPC1, SCP2, SPC3, among 50 independent trials. The test is on skewed data with size $1e6$ by 50. Within each plot, three sampling sizes are considered, namely, $1e4, 1e4, 1e5$.

for all the six methods used to generate Figure 1.) As expected, the two naive methods (NOCO and UNIF) run faster than other methods in most cases—since they don't perform the additional step of conditioning. For $s < 10^4$, among the conditioning-based methods, SPC1 runs fastest, followed by SPC3 and then SPC2. As s increases, however, the faster methods, including NOCO and UNIF, become relatively more expensive; and when $s \approx 5e5$, all of the curves, except for SPC1, reach almost the same point.

To understand what is happening here, recall that we accept the sampling size s as an input in our algorithm; and we then construct our sampling probabilities by $\hat{p}_i = \min\{1, s \cdot \lambda_i / \sum \lambda_i\}$, where λ_i is the estimation of the ℓ_1 norm of the i-th row of a well-conditioned basis. (See Step 4 in Algorithm 7.) Hence, the s is not the exact sampling size. Indeed, upon examination, in this regime when s is large, the actual sampling size is often much less than the input s. As a result, almost all the conditioning-based algorithms are solving a subproblem with size, say, $s/2 \times d$, while the two naive methods are are solving subproblem with size about $s \times d$. The difference of running time for solving problems with these sizes can be quite large when s is large. For conditioning-based algorithms, the running time mainly comes from the time for conditioning and solving the subproblem. Thus, since SPC1 needs the least time for conditioning, it should be clear why SPC1 needs much less time when s is very large.

WHEN THE HIGHER DIMENSION n CHANGES

Next, we compare the running time of our method with some competing methods when data size increases. The competing methods are the primal-dual method, referred to as

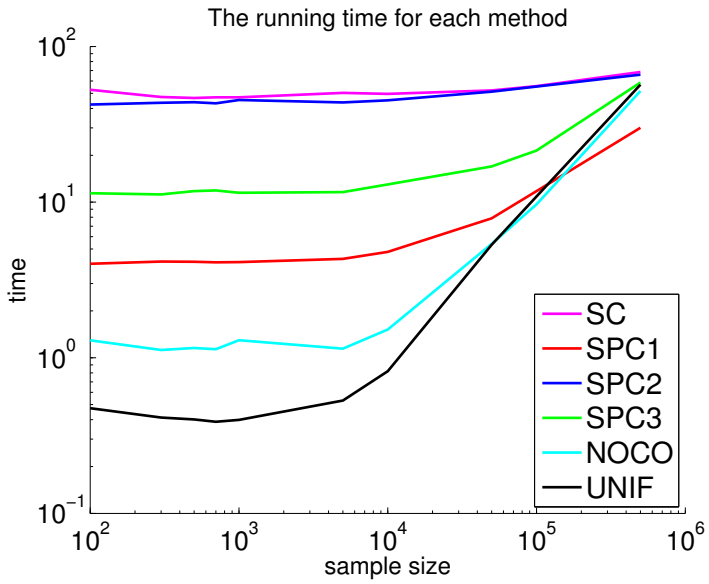


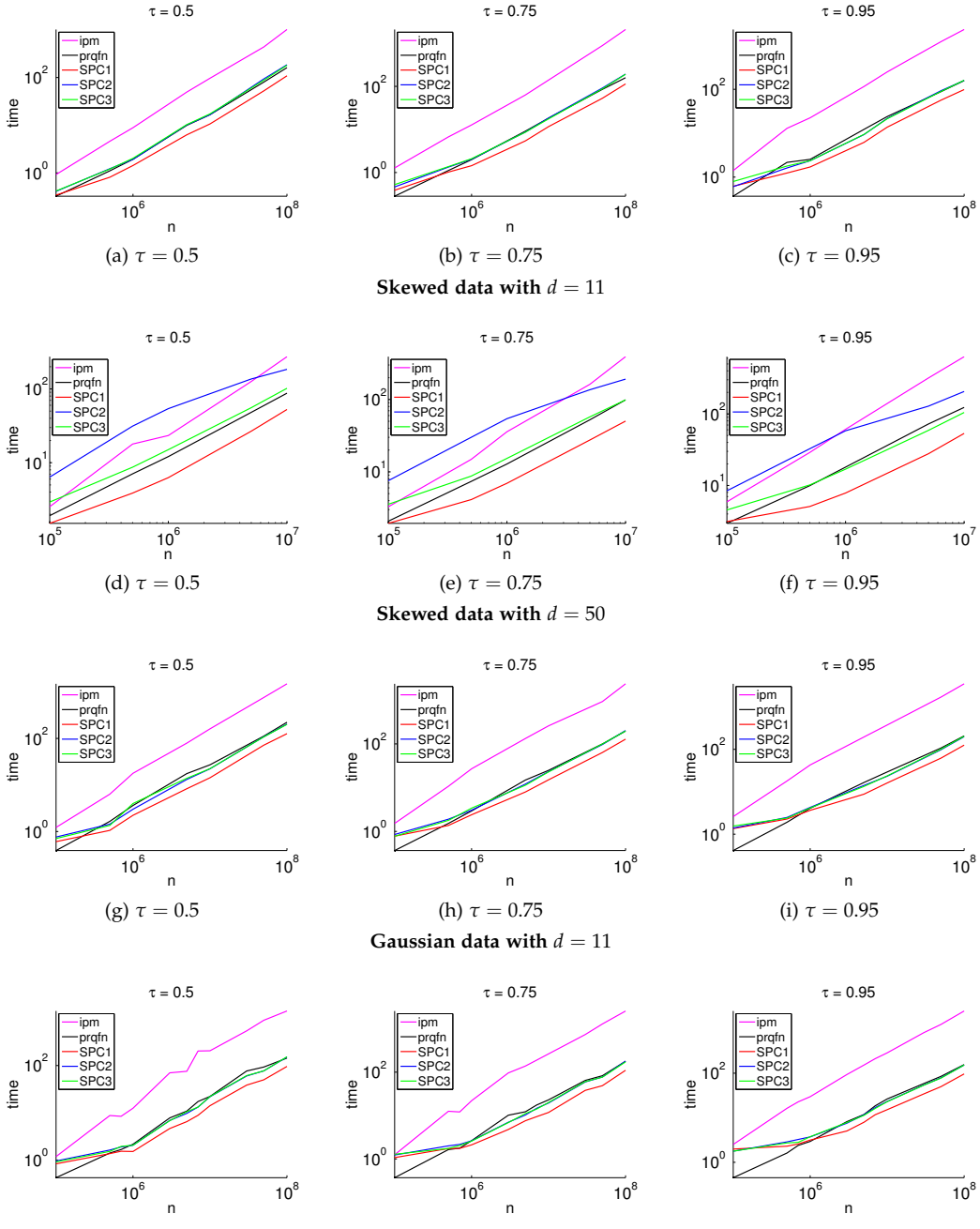
Figure 5: The running time for solving the three problems associated with three different τ values by using six methods, namely, SC, SPC1, SPC2, SPC3, NOCO, UNIF, when the sampling size s changes.

`ipm`, and that with preprocessing, referred to as `prqfn`; see [PK97] for more details on these two methods.

We let the large dimension n increase from $1e5$ to $1e8$, and we fix $s = 5e4$. For completeness, in addition to the skewed data, we will consider two additional data sets. First, we also consider a design matrix with entries generated from i.i.d. Gaussian distribution, where the response vector is generated in the same manner as the skewed data. Also, we will replicate the census data 20 times to obtain a data set with size $1e8$ by 11. For each n, we extract the leading $n \times d$ submatrix of the replicated matrix, and we record the corresponding running time. The results of running time on all three data sets are shown in Figure 6.

From the plots in Figure 6 we see, SPC1 runs faster than any other methods across all the data sets, in some cases significantly so. SPC2, SPC3 and `prqfn` perform similarly in most cases, and they appear to have a linear rate of increase. Also, the relative performance between each method does not vary a lot as the data type changes.

Notice that for the skewed data, when $d = 50$, SPC2 runs much slower than when $d = 10$. The reason for this is that, for conditioning-based methods, the running time is composed of two parts, namely, the time for conditioning and the time for solving the subproblem. For SPC2, an ellipsoid rounding needs to be applied on a smaller data set whose larger dimension is a polynomial of d. When the sampling size s is small, i.e., the size of the subproblem is not too large, the dominant running time for SPC2 will be the time for ellipsoid rounding, and as d increase (by, say, a factor of 5) we expect a worse running time. Notice also that, for all the methods, the running time does not vary a lot when τ changes. Finally, notice that all the conditioning-based methods run faster on skewed data, especially when d is small. The reason is that the running time for these three methods is of the order of input-sparsity time, and the skewed data are very sparse.



(a) $\tau = 0.5$ (b) $\tau = 0.75$ (c) $\tau = 0.95$

Skewed data with $d = 11$

(d) $\tau = 0.5$ (e) $\tau = 0.75$ (f) $\tau = 0.95$

Skewed data with $d = 50$

(g) $\tau = 0.5$ (h) $\tau = 0.75$ (i) $\tau = 0.95$

Gaussian data with $d = 11$

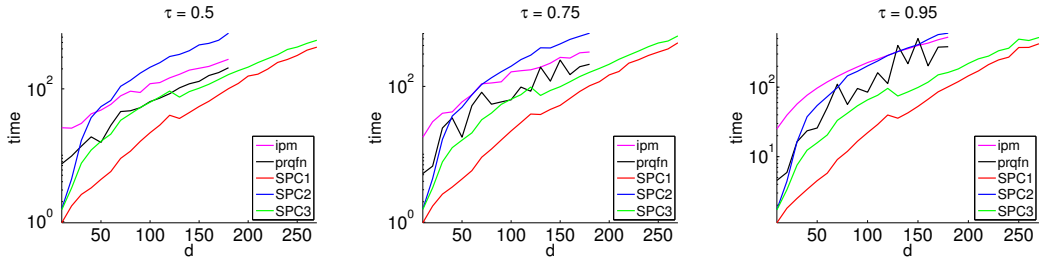
(j) $\tau = 0.5$ (k) $\tau = 0.75$ (l) $\tau = 0.95$

Replicated census data with $d = 11$

Figure 6: The running time for five methods (`ipm`, `prqfn`, SPC1, SPC2 and SPC3) on the same data set, with d fixed and n changing. The sampling size $s = 5e4$, and the three columns correspond to $\tau = 0.5, 0.75, 0.95$, respectively.

WHEN THE LOWER DIMENSION d CHANGES

Finally, we will describe the scaling of the running time as the lower dimension d changes. To do so, we fixed $n = 1e6$ and the sampling size $s = 1e4$. We let all five methods run on the data set with d varying from 5 up to 180. When $d \approx 200$, the scaling was such that all the methods except for SPC1 and SPC3 became too expensive. Thus, we let only SPC1 and SPC3 run on additional data sets with d up to 270. The plots are shown in Figure 7.



(a) $\tau = 0.5$ (b) $\tau = 0.75$ (c) $\tau = 0.95$

Figure 7: The running time for five methods (`ipm`, `prqfn`, SPC1, SPC2, and SPC3) for solving skewed data, with $n = 1e6$, $s = 1e4$, when d varies. SPC1 and SPC3 show better scaling than other methods when $d < 180$. For this reason, we keep running the experiments for SPC1 and SPC3 until $d = 270$. When $d < 100$, the three conditioning-based methods can yield 2-digit accuracy. When for $d \in [100, 180]$, they can yield 1-digit accuracy.

From the plots in Figure 7, we can see that when $d < 180$, SPC1 runs significantly faster than any other method, followed by SPC3 and `prqfn`. The performance of `prqfn` is quite variable. The reason for this is that there is a step in `prqfn` that involves uniform sampling, and the number of subproblems to be solved in each time might vary a lot. The scalings of SPC2 and `ipm` are similar, and when d gets much larger, say $d > 200$, they may not be favorable due to the running time. When $d < 180$, all the conditioning methods can yield at least 1-digit accuracy. Although one can only get an approximation to the true solution by using SPC1 and SPC3, they will be a good choice when d gets even larger, say up to several hundred, as we shown in Figure 7. We note that we could let d get even larger for SPC1 and SPC3, demonstrating that SPC1 and SPC3 is able to run with a much larger lower dimension than the other methods.

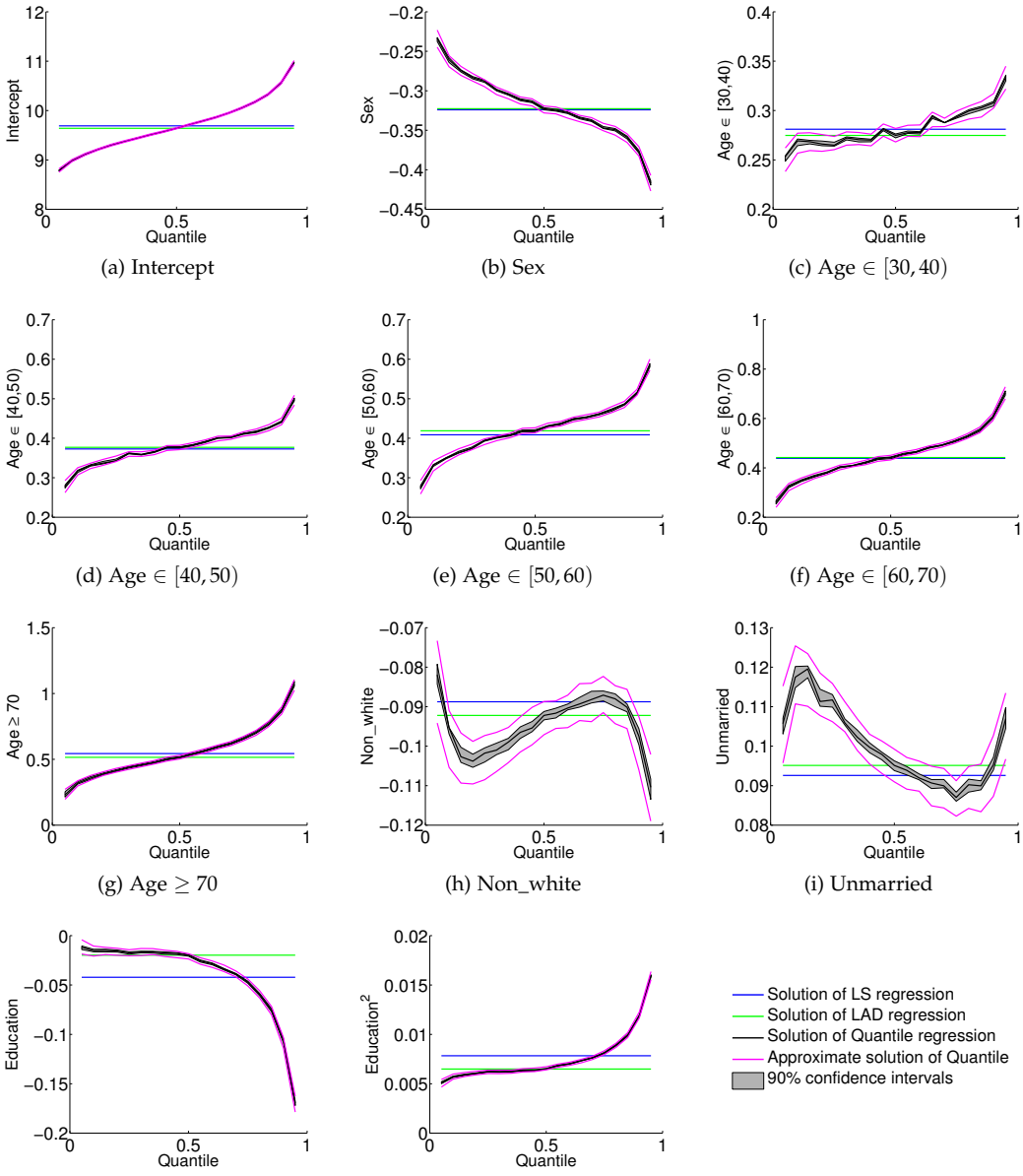
Remark 6. *One may notice a slight but sudden change in the running time for SPC1 and SPC3 at $d \approx 130$. After we traced down the reason, we found out that the difference come from the time in the conditioning step (since the subproblems they are solving have similar size), especially the time for performing the QR factorization. At this size, it will be normal to take more time to factorize a slightly smaller matrix due to the structure of cache line, and it is for this reason that we see that minor decrease in running time with increasing d. We point out that the running time of our conditioning-based algorithm is mainly affected by the time for the conditioning step. That is also the reason why it does not vary a lot when τ changes.*

3.3.6 *Evaluation on solution of Census data*

Here, we will describe more about the accuracy on the census data when SPC3 is applied to it. The size of the census data is roughly $5e6 \times 11$.

We will generate plots that are similar to those appeared in [KH01]. For each coefficient, we will compute a few quantities of it, as a function of τ, when τ varies from 0.05 to 0.95. We compute a point-wise 90 percent confidence interval for each τ by bootstrapping. These are shown as the shaded area in each subfigure. Also, we compute the quartiles of the approximated solutions by using SPC3 from 200 independent trials with sampling size $s = 5e4$ to show how close we can get to the confidence interval. In addition, we also show the solution to Least Square regression (LS) and Least Absolute Deviations regression (LAD) on the same problem. The plots are shown in Figure 8.

From these plots we can see that, although the two quartiles are not inside the confidence interval, they are quite close, even for this value of s. The sampling size in each trial is only $5e4$ which is about 1 percent of the original data; while for bootstrapping, we are resampling the same number of rows as in the original matrix with replacement. In addition, the median of these 50 solutions is in the shaded area and close to the true solution. Indeed, for most of the coefficients, SPC3 can generate 2-digit accuracy. Note that we also compute the exact values of the quartiles; we don't present them here since they are very similar to those in Table 21 in Section 6.2 (where empirical results of large-scale quantile regression problems are shown) in terms of accuracy. All in all, SPC3 performs quite well on this real data.



(a) Intercept (b) Sex (c) Age $\in [30, 40)$

(d) Age $\in [40, 50)$ (e) Age $\in [50, 60)$ (f) Age $\in [60, 70)$

(g) Age ≥ 70 (h) Non_white (i) Unmarried

(j) Education (k) Education^2 (l) Legend

Figure 8: Each subfigure is associated with a coefficient in the census data. The shaded area shows a point-wise 90% confidence interval. The black curve inside is the true solution when τ changes from 0.05 to 0.95. The blue and green lines correspond to the ℓ_2 and ℓ_1 solution, respectively. The two magenta curves show the first and third quartiles of solutions obtained by using SPC3, among 200 independent trials with sampling size $s = 5e4$ (about 1% of the original data).

4

WEIGHTED SGD FOR ℓ_p REGRESSION WITH RANDOMIZED PRECONDITIONING

Among many novel algorithms for large-scale data analysis and machine learning problems that have emerged in recent years, stochastic gradient descent (SGD) methods and randomized linear algebra (RLA) algorithms have received much attention—both for their strong performance in practical applications and for their interesting theoretical properties [Bot10; Mah11]. Consider the ubiquitous ℓ_p regression problems. For these regression problems, SGD algorithms are widely used in practice because of their scalability and efficiency. In contrast, RLA algorithms have better theoretical guarantees but (thus far) have been less flexible, e.g., in the presence of constraints.

In this chapter, we describe a novel RLA-SGD algorithm called pwSGD (preconditioned weighted SGD) for ℓ_p regression. Our new algorithm combines the advantages of both RLA and SGD methods for solving constrained overdetermined ℓ_p regression problems. We prove that PWSGD inherits fast convergence rates that only depend on the lower dimension of the linear system, while maintaining low computational complexity. In Section 4.1 we provide an overview of our algorithm and its connection to related algorithms. Main algorithm and theoretical results are presented in Section 4.2. Empirical evaluation of pwSGD is provided in Section 4.3. Finally, we discuss the connection between RLA algorithms and coreset methods for ℓ_p regression problems in Section 4.4. The material in this chapter appears in Yang et al. [Yan+16a; Yan+16b].

4.1 OVERVIEW AND CONNECTION TO RELATED ALGORITHMS

Consider the overdetermined ℓ_p regression problem, which is previously defined in Definition 2.24,

$$\min_{x \in \mathcal{Z}} f(x) = \| Ax - b \|_p, \tag{4.1}$$

where $p \in [1, \infty)$, $A \in \mathbb{R}^{n \times d}$, $b \in \mathbb{R}^n$ and $n \gg d$. When $\mathcal{Z} = \mathbb{R}^d$, i.e., the solution space is unconstrained, the cases $p \in \{1, 2\}$ are respectively known as the least absolute deviations (LAD or ℓ_1) and least-squares (LS or ℓ_2) regression problems.

As a point of potentially-independent interest, a connection between ℓ_p regression and stochastic optimization will allow us to unify our main algorithm pwSGD and some existing ℓ_p regression solvers under the same framework. In Figure 28, we present the basic structure of this framework, which provides a view of pwSGD from another perspective. To be more specific, we reformulate a (deterministic) overdetermined ℓ_p regression problem of the form (4.1) into a stochastic optimization problem of the form (4.2).

Lemma 4.1. *Let $U \in \mathbb{R}^{n \times d}$ be a basis of the range space of A in the form of $U = AF$, where $F \in \mathbb{R}^{d \times d}$. The constrained overdetermined ℓ_p regression problem (4.1) is equivalent to*

$$\min_{y \in \mathcal{Y}} \|Uy - b\|_p^p = \min_{y \in \mathcal{Y}} \mathbb{E}_{\xi \sim P} [H(y, \xi)], \tag{4.2}$$

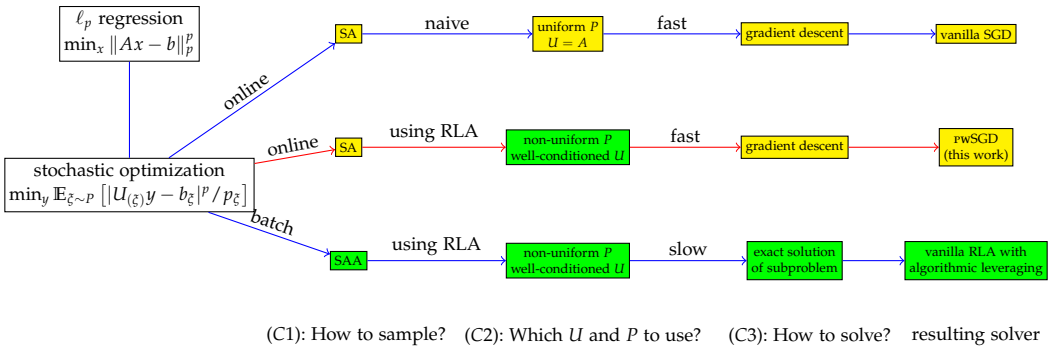


Figure 9: An overview of our framework for solving ℓ_p regression via stochastic optimization. To construct a solver, three choices have to be made. For (C1), the answer can be either SAA (Sampling Average Approximation, i.e., sample a batch of points and deal with the subproblem) or SA (Stochastic Approximation, i.e., sample a mini-batch in an online fashion and update the weight vector after extracting useful information). In (C2), the answer is determined by P, which denotes the underlying sampling distribution (uniform or non-uniform) and U, which denotes the basis with which to work (original or preconditioned system). Finally, for (C3), the answer determines how we solve the subproblem (in SAA) or what information we extract and how we update the weight (in SA).

where ξ is a random variable over $\{1,\ldots,n\}$ with distribution $P=\{p_i\}_{i=1}^n$, y is the decision variable in $\mathcal{Y}$, and $H(y,\xi)=|U_{(\xi)}y-b_\xi|^p/p_\xi$. The constraint set of y is given by $\mathcal{Y}=\{y\in\mathbb{R}^d \mid y=F^{-1}x, x\in\mathcal{Z}\}$.

As suggested in Figure 28, to solve this stochastic optimization problem, typically one needs to answer the following three questions:

(C1) How to sample: SAA (Sampling Average Approximation, i.e., draw samples in a batch mode and deal with the subproblem) or SA (Stochastic Approximation, i.e., draw a mini-batch of samples in an online fashion and update the weight after extracting useful information)?

(C2) Which probability distribution P (uniform distribution or not) and which basis U (preconditioning or not) to use?

(C3) Which solver to use (e.g., how to solve the subproblem in SAA or how to update the weight in SA)?

Some combinations of these choices may lead to existing solvers; see Figure 28 for more details. In the following, we briefly outline several connections. Using the SAA approach with a naive choice of $U=A$ and uniform distribution P leads to the vanilla subsampling algorithm. Importantly, such a simple algorithm might fail (ungracefully) for worst-case input. On the other hand, RLA methods (in particular, those that exploit algorithmic averaging discussed in Section 2.3.2.3 and Section 2.3.2.4) inherit strong theoretical guarantees because the underlying sampling distribution P captures most of the important information of the original system; moreover, such a carefully constructed leverage-based distribution is defined based on a well-conditioned basis U, e.g., an orthogonal matrix for $p=2$ (Theorem 2.19).

A natural question arises: can we leverage the algorithmic benefits of RLA preconditioning to improve the performance of SGD-type algorithms? One immediate idea is to

develop an SGD-like algorithm that uses the same choice of U and P as in RLA methods. Indeed, this simple idea leads to our main algorithm pwSGD, which is an online algorithm ($C1$) that uses a non-uniform sampling distribution ($C2$) and performs a gradient descent update ($C3$) on a preconditioned system ($C2$), as Figure 28 suggests.

Indeed, for least-squares problems (unconstrained ℓ_2 regression), pwSGD is highly related to the weighted randomized Kaczmarz (RK) algorithm [SV09; NWS14] in the way that both algorithms perform SGD updates with non-uniform sampling distribution P. An important difference is that pwSGD runs on a well-conditioned basis U while randomized RK doesn't involve preconditioning. In Section 4.2.6 we show that this preconditioning step dramatically reduces the number of iteration required for pwSGD to converge to a (fixed) desired accuracy.

4.2 MAIN ALGORITHM AND THEORETICAL RESULTS

4.2.1 *Our main algorithm:* pwSGD

Here, we state our main algorithm pwSGD (Algorithm 9) for solving the *constrained* overdetermined ℓ_1 and ℓ_2 regression problems. We summarize the main steps of our main algorithm as follows.

First, we compute a well-conditioned basis U for the range space of A implicitly via a conditioning method; see Table 2 for possible randomized preconditioning methods. We refer this as the "implicit" method, i.e., it focuses on computing $R \in \mathbb{R}^{d\times d}$ such that $U = AR^{-1}$.

Second, we either exactly compute or quickly approximate the ℓ_p leverage scores defined as $\{\|U_{(i)}\|_p^p\}_{i=1}^n$, as $\{\lambda_i\}_{i=1}^n$. To compute leverage scores exactly, we have to form the matrix U explicitly, which takes time $\mathcal{O}(nd^2)$. Alternatively, we can estimate the row norms of U without computing the product between A and R^{-1}, in order to reduce the running time as discussed in Section 2.3.2.3 and Section 2.3.2.4. We assume that $\{\lambda_i\}_{i=1}^n$ satisfy

$$(1-\gamma)\|U_{(i)}\|_p^p \le \lambda_i \le (1+\gamma)\|U_{(i)}\|_p^p, \tag{4.3}$$

where γ is the approximation factor of estimation. When the leverage scores are exact, the approximation factor $\gamma = 0$. From that, we can define a distribution P over $\{1,\dots,n\}$ based on $\{\lambda_i\}_{i=1}^n$ as follows:

$$p_i = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j}. \tag{4.4}$$

Third, in each iteration a new sample corresponding to a row of A is drawn according to distribution P and we apply an SGD process to solve the following equivalent problem with a specific choice of $F \in \mathbb{R}^{d\times d}$:

$$\min_{y\in\mathcal{Y}} h(y) = \|AFy - b\|_p^p = \mathbb{E}_{\xi\sim P}\left[|A_{(\xi)}Fy - b_\xi|^p / p_\xi\right]. \tag{4.5}$$

Here the matrix F is called the preconditioner for the linear system being solved; see Section 4.2.3 for several choices of F. Below, we show that with a suitable choice of F, the convergence rate of the SGD phase can be improved significantly. Indeed, we can perform

Algorithm 9 pwSGD— preconditioned weighted SGD for overdetermined ℓ_1 and ℓ_2 regression

Input: $A \in \mathbb{R}^{n\times d}$, $b \in \mathbb{R}^n$ with $\text{rank}(A) = d$, $x_0 \in \mathcal{Z}$, η and T.
Output: An approximate solution vector to problem $\min_{x\in\mathcal{Z}} \|Ax - b\|_p^p$ for $p = 1$ or 2.
Compute $R \in \mathbb{R}^{d\times d}$ such that $U = AR^{-1}$ is a well-conditioned basis U; see Table 2 for possible options.
Compute or estimate $\|U_{(i)}\|_p^p$ with leverage scores λ_i, for $i \in [n]$, that satisfies (4.3).
Let $p_i = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j}$, for $i \in [n]$.
Construct the preconditioner $F \in \mathbb{R}^{d\times d}$ based on R; see Section 4.2.3 for details.
for $t = 0, \ldots, T$ **do**
  Pick ξ_t from $[n]$ based on distribution $\{p_i\}_{i=1}^n$.

$$c_t = \begin{cases} \text{sgn}\left(A_{(\xi_t)}x_t - b_{\xi_t}\right)/p_{\xi_t} & \text{if } p = 1; \\ 2\left(A_{(\xi_t)}x_t - b_{\xi_t}\right)/p_{\xi_t} & \text{if } p = 2. \end{cases}$$

  Update x by

$$x_{t+1} = \begin{cases} x_t - \eta c_t H^{-1} A_{(\xi_t)} & \text{if } \mathcal{Z} = \mathbb{R}^d; \\ \arg\min_{x\in\mathcal{Z}} \ \eta c_t A_{(\xi_t)} x + \frac{1}{2}\|x_t - x\|_H^2 & \text{otherwise.} \end{cases} \tag{4.8}$$

  where $H = \left(FF^T\right)^{-1}$.
end for
Return $\bar{x}$ for $p = 1$ or x_T for $p = 2$.

the update rule in the original domain (with solution vector x instead of y), i.e., (4.8). Notice that if $\mathcal{Z} = \mathbb{R}^d$ and $F = I$, then the update rule can be simplified as

$$x_{t+1} = x_t - \eta c_t A_{(\xi_t)}. \tag{4.6}$$

If $\mathcal{Z} = \mathbb{R}^d$ and $F = R^{-1}$, then the update rule becomes

$$x_{t+1} = x_t - \eta c_t H^{-1} A_{(\xi_t)}, \tag{4.7}$$

where $H = (R^T R)^{-1}$. In the presence of constraints, (4.8) only needs to solve an optimization problem with a quadratic objective over the same constraints, whose size is independent of n.

Finally, the output is the averaged value over all iterates, i.e., $\bar{x} = \frac{1}{T}\sum_{t=1}^T x_t$, for ℓ_1 regression, or the last iterate, i.e., x_T, for ℓ_2 regression.

4.2.2 *Main results for ℓ_1 and ℓ_2 regression problems*

The quality-of-approximation of Algorithm 9 is presented in Theorem 4.2 and Theorem 4.3 for ℓ_1 and ℓ_2 regression, respectively, in which we give the expected number of iterations that pwSGD needs for convergence within small tolerance. We show that pwSGD inherits a

convergence rate of $\mathcal{O}\left(1/\sqrt{T}\right)$ for ℓ_1 regression and $\mathcal{O}\left(\log T/T\right)$ for ℓ_2 regression and the constant term only depends on the lower dimension d when $F = R^{-1}$. Worth mentioning is that for ℓ_2 regression, our bound on the solution vector is measured in prediction norm, i.e., $\|Ax\|_2$. All the proofs can be found in Appendix B.

In the following results, R is the matrix computed in step 3 in Algorithm 9, $\{\lambda_i\}_{i\in[n]}$, are the leverage scores computed in step 4, F is the preconditioner chosen in step 6 in Algorithm 9 and $H = \left(FF^T\right)^{-1}$. Denote by $\bar{\kappa}_p(U)$ the condition number of the well-conditioned basis $U = AR^{-1}$ and γ the approximation factor of the leverage scores λ_i, $i \in [n]$, that satisfies (4.3). For any vector $x \in \mathbb{R}^d$, denote by $\|x\|_H^2 = x^T Hx$ the ellipsoidal norm of x induced by matrix $H = H^T \succ 0$. For any nonsingular matrix A, denote $\kappa(A) = \|A\|_2\|A^{-1}\|_2$ and $\hat{\kappa}(A) = |A|_1|A^{-1}|_1$ where $|\cdot|_1$ is the element-wise ℓ_1 norm. The exact form of the step-sizes used can be found in the proofs.

Theorem 4.2. *For $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$, define $f(x) = \|Ax - b\|_1$ and suppose $f(x^*) > 0$. Then there exists a step-size η such that after*

$$T = d\bar{\kappa}_1^2(U)\hat{\kappa}^2(RF)\frac{c_1^2 c_2 c_3^2}{\epsilon^2}$$

iterations, Algorithm 9 with $p = 1$ returns a solution vector estimate $\bar{x}$ that satisfies the expected relative error bound

$$\frac{\mathbb{E}\left[f(\bar{x})\right] - f(x^*)}{f(x^*)} \leq \epsilon.$$

Here, the expectation is taken over all the samples $\xi_1, \ldots, \xi_T$ and x^* is the optimal solution to the problem $\min_{x\in\mathcal{Z}} f(x)$. The constants in T are given by $c_1 = \frac{1+\gamma}{1-\gamma}$, $c_2 = \frac{\|x^* - x_0\|_H^2}{\|x^*\|_H^2}$ and $c_3 = \|Ax^*\|_1/f(x^*)$.

Theorem 4.3. *For $A \in \mathbb{R}^{n\times d}$ and $b \in \mathbb{R}^n$, define $f(x) = \|Ax - b\|_2$ and suppose $f(x^*) > 0$. Then there exists a step-size η such that after*

$$T = c_1\bar{\kappa}_2^2(U)\kappa^2(RF)\cdot\log\left(\frac{2c_2\kappa^2(U)\kappa^2(RF)}{\epsilon}\right)\cdot\left(1 + \frac{\kappa^2(U)\kappa^2(RF)}{c_3\epsilon}\right)$$

iterations, Algorithm 9 with $p = 2$ returns a solution vector estimate x_T that satisfies the expected relative error bound

$$\frac{\mathbb{E}\left[\|A(x_T - x^*)\|_2^2\right]}{\|Ax^*\|_2^2} \leq \epsilon.$$

Furthermore, when $\mathcal{Z} = \mathbb{R}^d$ and $F = R^{-1}$, there exists a step-size η such that after

$$T = c_1\bar{\kappa}_2^2(U)\cdot\log\left(\frac{c_2\kappa^2(U)}{\epsilon}\right)\cdot\left(1 + \frac{2\kappa^2(U)}{\epsilon}\right)$$

iterations, Algorithm 9 with $p = 2$ returns a solution vector estimate x_T that satisfies the expected relative error bound

$$\frac{\mathbb{E}\left[f(x_T)\right] - f(x^*)}{f(x^*)} \leq \epsilon.$$

Here, the expectation is taken over all the samples $\xi_1, \ldots, \xi_T$, *and* x^* *is the optimal solution to the problem* $\min_{x \in \mathcal{Z}} f(x)$. *The constants in* T *are given by* $c_1 = \frac{1+\gamma}{1-\gamma}$, $c_2 = \frac{\|x^* - x_0\|_H^2}{\|x^*\|_H^2}$, $c_3 = \|Ax^*\|_2^2 / f(x^*)^2$.

The above results indicate two important properties of pwSGD. First recall that the condition number $\bar{\kappa}_p(U)$ of the well-conditioned basis U is a polynomial of d that is independent of n (Section 2.3). Thus with a preconditioner $F = R^{-1}$ and an appropriate step-size in pwSGD, the number of iterations T required to achieve an arbitrarily low relative error only depends on the *low dimension* d of the input matrix A. Second, pwSGD is *robust* to leverage score approximations, i.e., the expected convergence rate will only be affected by a small distortion factor even when the approximation has low accuracy, such as $\gamma = 0.5$.

Remark 7. *For constrained* ℓ_2 *regression, the bound is on the solution vector measured in prediction norm. By the triangular inequality, this directly implies* $(\mathbb{E}[f(x_T)] - f(x^*))/f(x^*) \leq \sqrt{c_3 \epsilon}$.

4.2.3 *The choice of the preconditioner F*

As we can see, the preconditioner F plays an important role in our algorithm. It converts the original regression problem in (4.1) to the stochastic optimization problem in (4.5). From Theorem 4.2 and Theorem 4.3, clearly, different choices of F lead to different convergence rates in the SGD phase (reflected in $\kappa(RF)$[1]) and additional computational costs (reflected in H in (4.8)).

When $F = R^{-1}$, the effect of $\kappa(RF)$ or $\hat{\kappa}(RF)$ on T vanishes. In this case, H is also a good approximation to the Hessian A^TA. This is because usually R is the R-factor in the QR decomposition of SA, where SA is a "sketch" of A satisfying (2.2) that shares similar properties with A. Together we have $H = R^TR = (SA)^T(SA) \approx A^TA$. This implies (4.7) is close to the Newton-type update. However, as a tradeoff, since H^{-1} is a $d \times d$ dense matrix, an additional $\mathcal{O}(d^2)$ cost per iteration is required to perform SGD update (4.8).

On the other hand, when $F = I$, no matrix-vector multiplication is needed in updating x. However, based on the discussion above, one should expect $\kappa(R) = \kappa(SA)$ to be close to $\kappa(A)$. Then the term $\kappa(RF) = \kappa(R)$ can be large if A is poorly conditioned, which might lead to undesirable performance in the SGD phase.

Besides the obvious choices of F such as R^{-1} and I, one can choose F to be a diagonal preconditioner D that scales R to have unit column norms. According to van der Sluis [van69], the condition number after preconditioning $\kappa(RD)$ is always upper bounded by the original condition number $\kappa(R)$, while the additional cost per iteration to perform SGD updates with diagonal preconditioner is only $\mathcal{O}(d)$. In Section 4.3 we illustrate the tradeoffs among these three choices of preconditioners empirically.

4.2.4 *Complexities*

Here, we discuss the complexity of pwSGD with $F = R^{-1}$. The running time of Algorithm 9 consists of three parts. First, for computing a matrix R such that $U = AR^{-1}$ is well-conditioned; see Table 2 for the running time $time(R)$ and preconditioning quality

1 It is also reflected in $\hat{\kappa}(RF)$; however, it depends on $\kappa(RF)$ because one can show $m_1\kappa(RF) \leq \hat{\kappa}(RF) \leq m_2\kappa(RF)$, where m_1, m_2 are constants derived using matrix norm equivalences.

$\bar{\kappa}_p(U)$ of various methods.[2] Second, to estimate the leverage scores, i.e., the row norms of AR^{-1}, Drineas et al. [Dri+12] and Clarkson et al. [Cla+13] proposed several algorithms for approximating the ℓ_1 and ℓ_2 leverage scores without forming matrix U. For a target constant approximation quality, e.g., $\gamma = 0.5$ and $c_1 = \frac{1+\gamma}{1-\gamma} = 3$, the running time of these algorithms is $\mathcal{O}(\log n \cdot \mathtt{nnz}(A))$. Third, Theorem 4.2 and Theorem 4.3 provide upper bounds for the expected algorithmic complexity of our proposed SGD algorithm when a target accuracy is fixed. Combining these, we have the following results.

Theorem 4.4. *Suppose the preconditioner in step 3 of Algorithm 9, is chosen from Table 2, with constant probability, one of the following events holds for* pwSGD *with* $F = R^{-1}$. *To return a solution* $\tilde{x}$ *with relative error* ϵ *on the objective,*

- *It runs in* $time(R) + \mathcal{O}(\log n \cdot nnz(A) + d^3\bar{\kappa}_1(U)/\epsilon^2)$ *for unconstrained* ℓ_1 *regression.*
- *It runs in* $time(R) + \mathcal{O}(\log n \cdot nnz(A) + time_{update} \cdot d\bar{\kappa}_1(U)/\epsilon^2)$ *for constrained* ℓ_1 *regression.*
- *It runs in* $time(R) + \mathcal{O}(\log n \cdot nnz(A) + d^3 \log(1/\epsilon)/\epsilon)$ *for unconstrained* ℓ_2 *regression.*
- *It runs in* $time(R) + \mathcal{O}(\log n \cdot nnz(A) + time_{update} \cdot d \log(1/\epsilon)/\epsilon^2)$ *for constrained* ℓ_2 *regression.*

In the above, $time(R)$ *denotes the time for computing the matrix* R *and* $time_{update}$ *denotes the time for solving the optimization problem in* (4.8).

Notice that, since $time_{update}$ only depends on d, an immediate conclusion is that by using sparse preconditioning methods, to find an ϵ-approximate solution, pwSGD runs in $\mathcal{O}(\log n \cdot \mathtt{nnz}(A) + \mathtt{poly}(d)/\epsilon^2)$ time for ℓ_1 regression and in $\mathcal{O}(\log n \cdot \mathtt{nnz}(A) + \mathtt{poly}(d)\log(1/\epsilon)/\epsilon)$ time for ℓ_2 regression (in terms of solution vector in prediction norm for constrained problems or objective value for unconstrained problems).

Also, as can be seen in Theorem 4.4, for the complexity for ℓ_1 regression, the tradeoffs in choosing preconditioners from Table 2 are reflected here. On the other hand, for ℓ_2 regression, as all the preconditioning methods in Table 2 provide similar preconditioning quality, i.e., $\kappa(U) = \kappa_2(U) = \mathcal{O}(1)$, $time(R)$ becomes the key factor for choosing a preconditioning method. In Table 8, we summarize the complexity of pwSGD using various underlying preconditioning methods for solving unconstrained ℓ_1 and ℓ_2 regression problems. We remark that, with decaying step-sizes, it is possible to improve the dependence on ϵ from $\log(1/\epsilon)/\epsilon$ to $1/\epsilon$ [RSS12].

To provide a quick overview of how pwSGD compares to existing algorithms, in Tables 9 and 10, we summarize the complexity required to compute a solution $\hat{x}$ with relative error $(f(\hat{x}) - f(x^*))/f(x^*) = \epsilon$, of several solvers for unconstrained ℓ_1 and ℓ_2 regression. In Table 9, RLA with algorithmic leveraging (RLA for short) [Cla+13; YMM14] is a popular method for obtaining a low-precision solution and randomized IPCPM is an iterative method for finding a higher-precision solution [MM13b] for unconstrained ℓ_1 regression. Clearly, pwSGD has a uniformly better complexity than that of RLA methods in terms of both d and ϵ, no matter which underlying preconditioning method is used. This makes pwSGD a more suitable candidate for getting a medium-precision, e.g., $\epsilon = 10^{-3}$, solution.

In Table 10, all the methods require constructing a sketch first. Among them, "low-precision" solvers refer to "sketching + direct solver" type algorithms; see [Dri+11; CW13a] for projection-based examples and [CW13a; Dri+12] for sampling-based examples. "High-precision" solvers refer to "sketching + preconditioning + iterative solver" type algorithms; see [AMT10; MSM14] for examples. One can show that, when $d \geq 1/\epsilon$ and

2 In Table 2, only values of κ_p are presented. However, using Lemma 2.5, one can compute the values of $\bar{\kappa}_p$ correspondingly.

Table 8: Summary of complexity of pwSGD with different underlying preconditioning methods when solving unconstrained ℓ_1 and ℓ_2 regression problems. The target is to return a solution $\tilde{x}$ with relative error ϵ on the objective. Here, the complexity of each algorithm is calculated by setting the failure probability to be a constant.

TYPE	PRECONDITIONER	COMPLEXITY	REFERENCE
ℓ_1	Cauchy	$\mathcal{O}(nd^2 \log n + d^3 \log d + d^{\frac{11}{2}} \log^{\frac{3}{2}} d/\epsilon^2)$	[SW11]
	Fast Cauchy	$\mathcal{O}(nd \log n + d^3 \log^5 d + d^{\frac{17}{2}} \log^{\frac{9}{2}} d/\epsilon^2)$	[Cla+13]
	Sparse Cauchy	$\mathcal{O}(\texttt{nnz}(A) \log n + d^7 \log^5 d + d^{\frac{19}{2}} \log^{\frac{11}{2}} d/\epsilon^2)$	[MM13a]
	Reciprocal exponential	$\mathcal{O}(\texttt{nnz}(A) \log n + d^3 \log d + d^{\frac{13}{2}} \log^{\frac{5}{2}} d/\epsilon^2)$	[WZ13]
	Lewis weights	$\mathcal{O}(\texttt{nnz}(A) \log n + d^3 \log d + d^{\frac{9}{2}} \log^{\frac{1}{2}} d/\epsilon^2)$	[CP15]
ℓ_2	Gaussian	$\mathcal{O}(nd^2 + d^3 \log(1/\epsilon)/\epsilon)$	[DS01]
	SRHT	$\mathcal{O}(nd \log n + d^3 \log n \log d + d^3 \log(1/\epsilon)/\epsilon)$	[Tro11]
	Sparse ℓ_2 embedding	$\mathcal{O}(\texttt{nnz}(A) \log n + d^3 \log d + d^3 \log(1/\epsilon)/\epsilon)$	[Coh16]
	Refinement sampling	$\mathcal{O}(\texttt{nnz}(A) \log(n/d) \log d + d^3 \log(n/d) \log d + d^3 \log(1/\epsilon)/\epsilon)$	[Coh+15]

$n \geq d^2/\epsilon$, pwSGD is asymptotically better than all the solvers shown in Table 10. Moreover, although high-precision solvers are more efficient when a high-precision solution is desired, usually they are designed for unconstrained problems, whereas pwSGD also works for constrained problems.

We remark that, compared to general SGD algorithms, our RLA-SGD hybrid algorithm pwSGD *works for problems in a narrower range, i.e., ℓ_p regression, but inherits the strong theoretical guarantees of RLA. When solving ℓ_2 regression, for which traditional RLA methods are well designed,* pwSGD *has a comparable complexity. On the other hand, when solving ℓ_1 regression, due to the efficiency of SGD update,* pwSGD *has a strong advantage over traditional RLA methods.*

Table 9: Summary of complexity of several unconstrained ℓ_1 solvers that use randomized linear algebra. The target is to find a solution $\hat{x}$ with accuracy $(f(\hat{x}) - f(x^*))/f(x^*) \leq \epsilon$, where $f(x) = \|Ax - b\|_1$. In the above, $time(R)$ denotes the time needed to compute a matrix R such that AR^{-1} is well-conditioned with condition number $\bar{\kappa}_1$ (Definition 2.4). The general complexity bound and the one using sparse reciprocal exponential transform (Lemma 2.18) as the underlying preconditioning method are presented. Here, we assume $n \gg d$ such that $n > d^3 \log d$ and the underlying ℓ_1 regression solver in RLA with algorithmic leveraging algorithm takes $\mathcal{O}(n^{\frac{5}{4}} d^3)$ time to return a solution [PK97]. The complexity of each algorithm is computed by setting the failure probability to be a constant.

SOLVER	COMPLEXITY (GENERAL)	COMPLEXITY (SPARSE)
RLA with algorithmic leveraging	$time(R) + \mathcal{O}(\texttt{nnz}(A) \log n + \bar{\kappa}_1^{\frac{5}{4}} d^{\frac{17}{4}} / \epsilon^{\frac{5}{2}})$	$\mathcal{O}(\texttt{nnz}(A) \log n + d^{\frac{69}{8}} \log^{\frac{25}{8}} d/\epsilon^{\frac{5}{2}})$
randomized IPCPM	$time(R) + nd^2 + \mathcal{O}((nd + \texttt{poly}(d)) \log(\bar{\kappa}_1 d/\epsilon))$	$nd^2 + \mathcal{O}((nd + \texttt{poly}(d)) \log(d/\epsilon))$
pwSGD	$time(R) + \mathcal{O}(\texttt{nnz}(A) \log n + d^3 \bar{\kappa}_1/\epsilon^2)$	$\mathcal{O}(\texttt{nnz}(A) \log n + d^{\frac{13}{2}} \log^{\frac{5}{2}} d/\epsilon^2)$

Table 10: Summary of complexity of several unconstrained ℓ_2 solvers that use randomized linear algebra. The target is to find a solution $\hat{x}$ with accuracy $(f(\hat{x}) - f(x^*))/f(x^*) \leq \epsilon$, where $f(x) = \|Ax - b\|_2$. Two sketching methods, namely, SRHT [Dri+11; Tro11] and sparse embedding [CW13a] are considered. Here, we assume $d \leq n \leq e^d$. The complexity of each algorithm is computed by setting the failure probability to be a constant.

SOLVER	COMPLEXITY (SRHT)	COMPLEXITY (SPARSE)
low-precision solvers (projection)	$\mathcal{O}\left(nd \log(d/\epsilon) + d^3 \log n (\log d + 1/\epsilon)\right)$	$\mathcal{O}\left(\texttt{nnz}(A) + d^4/\epsilon^2\right)$
low-precision solvers (sampling)	$\mathcal{O}\left(nd \log n + d^3 \log n \log d + d^3 \log d/\epsilon\right)$	$\mathcal{O}\left(\texttt{nnz}(A) \log n + d^4 + d^3 \log d/\epsilon\right)$
high-precision solvers	$\mathcal{O}\left(nd \log n + d^3 \log n \log d + nd \log(1/\epsilon)\right)$	$\mathcal{O}\left(\texttt{nnz}(A) + d^4 + nd \log(1/\epsilon)\right)$
pwSGD	$\mathcal{O}\left(nd \log n + d^3 \log n \log d + d^3 \log(1/\epsilon)/\epsilon\right)$	$\mathcal{O}\left(\texttt{nnz}(A) \log n + d^4 + d^3 \log(1/\epsilon)/\epsilon\right)$

4.2.5 *Complexity comparison between* pwSGD *and RLA*

As we pointed out in Section 4.1, pwSGD and RLA methods with algorithmic leveraging (RLA for short) are closely related as they can be viewed as methods using SA and SAA to solve the stochastic optimization problem (4.2). Omitting the time for computing basis U and sampling distribution P, the comparison of complexity boils down to comparing $time_{sub}(s,d)$ (for RLA) and $time_{update} \cdot T$ (for pwSGD) where $time_{sub}(s,d)$ is the time needed to solve the same constrained regression problem with size s by d and $time_{update}$ denotes the time needed for to solve the optimization problem in (4.8). According to the theory, for the same target accuracy, the required s (sampling size) and T (number of iterations) are the same asymptotically, up to logarithmic factors; see Theorem 2.22 for expression of s. When the problem is unconstrained, due to the efficiency of SGD, $time_{update} = \mathcal{O}(d^2)$ as indicated in (4.8). For ℓ_2 regression, due to the efficiency of the direct solver, $time_{sub}(s,d) = \mathcal{O}(sd^2)$. This explains why pwSGD and RLA (low-precision solvers (sampling)) have similar complexities as shown in Table 10. On the other hand, for unconstrained ℓ_1 regression, a typical ℓ_1 regression solver requires time $time_{sub}(s,d) > sd^2$. For example, if an interior point method is used [PK97], $time_{sub}(s,d)$ is not even linear in s. This explains the advantage pwSGD has over RLA as shown in Table 9. We also note that in the presence of constraints, pwSGD may still be more efficient for solving ℓ_1 regression because roughly speaking, $time_{sub}(s,d)/s > time_{update}$.

4.2.6 *Connection to weighted randomized Kaczmarz algorithm*

As mentioned in Section 4.1, our algorithm pwSGD for least-squares regression is related to the weighted randomized Kaczmarz (RK) algorithm [SV09; NWS14]. To be more specific, weighted RK algorithm can be viewed as an SGD algorithm with constant step-size that exploits a sampling distribution based on row norms of A, i.e., $p_i = \|A_{(i)}\|_2^2 / \|A\|_F^2$. In pwSGD, if the preconditioner $F = R^{-1}$ is used and the leverage scores are computed exactly, the resulting algorithm is equivalent to applying the weighted randomized Karczmarz algorithm on a well-conditioned basis $U = AR^{-1}$ since leverage scores are defined as the row norms of U.

Since the matrix A itself can be a basis for its range space, setting $U = A$ and $F = R = I$ in Theorem 4.3 indicates that weighted RK algorithm inherits a convergence rate that depends on condition number $\kappa(A)$ times the scaled condition number $\bar{\kappa}_2(A)$. Notice that in pwSGD, the preconditioning step implicitly computes a basis U such that both $\kappa(U)$ and $\bar{\kappa}(U)$ are low. One should expect the SGD phase in pwSGD inherits a faster convergence rate, as verified numerically in Section 4.3.

4.3 EMPIRICAL EVALUATION

In this section, we provide empirical evaluations of our main algorithm pwSGD. We evaluate its convergence rate and overall running time on both synthetic and real datasets. For pwSGD, we implement it with three different choices of the preconditioner F. Herein, throughout the experiments, by pwSGD-full, pwSGD-diag, pwSGD-noco, we respectively mean pwSGD with preconditioner $F = R^{-1}, D, I$; see Section 4.2.3 for details. Note that, for pwSGD, we use the methods from Clarkson and Woodruff [CW13a] for preconditioning. Also, we exactly compute the row norms of AR^{-1} and use them as the leverage scores. In each experiment, the initial solution vector estimate is set as zero. The above algorithms

are then run in the following manner. Each epoch contains $\lceil n/10 \rceil$ iterations. At the beginning of each epoch, we sample $\lceil n/10 \rceil$ indices according to the underlying distribution without replacement and update the weight using the $\lceil n/10 \rceil$ row samples from the data matrix. Finally, the plots are generated by averaging the results over 20 independent trials.

4.3.1 *Experiments on synthetic datasets*

Theoretically the major advantage of pwSGD is the fast convergence rate. To evaluate its performance, we compare the convergence rate of relative error, i.e., $|\hat{f} - f^*|/f^*$, with other competing algorithms including vanilla SGD and fully weighted randomized Kaczmarz (weighted-RK) algorithm ([NWS14; SV09]) for solving least-squares problem (unconstrained ℓ_2 regression). For each of these methods, given a target relative error $\epsilon = 0.1$ on the objective, i.e., $(\hat{f} - f^*)/f^* = 0.1$, we use the optimal step-size suggested in the theory. In particular, for pwSGD, we are showing the convergence rate of the SGD phase after preconditioning. We stop the algorithm when the relative error reaches ϵ. In this task, we use synthetic datasets for better control over the properties on input matrices A and b. Each dataset has size 1000 by 10. The design matrix A is of the form $A = U\Sigma V^T$ where $U \in \mathbb{R}^{1000\times 10}$ and $V \in \mathbb{R}^{10\times 10}$ are random orthonormal matrices and $\Sigma \in \mathbb{R}^{10\times 10}$ is a diagonal matrix that controls $\kappa(A)$. The true solution x^* is a standard Gaussian vector and the response vector b is set to be Ax^* corrupted by some Gaussian noise with standard deviation 0.1.

We investigate the relation between the condition number of A and convergence rate. By Theorem 4.3, for weighted SGD algorithm, the number of iterations required to solve an ℓ_2 regression problem is proportional to $\bar{\kappa}_2^2(A)\kappa^2(A) = \|(A^TA)^{-1}\|_2^2\|A\|_2^2\|A\|_F^2 \leq \bar{\kappa}_2^4(A)$. To verify this hypothesis, we generate a sequence of A matrices using `Synthetic 2` dataset with increasing $\bar{\kappa}_2^4(A)$ values such that U and V in the sequence are constants.[3] This construction ensures that all other properties such as leverage scores and coherence (the largest leverage score) remain unchanged. We present the experimental results (number of iterations required for different methods versus $\bar{\kappa}_2^4(A)$) for the synthetic dataset in Figure 10.

As shown in Figure 10, the required number of iterations of all the methods except for pwSGD-full scales linearly in $\bar{\kappa}_2^4(A)$. This phenomenon matches the result predicted in theory. A significant advantage of pwSGD-full over other methods is its robust convergence rate against variations in $\bar{\kappa}_2^4(A)$. This is mainly because its SGD phase operates on a well-conditioned basis after preconditioning and the preconditioning quality of pwSGD-full depends only on the low-dimension of A; thus increasing $\bar{\kappa}_2^4(A)$ has little effect on changing its convergence rate. Also, while the diagonal preconditioner in pwSGD-diag reduces the condition number, i.e., $\kappa(RD) \leq \kappa(R)$, its convergence rate still suffers from the increase of $\bar{\kappa}_2^4(A)$.

4.3.2 *Time-accuracy tradeoffs*

We present the time-accuracy tradeoffs among these methods on the following two datasets.

Here we test the performance of various methods in solving unconstrained ℓ_1 and ℓ_2 regression problems. Although there are no theoretical results to support the solution

3 In `Synthetic 2`, U and V are fixed. Σ is of the form $\mathrm{diag}(\sigma_1, \ldots, \sigma_d)$ where $\sigma_i = 1 + (i-1)q$ for $i \in [d]$. We solve for q such that $\sum_{i=1}^d \sigma_i^2 = \bar{\kappa}_2^2(U)$ for any desired value $\bar{\kappa}_2^2(U)$.

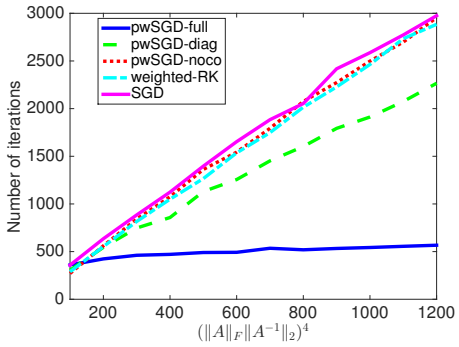


Figure 10: Convergence rate comparison of several SGD-type algorithms including pwSGD with three different choices of preconditioners for solving ℓ_2 regression on `Synthetic 2` datasets with increasing condition number. For each method, the optimal step-size is set according to the theory with target accuracy $|f(\hat{x}) - f(x^*)|/f(x^*) = 0.1$. The y-axis is showing the minimum number of iterations for each method to find a solution with the target accuracy.

Table 11: Summary of datasets.

NAME	#ROWS	# COLUMNS	$\kappa(A)$
`Year`[4]	5×10^5	90	2×10^3
`Buzz`[5]	5×10^5	77	10^8

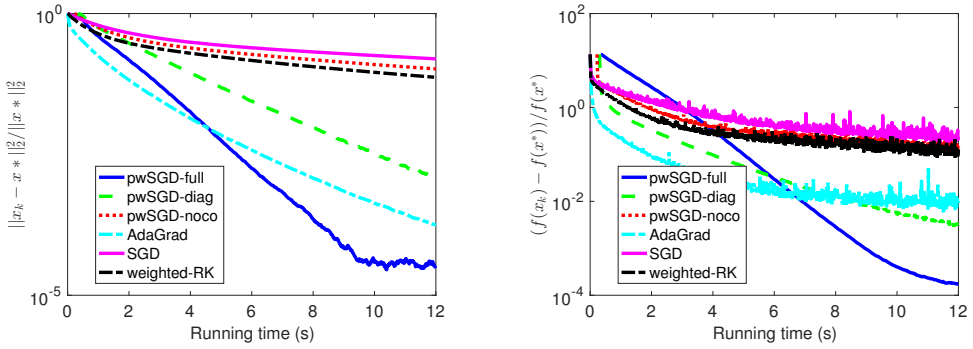
vector convergence on ℓ_1 regression problems with pwSGD, we still evaluate relative error in the solution vector. To further examine the performance of pwSGD methods, we also include AdaGrad [DHS11] and RLA methods with algorithmic leveraging (RLA for short) described in Section 2.3.2.3 and Section 2.3.2.4 for comparisons. For AdaGrad, we use diagonal scaling and mirror descent update rule. As for implementation details, in all SGD-like algorithms, step-size tuning is done by grid-searching where at each trial the algorithm is run with a candidate step-size for enough iterations. Then the step-size that yields the lowest error within 25 seconds is used. The time/accuracy pair at every 2000 iterations is recorded. For RLA, we choose s from a wide range of values and record the corresponding time/accuracy pairs. The results on the two datasets are presented in Figure 11 and Figure 12, respectively.

As we can see in Figure 11 and Figure 12, in our algorithm pwSGD, a faster convergence comes with the additional cost of preconditioning. For example, the preconditioning phase of pwSGD takes approximately 3 seconds. Nevertheless, with a faster convergence rate in a well-conditioned basis, pwSGD-full still outperforms other methods in converging to a higher-precision solution at a given time span. As pwSGD-diag balances convergence rate and computational cost, it outperforms pwSGD-full at the early stage and yields comparable results to AdaGrad. As expected, due to the poor conditioning, SGD, weighted-RK and pwSGD suffer from slow convergence rates. As for RLA methods, they have the same first step as in pwSGD, i.e., preconditioning and constructing the sampling distribution. For ℓ_1 regression, to obtain a fairly high-precision solution, the sampling size has to be fairly large, which might drastically increase the computation time for solving the sampled subproblem. This explains the advantage of pwSGD-full over RLA methods in Figure 11. It is worth mentioning that, although for ℓ_2 regression our theory provides

relative error bound on the solution vector measured in the prediction norm, here we also see that pwSGD-full and pwSGD-diag display promising performance in approximating the solution vector measured in ℓ_2 norm.

We also notice that on `Buzz` (Figure 12) , all the methods except for pwSGD-full and pwSGD-diag have a hard time converging to a solution with low solution error. This is due to the fact that `Buzz` has a high condition number. The advantage of applying a preconditioner is manifested.

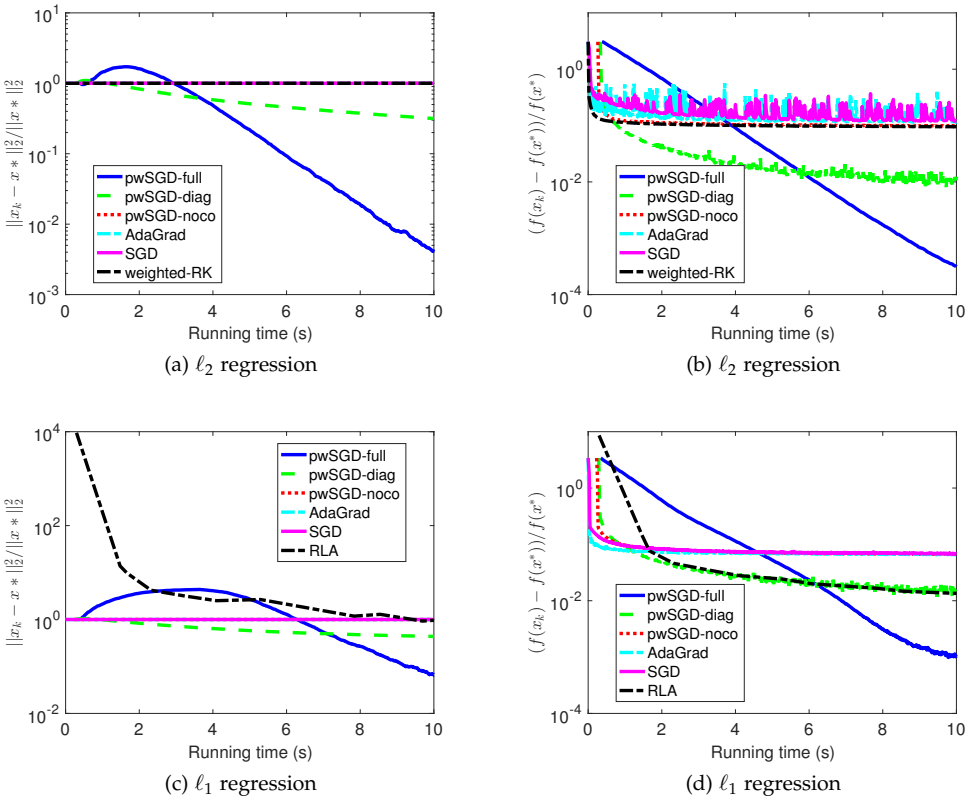
Finally, notice that RLA uses a high performance direct solver to solve the medium-size subsampled problem for ℓ_2 regression. In this case pwSGD methods do not show significant advantages over RLA in terms of speed. For this reason we have not included RLA results in Figure 11a and Figure 11b. Nevertheless, pwSGD methods may still be favorable over RLA in terms of speed and feasibility when the size of the dataset becomes increasingly larger, e.g., 10^7 by 500.



(a) ℓ_2 regression (b) ℓ_2 regression

(c) ℓ_1 regression (d) ℓ_1 regression

Figure 11: Time-accuracy tradeoffs of several algorithms including pwSGD with three different choices of preconditioners on `year` dataset. Both ℓ_1 and ℓ_2 regressions are tested and the relative error on both the objective value, i.e., $|f(\hat{x}) - f(x^*)|/f(x^*)$, and the solution vector, i.e., $\|\hat{x} - x^*\|_2^2/\|x^*\|_2^2$, are measured.



(a) ℓ_2 regression (b) ℓ_2 regression

(c) ℓ_1 regression (d) ℓ_1 regression

Figure 12: Time-accuracy tradeoffs of several algorithms including pwSGD with three different choices of preconditioners on buzz dataset. Both ℓ_1 and ℓ_2 regressions are tested and the relative error on both the objective value, i.e., $|f(\hat{x}) - f(x^*)|/f(x^*)$, and the solution vector, i.e., $\|\hat{x} - x^*\|_2^2/\|x^*\|_2^2$, are measured.

4.3.3 *Experiments with sparse ℓ_2 regression*

Finally, we evaluate our algorithm on a constrained problem—sparse ℓ_2 regression, which is a special case of (4.1). The problem formulation is as follows. Given a matrix $A \in \mathbb{R}^{n\times d}$ and a vector $b \in \mathbb{R}^n$, we want to solve the constrained problem

$$\min_{\|x\|_1 \le R} \|Ax - b\|_2, \tag{4.9}$$

where R controls the size of the ℓ_1-ball constraint.

When using pwSGD, according to (4.8) in Algorithm 9, at each iteration, a sparse ℓ_2 regression problem with size d by d needs to be solved. Here, to use the samples more efficiently, we use a mini-batch version of pwSGD. That is, in Step 8-10 of Algorithm 9, rather than picking only one row from A to compute the noisy gradient, we select m rows and average the scaled version of them. Doing this allows us to reduce the variance of the noisy gradient. In our experiments, we set $m = 200$.

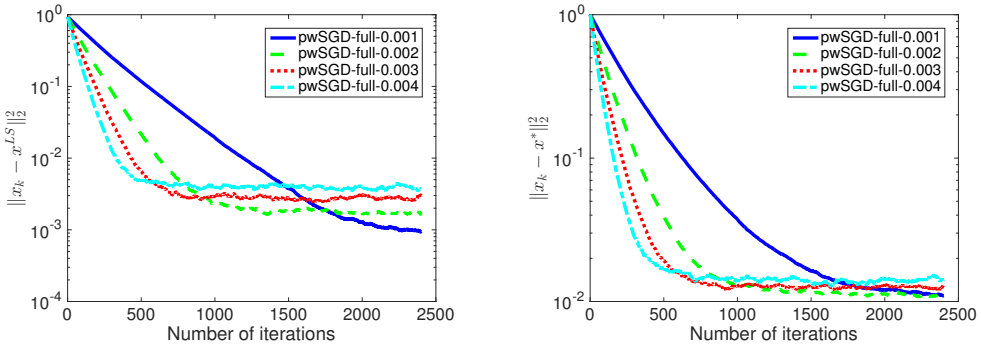
In this task, the observation model is generated in the following manner. $b = Ax^* + r$ where $A \in \mathbb{R}^{n\times d}$ has independent standard normal entries, x^* has s nonzero entries

and noise vector $r \in \mathbb{R}^n$ has independent standard normal entries. We evaluate both the optimization error $\|\hat{x} - x^{LS}\|_2$ and statistical error $\|\hat{x} - x^*\|_2$ of pwSGD-full with several choices of stepsize η where x^{LS} the optimal solution of problem (4.9). By [HTW15], the least-squares error of x^{LS} is $\|x^{LS} - x^*\|_2 \approx \sqrt{s\log(ed/s)/n}$. The statistical error can be bounded using triangle inequality as shown below,

$$\|\hat{x} - x^*\|_2 \leq \|\hat{x} - x^{LS}\|_2 + \|x^{LS} - x^*\|_2.$$

Therefore, the statistical error $\|\hat{x} - x^*\|_2$ is dominated by the least-squares error $\|x^{LS} - x^*\|_2$ when the optimization error $\|\hat{x} - x^{LS}\|_2$ is small.

In Figure 13, we show the results on a data instance with $n = 1e4$, $d = 400$ and $s = 30$. Here R is set to be $R = \|x^*\|_1$ for the experimental purpose. First, we briefly describe the effect of stepsize η. When a constant stepsize is used, typically, a smaller η allows the algorithm to converge to a more accurate solution with a slower convergence rate. This is verified by Figure 13a in which the performance of pwSGD-full with larger η's saturates earlier at a coarser level while $\eta = 0.001$ allows the algorithm to achieve a finer solution. Nevertheless, as discussed above, the statical error is typically dominated by the least-squares error. For our choice of (n, d, s), one can show that the least-squares error $\|x^{LS} - x^*\|_2^2 \approx 0.01$. Therefore, the statistical error shown in Figure 13b is around 0.01 when the optimization error is small enough.



(a) Optimization error $\|x_k - x^{LS}\|_2^2$ (b) Statistical error $\|x_k - x^*\|_2^2$

Figure 13: Performance of pwSGD-full on a synthetic sparse ℓ_2 regression problem with difference choices of stepsize η. Both optimization error and statistical error are shown.

4.4 CONNECTION WITH CORESET METHODS

After viewing RLA and SGD from the stochastic optimization perspective and using that to develop our main algorithm, a natural question arises: can we do this for other types of problems? To do so, we need to define "leverage scores" for them, since they play a crucial role in this stochastic framework. Here, we first describe the coreset framework of Feldman and Langberg [FL11]. Then we show that—on ℓ_p regression problems—two key notions (leverage scores from RLA and sensitivities from coresets) correspond. Finally we will show what amounts to a negative result (i.e., a lower bound) for other problems. Note

here, in this section, we work on constrained ℓ_p regression (4.1) with $p \in [1, \infty)$ and we use $\bar{A}$ to denote the augmented linear system $\begin{pmatrix} A & b \end{pmatrix}$.

4.4.1 *Short summary of coreset methods*

In [FL11], the authors propose a framework for computing a coreset of $\mathcal{F}$ to a given optimization problem of the form

$$\text{cost}(\mathcal{F}, x) = \min_{x \in \mathcal{X}} \sum_{f \in \mathcal{F}} f(x),$$

where $\mathcal{F}$ is a set of functions from a set $\mathcal{X}$ to $[0, \infty)$. By Lemma 4.1, it is not hard to see, the ℓ_p regression problem (4.1) can be written as

$$\min_{x \in \mathcal{C}} \sum_{i=1}^{n} f_i(x),$$

where $f_i(x) = |\bar{A}_{(i)} x|^p$ and $\mathcal{C} = \{x \in \mathbb{R}^{d+1} | x_{d+1} = -1\}$, in which case one can define a set of functions $\mathcal{F} = \{f_i\}_{i=1}^n$.

Central to the coreset method of [FL11] is the following notion of sensitivity, which is used to construct importance sampling probabilities, as shown in Algorithm 10, and the dimension of the given class of function, which is based as Definition 6.1 in [FL11]. They are defined below.

Definition 4.5. *Given a set of function $\mathcal{F} = \{f_i\}_{i=1}^n$, the* sensitivity *$m(f)$ of each function is defined as $m(f) = \lfloor \sup_{x \in \mathcal{X}} n \cdot \frac{f(x)}{\text{cost}(\mathcal{F}, x)} \rfloor + 1$, and the* total sensitivity *$M(\mathcal{F})$ of the set of functions is defined as $M(\mathcal{F}) = \sum_{f \in \mathcal{F}} m(f)$.*

Definition 4.6. *The dimension of $\mathcal{F}$ is defined as the smallest integer d such that for any $G \subset \mathcal{F}$,*

$$|\{\textbf{Range}(G, x, r) \mid x \in \mathcal{X}, r \geq 0\}| \leq |G|^d,$$

where $\textbf{Range}(G, x, r) = \{g \in G \mid g(x) \leq r\}$.

The algorithm proposed in [FL11] is summarized in Algorithm 10 below, and the corresponding result of quality of approximation is presented in Theorem 4.7.

Theorem 4.7. *Given a set of functions $\mathcal{F}$ from $\mathcal{X}$ to $[0, \infty]$, if $s \geq \frac{cM(\mathcal{F})}{\epsilon^2}(\dim(\mathcal{F}') + \log\left(\frac{1}{\delta}\right))$, then with probability at least $1 - \delta$, Algorithm 10 returns an ϵ-coreset for $\mathcal{F}$. That is,*

$$(1 - \epsilon) \sum_{f \in \mathcal{F}} f(x) \leq \sum_{f \in \mathcal{D}} f(x) \leq (1 + \epsilon) \sum_{f \in \mathcal{F}} f(x),$$

where $\mathcal{F}' = \{f / s(f) \mid f \in \mathcal{F}\}$ is a rescaled version of $\mathcal{F}$.

4.4.2 *Connections between RLA and coreset methods*

In the following, we present two results on the connection between RLA with algorithmic leveraging, i.e., with sampling based on exact or approximate leverage scores, and coreset

Algorithm 10 Compute ϵ-coreset

Input: A class of functions $\mathcal{F}$, sampling size s.
Output: An ϵ-coreset to $\mathcal{F}$.
Initialize $\mathcal{D}$ as an empty set.
Compute the sensitivity $m(f)$ for each function $f \in \mathcal{F}$.
$M(\mathcal{F}) \leftarrow \sum_{f\in\mathcal{F}} m(f)$.
for $f \in \mathcal{F}$ **do**
    Compute probabilities $p(f) = \frac{m(f)}{M(\mathcal{F})}$.
end for
for $i = 1, \ldots, s$ **do**
    Pick f from $\mathcal{F}$ with probability $p(f)$.
    Add $f/(s \cdot p(f))$ to $\mathcal{D}$.
end for
Return $\mathcal{D}$.

methods. These results originally appeared in [VX12]. We include them here and give different proofs.

The first result shows that the sensitivities are upper bounded by a constant factor times the ℓ_p leverage scores. With this connection between leverage scores and sensitivities, it is not hard to see that applying Algorithm 10 to ℓ_p regression is exactly the same as applying RLA sampling algorithm described in Theorem 2.22 and Section 2.4.1.1.

Proposition 4.8. *Given $\bar{A} \in \mathbb{R}^{n\times(d+1)}$, let $f_i(x) = |\bar{A}_{(i)}x|^p$, for $i \in [n]$. Let λ_i be the i-th leverage score of $\bar{A}$. Then, the i-th sensitivity satisfies*

$$m(f_i) \leq n\beta^p\lambda_i + 1$$

for $i \in [n]$, and the total sensitivity satisfies

$$M(\mathcal{F}) \leq n((\alpha\beta)^p + 1).$$

The second result is that, for the ℓ_p regression problem, the dimension of the class of functions $\dim(\mathcal{F}')$ is the same as the dimension of the subspace being considered, which is $\mathcal{O}(d)$. To be more specific, since all the $f \in \mathcal{F}'$ here are of the form $f(x) = |a^Tx|^p$ for some vector $a \in \mathbb{R}^d$, we consider a broader class of functions, namely $\mathcal{A} = \{|a^Tx|^p \mid a \in \mathbb{R}^d\}$, and compute its dimension.

Proposition 4.9. *Let $\mathcal{A} = \{|a^Tx|^p \mid a \in \mathbb{R}^d\}$. We have*

$$\dim(\mathcal{A}) \leq d + 1.$$

With these results, in combine with Theorem 4.7, we can see that to compute a coreset $\mathcal{D}$, which leads to a $\left(\frac{1+\epsilon}{1-\epsilon}\right)$-approximate solution the ℓ_p regression using coreset method of [FL11], the required sampling complexity is the same (up to constants) as that of RLA sampling algorithm, as indicated by Theorem 2.22.

4.4.3 *Limitation of our approach*

From the above, we see that for ℓ_p regression, a small coreset exists whose size only depends on d, and by solving it we can get a $(1+\epsilon)$-approximation solution. This results in the same sampling algorithm as in RLA. Also, the sensitivities defined in the framework can be used as a distribution when one converts a deterministic problem into a stochastic optimization problem. We want to see whether we can extend this scheme to other problems. Indeed, beyond ℓ_p regression, the coreset methods work for any kind of convex loss function [FL11]. However, since it depends on the total sensitivity, the size of the coreset is not necessarily small. For RLA, this translates into requiring a very large sample size to construct a good subproblem. For example, for hinge loss, we have the following example showing that the size of the coreset has an exponential dependency on d.

Proposition 4.10. *Define $f_i(x) = f(x, a_i) = (x^T a_i)^+$, where $x, a_i \in \mathbb{R}^d$ for $i \in [n]$. There exists a set of vectors $\{a_i\}_{i=1}^d$ such that the total sensitivity of $\mathcal{F} = \{f_i\}_{i=1}^n$ is approximately 2^d.*

This result indicates that new ideas will be needed to extend RLA preconditioning ideas to weighted SGD algorithms for other types of convex optimization problems. This should not be surprising, because RLA methods have been developed for randomized *linear* algebra problems, but it suggests several directions for follow-up work.

5

SUB-SAMPLED NEWTON METHODS WITH NON-UNIFORM SAMPLING

Consider the optimization problem

$$\min_{w \in \mathcal{C}} F(w) := \sum_{i=1}^{n} f_i(w) + R(w), \tag{5.1}$$

where $f_i(w), R(w)$ are smooth functions and $\mathcal{C} \subseteq \mathbb{R}^d$ is a convex constraint set. Many machine learning and scientific computing problems involve finding an approximation of the minimizer of (5.1) to *high precision*. For example, consider any machine learning application where each f_i is a loss function corresponding to the i-th data point and F is the *empirical risk*. The goal of solving (5.1) is to obtain a solution with small generalization error, i.e., high predictive accuracy on "unseen" data. One can show that the generalization error of the empirical risk minimizer is to within $\mathcal{O}(1/\sqrt{n})$ additive error of that of the true population risk minimizer; e.g., see [BE02; CBCG04]. The generalization error then depends on the sum of the optimization error of the empirical risk minimization problem and $\mathcal{O}(1/\sqrt{n})$. As a result, in the large-scale regime that we consider where there are many data points available, i.e., $n \gg 1$, obtaining a solution to (5.1) to high precision would indeed guarantee a low generalization error. A second example is that in many problems in which the optimization variable w contains specific meanings, a high-precision solution to (5.1) is necessary for interpreting the results. Examples of such settings arise frequently in machine learning such as sparse least-squares [Tib96], generalized linear models (GLMs) [FHT01], and metric learning problems [Kul13] among many more modern large-scale problems.

In this chapter, we describe a class of randomized Newton-type algorithms that exploit non-uniform sub-sampling of $\{\nabla^2 f_i(w)\}_{i=1}^n$ as well as inexact updates to reduce the computational complexity. We show that these techniques drastically drive down the complexity of Newton's method while maintaining a fast convergence rate. We also demonstrate that these algorithms are more robust to the ill-conditioning of the problem both theoretically and empirically. In Section 5.1, we provide background on Newton's method. We formally state our main algorithm in Section 5.2. The convergence rate results are presented in Section 5.3 and empirical results are shown in Section 5.4. The material in this chapter appears in Xu et al. [Xu+16].

5.1 BACKGROUND ON NEWTON'S METHOD

There is a plethora of first-order optimization algorithms [Bub14; NW06] for solving (5.1). However, for ill-conditioned problems, it is often the case that first-order methods return a solution far from the minimizer, w^*, albeit a low objective value. (See Figure 15 in Section 5.4 for example.) On the other hand, most second-order algorithms prove to be more robust to such ill conditioning. This is because, using the curvature information, second-order methods properly rescale the gradient such that it is a more appropriate

direction to follow. For example, take the canonical second-order method, i.e., *Newton's method*, which in the unconstrained case has updates of the form

$$w_{t+1} = w_t - [H(w_t)]^{-1} g(w_t), \tag{5.2}$$

where $g(w_t)$ and $H(w_t)$ denote the gradient and Hessian of F at w_t, respectively. Classical results indicate that under certain assumptions, Newton's method can achieve a locally superlinear convergence rate, which can be shown to be *problem independent*! Nevertheless, the cost of forming and inverting the Hessian is a major drawback in using Newton's method in practice.

In this regard, there has been a long line of work that tries to provide sufficient second-order information with feasible computations. For example, among the class of quasi-Newton methods, the BFGS algorithm [NW06] and its limited memory version [LN89] are the most celebrated. However, the convergence guarantee of these methods can be much weaker than Newton's methods. More recently, authors in [PW15; EM15; RKM16a] considered using sketching and sampling techniques to construct an approximate Hessian matrix and using it in the update rule (5.2). They showed that such algorithms inherit a local linear-quadratic convergence rate with a substantial computational gain.

Here, we propose novel, robust and highly efficient non-uniformly sub-sampled Newton methods (SSN) for a large sub-class of problem (5.1), where the Hessian of $F(w)$ in (5.1) can be written as

$$H(w) = \sum_{i=1}^{n} A_i^T(w) A_i(w) + Q(w), \tag{5.3}$$

where $A_i(w) \in \mathbb{R}^{k_i \times d}$, $i = 1, 2, \ldots, n$, are readily available and $Q(w)$ is some positive semidefinite matrix. This situation arises very frequently in applications such as machine learning. For example, take any problem where $f_i(w) = \ell(x_i^T w)$, $\ell(\cdot)$ is any convex loss function and x_i's are data points, and $R(w) = \frac{\lambda}{2}\|w\|^2$. In such situations, $A_i(w)$ is simply $\sqrt{\ell''(x_i^T w)}x_i^T$ and $Q(w) = \lambda I$.

First, we choose a sampling scheme $\mathcal{S}$ that constructs an appropriate non-uniform sampling distribution $\{p_i\}_{i=1}^n$ over $\{A_i(w)\}_{i=1}^n$ and samples s terms from $\{A_i(w)\}_{i=1}^n$. The approximate Hessian constructed as $\tilde{H}(w_t) = \sum_{i \in \mathcal{I}} A_i^T(w_t) A_i(w_t)/p_i + Q(w_t)$, where $\mathcal{I}$ denotes the set of sub-sampled indices, is then used to update the current iterate as

$$w_{t+1} = w_t - [\widetilde{H}(w_t)]^{-1} g(w_t). \tag{5.4}$$

Second, when the dimension of the problem, i.e., d, is so large that solving the above linear system, i.e., (5.4), becomes infeasible, we consider solving (5.4) inexactly by using an iterative solver $\mathcal{A}$, e.g., Conjugate Gradient or Stochastic Gradient Descent, with a few iterations such that a high-quality approximate solution can be produced with a lower complexity. Such inexact updates used in many second-order optimization algorithms have been well studied in [Byr+11; DES82].

5.2 MAIN ALGORITHM

5.2.1 *Notation and assumptions*

Given a function F, the gradient, the exact Hessian and the approximate Hessian are denoted by g, H, and $\widetilde{H}$, respectively. Iteration counter is denoted by subscript, e.g., w_t.

By a matrix A having n blocks, we mean that A has a block structure and can be viewed as $A = \left(A_1^T \cdots A_n^T\right)^T$, for appropriate size blocks A_i.

Definition 5.1 (Tangent cone). *Let $\mathcal{K}$ be the tangent cone of constraints $\mathcal{C}$ at the optimum w^*, i.e., $\mathcal{K} = \{\Delta \mid w^* + t\Delta \in \mathcal{C} \text{ for some } t > 0\}$.*

Definition 5.2 ($\mathcal{K}$-restricted maximum and minimum eigenvalues). *Given a symmetric matrix A and a cone $\mathcal{K}$, we define the $\mathcal{K}$-restricted maximum and minimum eigenvalues as*

$$\lambda_{\min}^{\mathcal{K}}(A) = \min_{x \in \mathcal{K} \setminus \{0\}} \frac{x^T A x}{x^T x}, \quad \lambda_{\max}^{\mathcal{K}}(A) = \max_{x \in \mathcal{K} \setminus \{0\}} \frac{x^T A x}{x^T x}.$$

Definition 5.3 (Stable rank). *Given a matrix $A \in \mathbb{R}^{N \times d}$, the stable rank of A is defined as*

$$\mathbf{sr}(A) = \frac{\|A\|_F^2}{\|A\|_2^2}.$$

Throughout this chapter, we use the following assumptions regarding the properties of the problem.

Assumption 1 (Lipschitz continuity). *$F(w)$ is convex and twice differentiable. The Hessian is L-Lipschitz continuous, i.e.*

$$\|\nabla^2 F(u) - \nabla^2 F(v)\| \leq L\|u - v\|, \quad \forall u, v \in \mathcal{C}.$$

Assumption 2 (Local regularity). *$F(x)$ is locally strongly convex and smooth, i.e.,*

$$\mu = \lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w^*)) > 0, \quad \nu = \lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w^*)) < \infty.$$

Here we define the local condition number of the problem as $\kappa := \nu / \mu$.

Assumption 3 (Hessian decomposition). *For each $f_i(w)$ in (5.1), define $\nabla^2 f_i(w) := H_i(w) := A_i^T(w) A_i(w)$. For simplicity, we assume $k_1 = \cdots = k_n = k$ and k is independent of d. Furthermore, we assume that given w, computing $A_i(w)$, $H_i(w)$, and $g(w)$ takes $\mathcal{O}(d)$, $\mathcal{O}(d^2)$, and $\mathcal{O}(\mathsf{nnz}(A))$ time, respectively. We call the matrix $A(w) = \left(A_1^T, \ldots, A_n^T\right)^T \in \mathbb{R}^{nk \times d}$ the augmented matrix of $\{A_i(w)\}$. Note that $H(w) = A(w)^T A(w) + Q(w)$.*

Finally, for simplicity, we also assume that $n \geq d^3 \log d$.

5.2.2 *Algorithm description*

Our proposed SSN method with non-uniform sampling is given in Algorithm 11. The core of our algorithm is based on a sampling scheme $\mathcal{S}$ that, at every iteration, constructs a non-uniform sampling distribution $\{p_i\}_{i=1}^n$ over $\{A_i(w_t)\}_{i=1}^n$ and then samples from $\{A_i(w_t)\}_{i=1}^n$ to form the approximate Hessian, $\widetilde{H}(w_t)$. The sampling sizes s needed for different sampling distributions are discussed in Sections 5.3.2 and 5.3.3. Since $H(w) = \sum_{i=1}^n A_i^T(w) A_i(w) + Q(w)$, the Hessian approximation boils down to a matrix approximation problem. Here, we generalize the two popular non-uniform sampling strategies, i.e., leverage score sampling and block norm squares sampling, which are commonly used in the field of randomized linear algebra, particularly for matrix approximation

problems [HI15; Mah11]. With an approximate Hessian constructed via non-uniform sampling, we may choose an appropriate solver $\mathcal{A}$ to solve the subproblem in Step 11 of Algorithm 11. Below we elaborate on the construction of the two non-uniform sampling schemes. Indeed, the sampling distribution is defined based on the matrix representation of $\{H_i(w_t)\}_{i=1}^n$ — its augmented matrix defined as follows.

Definition 5.4 (Augmented matrix). *Define the augmented matrix of* $\{H_i(w_t)\}_{i=1}^n$ *as*

$$A(w) = \begin{pmatrix} A_1^T & \cdots & A_n^T \end{pmatrix}^T \in \mathbb{R}^{kn \times d}.$$

For the ease of presentation, throughout the rest of this section and next section, we use A and Q to denote $A(w)$ and $Q(w)$, respectively, as long as it is clear in the text.

Algorithm 11 Sub-sampled Newton method(SSN) with non-uniform sampling

1: **Input:** Initialization point w_0, number of iteration T, sampling scheme $\mathcal{S}$ and solver $\mathcal{A}$.
2: **Output:** w_T
3: **for** $t = 0, \ldots, T-1$ **do**
4: Construct the non-uniform sampling distribution $\{p_i\}_{i=1}^n$ as described in Section 5.2.2.
5: **for** $i = 1, \ldots, n$ **do**
6: $q_i = \min\{s \cdot p_i, 1\}$, where s is the sampling size.
7: $\widetilde{A}_i(w_t) = \begin{cases} A_i(w_t)/\sqrt{q_i}, & \text{with probability } q_i, \\ 0, & \text{with probability } 1 - q_i. \end{cases}$
8: **end for**
9: $\widetilde{H}(w_t) = \sum_{i=1}^n \widetilde{A}_i^T(w_t)\widetilde{A}_i(w_t) + Q(w_t)$.
10: Compute $g(w_t)$
11: Use solver $\mathcal{A}$ to solve inexactly the subproblem

$$w_{t+1} \approx \arg\min_{w \in \mathcal{C}} \{\frac{1}{2}\langle (w - w_t), \widetilde{H}(w_t)(w - w_t)\rangle + \langle g(w_t), w - w_t\rangle\}. \tag{5.5}$$

12: **end for**
13: **Return** w_T.

BLOCK NORM SQUARES SAMPLING The first option is to construct a sampling distribution based on the magnitude of A_i. That is, define

$$p_i = \frac{\|A_i\|_F^2}{\|A\|_F^2}, \quad i = 1, \ldots, n. \tag{5.6}$$

This is an extension to the row norm squares sampling in which the intuition is to capture the importance of the blocks based on the "magnitudes" of the sub-Hessians.

BLOCK PARTIAL LEVERAGE SCORES SAMPLING The second option is to construct a sampling distribution based on leverage scores (Definition 2.1). Compared to the traditional matrix approximation problem, this problem has two major difficulties. First, here

blocks are being sampled, not single rows. Second, the matrix being approximated involves not only A but also Q.

To address the first difficulty, we follow the work by Carli Silva, Harvey, and Sato [CSHS11] in which a sparse sum of semidefinite matrices is found by sampling based on the trace of each semidefinite matrix after a proper transformation. By expressing $A^T A = \sum_{i=1}^{n} A_i^T A_i$, one can show that their approach is essentially sampling based on the sum of leverage scores that correspond to each block. For the second difficulty, inspired by the recently proposed ridge leverage scores [EAM15; CMM15], we consider the leverage scores of a matrix that concatenates A and $Q^{\frac{1}{2}}$. Combining these motivates us to define a new notion of leverage scores, block partial leverage scores, which is defined formally as follows.

Definition 5.5 (Block partial leverage scores)**.** *Given a matrix $A \in \mathbb{R}^{kn \times d}$ with n blocks and a matrix $Q \in \mathbb{R}^{d \times d}$ satisfying $Q \succeq 0$, let $\{\tau_i\}_{i=1}^{kn+d}$ be the leverage scores of the matrix $\begin{pmatrix} A \\ Q^{\frac{1}{2}} \end{pmatrix}$. Define the block partial leverage score for the i-th block as*

$$\tau_i^Q(A) = \sum_{j=k(i-1)+1}^{ki} \tau_j.$$

Then the sampling distribution is defined as

$$p_i = \frac{\tau_i^Q(A)}{\sum_{j=1}^{n} \tau_j^Q(A)}, \quad i = 1, \ldots, n. \tag{5.7}$$

Remark. When each block of A has only one row and $Q = 0$, the partial block leverage scores are equivalent to the ordinary leverage scores.

5.3 THEORETICAL RESULTS

In this section we provide detailed theoretical analysis to describe the complexity of our algorithm.[1] Different choices of sampling scheme $\mathcal{S}$ and the subproblem solver $\mathcal{A}$ lead to different complexities in SSN. More precisely, total complexity is characterized by the following four factors: **(i)** total number of iterations T determined by the convergence rate, which is affected by the choice of $\mathcal{S}$ and $\mathcal{A}$; **(ii)** the time t_{grad} it takes to compute the full gradient $g(w_t)$ (Step 10 in Algorithm 11); **(iii)** the time t_{const} to construct the sampling distribution $\{p_i\}_{i=1}^{n}$ and sample s terms at each iteration (Steps 4-8 in Algorithm 11), which is determined by $\mathcal{S}$; and **(iv)** the time t_{solve} needed to (implicitly) form $\tilde{H}$ and (inexactly) solve the subproblem at each iteration (Steps 9 and 11 in Algorithm 11), which is affected by the choice of both $\mathcal{S}$ (manifested in the sampling size s) and $\mathcal{A}$. With these, the total complexity can be expressed as

$$T \cdot (t_{grad} + t_{const} + t_{solve}). \tag{5.8}$$

1 We only focus on local convergence guarantees for Algorithm 11. To ensure global convergence, one can incorporate an existing globally convergent method, e.g. [RKM16b], as initial phase and switch to Algorithm 11 once the iterate is "close enough" to the optimum; see Lemma 5.6.

Below we study these contributing factors. Lemma 5.6 in Section 5.3.1 gives a general structural lemma that characterizes the convergence rates which determines T. In Sections 5.3.2 and 5.3.3, Lemmas 5.7 and 5.9 discuss t_{const} for the two sampling schemes respectively, while Lemmas 5.8 and 5.10 give the required sampling size s for the two sampling schemes respectively, which directly affects t_{solve}. Furthermore, t_{solve} is affected by the choice of solver, as discussed in Section 5.3.4. Finally, the complexity results are summarized in Section 5.3.5 and a comparison with other methods is provided in Section 5.3.6. The proofs of all the results can be found in Appendix C.

5.3.1 *Sufficient conditions for local linear-quadratic convergence*

Before diving into details of the complexity analysis, we state a structural lemma that characterizes the local convergence rate of our main algorithm, i.e., Algorithm 11. As discussed earlier, there are two layers of approximation in Algorithm 11: approximation of the Hessian by sub-sampling, and inexactness of solving (5.5). For the first layer, we require the approximate Hessian to satisfy one of the following two conditions (in Sections 5.3.2 and 5.3.3 we shall see our construction of approximate Hessian via non-uniform sampling can achieve these conditions with a sampling size independent of n):

$$\|\widetilde{H}(w_t) - H(w_t)\| \leq \epsilon \cdot \|H(w_t)\|, \tag{C1}$$

or

$$|x^T(\widetilde{H}(w_t) - H(w_t))y| \leq \epsilon \cdot \sqrt{x^T H(w_t) x} \cdot \sqrt{y^T H(w_t) y}, \quad \forall x, y \in \mathcal{K}. \tag{C2}$$

Note that (C1) and (C2) are two commonly seen guarantees for matrix approximation problems. In particular, (C2) is stronger in the sense that the spectrum of the approximated matrix $H(w_t)$ is well preserved. Below in Lemma 5.6, we see such a stronger condition ensures a better dependence on the condition number in terms of the convergence rate. For the second layer of approximation, we require the solver to produce an ϵ_0-approximate solution w_{t+1} satisfying

$$\|w_{t+1} - w_{t+1}^*\| \leq \epsilon_0 \cdot \|w_t - w_{t+1}^*\|, \tag{5.9}$$

where w_{t+1}^* is the exact optimal solution to (5.5). Note that (5.9) implies an ϵ_0-relative error approximation to the exact update direction, i.e., $\|v - v^*\| \leq \epsilon_0 \|v^*\|$, where $v = w_{t+1} - w_t$, $v^* = w_{t+1}^* - w_t$.

Remark 8. *When the problem is unconstrained, i.e., $\mathcal{C} = \mathbb{R}^d$, solving the subproblem (5.5) is equivalent to solving*

$$\tilde{H}_t v = -\nabla F(w_t).$$

Then requirement (5.9) is equivalent to finding an approximation solution v such that

$$\|v - v^*\| \leq \epsilon_0 \|v^*\|.$$

Lemma 5.6 (Structural result). *Let $\{w_t\}_{i=1}^T$ be the sequence generated based on update rule (5.5) with initial point w_0 satisfying $\|w_0 - w^*\| \leq \frac{\mu}{4L}$. Under Assumptions 1 and 2, if condition (C1) or (C2) is met, we have the following results.*

- *If the subproblem is solved exactly, then the solution error satisfies the recursion*

$$\|w_{t+1} - w^*\| \leq C_q \cdot \|w_t - w^*\|^2 + C_l \cdot \|w_t - w^*\|, \tag{5.10}$$

  where C_q and C_l are specified in (5.12) or (5.13) below.

- *If the subproblem is solved approximately and w_{t+1} satisfies (5.9), then the solution error satisfies the recursion*

$$\|w_{t+1} - w^*\| \leq (1+\epsilon_0)C_q \cdot \|w_t - w^*\|^2 + (\epsilon_0 + (1+\epsilon_0)C_l) \cdot \|w_t - w^*\|, \tag{5.11}$$

  where C_q and C_l are specified in (5.12) or (5.13) below.

Specifically, given any $\epsilon \in (0, 1/2)$,

- *If the approximate Hessian $\tilde{H}_t$ satisfies* (**C1**)*, then in (5.10) and (5.11),*

$$C_q = \frac{2L}{(1-2\epsilon\kappa)\mu}, \quad C_l = \frac{4\epsilon\kappa}{1-2\epsilon\kappa}. \tag{5.12}$$

- *If the approximate Hessian $\tilde{H}_t$ satisfies* (**C2**)*, then in (5.10) and (5.11),*

$$C_q = \frac{2L}{(1-\epsilon)\mu}, \quad C_l = \frac{3\epsilon}{1-\epsilon}\sqrt{\kappa}. \tag{5.13}$$

We remark that Lemma 5.6 is applicable to $F(w_t)$ and $\tilde{H}_t$ of any form. In our case, specifically, since $\nabla^2 F(w_t) = A^T A + Q$ and $\tilde{H}_t = A^T S^T S A + Q$, where S is the resulting sampling matrix, (**C1**) is equivalent to

$$\|(A^T S^T S A + Q) - (A^T A + Q)\| \leq \epsilon \|A^T A + Q\|, \tag{5.14}$$

and due to Lemma C.2 in Appendix C, (**C2**) is equivalent to

$$-\epsilon(A^T A + Q) \preceq (A^T S^T S A + Q) - (A^T A + Q) \preceq \epsilon(A^T A + Q). \tag{5.15}$$

From this it is not hard to see that (**C2**) is strictly stronger than (**C1**). Also, in this case the Hessian approximation problem boils down to a matrix approximation problem. That is, given A and Q, we want to construct a sampling matrix S efficiently such that the matrix $A^T A + Q$ is well preserved. As we mentioned, leverage scores sampling and block norm squares sampling are two popular ways for this task. In the next two subsections we focus on the theoretical properties of these two schemes.

5.3.2 *Results for block partial leverage scores sampling*

5.3.2.1 *Construction*

Since the block partial leverage scores are defined as the standard leverage scores of some matrix, we can make use of the fast approximation algorithm for standard leverage scores introduced in Section 2.3.2.3. Specifically, apply a variant of the algorithm in [Dri+12] by using the sparse subspace embedding [CW13a; Coh16] as the underlying sketching method to further speed up the computation.

Theorem 5.7. *Given w, under Assumption 3, with high probability, it takes $t_{const} = \mathcal{O}(nnz(A)\log n)$ time to construct a set of approximate leverage scores $\{\hat{\tau}_i^Q(A)\}_{i=1}^n$ that satisfy $\tau_i^Q(A) \leq \hat{\tau}_i^Q(A) \leq \beta \cdot \tau_i^Q(A)$ where $\{\tau_i\}_{i=1}^n$ are the block partial leverage scores of $H(w) = \sum_{i=1}^n H_i(w) + Q(w)$, where A is the augmented matrix of $\{H_i(w)\}_{i=1}^n$, and β is a constant.*

5.3.2.2 *Sampling size*

The following theorem indicates that if we sample the blocks of A based on block partial leverage scores with large enough sampling size, (5.15) holds with high probability.

Theorem 5.8. *Given A with n blocks, $Q \succeq 0$ and $\epsilon \in (0,1)$, let $\{\tau_i^Q(A)\}_{i=1}^n$ be its block partial leverage scores and $\{\hat{\tau}_i^Q(A)\}_{i=1}^n$ be their overestimates, i.e., $\hat{\tau}_i^Q(A) \geq \tau_i^Q(A), i = 1, \cdots, n$. Let $p_i = \frac{\hat{\tau}_i^Q(A)}{\sum_{j=1}^n \hat{\tau}_j^Q(A)}$. Construct SA by sampling the i-th block of A with probability $q_i = \min\{s \cdot p_i, 1\}$ and rescaling it by $1/\sqrt{q_i}$. Then if*

$$s \geq 4\left(\sum_{i=1}^n \hat{\tau}_i^Q(A)\right) \cdot \log\frac{4d}{\delta} \cdot \frac{1}{\epsilon^2}, \tag{5.16}$$

with probability at least $1-\delta$, (5.15) holds, and thus (C2) holds.

Remark 9. *When $\{\tau_i^Q(A)\}_{i=1}^n$ are the exact scores, since $\sum_{i=1}^n \tau_i^Q(A) \leq \sum_{i=1}^{N+d} \tau_i(\bar{A}) = d$ with $\bar{A} = \begin{pmatrix} A \\ Q^{\frac{1}{2}} \end{pmatrix}$, the above theorem indicates that less than $\mathcal{O}(d\log d/\epsilon^2)$ blocks are needed for (5.15) to hold.*

5.3.3 *Results for block norm squares sampling*

5.3.3.1 *Construction*

To sample based on block norm squares, one must first compute the Frobenius norm of every block in the augmented matrix A. This requires $\mathcal{O}(\mathsf{nnz}(A))$ time.

Theorem 5.9. *Given w, under Assumption 3, it takes $t_{const} = \mathcal{O}(nnz(A))$ time to construct a block norm squares sampling distribution for $H(w) = \sum_{i=1}^n H_i(w) + Q(w)$, where A is the augmented matrix of $\{H_i(w)\}_{i=1}^n$.*

5.3.3.2 *Sampling size*

The following theorem [HI15] show the approximation error bound for the Gram matrix. Here we extend it to our augmented matrix setting as follows.

Theorem 5.10 ([HI15])**.** *Given A with n blocks, $Q \succeq 0$ and $\epsilon \in (0,1)$, for $i = 1, \ldots, n$, let $r_i = \|A_i\|_F^2$. Let $p_i = \frac{r_i}{\sum_{j=1}^n r_j}$. Construct SA by sampling the i-th block of A with probability $q_i = \min\{s \cdot p_i, 1\}$ and rescaling it by $1/\sqrt{q_i}$. Then if*

$$s \geq 4\mathbf{sr}(A) \cdot \log\frac{\min\{4\mathbf{sr}(A), d\}}{\delta} \cdot \frac{1}{\epsilon^2}, \tag{5.17}$$

with probability at least $1-\delta$, (5.14) holds, and thus (C1).

Table 12: Comparison of different solvers for the subproblem. Here $\tilde{\kappa}_t = \lambda^{\mathcal{K}}_{\max}(\tilde{H}_t)/\lambda^{\mathcal{K}}_{\min}(\tilde{H}_t)$.

SOLVER $\mathcal{A}$	$\mathcal{T}(\mathcal{A}, \mathbb{R}^d, s, d)$	ϵ_0	REFERENCE
direct	$\mathcal{O}(sd^2)$	0	[GVL96]
CG	$\mathcal{O}(sd\sqrt{\tilde{\kappa}_t}\log(1/\epsilon))$	$\sqrt{\tilde{\kappa}_t}\epsilon$	[GVL96]
GD	$\mathcal{O}(sd\tilde{\kappa}_t\log(1/\epsilon))$	ϵ	[Nes13, Theorem 2.1.15]
ACDM	$\mathcal{O}(s\mathbf{sr}(SA)\sqrt{\tilde{\kappa}_t}\log(1/\epsilon))$	$\sqrt{\tilde{\kappa}_t}\epsilon$	[LS13]

5.3.4 *The choice of solver*

Here we discuss the effect of the choice of the solver $\mathcal{A}$ in Algorithm 11. Specifically, after an approximate Hessian $\tilde{H}_t$ is constructed in Algorithm 11, we look at how various solvers affect t_{solve} in (5.8). Since the approximate Hessian $\tilde{H}_t$ is of the form $A^T S^T SA + Q$ with $SA \in \mathbb{R}^{s\times d}$, the complexity for solving the subproblem (5.5) essentially depends on s and d. Given SA and Q, for ease of notation, we use

$$t_{solve} = \mathcal{T}(\mathcal{A}, \mathcal{C}, s, d)$$

to denote the time it needs to solve the subproblem (5.5) using solver $\mathcal{A}$.

For example, when the problem is unconstrained, i.e., $\mathcal{C} = \mathbb{R}^d$, subproblem (5.5) reduces to a linear regression problem with size s by d and direct solver costs $\mathcal{O}(sd^2)$. Alternatively, one can use an iterative solver such as Conjugate Gradient (CG) to obtain an approximate solution. In this case, the complexity for solving the subproblem becomes $\mathcal{O}(sd\sqrt{\tilde{\kappa}_t}(\log\frac{1}{\epsilon_0} + \log\tilde{\kappa}_t))$ to produce an ϵ_0 solution to (5.5), where $\tilde{\kappa}_t$ is the condition number of the problem. We see that CG is advantageous when the low dimension d is large and the linear system is fairly well-conditioned.

There are also many solvers that are suitable. In Table 12, we give a few examples for the unconstrained case ($\mathcal{C} = \mathbb{R}^d$) by summarizing the complexity and the resulting approximation quality ϵ_0 in (5.9).

5.3.5 *Complexities*

Again, recall that in (5.8) the complexity of the sub-sampled Newton methods can be expressed as $T \cdot (t_{const} + t_{grad} + t_{solve})$. Combining the results from the previous few subsections, we have the following lemma characterizing the total complexity.

Theorem 5.11. *For Algorithm 11 with sampling scheme $\mathcal{S}$ and solver $\mathcal{A}$, the total complexity is*

$$T \cdot (t_{const} + \mathcal{T}(\mathcal{A}, \mathcal{C}, s, d)),$$

and the solution error is specified in Lemma 5.6. In the above, t_{const} is specified in Theorem 5.7 and Theorem 5.9 and s is specified in Theorem 5.8 and Theorem 5.10 depending on the choice of $\mathcal{S}$; $\mathcal{T}(\mathcal{A}, \mathcal{C}, s, d)$ is discussed in Section 5.3.4.

Indeed, Lemma 5.6 implies that the sub-sampled Newton method inherits a local constant linear convergence rate. This can be shown by choosing specific values for ϵ and ϵ_0 in Lemma 5.6. The results are presented in the following corollary.

Corollary 5.12. *Suppose $\mathcal{C} = \mathbb{R}^d$. In Algorithm 11, assume that CG is used to solve subproblem (5.5). Then under Assumption 3,*

- *if block partial leverage scores sampling is used, the complexity per iteration in the local phase is*

$$\tilde{\mathcal{O}}(nnz(A)\log n + d^2\kappa^{3/2}); \tag{5.18}$$

- *if block norm squares sampling is used, the complexity per iteration in the local phase is*

$$\tilde{\mathcal{O}}(nnz(A) + \mathbf{sr}(A)d\kappa^{5/2}) \tag{5.19}$$

and the solution error satisfies

$$\|w_{t+1} - w^*\| \leq \rho \cdot \|w_t - w^*\|, \tag{5.20}$$

where $\rho \in (0,1)$ is a constant.[2]

5.3.6 *Comparisons*

5.3.6.1 *Comparison between different sampling schemes*

As discussed above, the sampling scheme $\mathcal{S}$ plays a crucial role in sub-sampled Newton methods. Here, we compare the two proposed non-uniform sampling schemes, namely, block partial leverage scores sampling and block norm squares sampling, with uniform sampling. SSN with uniform sampling was discussed in [RKM16a]. For completeness, we state the sampling size bound for uniform sampling. Note that this upper bound for s is tighter than in the original analysis [RKM16a].

Theorem 5.13. *Given A with n blocks, $Q \succeq 0$ and $\epsilon \in (0,1)$, construct SA by uniform sampling s blocks from A and rescaling it by $\sqrt{n/s}$. Then if*

$$s \geq 4n \cdot \frac{\max_i \|A_i\|^2}{\|A\|^2} \cdot \log\frac{d}{\delta} \cdot \frac{1}{\epsilon^2}, \tag{5.21}$$

*with probability at least $1-\delta$, (5.14) holds, and thus (**C1**) holds.*

This result allows us to compare the three sampling schemes in terms of the three main complexities, i.e., t_{const}, t_{solve} and T (manifested in C_q and C_l), as shown in Table 13. Note here, to evaluate the effect of the sampling scheme $\mathcal{S}$ only, we assume a direct solver is used for subproblem (5.5) because in this case t_{solve} is directly controlled by the sampling size s, independent of the solver $\mathcal{A}$. Also, for simplicity, we assume that $\mathcal{C} = \mathbb{R}^d$. The analysis is similar for general cases. In Table 13, C_q and C_l are defined based on two problem properties κ and $\tilde{\kappa}$:

$$\tilde{\kappa} = L/\mu, \quad \kappa = \nu/\mu, \tag{5.22}$$

where constants L, μ, ν are defined in Assumptions 1 and 2. Also, throughout this subsection, for randomized algorithms we choose parameters such that the failure probability is a constant.

As can be seen in Table 13, the greatest advantage of uniform sampling scheme comes from its simplicity of construction. On the other hand, as discussed in Sections 5.3.2.1 and 5.3.3.1, it takes nearly input-sparsity time to construct the leverage scores sampling distribution or the block norm squares sampling distribution. When it comes to the sampling size s for achieving (5.14) or (5.15), as suggested in (5.21), the one for uniform sampling

2 Here, $\tilde{\mathcal{O}}(\cdot)$ hides logarithmic factors of d, κ and $1/\delta$.

Table 13: Comparison between standard Newton's method and sub-sampled Newton methods (SSN) with different sampling schemes. In the above, C_q and C_l are the constants achieved in (5.10); κ and $\tilde{\kappa}$ are defined in (5.22); $A \in \mathbb{R}^{\mathcal{O}(n)\times d}$ is the augmented matrix in the current iteration (Definition 5.4) that satisfies $A^T A = \sum_{i=1}^n H_i(w_t)$; $\mathbf{sr}(A)$ is the stable rank of A satisfying $\mathbf{sr}(A) \leq d$; $\mathtt{nnz}(A)$ denote the number of non-zero elements in A. Note here, to remove the effect of solver $\mathcal{A}$, we assume the subproblem (5.4) is solved exactly. Also, we assume the problem is unconstrained ($\mathcal{C} = \mathbb{R}^d$) so that $t_{solve} = sd^2$.

NAME	t_{const}	$t_{solve} = sd^2$	C_q	C_l
Newton's method	0	$\mathcal{O}(nd^2)$	$\tilde{\kappa}$	0
SSN (leverage scores)	$\mathcal{O}(\mathtt{nnz}(A)\log n)$	$\tilde{\mathcal{O}}((\sum_i \tau_i^Q(A))d^2/\epsilon^2)$	$\frac{\tilde{\kappa}}{1-\epsilon}$	$\frac{\epsilon\sqrt{\kappa}}{1-\epsilon}$
SSN (block norm squares)	$\mathcal{O}(\mathtt{nnz}(A))$	$\tilde{\mathcal{O}}(\mathbf{sr}(A)d^2/\epsilon^2)$	$\frac{\tilde{\kappa}}{1-\epsilon\kappa}$	$\frac{\epsilon\kappa}{1-\epsilon\kappa}$
SSN (uniform)	$\mathcal{O}(1)$	$\tilde{\mathcal{O}}\left(nd^2\frac{\max_i \|A_i\|^2}{\|A\|^2}/\epsilon^2\right)$	$\frac{\tilde{\kappa}}{1-\epsilon\kappa}$	$\frac{\epsilon\kappa}{1-\epsilon\kappa}$

can become $\Omega(n)$ when A is very non-uniform, i.e., $\max_i \|A_i\| \approxeq \|A\|$. It can be shown that for a given ϵ, block norm squares sampling requires the smallest sampling size, which leads to the smallest value of t_{solve} in Table 13.

It is worth pointing that, although either (5.14) or (5.15) is sufficient to yield a local linear-quadratic convergence rate, as (5.15) is essentially a stronger condition, it has better constants, i.e., C_q and C_l. This fact is reflected in Table 13. The constants C_q and C_l for leverage scores sampling have a better dependence on the local condition number κ than the other two schemes because leverage scores sampling yields a sampling matrix that satisfies the spectral approximation guarantee (5.15). In fact, this difference can dramatically affect the performance of the algorithm on ill-conditioned problems. This is verified by numerical experiments; see Figure 14 in Section 5.4 for details.

5.3.6.2 *Comparison between various methods*

Next, we compare our main algorithm with other stochastic second-order methods including [RKM16a; ABH16]. Since these essentially imply a constant linear convergence rate, i.e.,

$$\|w_{t+1} - w^*\| \leq \rho \cdot \|w_t - w^*\|, \quad 0 < \rho < 1, \tag{5.23}$$

we compare the complexity per iteration needed in each algorithm when such a rate (5.23) is desired. Note here, for ease of comparison, we assume that $\mathcal{C} = \mathbb{R}^d$, $R(w) = 0$, and CG is used for solving subproblems in SSN so that the complexities can be easily expressed. This is the same setting as in [ABH16].[3] Analysis for general cases is similar.

Note that the results in the related works are stated in terms of condition numbers that are defined differently from the local condition number (Assumption 2) used. To be precise, besides the standard definition, i.e., κ, for any $w \in \mathbb{R}^d$, define

$$\kappa(w) = \frac{\lambda_{\max}(\sum_{i=1}^n H_i(w))}{\lambda_{\min}(\sum_{i=1}^n H_i(w))}, \tag{5.24a}$$

$$\hat{\kappa}(w) = n \cdot \frac{\max_i \lambda_{\max}(H_i(w))}{\lambda_{\min}(\sum_{i=1}^n H_i(w))}, \tag{5.24b}$$

$$\bar{\kappa}(w) = \frac{\max_i \lambda_{\max}(H_i(w))}{\min_i \lambda_{\min}(H_i(w))}. \tag{5.24c}$$

3 In [ABH16], the authors also considered the ridge penalty term but they absorb the penalty term into the summation, which still makes the objective in the form of an average/sum of functions.

An immediate relationship between the three condition numbers is $\kappa(w) \leq \hat{\kappa}(w) \leq \bar{\kappa}(w)$. The connections between these condition numbers depend on the properties of $H_i(w)$. Roughly speaking, when all $H_i(w)$'s are "close" to each other, then $\lambda^{\mathcal{K}}_{\max}(\sum_{i=1}^n H_i(w)) \approx \sum_{i=1}^n \lambda^{\mathcal{K}}_{\max}(H_i(w)) \approx n \cdot \max_i \lambda^{\mathcal{K}}_{\max}(H_i(w))$, and thus $\kappa \approx \hat{\kappa}$. And similarly, $\kappa \approx \bar{\kappa}$. While in many cases, some $H_i(w)$'s can be very different from the rest. For example, in linear regression, the Hessian is $H(w) = A^T A$, where A is the data matrix with each row as a data point. When the rows are not very uniform, it can be the case that κ is smaller than $\hat{\kappa}$ and $\bar{\kappa}$ by a factor of n.

Table 14: Complexity per iteration of different methods to obtain a problem independent local linear convergence rate. The quantities κ, $\hat{\kappa}$, and $\bar{\kappa}$ are the local condition numbers, defined in (5.24) at the optimum w^*, satisfying $\kappa \leq \hat{\kappa} \leq \bar{\kappa}$; $A \in \mathbb{R}^{\mathcal{O}(n)\times d}$ is the augmented matrix in the current iteration (Definition 5.4) that satisfies $A^T A = \sum_{i=1}^n H_i(w_t)$; $\mathbf{sr}(A)$ is the stable rank of A satisfying $\mathbf{sr}(A) \leq d$; $\mathtt{nnz}(A)$ denote the number of non-zero elements in A. Note here, for ease of comparison, we assume $\mathcal{C} = \mathbb{R}^d$, $R(w) = 0$, and CG is used for solving subproblems in SSN so that the complexities can be easily expressed.

name	COMPLEXITY PER ITERATION	REFERENCE
Newton-CG method	$\tilde{\mathcal{O}}(\mathtt{nnz}(A)\sqrt{\kappa})$	[NW06]
SSN (leverage scores)	$\tilde{\mathcal{O}}(\mathtt{nnz}(A)\log n + d^2\kappa^{3/2})$	**This work**
SSN (block norm squares)	$\tilde{\mathcal{O}}(\mathtt{nnz}(A) + \mathbf{sr}(A)d\kappa^{5/2})$	**This work**
Newton Sketch (SRHT)	$\tilde{\mathcal{O}}(nd(\log n)^4 + d^2(\log n)^4\kappa^{3/2})$	[PW15]
SSN (uniform)	$\tilde{\mathcal{O}}(\mathtt{nnz}(A) + d\hat{\kappa}\kappa^{3/2})$	[RKM16b]
LiSSA	$\tilde{\mathcal{O}}(\mathtt{nnz}(A) + d\hat{\kappa}\bar{\kappa}^2)$	[ABH16]

Given the notation we defined, we summarize the complexities of different algorithms in Table 14 including Newton methods with CG solving the subproblem. One immediate conclusion we can draw is that compared to Newton's methods, these stochastic second-order methods trade the coefficient of the leading term $\mathcal{O}(nd)$ with some lower order terms that only depend on d and condition numbers (assuming $\mathtt{nnz}(A) \approx nd$). Therefore, one should expect these algorithm to perform well when $n \gg d$ and the problem is fairly well-conditioned.

Although SSN with non-uniform sampling has a quadratic dependence on d, its dependence on the condition number is better than the other methods. There are two main reasons. First, the total power of the condition number is lower, regardless the versions of the condition number needed. Second, SSN (leverage scores) and SSN (block norm squares) only depend on κ, which can be significantly lower than the other two definitions of condition number according to the discussion above. Overall, SSN (leverage scores) is more robust on ill-conditioned problems.

5.4 EMPIRICAL EVALUATION

We consider an estimation problem in GLMs with Gaussian prior. Assume $x \in \mathbb{R}^{n\times d}, y \in \mathcal{Y}^n$ are the data matrix and response vector. The problem of minimizing the negative log-likelihood with ridge penalty can be written as

$$\min_{w\in\mathbb{R}^d} \sum_{i=1}^n \psi(x_i^T w, y_i) + \lambda\|w\|_2^2,$$

Table 15: Datasets used in ridge logistic regression, where κ and $\hat{\kappa}$ are the local condition numbers of ridge logistic regression problem with $\lambda = 0.01$ as defined in (5.24).

DATASET	CT slices	Forest	Adult	Buzz
n	53500	581012	32561	59535
d	385	55	123	78
κ	368	221	182	37
$\hat{\kappa}$	47078	322370	69359	384580

where $\psi : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$ is a convex cumulant generating function and $\lambda \geq 0$ is the ridge penalty parameter. In this case, the Hessian is $H(w) = \sum_{i=1}^{n} \psi''(x_i^T w, y_i) x_i x_i^T + \lambda I := x^T D^2(w) x + \lambda I$, where x_i is i-th column of x^T and $D(w)$ is a diagonal matrix with the diagonal $[D(w)]_{ii} = \sqrt{\psi''(x_i^T w, y_i)}$. The augmented matrix of $\{A_i(w)\}$ can be written as $A(w) = D(w)x \in \mathbb{R}^{n \times d}$, where $A_i(w) = [D(w)]_{ii} x_i^T$.

For our numerical simulations, we consider a very popular instance of GLMs, namely, logistic regression, where $\psi(u, y) = \log(1 + \exp(-uy))$ and $\mathcal{Y} = \{\pm 1\}$. Table 15 summarizes the datasets used in our experiments.

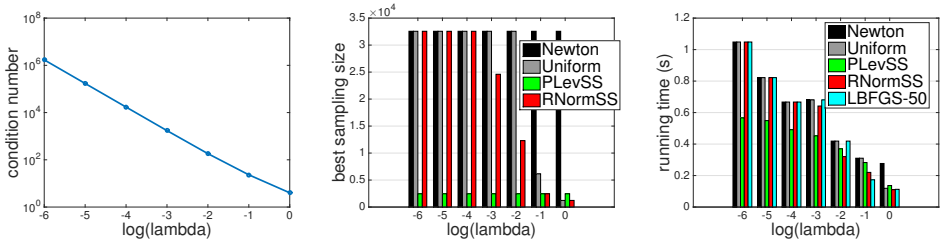
We compare the performance of the following five algorithms: (i) *Newton*: the standard Newton's method, (ii) *Uniform*: SSN with uniform sampling, (iii) *PLevSS*: SSN with partial leverage scores sampling, (iv) *RNormSS*: SSN with block (row) norm squares sampling, and (v) *LBFGS-k*: standard L-BFGS method [LN89] with history size k, (vi) *GD*: Gradient Descent, (vii) *AGD*: Accelerated Gradient Descent (AGD) [Nes13]. Note that, despite all of our effort, we could not compare with methods introduced in [ABH16; EM15] as they seem to diverge in our experiments. All algorithms are initialized with a zero vector.[4] We also use CG to solve the subproblem approximately to within 10^{-6} relative residue error. In order to compute the relative error $\|w_t - w^*\| / \|w^*\|$, an estimate of w^* is obtained by running the standard Newton's method for sufficiently long time. Note here, in SSN with partial leverage score sampling, we recompute the leverage scores every 10 iterations. Roughly speaking, these "stale" leverage scores can be viewed as approximate leverage scores for the current iteration with approximation quality that can be upper bounded by the change of the Hessian and such quantity is often small in practice. Reusing the leverage scores allows us to further drive down the running time.

We first investigate the effect of the condition number, controlled by varying λ, on the performance of different methods, and the results are depicted in Figure 14. It can be seen that in well-conditioned cases, all sampling schemes work equally well. However, as the condition number gets larger, the performance of uniform sampling deteriorates, while non-uniform sampling, in particular leverage score sampling, shows a great degree of robustness to such ill-conditioning effect. The experiments shown in Figure 14 are consistent with the theoretical results of Table 13.

Next, we compare the performance of various methods as measured by relative-error of the solution vs. running time. First, we provide a set of empirical comparison between first-order and second-order methods in Figure 15.[5] This is on dataset `CT Slice` with two different λ's. As can be seen clearly in Figure 15, SSN with non-uniform sampling not

4 Theoretically, the suitable initial point for all the algorithms is the one with which the standard Newton's method converges with a unit stepsize. Here, $w_0 = 0$ happens to be one such good starting point.

5 For each sub-sampled Newton method, the sampling size is determined by choosing the best value from $\{10d, 20d, 30d, \ldots, 100d, 200d, 300d, \ldots, 1000d\}$ in the sense that the objective value drops to $1/3$ of the initial function value first.

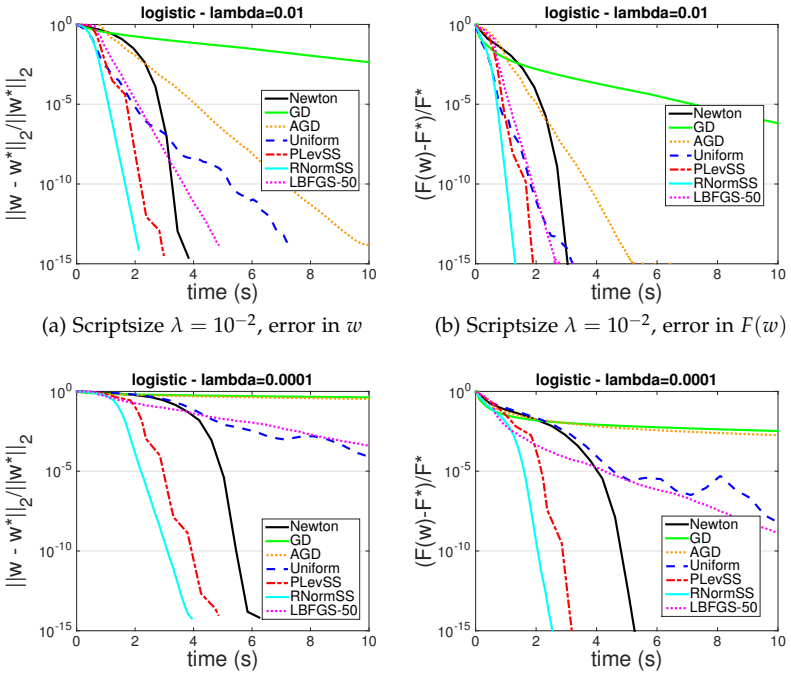


(a) condition number (b) sampling size (c) running time

Figure 14: Ridge logistic regression on `Adult` with different λ's: (a) local condition number κ, (b) sample size for different SSN methods giving the best overall running time, (c) running time for different methods to achieve 10^{-8} relative error.

only drives down the loss function $F(w)$ to an arbitrary precision much more quickly, but also recovers the minimizer w^* to high precision while first-order methods such as Gradient Descent converge very slowly. More importantly, unlike SSN with uniform sampling and LBFGS, non-uniform SSN exhibits a better robustness to condition number as its performance doesn't deteriorate much when the problem becomes more ill-conditioned (by setting the regularization λ smaller in Figures 15c and Figure 15d). This robustness to condition number allows our approach to excel for a wider range of models.

A more comprehensive comparison among various second-order methods on the four datasets is presented in Figure 16. It can be seen that, in most cases, SSN with non-uniform sampling schemes, i.e., PLevSS and RNormSS, outperform the other algorithms, especially Newton's method. In particular, they can be as twice fast as Newton's method. This is because on datasets with large n, the computational gain of our sub-sampled Newton methods in forming the (approximate) Hessian is significant while their convergence rate is only slightly worse than Newton's method (not shown here). Moreover, recall that in Section 5.3.6 we discussed that the convergence rate of SSN with uniform sampling relies on $\bar{\kappa}$. When the problem exhibits a high non-uniformity among data points, i.e., $\bar{\kappa}$ is much higher than κ as shown in Table 15, uniform sampling scheme performs poorly, e.g., in Figure 16b.



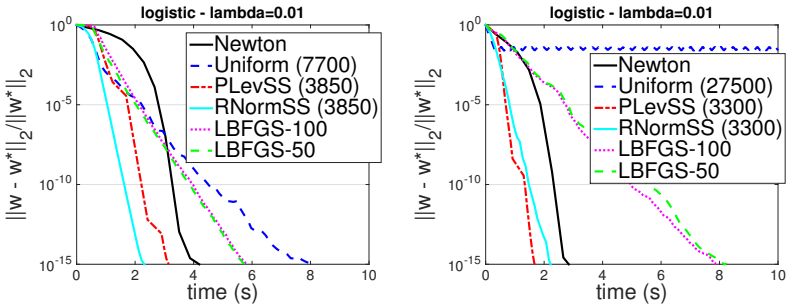
(a) Scriptsize $\lambda = 10^{-2}$, error in w

(b) Scriptsize $\lambda = 10^{-2}$, error in $F(w)$

(c) Scriptsize $\lambda = 10^{-4}$, error in w

(d) Scriptsize $\lambda = 10^{-4}$, error in $F(w)$

Figure 15: Iterate relative error vs. time(s) for a ridge logistic regression problem with two choices of regularization parameter λ on a real dataset `CT Slice`. Various second-order methods including standard Newton, LBFGS, SSN with uniform sampling (Uniform), partial leverage scores sampling (PLevSS) and row norm squares sampling (RNormSS), as well as gradient descent (GD) and its accelerated version (AGD) as representatives of first-order methods are implemented. Here, when $\lambda = 10^{-2}$, $\kappa = 386$; when $\lambda = 10^{-4}$, $\kappa = 1.387 \times 10^4$.



(a) `CT Slice` (b) `Forest`

(c) `Adult` (d) `Buzz`

Figure 16: Iterate relative solution error vs. time(s) for various second-order methods on four datasets with ridge penalty parameter $\lambda = 0.01$. The values in brackets denote the sample size used for each method.

Part II

IMPLEMENTATIONS AND APPLICATIONS

6 IMPLEMENTING RLA REGRESSION ALGORITHMS IN PARALLEL AND DISTRIBUTED ENVIRONMENTS

As pointed out earlier, one of the greatest advantages of RLA algorithms is that they are amenable to distributed computing platforms. This property is extremely valuable in the case where the dataset is too large to fit into the memory of a single machine. In this chapter, we present detailed empirical evaluations of RLA algorithms for large-scale overdetermined regression and matrix decomposition problems in parallel and distributed environments with terabyte-sized datasets. These problems are ubiquitous in many fields to which machine learning and data analysis methods are routinely applied, such as geophysical applications and high-frequency trading. We show that these algorithms are well suited to distributed computing platforms. In particular, we demonstrate that least squares problems with up to terabyte-sized data can be solved to low, medium, or high precision on existing distributed systems. For matrix decomposition, in order to assess the relative performance on various hardware, we consider three contemporary platforms. We report results on these platforms and their implications on the hardware and software issues arising in supporting data-centric workloads.

In Sections 6.1 – Section 6.3, we present computational results for ℓ_2 regression, quantile regression, and CX decomposition with terabyte-sized dataset, respectively. The material in this chapter appears in Yang, Meng, and Mahoney [YMM16; YMM14] and Gittens et al. [Git+16a].

6.1 COMPUTATIONAL RESULTS FOR ℓ_2 REGRESSION

First, we focus on very overdetermined very large-scale ℓ_2 regression problems. Recall that the subspace embedding that is a crucial part of RLA algorithms can be data-aware (i.e., a sampling algorithm) or data-oblivious (i.e., a projection algorithm). Recall also that, as discussed in Section 2.4.1, after obtaining a subspace embedding matrix, one can obtain a low-precision solution by solving the resulting subproblem, or one can obtain a high-precision solution by invoking an iterative solver, e.g., LSQR [PS82] for ℓ_2 regression, with a preconditioner constructed by the embedding. Thus, in this empirical evaluation, we consider both random sampling and random projection algorithms, and we consider solving the problem to low-precision, medium-precision, and high-precision on a suite or data sets chosen to be challenging for different classes of algorithms. We consider a range of matrices designed to "stress test" all of the variants of the basic algorithms that we have been describing, and we consider matrices of size ranging up to just over the terabyte scale.

6.1.1 *Experimental setup*

In order to illustrate a range of uniformity and non-uniformity properties for both the leverage scores and the condition number, we considered the following four types of dataset.

- UG (matrices with *uniform* leverage scores and *good* condition number);

Table 16: Commands (presented in MATLAB format) used to generate matrices with uniform leverage scores, i.e., the UG and UB matrices. Here, `kappa` is a parameter used to determine the condition number of the generated matrix.

```
U = orth(randn(m,n));
S = diag(linspace(1,1/kappa,n));
V = orth(randn(n,n));
A = U*S*V';
x = randn(n,1);
b = A*x;
err = randn(m,1);
b = b+0.25*norm(b)*err/norm(err);
```

- UB (matrices with *uniform* leverage scores and *bad* condition number);
- NG (matrices with *non-uniform* leverage scores and *good* condition number);
- NB (matrices with *non-uniform* leverage scores and *bad* condition number).

These matrices are generated in the following manner. For matrices with uniform leverage scores, we used the commands in Table 16. For matrices with non-uniform leverage scores, we considered matrices with the following structure:

$$A = \begin{pmatrix} \alpha B & R \\ \mathbf{0} & I \end{pmatrix},$$

where $B \in \mathbb{R}^{(n-d/2)\times(d/2)}$ is a random matrix with each element sampled from $\mathcal{N}(0,1)$, $I \in \mathbb{R}^{(d/2)\times(d/2)}$ is the identity matrix, and $R \in \mathbb{R}^{(n-d/2)\times(d/2)}$ is a random matrix generated using `1e-8 * rand(n-d/2,d/2)`. In this case, the condition number of A is controlled by α. It is worth mentioning that the last $d/2$ rows of the above matrix have leverage scores exactly 1 and the rest are approximately $d/2/(n-d/2)$. Also, for matrices with bad condition number, the condition number is approximately $1e6$ (meaning 10^6); while for matrices with good condition number, the condition number is approximately 5.

To generate a large-scale matrix that is beyond the capacity of RAM, and to evaluate the quality of the solution for these larger inputs, we used two methods. First, we replicate the matrix (and the right-hand side vector, when it is needed to solve regression problems) `REPNUM` times, and we "stack" them together vertically. We call this naïve way of stacking matrices as `STACK1`. Alternatively, for NB or NG matrices, we can stack them in the following manner:

$$\tilde{A} = \begin{pmatrix} \alpha B & R \\ \cdots & \\ \alpha B & R \\ \mathbf{0} & I \end{pmatrix}.$$

We call this stacking method `STACK2`. The two different stacking methods lead to different properties for the linear system being solved—we summarize these in Table 17—and, while they yielded results that were usually similar, as we mention below, the results were different in certain extreme cases. With either method of stacking matrices, the optimal solution remains the same, so that we can evaluate the approximate solutions of the new

large least-squares problems. We considered these and other possibilities, but in the results reported below, unless otherwise specified we choose the following: for large-scale UG and UB matrices, we use `STACK1` to generate the data; and for large-scale NG and NB matrices, we use `STACK2` to generate the data.

Table 17: Summary of methods for stacking matrices, to generate matrices too large to fit into RAM; here, `REPNUM` denotes the number of replications and coherence is defined as the largest leverage score of the matrix.

NAME	CONDITION NUMBER	LEVERAGE SCORES	COHERENCE
STACK1	unchanged	divided by `REPNUM`	divided by `REPNUM`
STACK2 (for NB and NG only)	increased	unknown	always 1

Recall that Table 3 provides several methods for computing an ℓ_2 subspace embedding matrix. Since a certain type of random projection either can be used to obtain an embedding directly or can be used (with the algorithm of [Dri+12]) to approximate the leverage scores for use in sampling, we consider both data-aware and data-oblivious methods. Throughout our evaluation, we use the following notation to denote various ways of computing the subspace embedding.

- `PROJ CW` — Random projection with the sparse ℓ_2 embedding (Lemma 2.14)
- `PROJ GAUSSIAN` — Random projection with Gaussian transform (Lemma 2.12)
- `PROJ RADEMACHER` — Random projection with Rademacher transform (similar to `PROJ GAUSSIAN`)
- `PROJ SRDHT` — Random projection with subsampled randomized discrete Hartley transform [Bra83] (Similar to SRHT in Lemma 2.13)
- `SAMP APPR` — Random sampling based on approximate leverage scores (general version of Lemma 2.20)
- `SAMP UNIF` — Random sampling with uniform distribution (for completeness)

Note that, instead of using a vanilla SRHT, we perform our evaluation with a SRDHT (i.e., a subsampled randomized discrete Hartley transform). (An SRDHT is a related FFT-based transform that has similar properties to a SRHT in terms of speed and accuracy but doesn't have the restriction on the dimension to be a power of 2.) Also note that, instead of using a distributed FFT-based transform to implement SRDHT, we treat the transform as a dense matrix-matrix multiplication; hence we should not expect SRDHT to have computational advantage over other transforms.

Throughout this section, by embedding dimension (or sketch size in the figures), we mean the projection size for projection based methods and the sampling size for sampling based methods. Also, it is worth mentioning that for a sampling algorithm with approximate leverage scores, we fix the underlying embedding method to be `PROJ CW` and the projection size c to be $d^2/4$. In our experiments, we found that—when they were approximated sufficiently well—the precise quality of the approximate leverage scores does not have a strong influence on the quality of the solution obtained by the sampling algorithm. We elaborate this more in Section 6.1.3.

The computations for Table 18, Figure 18, and Table 19 below (i.e., for the smaller-sized problems) were performed on a shared-memory machine with 12 Intel Xeon CPU cores at clock rate 2GHz with 128GB RAM. In these cases, the algorithms are implemented in MATLAB. All of the other computations (i.e., for the larger problems) were performed on a cluster with 16 nodes (1 master and 15 slaves), each of which has 8 CPU cores at clock rate 2.5GHz with 25GB RAM. For all these cases, the algorithms are implemented in Apache Spark[1] via a Python API.

Spark provides a high-level programming model and execution engine for fault-tolerant parallel and distributed computing, based on a core abstraction called *resilient distributed dataset (RDD)*. RDDs are immutable lazily materialized distributed collections supporting functional programming operations such as `map`, `filter`, and `reduce`, each of which returns a new RDD. RDDs may be loaded from a distributed file system, computed from other RDDs, or created by parallelizing a collection created within the user's application. RDDs of key-value pairs may also be treated as associative arrays, supporting operations such as `reduceByKey`, `join`, and `cogroup`. Spark employs a lazy evaluation strategy for efficiency. Another major benefit of Spark over MapReduce is the use of in-memory caching and storage so that data structures can be reused.

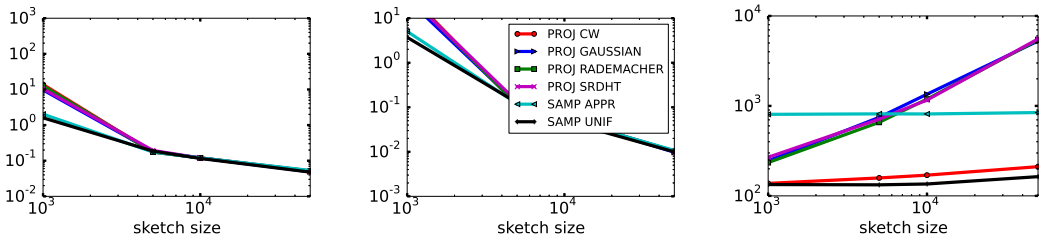
6.1.2 *Overall performance of low-precision solvers*

Here, we evaluate the performance of the 6 kinds of embedding methods described above (with different embedding dimension) on the 4 different types of dataset described above (with size $1e7$ by 1000). For dense transforms, e.g., `PROJ GAUSSIAN`, due to the memory capacity, the largest embedding dimension we can handle is $5e4$. For each dataset and each kind of the embedding, we compute the following three quantities: relative error of the objective $|f - f^*|/f^*$; relative error of the solution certificate $\|x - x^*\|_2/\|x^*\|_2$; and the total running time to compute the approximate solution. The results are presented in Figure 17.

As we can see, when the matrices have uniform leverage scores, all the methods including `SAMP UNIF` behave similarly. As expected, `SAMP UNIF` runs fastest, followed by `PROJ CW`. On the other hand, when the leverages scores are non-uniform, `SAMP UNIF` breaks down even with large sampling size. Among the projection based methods, the dense transforms, i.e., `PROJ GAUSSIAN`, `PROJ RADEMACHER` and `PROJ SRDHT`, behave similarly. Although `PROJ CW` runs much faster, it yields very poor results until the embedding dimension is large enough, i.e., $c = 3e5$. Meanwhile, the sampling algorithm with approximate leverage scores, i.e., `SAMP APPR`, tends to give very reliable solutions. (This breaks down if the embedding dimension in the approximate leverage score algorithm is chosen too small.) In particular, the relative error is much lower throughout all choices of the embedding dimension. This can be understood in terms of the theory; see [DMM06; Dri+11] for details. In addition, its running time becomes more favorable when the embedding dimension is larger.

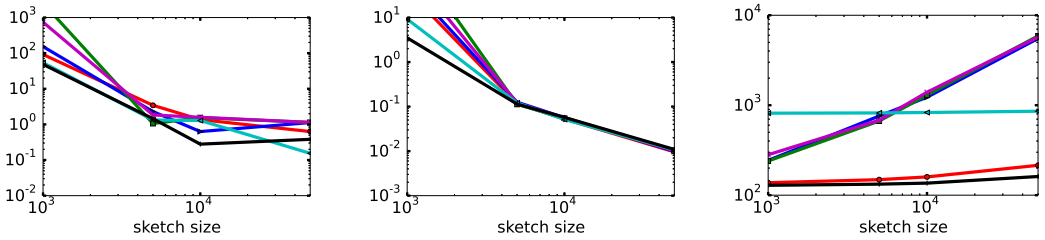
As a more minor point, theoretical results also indicate that the upper bound on the relative error of the solution vector depends on the condition number of the system as well as the amount of mass of b that lies in the range space of A, denote by γ [Dri+12]. Across the four datasets, γ is roughly the same. This is why we see that the relative error of the certificate, i.e., the vector achieving the minimum solution, tends to be larger when the condition number of the matrix becomes higher.

1 Apache Spark, http://spark.apache.org/



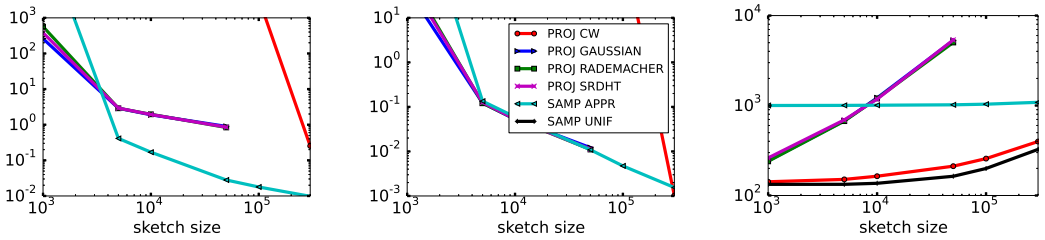
(a) $\|x - x^*\|_2 / \|x^*\|_2$ (b) $|f - f^*|/|f^*|$ (c) Running time(sec)

$1e7 \times 1000$ **UG matrix**



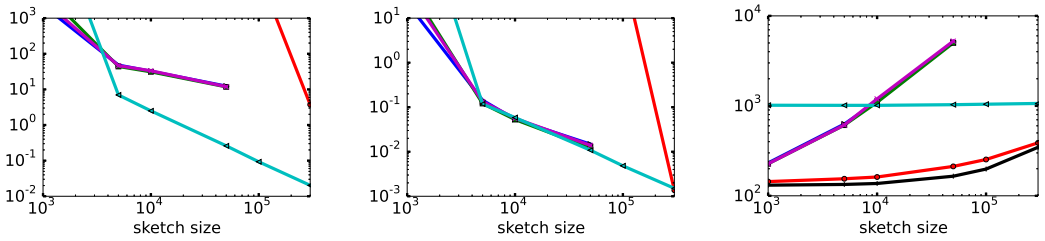
(d) $\|x - x^*\|_2 / \|x^*\|_2$ (e) $|f - f^*|/|f^*|$ (f) Running time(sec)

$1e7 \times 1000$ **UB matrix**



(g) $\|x - x^*\|_2 / \|x^*\|_2$ (h) $|f - f^*|/|f^*|$ (i) Running time(sec)

$1e7 \times 1000$ **NG matrix**



(j) $\|x - x^*\|_2 / \|x^*\|_2$ (k) $|f - f^*|/|f^*|$ (l) Running time(sec)

$1e7 \times 1000$ **NB matrix**

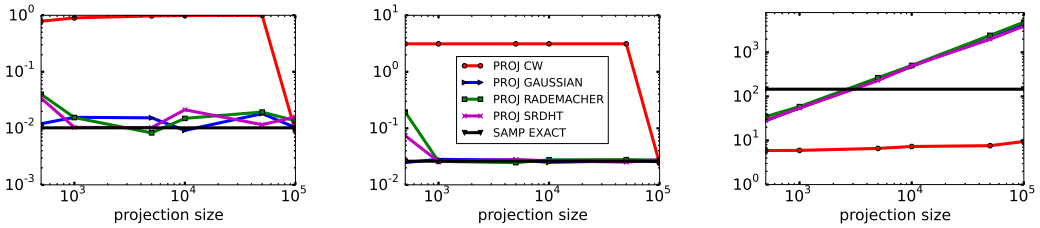
Figure 17: Evaluation of all 6 of the algorithms on the 4 different types of matrices of size $1e7$ by 1000. For each method, the following three quantities are computed: relative error of the objective $|f - f^*|/f^*$; relative error of the certificate $\|x - x^*\|_2 / \|x^*\|_2$; and the running time to compute the approximate solution. Each subplot shows one of the above quantities versus the embedding dimension, respectively. For each setting, 3 independent trials are performed and the median is reported.

6.1.3 *Quality of the approximate leverage scores*

Here, we evaluate the quality of the fast approximate leverage score algorithm of [Dri+12], and we investigate the quality of the approximate leverage scores with several underlying embeddings. (The algorithm of [Dri+12] considered only Hadamard-based projections, but other projection methods could be used, leading to similar approximation quality but different running times.) We consider only an NB matrix since leverage scores with non-uniform distributions are harder to approximate. In addition, the size of the matrix we considered is only rather small, $1e6$ by 500, due to the need to compute the exact leverage scores for comparison. Our implementation follows closely the main algorithm of [Dri+12], except that we consider other random projection matrices. In particular, we used the following four ways to compute the underlying embedding: namely, `PROJ CW`, `PROJ GAUSSIAN`, `PROJ RADEMACHER`, and `PROJ SRDHT`. For each kind of embedding and embedding dimension, we compute a series of quantities which characterize the statistical properties of the approximate leverage scores. The results are summarized in Table 18.

As we can see, when the projection size is large enough, all the projection-based methods to compute approximations to the leverage scores produce highly accurate leverage scores. Among these projection methods, `PROJ CW` is typically faster but also requires a much larger projection size in order to yield reliable approximate leverage scores. The other three random projections perform similarly. In general, the algorithms approximate the large leverage scores (those that equal or are close to 1) better than the small leverage scores, since α_L and β_L are closer to 1. This is crucial when calling `SAMP APPR` since the important rows shall not be missed, and it is a sufficient condition for the theory underlying the algorithm of [Dri+12] to apply.

Next, we invoke the sampling algorithm for the ℓ_2 regression problem, with sampling size $s = 1e4$ by using these approximate leverage scores. We evaluate the relative error on both the solution vector and objective and the total running time. For completeness and in order to evaluate the quality of the approximate leverage score algorithm, we also include the results by using the exact leverage scores. The results are presented in Figure 18.



(a) $\|x - x^*\|_2 / \|x^*\|_2$ (b) $|f - f^*|/|f^*|$ (c) Running time(sec)

Figure 18: Performance of sampling algorithms with approximate leverage scores, as computed by several different underlying projections. The test was performed on an NB matrix of size $1e6$ by 500 and the sampling size was $1e4$. Each subplot shows one of the following three quantities versus the projection size used in the underlying random projection phase: relative error of the objective $|f - f^*|/f^*$; relative error of the certificate $\|x - x^*\|_2 / \|x^*\|_2$; and the running time. For each setting, 5 independent trials are performed and the median is reported.

These results suggest that the precise quality of the approximate leverage scores does not substantially affect the downstream error, i.e., sampling-based algorithms are robust

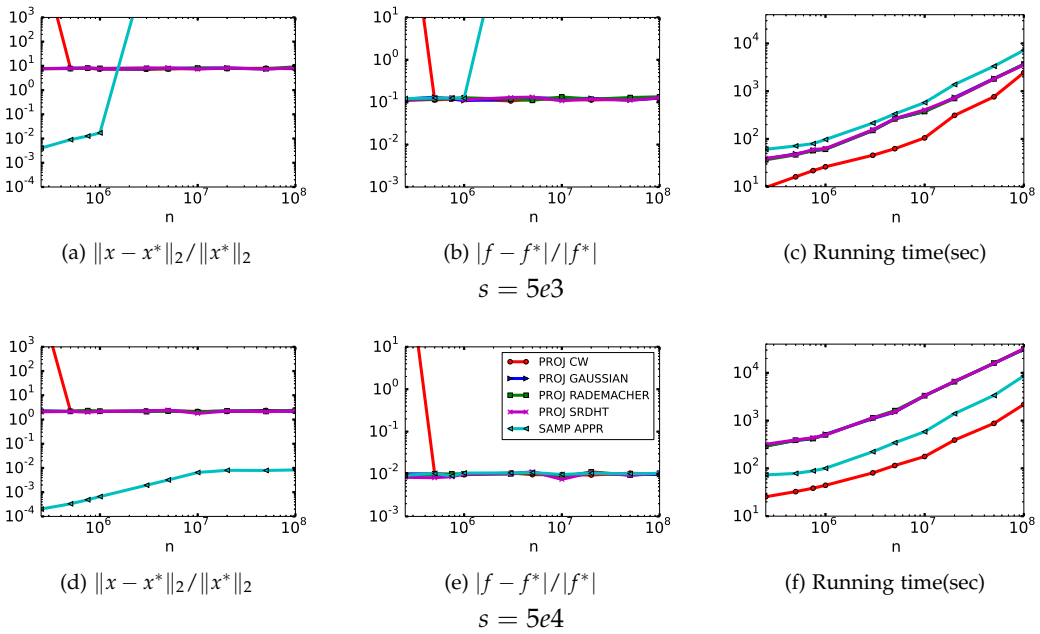
Table 18: Quality of the approximate leverage scores. The test was performed on an NB matrix with size $1e6$ by 500. In above, $\hat{p}$ denotes the distribution by normalizing the approximate leverage scores and p^* denotes the exact leverage score distribution. $D_{KL}(p||q)$ is the KL divergence [KL51] of q from p defined as $\sum_i p_i \ln \frac{p_i}{q_i}$. Let $L = \{i|p_i^* = 1\}$ and $S = \{i|p_i^* < 1\}$. In this case, $\hat{p}^L$ denotes the corresponding slice of $\hat{p}$, and the quantities $\hat{p}^S$, $p^{*,L}$, $p^{*.S}$ are defined similarly.

c	PROJ CW	PROJ GAUSSIAN	PROJ RADEMACHER	PROJ SRDH
	$\|\hat{p} - p^*\|_2 / \|p^*\|_2$			
5e2	0.9205	0.7738	0.7510	0.5008
1e3	0.9082	0.0617	0.0447	0.0716
5e3	0.9825	0.0204	0.0072	0.0117
1e4	0.9883	0.0143	0.0031	0.0075
5e4	0.9962	0.0061	0.0006	0.0030
1e5	0.0016	0.0046	0.0003	0.0023
	$D_{KL}(p^*\|\|\hat{p})$			
5e2	18.5241	0.0710	0.6372	0.1852
1e3	19.7773	0.0020	0.0015	0.0029
5e3	20.3450	0.0002	0.0001	0.0001
1e4	20.0017	0.0001	0.0001	0.0001
5e4	19.2417	1.9e-5	1.0e-5	1.0e-5
1e5	0.0001	1.0e-5	5e-6	5e-6
	$\alpha_L = \max_i\{\hat{p}_i^L / p_i^{*,L}\}$			
5e2	28.6930	7.0267	7.3124	4.0005
1e3	11.4425	1.1596	1.1468	1.2201
5e3	50.3311	1.0584	1.0189	1.0379
1e4	82.6574	1.0449	1.0099	1.0199
5e4	218.9658	1.0192	1.0018	1.0094
1e5	1.0016	1.0108	1.0009	1.0060
	$\alpha_S = \max_i\{\hat{p}_i^S / p_i^{*,S}\}$			
5e2	0	24.4511	16.8698	4.5227
1e3	0	1.3923	1.3718	1.3006
5e3	0	1.1078	1.1040	1.1077
1e4	0	1.0743	1.0691	1.0698
5e4	0	1.0332	1.0317	1.0310
1e5	1.0236	1.0220	1.0218	1.0198
	$\beta_L = \min_i\{\hat{p}_i^L / p_i^{*,L}\}$			
5e2	0	0.0216	0.0448	0.4094
1e3	0	0.8473	0.8827	0.8906
5e3	0	0.9456	0.9825	0.9702
1e4	0	0.9539	0.9916	0.9827
5e4	0	0.9851	0.9982	0.9922
1e5	0.9969	0.9878	0.9993	0.9934
	$\beta_S = \min_i\{\hat{p}_i^S / p_i^{*,S}\}$			
5e2	0	0.0077	0.0141	0.1884
1e3	0	0.7503	0.7551	0.7172
5e3	0	0.9037	0.9065	0.9065
1e4	0	0.9328	0.9306	0.9356
5e4	0	0.9704	0.9691	0.9710
1e5	0.9800	0.9787	0.9789	0.9803

to imperfectly-approximated leverage scores, as long as the largest scores are not too poorly approximated. (Clearly, however, we could have chosen parameters such that some of the larger scores were very poorly approximated, e.g., by choosing the embedding dimension to be too small, in which case the quality would matter. In our experience, the quality matters less since these approximate leverage scores are sufficient to solve ℓ_2 regression problems.) Finally, and importantly, note that the solution quality obtained by using approximate leverage scores is as good as that of using exact leverage scores, while the running time can be much less.

6.1.4 *Performance of low-precision solvers when n changes*

Here, we explore the scalability of the low-precision solvers by evaluating the performance of all the embeddings on NB matrices with varying n. We fix $d = 1000$ and let n take values from $2.5e5$ to $1e8$. These matrices are generated by stacking an NB matrix with size $2.5e5$ by 1000 `REPNUM` times, with `REPNUM` varying from 1 to 400 using `STACK1`. For conciseness, we fix the embedding dimension of each method to be either $5e3$ or $5e4$. The relative error on certificate and objective and running time are evaluated. The results are presented in Figure 19.



(a) $\|x - x^*\|_2/\|x^*\|_2$ (b) $|f - f^*|/|f^*|$ (c) Running time(sec)

$s = 5e3$

(d) $\|x - x^*\|_2/\|x^*\|_2$ (e) $|f - f^*|/|f^*|$ (f) Running time(sec)

$s = 5e4$

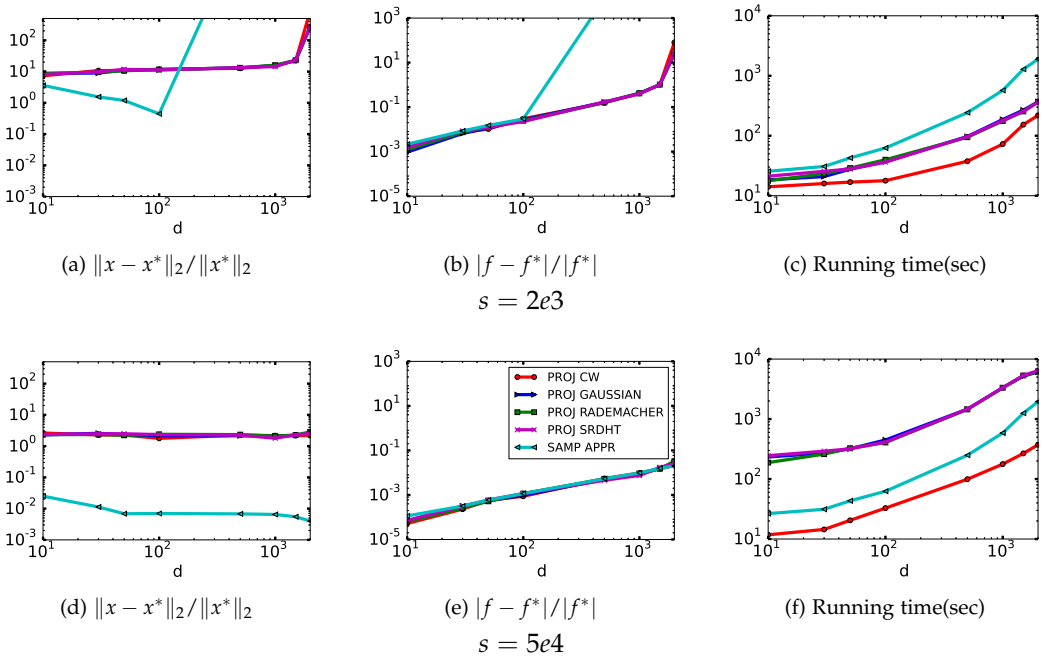
Figure 19: Performance of all the algorithms on NB matrices with varying n from $2.5e5$ to $1e8$ and fixed $d = 1000$. The matrix is generated using `STACK1`. For each method, the embedding dimension is fixed to be $5e3$ or $5e4$. The following three quantities are computed: relative error of the objective $|f - f^*|/f^*$; relative error of the certificate $\|x - x^*\|_2/\|x^*\|_2$; and the running time to compute the approximate solution. For each setting, 3 independent trials are performed and the median is reported.

Especially worthy mentioning is that when using `STACK1`, by increasing `REPNUM`, as we pointed out, the coherence of the matrix, i.e., the maximum leverage score, is decreasing, as the size is increased. We can clearly see that, when $n = 2.5e5$, i.e., the coherence is

1, PROJ CW fails. Once the coherence gets smaller, i.e., n gets larger, the projection-based methods behave similarly and the relative error remains roughly the same as we increased n. This is because STACK1 doesn't alter the condition number and the amount of mass of the right hand side vector that lies in the range space of the design matrix and the lower dimension d remains the same. However, SAMP APPR tends to yield larger error on approximating the certificate as we increase REPNUM, i.e., the coherence gets smaller. Moreover, it breaks down when the embedding dimension is very small.

6.1.5 *Performance of low-precision solvers when d changes*

Here, we evaluate the performance of the low-precision solvers by evaluating the performance of all the embeddings on NB matrices with changing d. We fix $n = 1e7$ and let d take values from 10 to 2000. For each d, the matrix is generated by stacking an NB matrix with size $2.5e5$ by d 40 times using STACK1, so that the coherence of the matrix is $1/40$. For conciseness, we fix the embedding of each method to be $2e3$ or $5e4$. The relative error on certificate and objective and running time are evaluated. The results are shown in Figure 20.



(a) $\|x - x^*\|_2/\|x^*\|_2$ (b) $|f - f^*|/|f^*|$ (c) Running time(sec)

$s = 2e3$

(d) $\|x - x^*\|_2/\|x^*\|_2$ (e) $|f - f^*|/|f^*|$ (f) Running time(sec)

$s = 5e4$

Figure 20: Performance of all the algorithms on NB matrices with varying d from 10 to 2000 and fixed $n = 1e7$. The matrix is generated using STACK1. For each method, the embedding dimension is fixed to be $2e3$ or $5e4$. The following three quantities are computed: relative error of the objective $|f - f^*|/f^*$; relative error of the certificate $\|x - x^*\|_2/\|x^*\|_2$; and the running time to compute the approximate solution. For each setting, 3 independent trials are performed and the median is reported.

As can be seen, overall, all the projection-based methods behave similarly. As expected, the relative error goes up as d gets larger. Meanwhile, SAMP APPR yields lower error as d

increases. However, it seems to have a stronger dependence on the lower dimension of the matrix, as it breaks down when d is 100 for small sampling size, i.e., $s = 2e3$.

6.1.6 *Performance of high precision solvers*

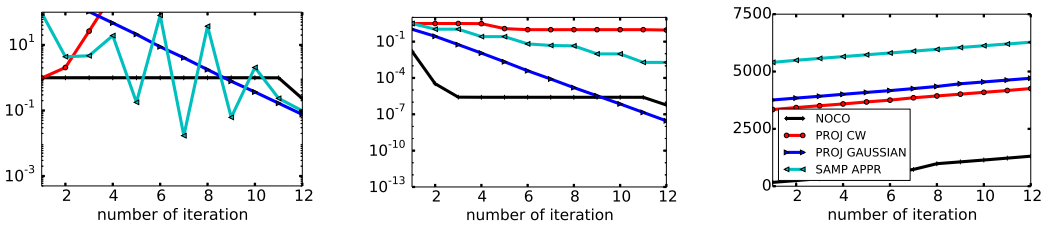
Here, we evaluate the use of these methods as preconditioners for high-precision iterative solvers. Since the embedding can be used to compute a preconditioner for the original linear system, one can invoke iterative algorithms such as LSQR [PS82] to solve the preconditioned least-squares problem. Here, we will use LSQR. We first evaluate the conditioning quality, i.e., $\kappa(AR^{-1})$, on an NB matrix with size $1e6$ by 500 using several different ways for computing the embedding. The results are presented in Table 19. Then we test the performance of LSQR with these preconditioners on an NB matrix with size $1e8$ by 1000 and an NG matrix with size $1e7$ by 1000. For simplicity, for each method of computing the embedding, we try a small embedding dimension where some of the methods fail, and a large embedding dimension where most of the methods succeed. See Figure 21 and Figure 22 for details.

Table 19: Quality of preconditioning on an NB matrix with size $1e6$ by 500 using several kinds of embeddings. For each setting, 5 independent trials are performed and the median is reported.

c	PROJ CW	PROJ GAUSSIAN	PROJ RADEMACHER	PROJ SRDHT	SAMP APPR
5e2	1.08e8	2.17e3	1.42e3	1.19e2	1.21e2
1e3	1.1e6	5.7366	5.6006	7.1958	75.0290
5e3	5.5e5	1.9059	1.9017	1.9857	25.8725
1e4	5.1e5	1.5733	1.5656	1.6167	17.0679
5e4	1.8e5	1.2214	1.2197	1.2293	6.9109
1e5	1.1376	1.1505	1.1502	1.1502	4.7573

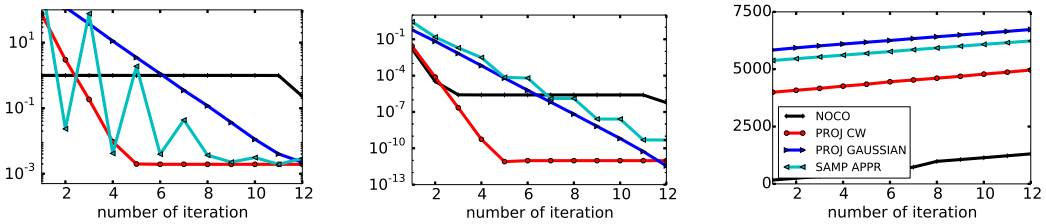
The convergence rate of the LSQR phase depends on the preconditioning quality, i.e., $\kappa(AR^{-1})$ where R is obtained by the QR decomposition of the embedding of A, ΦA. See Section 2.3 for more details. Table 19 implies that all the projection-based methods tend to yield preconditioners with similar condition numbers once the embedding dimension is large enough. Among them, `PROJ CW` needs a much larger embedding dimension to be reliable (clearly consistent with its use in low-precision solvers). In addition, overall, the conditioning quality of the sampling-based embedding method, i.e., `SAMP APPR` tends to be worse than that of projection-based methods.

As for the downstream performance, from Figure 21 we can clearly see that, when a small embedding dimension is used, i.e., $s = 5e3$, `PROJ GAUSSIAN` yields the best preconditioner, as its better preconditioning quality translates immediately into fewer iterations for LSQR to converge. This is followed by `SAMP APPR`. This relative order is also suggested by Table 19. As the embedding dimension is increased, i.e., using large embedding dimension, all the method yield significant improvements and produce much more accurate solutions compared to that of `NOCO` (LSQR without preconditioning), among which `PROJ CW` with embedding dimension $3e5$ converges to a nearly machine-precision solution within only 5 iterations. As for the running time, since each iteration of LSQR only involves with two matrix-vector multiplications (costs less than 2 minutes in our experiments), the overall running time is dominated by the time for computing the preconditioner. As expected,



(a) $\|x - x^*\|_2 / \|x^*\|_2$ (b) $|f - f^*|/|f^*|$ (c) Running time(sec)

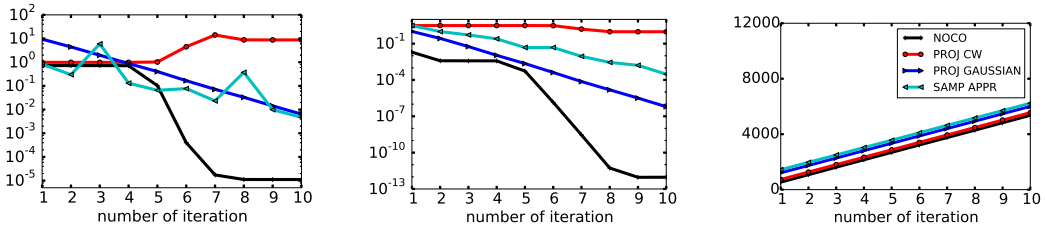
small embedding dimension



(d) $\|x - x^*\|_2 / \|x^*\|_2$ (e) $|f - f^*|/|f^*|$ (f) Running time(sec)

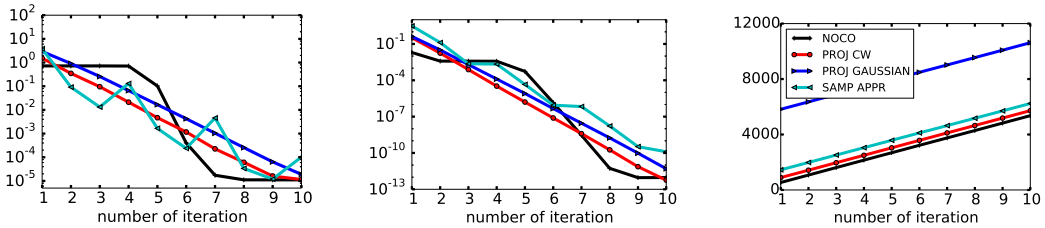
large embedding dimension

Figure 21: Evaluation of LSQR with randomized preconditioner on an NB matrix with size $1e8$ by 1000 and condition number $1e6$. Here, several ways for computing the embedding are implemented. In `SAMP APPR`, the underlying random projection is `PROJ CW` with projection dimension $3e5$. For completeness, LSQR without preconditioner is evaluated, denoted by `NOCO`. In above, by small embedding dimension, we mean $5e3$ for all the methods. By large embedding dimension, we mean $3e5$ for `PROJ CW`, $1e4$ for `PROJ GAUSSIAN` and $5e4$ for `SAMP APPR`. For each method and embedding dimension, the following three quantities are computed: relative error of the objective $|f - f^*|/f^*$; relative error of the certificate $\|x - x^*\|_2 / \|x^*\|_2$; and the running time to compute the approximate solution. Each subplot shows one of the above quantities versus number of iteration, respectively. For each setting, only one trial is performed.



(a) $\|x - x^*\|_2 / \|x^*\|_2$ (b) $|f - f^*|/|f^*|$ (c) Running time(sec)

small embedding dimension



(d) $\|x - x^*\|_2 / \|x^*\|_2$ (e) $|f - f^*|/|f^*|$ (f) Running time(sec)

large embedding dimension

Figure 22: Evaluation of LSQR with randomized preconditioner on an NB matrix with size $1e7$ by 1000 and condition number 5. Here, several ways for computing the embedding are implemented. In `SAMP APPR`, the underlying random projection is `PROJ CW` with projection dimension $3e5$. For completeness, LSQR without preconditioner is evaluated, denoted by `NOCO`. In above, by small embedding dimension, we mean $5e3$ for all the methods. By large embedding dimension, we mean $3e5$ for `PROJ CW` and $5e4$ for the rest. For each method and embedding dimension, the following three quantities are computed: relative error of the objective $|f - f^*|/f^*$; relative error of the certificate $\|x - x^*\|_2 / \|x^*\|_2$; and the running time to compute the approximate solution. Each subplot shows one of the above quantities versus number of iteration, respectively. For each setting, 3 independent trials are performed and the median is reported.

PROJ CW runs the fastest and the running time of PROJ GAUSSIAN scales linearly in the embedding dimension. In SAMP APPR, the sampling process needs to make 1-2 passes over the dataset but the running time is relatively stable regardless of the sampling size, as reflected in Figure 21c and Figure 21f. Finally, note that the reason that the error does not drop monotonically in the solution vector is the following. With the preconditioners, we work on a transformed system, and the theory only guarantees monotonicity in the decreasing of the relative error of the certificate of the transformed system, not the original one.

Finally, a minor but potentially important point should be mentioned as a word of caution. As expected, when the condition number of the linear system is large, vanilla LSQR does not converge at all. On the other hand, when the condition number is very small, from Figure 22, there is no need to precondition. If, in this latter case, a randomized preconditioning method is used, then the embedding dimension must be chosen to be sufficiently large: unless the embedding dimension is large enough such that the conditioning quality is sufficiently good, then preconditioned LSQR yields larger errors than even vanilla LSQR.

6.2 COMPUTATIONAL RESULTS FOR QUANTILE REGRESSION

Next, recall that in Chapter 3 we present a fast RLA algorithm for solving large-scale quantile regression. In this section, we continue our empirical evaluation with an evaluation of our main algorithm applied to terabyte-scale problems. As the algorithms for ℓ_2 regression, our sampling algorithms for quantile regression only needs several passes through the data and it is embarrassingly parallel, we evaluate them (Algorithm 8 with different conditioning methods) in Apache Hadoop.[2] Hadoop is an open source implementation of MapReduce computational framework which is the *de facto* standard parallel environment for large data analysis. In this section, we continue to use the notation used in Section 3.3.

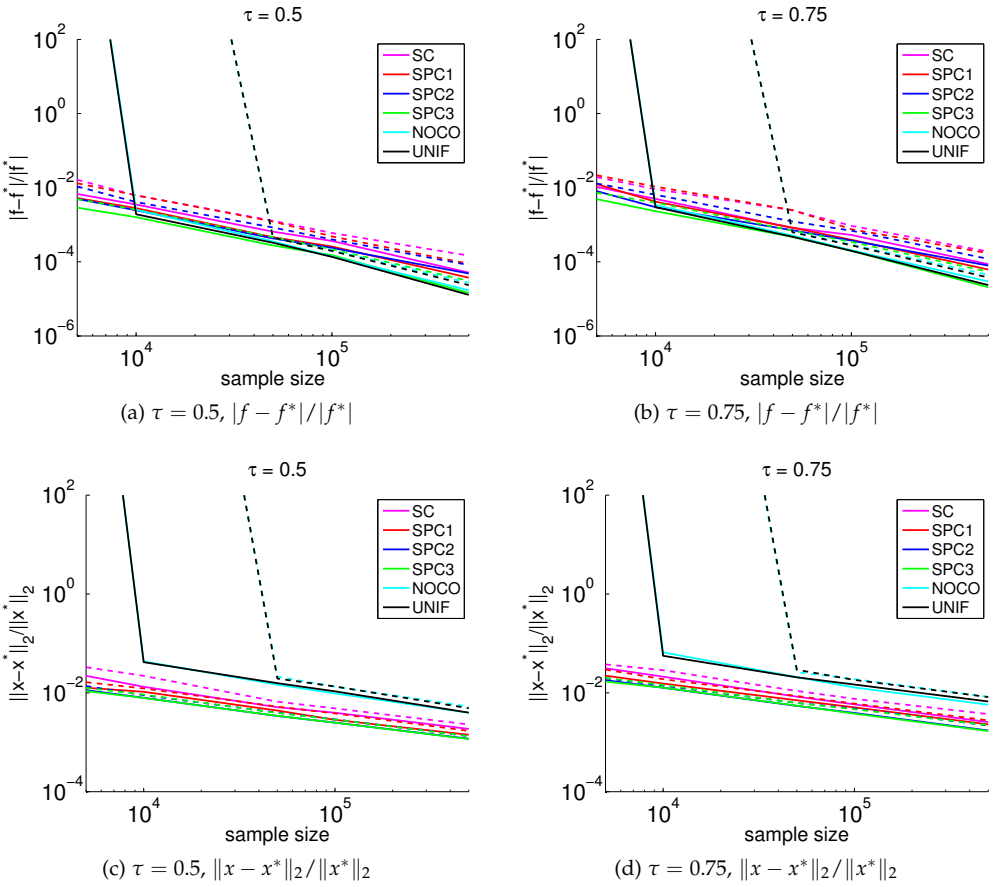
6.2.1 *Quality of approximation when the sampling size s changes*

In order to show the evaluations similar to Figure 1, we still implement SC, SPC1, SPC2, SPC3, NOCO and UNIF. Here, the datasets are generated by "stacking" the medium-scale data a few thousand times. Although this leads to "redundant" data, which may favor sampling methods, this has the advantage that it leads terabyte-sized problems whose optimal solution at different quantiles are known. For a skewed data with size $1e6 \times 50$, we stack it vertically 2500 times. This leads to a data with size $2.5e9 \times 50$.

Figure 23 shows the relative errors on the replicated skewed data set by using the six methods. We only show the results for $\tau = 0.5$ and 0.75 since the conditioning methods tend to generate abnormal results when $\tau = 0.95$. These plots correspond with and should be compared to the four subfigures in the first two rows and columns of Figure 1.

As can be seen, the method preserves the same structure as when the method is applied to the medium-scale data. Still, SPC2 and SPC3 performs slightly better than other methods when s is large enough. In this case, as before, NOCO and UNIF are not reliable when $s < 1e4$. When $s > 1e4$, NOCO and UNIF perform sufficiently closely to the conditioning-based methods on approximating the objective value. However, the gap between the performance on approximating the solution vector is significant.

2 Apache Hadoop, http://hadoop.apache.org/



(a) $\tau = 0.5$, $|f - f^*|/|f^*|$

(b) $\tau = 0.75$, $|f - f^*|/|f^*|$

(c) $\tau = 0.5$, $\|x - x^*\|_2/\|x^*\|_2$

(d) $\tau = 0.75$, $\|x - x^*\|_2/\|x^*\|_2$

Figure 23: The first (solid lines) and the third (dashed lines) quartiles of the relative errors of the objective value (namely, $|f - f^*|/|f^*|$) and solution vector (namely, $\|x - x^*\|_2/\|x^*\|_2$), by using 6 different methods, among 30 independent trials, as a function of the sample size s. The test is on replicated skewed data with size 2.5e9 by 50. The three different columns correspond to $\tau = 0.5, 0.75$, respectively.

In order to show more detail on the quartiles of the relative errors, in Table 20 records the quartiles of relative errors on vectors, measured in ℓ_1, ℓ_2, and ℓ_∞ norms by using the six methods when the sampling size $s = 5e4$ and $\tau = 0.75$. Conditioning-based methods can yield 2-digit accuracy when $s = 5e4$ while NOCO and UNIF cannot. Also, the relative error is somewhat higher when measured in ℓ_∞ norm.

Table 20: The first and the third quartiles of relative errors of the solution vector, measured in ℓ_1, ℓ_2, and ℓ_∞ norms. The test is on replicated synthetic data with size $2.5e9$ by 50, the sampling size $s = 5e4$, and $\tau = 0.75$.

	$\|x - x^*\|_2 / \|x^*\|_2$	$\|x - x^*\|_1 / \|x^*\|_1$	$\|x - x^*\|_\infty / \|x^*\|_\infty$
SC	[0.0084, 0.0109]	[0.0075, 0.0086]	[0.0112, 0.0159]
SPC1	[0.0071, 0.0086]	[0.0066, 0.0079]	[0.0080, 0.0105]
SPC2	[0.0054, 0.0063]	[0.0053, 0.0061]	[0.0050, 0.0064]
SPC3	[0.0055, 0.0062]	[0.0054, 0.0064]	[0.0050, 0.0067]
NOCO	[0.0207, 0.0262]	[0.0163, 0.0193]	[0.0288, 0.0397]
UNIF	[0.0206, 0.0293]	[0.0175, 0.0200]	[0.0242, 0.0474]

6.2.2 *Quality of approximation when the sampling size d changes*

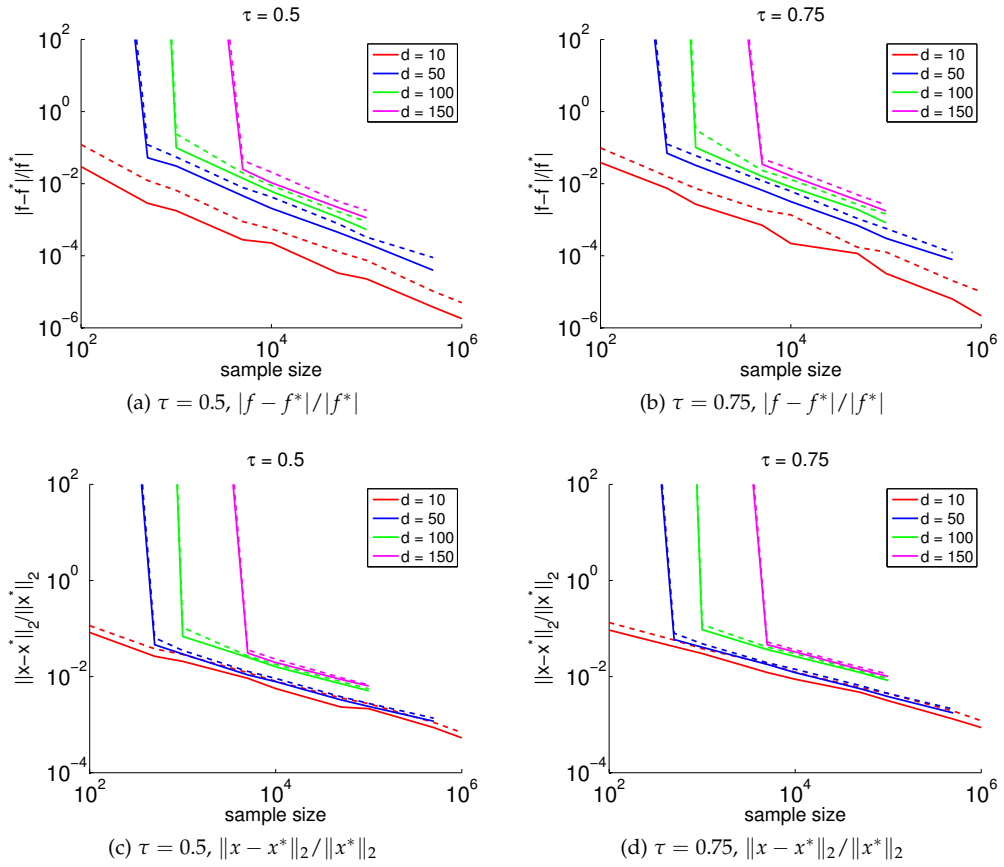
Next, we explore how the accuracy may change as the lower dimension d varies, and the capacity of our large-scale version algorithm. In this experiment, we fix the higher dimension of the replicated skewed data to be $1e9$, and let d take values in $10, 50, 100, 150$. We will only use SPC2 as it has the relative best condition number. Figure 24 shows the results of the experiment described above.

From Figure 24, except for some obvious fact such as the accuracies become lower as d increases when the sampling size is unchanged, we should also notice that, the lower d is, the higher the minimum sampling size required to yield acceptable relative errors will be. For example, when $d = 150$, we need to sample at least $1e4$ rows in order to obtain at least one digit accuracy.

Notice also that, there are some missing points in the plot. That means we cannot solve the subproblem at that sampling size with certain d. For example, solving a subproblem with size $1e6$ by 100 is unrealistic on a single machine. Therefore, the corresponding point is missing. Another difficulty we encounter is the capability of conditioning on a single machine. Recall that, in Algorithm 5 or Algorithm 6, we need to perform QR factorization or ellipsoid rounding on a matrix, say SA, whose size is determined by d. In our large-scale version algorithm, since these two procedures are not parallelizable, we have to perform these locally. When $d = 150$, the higher dimension of SA will be over $1e7$. Such size has reached the limit of RAM for performing QR factorization or ellipsoid rounding. Hence, it prevents us from increasing the lower dimension d.

6.2.3 *Evaluation on solution of Census data*

For the census data, we stack it vertically 2000 times to construct a realistic data set whose size is roughly $1e10 \times 11$. In Table 21, we present the solution computed by our randomized algorithm with a sample size $1e5$ at different quantiles, along with the corresponding



(a) $\tau = 0.5$, $|f - f^*|/|f^*|$

(b) $\tau = 0.75$, $|f - f^*|/|f^*|$

(c) $\tau = 0.5$, $\|x - x^*\|_2/\|x^*\|_2$

(d) $\tau = 0.75$, $\|x - x^*\|_2/\|x^*\|_2$

Figure 24: The first (solid lines) and the third (dashed lines) quartiles of the relative errors of the objective value (namely, $|f - f^*|/|f^*|$) and solution vector (namely, $\|x - x^*\|_2/\|x^*\|_2$), by using SPC2, among 30 independent trials, as a function of the sample size s. The test is on replicated skewed data with $n = 1e9$ and $d = 10, 50, 100, 150$. The two different columns correspond to $\tau = 0.5, 0.75$, respectively. The missing points mean that the subproblem on such sampling size with corresponding d is unsolvable in RAM.

optimal solution. As can be seen, for most coefficients, our algorithm provides at least 2-digit accuracy. Moreover, in applications such as this, the quantile regression result reveals some interesting facts about these data. For example, for these data, marriage may entail a higher salary in lower quantiles; Education^2, whose value ranged from 0 to 256, has a strong impact on the total income, especially in the higher quantiles; and the difference in age doesn't affect the total income much in lower quantiles, but becomes a significant factor in higher quantiles.

Table 21: Quantile regression results for the U.S. Census 2000 data. The response is the total annual income. Except for the intercept and the terms involved with education, all the covariates are $\{0, 1\}$ binary indicators.

Covariate	$\tau = 0.1$	$\tau = 0.25$	$\tau = 0.5$	$\tau = 0.75$	$\tau = 0.9$
intercept	8.9812	9.3022	9.6395	10.0515	10.5510
	[8.9673, 8.9953]	[9.2876, 9.3106]	[9.6337, 9.6484]	[10.0400, 10.0644]	[10.5296, 10.5825]
female	-0.2609	-0.2879	-0.3227	-0.3472	-0.3774
	[-0.2657, -0.2549]	[-0.2924, -0.2846]	[-0.3262, -0.3185]	[-0.3481, -0.3403]	[-0.3792, -0.3708]
Age $\in [30, 40)$	0.2693	0.2649	0.2748	0.2936	0.3077
	[0.2610, 0.2743]	[0.2613, 0.2723]	[0.2689, 0.2789]	[0.2903, 0.2981]	[0.3027, 0.3141]
Age $\in [40, 50)$	0.3173	0.3431	0.3769	0.4118	0.4416
	[0.3083, 0.3218]	[0.3407, 0.3561]	[0.3720, 0.3821]	[0.4066, 0.4162]	[0.4386, 0.4496]
Age $\in [50, 60)$	0.3316	0.3743	0.4188	0.4612	0.5145
	[0.3190, 0.3400]	[0.3686, 0.3839]	[0.4118, 0.4266]	[0.4540, 0.4636]	[0.5071, 0.5230]
Age $\in [60, 70)$	0.3237	0.3798	0.4418	0.5072	0.6027
	[0.3038, 0.3387]	[0.3755, 0.3946]	[0.4329, 0.4497]	[0.4956, 0.5162]	[0.5840, 0.6176]
Age ≥ 70	0.3206	0.4132	0.5152	0.6577	0.8699
	[0.2962, 0.3455]	[0.4012, 0.4359]	[0.5036, 0.5308]	[0.6371, 0.6799]	[0.8385, 0.8996]
non_white	-0.0953	-0.1018	-0.0922	-0.0871	-0.0975
	[-0.1023, -0.0944]	[-0.1061, -0.0975]	[-0.0985, -0.0902]	[-0.0932, -0.0860]	[-0.1041, -0.0932]
married	0.1175	0.1117	0.0951	0.0870	0.0953
	[0.1121, 0.1238]	[0.1059, 0.1162]	[0.0918, 0.0989]	[0.0835, 0.0914]	[0.0909, 0.0987]
education	-0.0152	-0.0175	-0.0198	-0.0470	-0.1062
	[-0.0179, -0.0117]	[-0.0200, -0.0149]	[-0.0225, -0.0189]	[-0.0500, -0.0448]	[-0.1112, -0.1032]
education^2	0.0057	0.0062	0.0065	0.0081	0.0119
	[0.0055, 0.0058]	[0.0061, 0.0064]	[0.0064, 0.0066]	[0.0080, 0.0083]	[0.0117, 0.0122]

To summarize our large-scale evaluation, our main algorithm (Algorithm 8) can handle terabyte-sized quantile regression problems easily, obtaining, e.g., 2 digits of accuracy by sampling about $1e5$ rows on a problem of size $1e10 \times 11$. In addition, its running time is competitive with the best existing random sampling algorithms, and it can be applied in

parallel and distributed environments. However, its capability is restricted by the size of RAM since some steps of the algorithms are needed to be performed locally.

6.3 COMPUTATIONAL RESULTS FOR CX DECOMPOSITION

Recall that in Section 2.4.2, we describe CX decompositions. CX decompositions are low-rank matrix decompositions that are expressed in terms of a small number of actual columns/rows. As such, they have found applicability in various scientific applications. Here in this section, we present computational results of implementation of CX decomposition (Algorithm 4) in parallel and distributed environments using Spark with 1TB-sized mass spectrometry imaging data sets.

6.3.1 *Implementation of CX decompositions in Spark*

The main consideration when implementing CX decompositions are efficient implementations of operations involving the data matrix A. All access of A by the CX occurs through the RANDOMIZEDSVD routine (Algorithm 2). RANDOMIZEDSVD in turn accesses A only through the MULTIPLYGRAMIAN routine, with repeated invocations of MULTIPLYGRAMIAN accounting for the majority of the execution time.

The matrix A is stored as an RDD containing one `IndexedRow` per row of the input matrix, where each `IndexedRow` consists of the row's index and corresponding data vector. This is a natural storage format for many datasets stored on a distributed or shared file system, where each row of the matrix is formed from one record of the input dataset, thereby preserving locality by not requiring data shuffling during construction of A.

We then express MULTIPLYGRAMIAN in a form amenable to efficient distributed implementation by exploiting the fact that the matrix product $A^T AB$ can be written as a sum of outer products, as shown in Algorithm 3. This allows for full parallelism across the rows of the matrix with each row's contribution computed independently, followed by a summation step to accumulate the result. This approach may be implemented in Spark as a `map` to form the outer products followed by a `reduce` to accumulate the results:

```
def multiplyGramian(A: RowMatrix, B: LocalMatrix) =
  A.rows.map(row => row * row.t * B).reduce(_ + _)
```

However, this approach forms $2m$ unnecessary temporary matrices of same dimension as the output matrix $n \times k$, with one per row as the result of the `map` expression, and the `reduce` is not done in-place so it too allocates a new matrix per row. This results in high Garbage Collection (GC) pressure and makes poor use of the CPU cache, so we instead remedy this by accumulating the results in-place by replacing the `map` and `reduce` with a single `treeAggregate`. The `treeAggregate` operation is equivalent to a map-reduce that executes in-place to accumulate the contribution of a single worker node, followed by a tree-structured reduction that efficiently aggregates the results from each worker. The reduction is performed in multiple stages using a tree topology to avoid creating a single bottleneck at the driver node to accumulate the results from each worker node. Each worker emits a relatively large result with dimension $n \times k$, so the communication latency savings of having multiple reducer tasks is significant.

```
def multiplyGramian(A: RowMatrix, B: LocalMatrix) = {
  A.rows.treeAggregate(LocalMatrix.zeros(n, k))(
```

Table 22: Specifications of the three hardware platforms used in these performance experiments.

PLATFORM	TOTAL CORES	CORE FREQUENCY	INTERCONNECT	DRAM	SSDs
Amazon EC2 `r3.8xlarge`	960 (32 per-node)	2.5 GHz	10 Gigabit Ethernet	244 GiB	2 x 320 GB
Cray XC40	960 (32 per-node)	2.3 GHz	Cray Aries [Alv+12; Faa+12]	252 GiB	None
Experimental Cray cluster	960 (24 per-node)	2.5 GHz	Cray Aries [Alv+12; Faa+12]	126 GiB	1 x 800 GB

```
    seqOp = (X, row) => X += row * row.t * B,
    combOp = (X, Y) => X += Y
  )
}
```

6.3.2 *Experimental setup*

6.3.2.1 *MSI Datasets*

We experiment with mass spectrometry imaging (MSI) datasets. Mass spectrometry measures ions that are derived from the molecules present in a complex biological sample. These spectra can be acquired at each location (pixel) of a heterogeneous sample, allowing for collection of spatially resolved mass spectra. This mode of analysis is known as *mass spectrometry imaging (MSI)*. The addition of *ion-mobility separation (IMS)* to MSI adds another dimension, drift time The combination of IMS with MSI is finding increasing applications in the study of disease diagnostics, plant engineering, and microbial interactions. See Chapter 7 for more details about mass spectrometry imaging.

Unfortunately, the scale of MSI data and complexity of analysis presents a significant challenge to scientists: a single 2D-image may be many gigabytes and comparison of multiple images is beyond the capabilities available to many scientists. The addition of IMS exacerbates these problems. Dimensionality reduction techniques can help reduce MSI datasets to more amenable sizes. Typical approaches for dimensionality reduction include PCA and NMF, but interpretation of the results is difficult because the components extracted via these methods are typically combinations of many hundreds or thousands of features in the original data. CX decompositions circumvent this problem by identifying small numbers of columns in the original data that reliably explain a large portion of the variation in the data. This facilitates rapidly pinpointing important ions and locations in MSI applications.

Here, we analyze one of the largest (1TB-sized) mass-spec imaging datasets in the field. The sheer size of this dataset has previously made complex analytics intractable. Here we mainly present the computational results. Science results can be found in Chapter 7.

6.3.2.2 *Platforms*

In order to assess the relative performance of CX matrix factorization on various hardware, we choose the following contemporary platforms:

- a Cray® XC40™ system [Alv+12; Faa+12],
- an experimental Cray cluster, and
- an Amazon EC2 `r3.8xlarge` cluster.

For all platforms, we size the Spark job to use 960 executor cores (except as otherwise noted). Table 22 shows the full specifications of the three platforms. Note that these are state-of-the-art configurations in data centers and high performance computing centers.

6.3.2.3 *CX Spark Phases*

Our implementations of CX decompositions execute RANDOMIZEDSVD subroutine, which accounts for the bulk of the runtime and all of the distributed computations. The execution of RANDOMIZEDSVD proceeds in four distributed phases listed below, along with a small amount of additional local computation.

1. **Load Matrix Metadata** The dimensions of the matrix are read from the distributed filesystem to the driver.
2. **Load Matrix** A distributed read is performed to load the matrix entries into an in-memory cached RDD containing one entry per row of the matrix.
3. **Power Iterations** The MULTIPLYGRAMIAN loop of RANDOMIZEDSVD is run to compute an approximate Q of the dominant right singular subspace.
4. **Finalization (Post-Processing)** Right multiplication by Q of RANDOMIZEDSVD to compute C.

6.3.3 *Strong scaling test*

Figure 25 shows how the distributed Spark portion of our code scales. We consider 240, 480, and 960 cores. An additional doubling (to 1920 cores) would be ineffective as there are only 1654 partitions, so many cores would remain unused. When we go from 240 to 480 cores, we achieve a speedup of 1.6x: 233 seconds versus 146 seconds. However, as the number of partitions per core drops below two, and the amount of computation-per-core relative to communication overhead drops, the scaling slows down (as expected). This results in a lower speedup of 1.4x (146 seconds versus 102 seconds) from 480 to 960 cores.

6.3.4 *CX Performance across multiple platforms*

Table 23 shows the total runtime of CX for the 1 TB dataset on our three platforms. The distributed Spark portion of the computation is also depicted visually in Figure 26 for $k = 16$ and $k = 32$ on the 1 TB dataset. All three platforms were able to successfully process the 1 TB dataset in under 25 minutes. As the table and figure illustrates, most of the variation between the platforms occurred during the `MultiplyGramian` iterations. We now explore how these difference relate to the performance of the matrix iterations.

Spark divides each iteration into two stages. The first *local* stage computes each row's contribution, sums the local results (the rows computed by the same worker node), and records these results. The second *aggregation* stage combines all of the workers' locally-aggregated results using a tree-structured reduction. Most of the variation between platforms occurs during the aggregation phase, where data from remote worker nodes is fetched and combined. In Spark, all inter-node data exchange occurs via *shuffle operations*. In a shuffle, workers with data to send write the data to their local scratch space. Once all data has been written, workers with data to retrieve from remote nodes request that

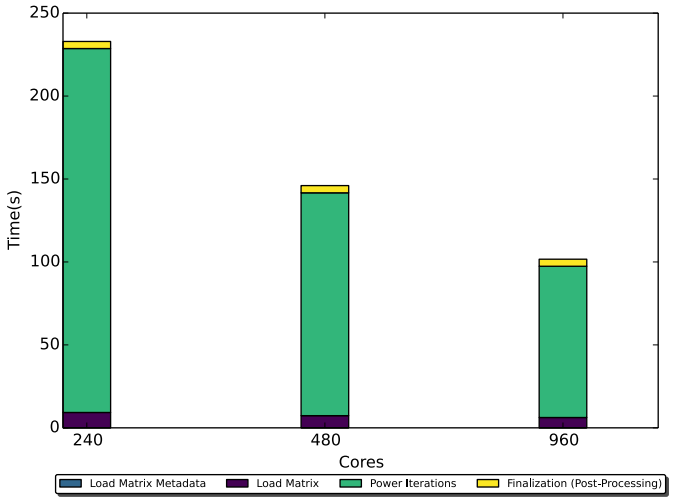


Figure 25: Strong scaling for the 4 phases of CX on an XC40 for 100GB dataset at $k = 32$ and default partitioning as concurrency is increased.

Table 23: Total runtime for the 1 TB dataset ($k = 16$), broken down into load time and per-iteration time. The per-iteration time is further broken down into the average time for each task of the local stage and each task of the aggregation stage. We also show the average amount of time spent waiting for a network fetch, to illustrate the impact of the interconnect.

PLATFORM	TOTAL RUNTIME	LOAD TIME	TIME PER ITERATION	AVERAGE LOCAL TASK	AVERAGE AGGREGATION TASK	NETWORK WAIT
Amazon EC2 `r3.8xlarge`	24.0 min	1.53 min	2.69 min	4.4 sec	27.1 sec	21.7 sec
Cray XC40	23.1 min	2.32 min	2.09 min	3.5 sec	6.8 sec	1.1 sec
Experimental Cray cluster	15.2 min	0.88 min	1.54 min	2.8 sec	9.9 sec	2.7 sec

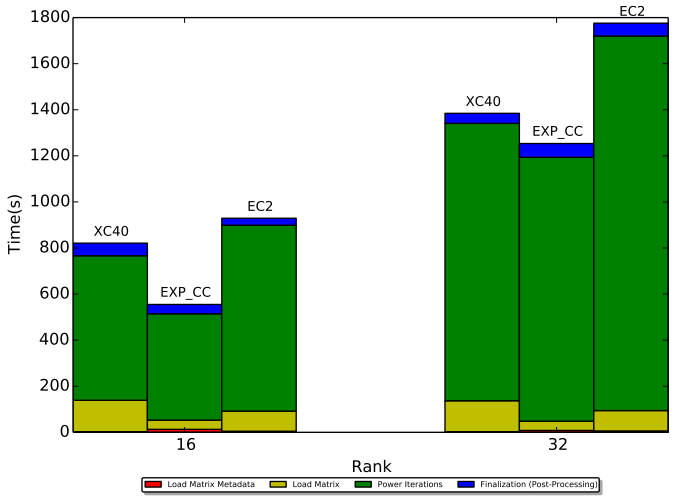


Figure 26: Run times for the various stages of computation of CX on the three platforms using $k = 16$ and $k = 32$ on the 1 TB size dataset, using the default partitioning on each platform.

data from the sender's block manager, which in turns retrieves if from the senders local scratch space, and sends it over the interconnect to the receiving node.

Examining our three platforms (Table 22), we notice two key hardware differences that impact shuffle operations:

- First, both the EC2 nodes and the experimental Cray cluster nodes have fast SSD storage local to the compute nodes that they can use to store Spark's shuffle data. The Cray® XC40™ system's [Alv+12; Faa+12] nodes, on the other hand, have no local persistent storage devices. Thus we must emulate local storage with a remote Lustre filesystem. The impacts of this can be somewhat mitigated, however, by leaving sufficient memory to store some of the data in a local RAM disk, and/or to locally cache some of the remote writes to Lustre.[3]

- Second, the Cray XC40 and the experimental Cray cluster both communicate over the HPC-optimized Cray Aries interconnect [Alv+12; Faa+12], while the EC2 nodes use 10 Gigabit Ethernet.

We can see the impact of differing interconnect capabilities in the Average Network Wait column in Table 23. These lower average network wait times explain why the two Cray platforms outperform the EC2 instance (with the experimental cluster achieving a speedup of roughly 1.5x over EC2).

The XC40 is still slightly slower than the experimental Cray cluster, however. Part of this difference is due to the slower matrix load phase on the XC40. On EC2 and the experimental Cray cluster, the input matrix is stored in SSDs on the nodes running the Spark executors. Spark is aware of the location of the HDFS blocks, and attempts to schedule tasks on the same nodes as their input. The XC40, however, lacks SSDs on its compute

3 This is an ideal usage of caching, since Spark assumes the scratch space is only locally accessible; thus we are guaranteed that the only node that reads a scratch file will be the same node that wrote it.

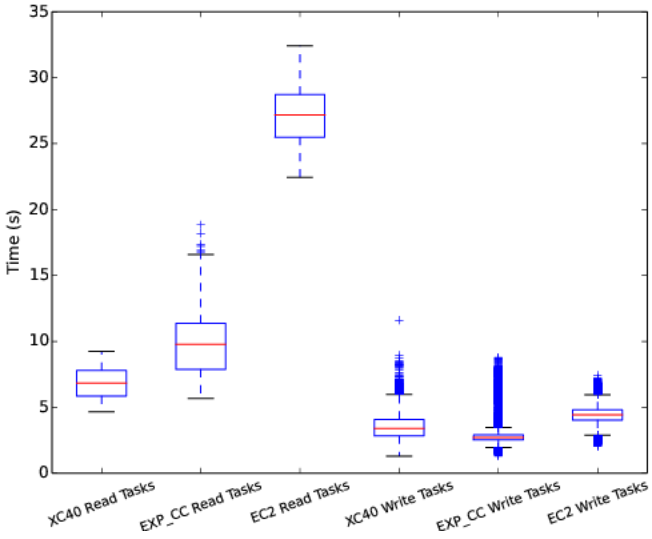


Figure 27: A box and whisker plot of the distribution of local (write) and aggregation (read) task times on our three platforms for the 1TB dataset with $k = 16$. The boxes represent the 25th through 75th percentiles, and the lines in the middle of the boxes represent the medians. The whiskers are set at 1.5 box widths outside the boxes, and the crosses are outliers (results outside the whiskers). Note that each iteration has 4800 write tasks and just 68 read tasks.

nodes, so the input matrix is instead stored on a parallel Lustre file system. The increased IO latency slows the input tasks. The rest of the difference in performance can be understood by looking at the distribution of local (write) task times in the box and whiskers plot in Figure 27. The local/write tasks are much more numerous than the aggregation/read tasks (4800 vs 68 per iteration), thus they have a more significant impact on performance. We see that the XC40 write tasks had a similar median time to the experimental cluster's write tasks, but a much wider distribution. The large tail of slower "straggler" tasks is the result of some shuffle data going to the remote Lustre file system rather than being cached locally. We enabled Spark's optional speculative re-execution (`spark.speculation`) for the XC40 runs, and saw that some of these tasks were successfully speculatively executed on alternate nodes with more available OS cache, and in some case finished earlier. This eliminated many of the straggler tasks and brought our performance closer to the experimental Cray cluster, but still did not match it (the results in Figure 26 and Table 23 include this configuration optimization).

7 APPLICATIONS OF CX DECOMPOSITION TO BIOIMAGING

Recent advances in chemical imaging techniques have enabled detailed investigation of metabolic processes at length scales ranging from sub-cellular to centimeter resolution. One of the most promising chemical imaging techniques is mass spectrometry imaging (MSI) [CFG97; MH07].

In this chapter, we apply CX matrix decompositions (described in Section 2.4.2) to mass spectrometry imaging datasets, and we show that this can lead to effective prioritization of information, both in terms of identifying important ions as well as in terms of identifying important positions. Previously, this approach has been applied to the study of gene expression [MD09; Pas+07] and astronomy [Yip+14]. The material in this chapter appears in Yang et al. [Yan+15] and Gittens et al. [Git+16a].

7.1 BACKGROUND ON MSI DATASETS AND METHODS

Typically in mass spectrometry imaging, a laser or ion beam is raster scanned across a surface. At each location, molecules are desorbed from the surface, often with the assistance of a matrix coating or specially prepared surface that enables the formation of gas phase ions. These ions are collected and analyzed by mass spectrometry [CH10]. MSI presents many data analysis and interpretation challenges due to the size and complexity of the data. MSI acquires one or more mass spectra at each location. Each spectrum is digitized into 10^4 to 10^6 m/z bins. Depending on the sample and analysis technique, it is common to have tens of thousands of intense, sharp peaks at each location. Likewise, MSI datasets containing up to a million pixels are possible with existing technology. This results in a situation where careful analysis requires sophisticated computational tools, infrastructure, and algorithms to reduce the large volume of measured data into easier to interpret smaller blocks with the goal of prioritizing ions and positions according to their importance.

Two widely used techniques for this are principle component analysis (PCA) [Jol86] and non-negative matrix factorization (NMF) [LS01], both of which express the original data in terms of concise but in general difficult-to-interpret components [Jon+12; Rei+11] since, for example, eigenvectors are often not meaningful in terms of the physical processes of metabolism, sample preparation, and data collection; and in addition, it is not always clear whether a single ion is the distinguishing characteristic of a region or whether it is a complex combination of relative ion-intensities that distinguish regions.

In contrast, CUR and the related CX matrix decompositions (described in Section 2.4.2) are relatively new algorithmic approaches that allow scientists to provide a low-rank approximation of the measured data that is expressed in terms of actual data elements [MD09; DMM08]. CX and CUR decompositions are provably almost as good as the low-rank approximation provided by the SVD, but instead of the blocks containing eigenions and eigenpositions, as they do with the SVD, the low rank approximation provided by CX/CUR is expressed in terms of actual rows and/or columns, i.e., actual ions and/or actual positions. In this chapter, we focus on CX decomposition. As is illustrated in Figure 28, the

CX decomposition uses the leverage score structure within SVD to find actual rows and actual columns of an MSI matrix that are most informative.

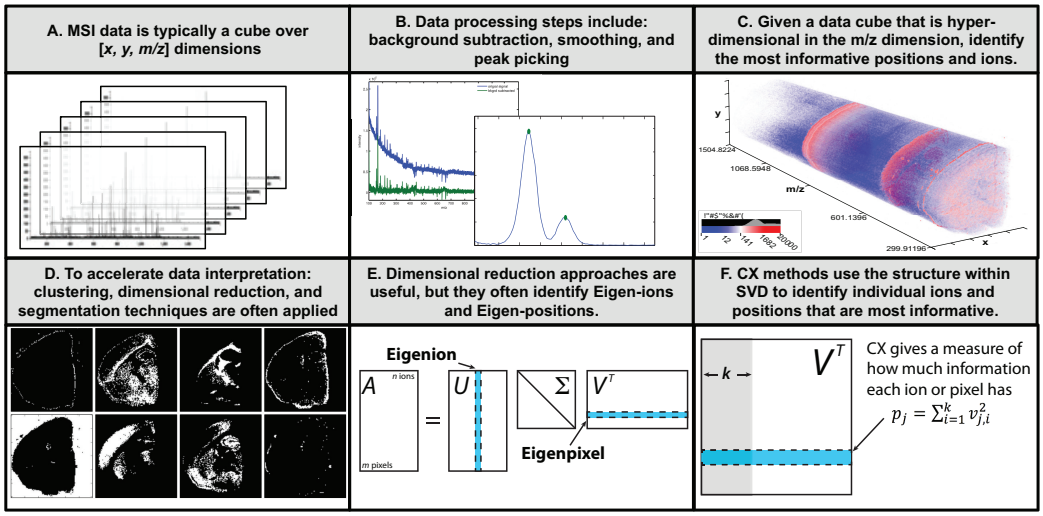


Figure 28: Mass spectrometry imaging collects one or more spectra at each location in a sample. Because of the scale and complexity of MSI data, computational tools are required to reach an understanding of the underlying physical processes. Panel A-D: A traditional processing workflow where raw data is cleaned and processed using traditional clustering and dimension reduction methods. Panel E: Multivariate statistics, such as PCA, yield informative combinations of ions and pixels, but they do not lend themselves to intuitive interpretation in terms of the biological processes generating the data. Panel F: In contrast, CX decomposition yields the most informative actual ions and actual positions instead of linear combinations of ions and positions.

Recall that in Section 2.4.2 we provided a brief overview of CX decomposition with approximate leverage scores; a more detailed introduction can be found in [DMM08]. Here in our experiments, we consider using several variants of CX decomposition described in Algorithm 4.

More specifically, as the statistical leverage scores are central to CX decomposition, for completeness, we use CX decomposition with exact leverage scores on smaller datasets, but to apply CX decompositions to larger datasets we use CX decompositions with approximate leverage scores (Algorithm 4). Then, with these (normalized) leverage scores at hand, one can select columns from A either by viewing p_j's as an importance sampling distribution over the columns and randomly sampling columns according to it (Algorithm 4), or by viewing p_j's as a ranking function and greedily selecting the columns with highest scores. We name them RANDCOLSELECT and DETERCOLSELECT, respectively.

RANDCOLSELECT Select c columns from A, each of which is randomly sampled according to the normalized leverage scores $\{p_j\}_{j=1}^{n}$.

DETERCOLSELECT Select the c columns of A corresponding to the largest c normalized leverage scores p_js.

Finally, our main algorithm can be summarized as follows. It takes as input an $m \times n$ matrix A, a rank parameter k, and desired number of columns c as inputs.

1. Compute the (normalized) leverage scores associated with rank k either exactly or approximately.

2. Select c columns from A according to RandColSelect or DeterColSelect.
3. Let $X = C^{\dagger}A$.

In each computation that is described in the next section, after having specified which scheme is used to compute the leverage scores, we respectively use *randomized CX decomposition* and *deterministic CX decomposition* to denote the algorithm CX Decomposition with RandColSelect or DeterColSelect scheme.

7.2 RESULTS AND DISCUSSION

In the following we use three datasets to demonstrate the utility of CX decompositions for MSI. Two of them are publicly available on the OpenMSI web gateway, and they are selected from two diverse acquisition modalities, including one NIMS image of the left coronal hemisphere of a mouse brain acquired using a time-of-flight (TOF) mass analyzer and one MSI dataset of a lung acquired using an Orbitrap mass analyzer [Lou+13; R̈13]. These files are previously described elsewhere and were chosen because of the commonality of brain-lipid images and the large number of m/z bins generated by Orbitrap detectors, respectively. The other dataset is the 1TB-sized one mentioned in Section 6.3. This dataset is provided by Norman Lewis's group at Washington State University and is derived from a MALDI-IMS-MSI analysis of a rhizome of the drug-producing green shrub Podophyllum hexandrum. For simplicity, we call them Brain, Lung, and Plant dataset, respectively, and present their science results in the next three subsections respectively.

7.2.1 *Results on Brain dataset*

To illustrate the utility of CX decompositions, we focus initially on results obtained from the NIMS image of a coronal brain section. For the analyses described for the NIMS brain image, the data were processed using peak-finding. The peak-finding identifies the most intense ions and integrates the peaks, so that each peak is represented by a single image, rather than a series of images spanning a range of m/z values. Using this approach, the original data is reduced from $\mathcal{O}(100,000)$ m/z values to the most intense ions. The size of the brain section dataset is $(122 \times 120 \times 1926)$. The (i, j, l)-th value of the matrix represents the intensity of the ion with the l-th m/z value at position (i, j) in a (122×120) regular lattice which discretizes physical space. To compute the CX decomposition and select ions and spectra, we reshape the three-dimensional MSI data cube into a two-dimensional (14640×1926) matrix A, where each row of A corresponds to the spectrum of a pixel in the image, and where each column of A corresponds to the intensities of an ion over all pixels, describing the distribution of the ion in physical space. For finding informative ions and pixels, we perform CX decomposition with exact computations for leverage scores on A and A^T, respectively. In each case, for clarity, we only report the results with a fixed small value of the rank parameter k. Varying in a range of small values does not have a large effect on the reconstruction errors. This behavior may indicate that the information that the corresponding top-k singular spaces contain does not vary a lot as k varies in this range.

7.2.1.1 *Finding Important Ions*

Figure 29A shows the reconstruction errors $\|A - CX\|_F / \|A\|_F$ using CX decomposition for selection of $c = 20, 30, 40, 50, 60$ ions, using a rank parameter $k = 5$ and using both

randomized and deterministic CX decompositions. For completeness, we also show the reconstruction errors using uniform sampling for varying numbers of selected ions and that of the optimal rank-k approximation of A. Figure 29B and 29C show the distribution of the leverage scores of A, relative to the best rank-k space, and their relative magnitudes. Figure 30 then presents the spatial distributions of the 30 most important ions selected using deterministic CX decomposition with $k = 5$ and $c = 20$.

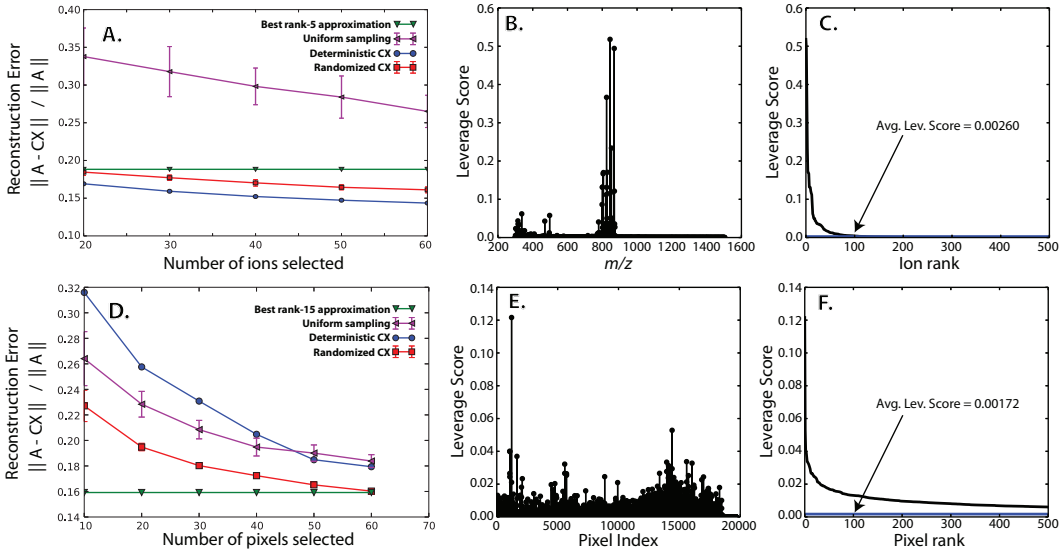


Figure 29: Analysis of the reconstruction error used to determine the most appropriate CX-based schemes and settings for selection of ions and locations/spectra. Panel A,D: Reconstruction error of the CX decomposition for selection of ions (panel A) and locations (panel D) using randomized and deterministic selection schemes with varying parameter c. Panel B,E: Distribution of leverage scores of A and A^T, relative to the best rank-k space, respectively. Panel C,F: Sorted distribution of these leverage scores of A and A^T, respectively. The blue horizontal line denotes the mean/average leverage score. Due to the fairly non-uniform shape of the leverage score distribution for ions, deterministic CX selection outperforms randomized CX sampling for ions. In contrast, pixel selection is best achieved by randomized CX sampling, since the leverage score distribution for pixels is much more uniform.

The selection of important ions from the brain dataset (Figure 29A) shows clearly that using deterministic CX decomposition leads to a smaller error than using randomized CX decomposition with the same parameters. The reason for this behavior lies in distribution of the leverage scores for the ions, as shown in Figure 29B and 29C. These leverage scores are very non-uniform: a few dozen leverage scores are much larger, e.g., 50 times larger, than the average score. Hence, since the leverage scores are highly non-uniform, the corresponding ions can be considered as very informative in reconstructing the matrix, and keeping the ions with the top leverage scores leads a good basis. The randomized CX decomposition carries a large variance—for the values of the parameters used here—since in many trials it failed to select those important ions, and thus it resulted in a large error. Not surprising, uniformly selecting columns do not give particularly meaningful results, i.e., many irrelevant ions were chosen and informative ions were not chosen.

As for the absolute magnitude of the error, we use that of the best rank-k approximation of A, i.e., A_k, as a reference scale suggested by (2.11). In Figure 29A and 29D, we can see

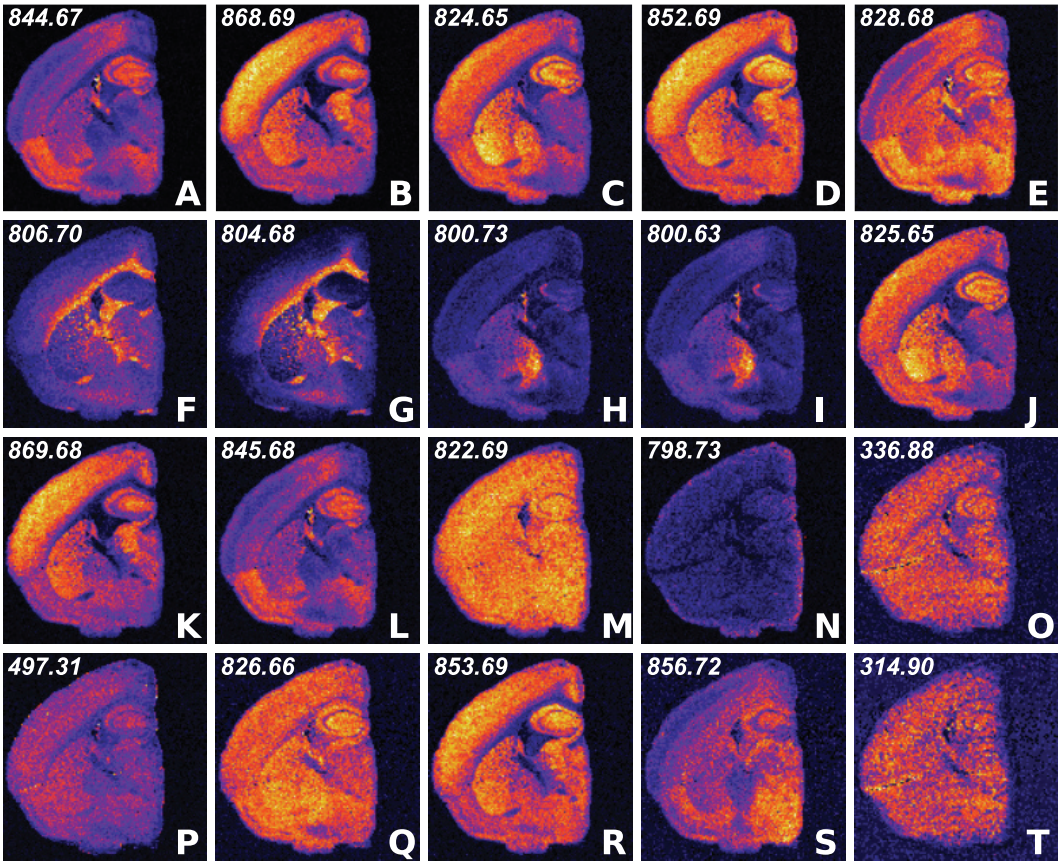


Figure 30: Ion-intensity visualization of the 20 most important ions selected via deterministic CX decomposition with $k = 5$ and $c = 20$. The distribution of leverage scores is presented in Figure 29B. Some of these ions map to distinct regions in the brain. Particular regions of the cortex, pons, and corpus collosum stand out as distinct anatomically identifiable regions. Also in the list are likely background ions and contaminants from the embedding material. Of the 20 ions, little redundancy is present, pointing to the effectiveness of the CX approach for information prioritization.

that the reconstruction error of the CX decomposition is close to that of A_k. In some cases, CX decomposition can even produce a lower error. This is because the matrix CX returned by CX decomposition is a rank-c matrix with $c > k$. It is possible to choose X to be a rank-k matrix; see Section 4.3 in [DMM08] for detailed construction.

7.2.1.2 *Finding Important Pixel/Spectra*

Similar to Figure 29A-C, Figure 29D-F provide an overview of the reconstruction errors and the distribution and magnitude of the leverage scores, relative to the best rank-k approximation, for the application of CX decomposition to A^T for selection of pixel. In Figure 31, we illustrate the application of both randomized and deterministic CX decompositions, with $k = 15$ and $c = 20$, on A^T for finding informative pixels. The first subplot (Figure 31A) shows the result returned by the deterministic CX decomposition, meaning the pixels with the top leverage scores are greedily selected and plotted. The remaining

subplots in Figure 31B-F we show the results returned by running randomized CX decomposition in five independent trials.

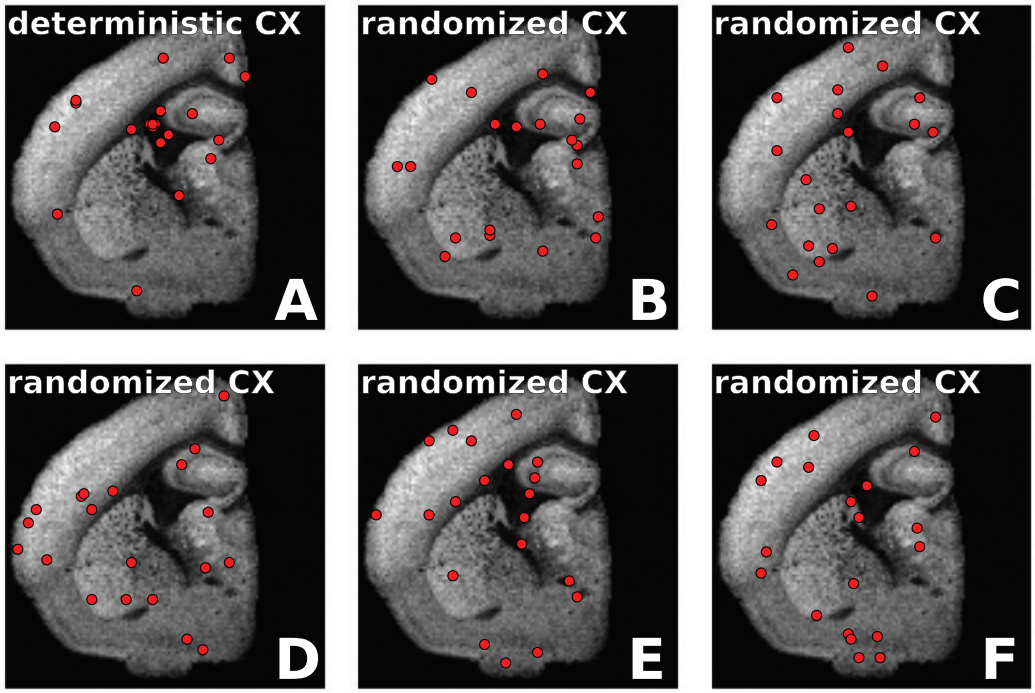


Figure 31: Visualization of the selection of important pixels using CX decompositions. All visualizations show a gray-scale image of a selected ion as context, and the 20 locations selected using the CX decomposition with $k = 15$ and $c = 20$ are highlighted via red circles. Panel A shows the result of using the deterministic CX decomposition. With this approach the algorithm selects locations clustered around a few regions. In comparison, panels B-F show the results from five independent trials using the randomized CX decomposition. Because of the uniformity in leverage scores for pixels, the randomized selection outperforms the deterministic approach for comprehensive sampling of important locations. The distribution of leverage scores is presented in Figure 29E.

In contrast with the selection of ions, deterministic CX decomposition results in larger reconstruction errors than randomized CX decomposition (Figure 29D). Also, the pixels selected using CX tend to be more localized in specific regions of the images, rather than selecting characteristic pixels from different physical components of the sample images. The reason for this behavior lies in the distribution of the leverage scores for the pixels, as shown in Figures 29E and 29F. These leverage scores are fairly uniform: most of them are less than 20 times the average. Also, there are many more pixels than ions, and thus we can consider the distribution of leverage scores to be fairly uniform. Furthermore, since each row in A represents a pixel in the image, many rows will contain a similar spectra. Similar locations tend to "split up" the leverage scores, resulting in smaller values for the score at each location. Importantly, applying random sampling here may still be able to identify pixels from the important regions (i.e., those with high total leverage scores), even when the value of any of its single pixel is small.

7.2.2 *Results on Lung dataset*

Next, we investigate the quality of the approximation of the leverage scores by which we mean Algorithm 4. We consider a moderately large dataset on which performing the full SVD exactly will take hours to finish. In particular, we present the result on the raw lung data before peak-finding, which has size approximately $20,000$ by $500,000$. We apply Algorithm 4 with $k = 15$ and with $q = 5$ on the raw lung dataset.

As no peak-finding was done on the raw dataset, some ions with high leverage scores have similar m/z values. In Figure 32a, we present the spatial distributions of the 4 most representative ions selected from different groups. In Figures 32b and 32c, the approximate leverage scores and the total sensitivities versus the m/z values are plotted, respectively. In addition, a "zoom-in" version of the above two plots, overlaid on each other, on ions with m/z values in the range between 866.02 and 866.75 is shown in Figure 32d.

The results suggest that the ion at $m/z = 392$ (a drug administered to the tissue) was identified as the highest leverage ion, and ions specific to regions of the lung were also identified. That the administered drug was identified as the highest importance ion could be significant for pharmacokinetics/pharmacology and could also be a marker to accelerate identification of degradation products or byproducts that are of unexpected/unpredetermined m/z values.

What is most significant in this approach is the lack of reliance on peak-finding. By applying scalable factorization approaches like CX and CUR to raw, profile spectra, a multitude of previously-ignored features can be considered. As can be seen in Figure 32d, the zoomed in portion of the leverage score overlaid with the total intensity spectra shows a large number of recognizable features with high intensity. Strikingly, only one of these features has a high leverage score. This prioritization allows accelerated interpretation of results by pointing a researcher towards which ions might be most informative in a mathematically more objective manner.

7.2.3 *Results on Plant dataset*

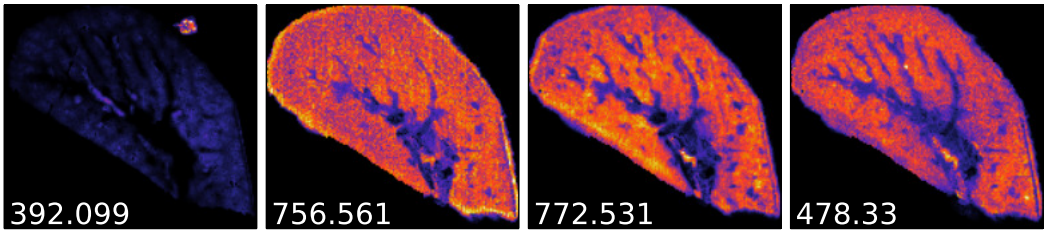
Finally, we present the science results for the terabyte-sized plant dataset used in Section 6.3. The size of the dataset is approximately $2.5e5$ by $1.6e7$ (511-by-507 spatial pixels by 460342 m/z channels by 200 ion mobility channels).

Again, the rows and columns of our data matrix A correspond to pixels and $(\tau, m/z)$ values of ions, respectively. We compute the CX decompositions of both A and A^T in order to identify important ions in addition to important pixels.

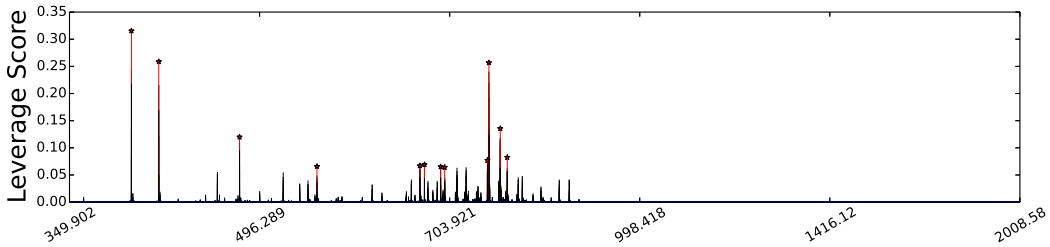
In Figure 33a, we present the distribution of the normalized ion leverage scores marginalized over τ. That is, each score corresponds to an ion with m/z value shown in the x-axis. Leverage scores of ions in three narrow regions have significantly larger magnitude than the rest. This indicates that these ions are more informative and should be kept as basis for reconstruction. Encouragingly, several other ions with significant leverage scores are chemically related to the ions with highest leverage scores. For example, the ion with an m/z value of 453.0983 has the second highest leverage score among the CX results. Also identified as having significant leverage scores are ions at m/z values of 439.0819, 423.0832, and 471.1276, which correspond to neutral losses of CH_2, CH_2O, and a neutral "gain" of H_2O from the 453.0983 ion. These relationships indicate that this set of ions, all identified by CX as having significant leverage scores, are chemically related. That fact

indicates that these ions may share a common biological origin, despite having distinct spatial distributions in the plant tissue sample.

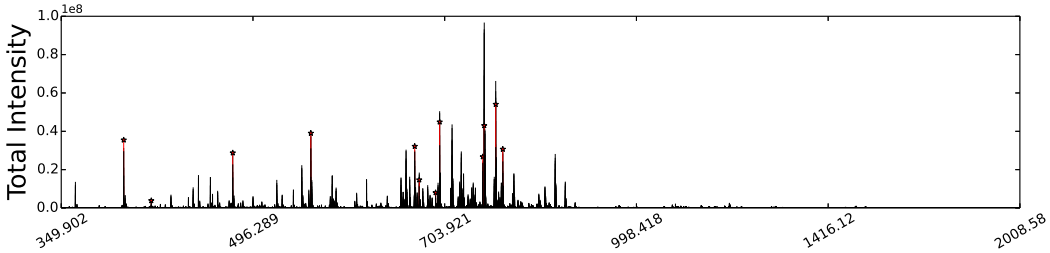
On the other hand, we observed that the pixel leverage scores are fairly uniform (not shown here). This is not surprising since similar pixels tend to have similar and small individual scores. An implication is that the length scales of spatial variation for most detected ions is much larger than the pixel size used for data acquisition. However, for each region that contains similar pixels, the total leverage score (sampling probability) will still be higher if such a region is more important in the reconstruction. Therefore, a random sample is still able to capture which larger-scale regions contain higher densities of important pixels, by sampling more pixels from the region as shown in Figure 33b.



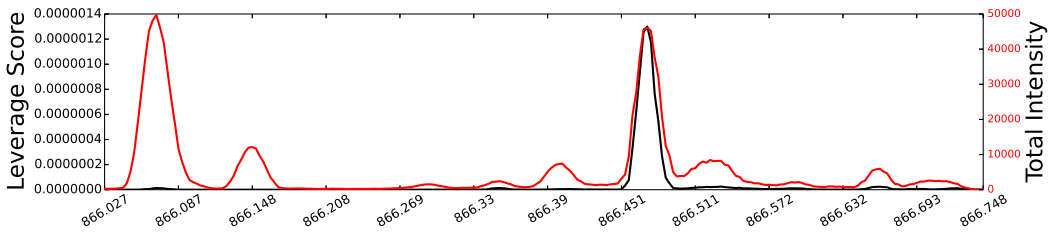
(a) Ion-intensity visualization of 4 important ions



(b) Leverage score as a function of m/z value

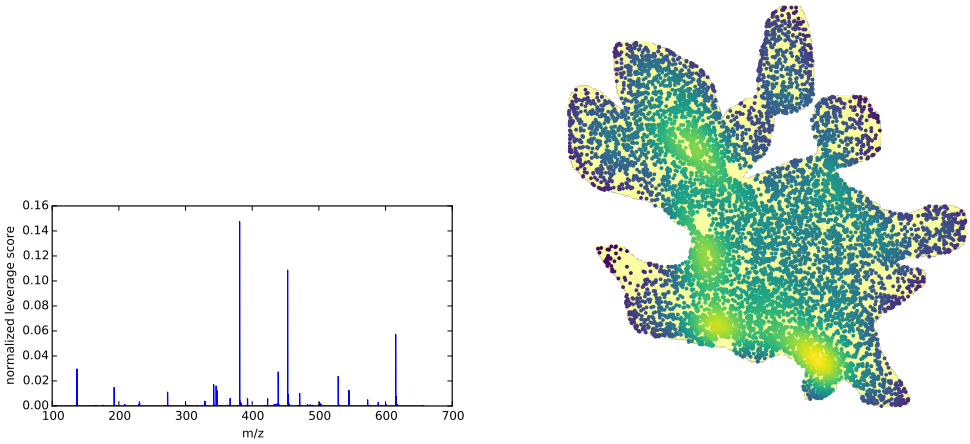


(c) Total intensity as a function of m/z value



(d) Zoom-in version of panel (b) and (c) with m/z value ranging from 866.02 to 866.75

Figure 32: Quality of the normalized leverage scores using Algorithm 4 with $q = 5$ on the raw lunch dataset. In panel A, we select 4 ions that are the most representative from the 30 most important ions returned by running deterministic CX decomposition. In panel B, we plot the approximate normalized leverage scores versus the m/z value. The ions with highest leverage scores are marked by red stars. Note, for a group of ions with similar m/z values and high leverage scores, only the one with the highest leverage score is plotted. In panel C, the total sensitivities are plotted. The same ions marked in panel B are marked. In panel D, a zoom-in version of panel B,C when the m/z value is ranging from 866.02 to 866.75 is shown. The black and red curves are the leverage scores and sensitivities, respectively.



(a) Normalized ion leverage scores

(b) Visualization of important pixels

Figure 33: (a) Normalized leverage scores (sampling probabilities) for m/z marginalized over τ. Three narrow regions of m/z account for 59.3% of the total probability mass; (b) Plot of 10000 points sampled by leverage score. Color and luminance of each point indicates density of points at that location as determined by a Gaussian kernel density estimate.

Part III

OTHER WORKS FOR LARGE-SCALE DATA APPLICATIONS

8

QUASI-MONTE CARLO FEATURE MAPS FOR SHIFT-INVARIANT KERNELS

Kernel methods [SS02; Wah90] offer a comprehensive suite of mathematically well-founded non-parametric modeling techniques for a wide range of problems in machine learning. These include nonlinear classification, regression, clustering, semi-supervised learning [BNS06], time-series analysis [Par70], sequence modeling [Son+10], dynamical systems [BGG13], hypothesis testing [Har+13], causal modeling [Zha+11] and many more.

However, there is a steep price to these elegant generalizations in terms of scalability. Consider, for example, least squares regression given n data points $\{(x_i, y_i)\}_{i=1}^n$ and assume that $n \gg d$. The complexity of linear regression training using standard least squares solvers is $O(nd^2)$ with $O(nd)$ memory requirements and $O(d)$ prediction speed on a test point. Its kernel-based nonlinear counterpart, however, requires solving a linear system involving the Gram matrix of the kernel function (defined by $K_{ij} = k(x_i, x_j)$). In general, this incurs $O(n^3 + n^2 d)$ complexity for training, $O(n^2)$ memory requirements, and $O(nd)$ prediction time for a single test point—none of which is particularly appealing in "big data" settings. Similar conclusions apply to other algorithms such as Kernel PCA.

This is rather unfortunate because non-parametric models, such as the ones produced by kernel methods, are particularly appealing in big data settings as they can adapt to the full complexity of the underlying domain, as uncovered by increasing dataset sizes. It is well known that imposing strong structural constraints upfront for the purpose of allowing an efficient solution (in the above example: a linear hypothesis space) often limits, both theoretically and empirically, the potential to deliver value on large amounts of data. Thus, as big data becomes pervasive across a number of application domains, it has become necessary to develop highly scalable algorithms for kernel methods.

In this chapter, we consider the problem of improving the efficiency of randomized Fourier feature maps [RR08] to accelerate training and testing speed of kernel methods on large datasets. These approximate feature maps arise as Monte Carlo approximations to integral representations of shift-invariant kernel functions (e.g., Gaussian kernel). We propose to use Quasi-Monte Carlo (QMC) approximations instead, where the relevant integrands are evaluated on a low-discrepancy sequence of points as opposed to random point sets as in the Monte Carlo approach. We derive a new discrepancy measure called *box discrepancy* based on theoretical characterizations of the integration error with respect to a given sequence.

We provide background on kernel methods, random Fourier feature maps, and Quasi-Monte Carlo technique in Section 8.1 and Section 8.2. Main algorithm and main theoretical results are presented in Section 8.3. In Section 8.4 we discuss learning adaptive QMC sequences to improve the algorithm further. Finally, empirical results are presented in Section 8.5. The material in this chapter appears in Yang et al. [Yan+14a] and Avron et al. [Avr+16].

Note that in this chapter, we use slightly different notation. In a sequence of vectors, we use w^i to denote the i-th element of the sequence and use w_j^i to denote the j-th coordinate of vector w^i. Given a vector x, we use x_i to denote the i-th coordinate of x.

8.1 BACKGROUND ON KERNEL METHODS AND RANDOM FOURIER FEATURE MAPS

The central object of kernel methods is a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ defined on an input domain $\mathcal{X} \subset \mathbb{R}^d$.[1] The kernel k is (non-uniquely) associated with an embedding of the input space into a high-dimensional Hilbert space $\mathcal{H}$ (with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$) via a feature map, $\Psi : \mathcal{X} \to \mathcal{H}$, such that

$$k(x, z) = \langle \Psi(x), \Psi(z) \rangle_{\mathcal{H}}.$$

Standard regularized linear statistical models in $\mathcal{H}$ then provide nonlinear inference with respect to the original input representation. The algorithmic basis of such constructions are classical Representer Theorems [Wah90; SS02] that guarantee finite-dimensional solutions of associated optimization problems, even if $\mathcal{H}$ is infinite-dimensional.

Recent years have seen intensive research on improving the scalability of kernel methods. In this work, we revisit one of the most successful techniques, namely the randomized construction of a family of low-dimensional approximate feature maps proposed by Rahimi and Recht [RR08]. These randomized feature maps, $\hat{\Psi} : \mathcal{X} \to \mathbb{C}^s$, provide low-distortion approximations for (complex-valued) kernel functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$:

$$k(x, z) \approx \langle \hat{\Psi}(x), \hat{\Psi}(z) \rangle_{\mathbb{C}^s}, \tag{8.1}$$

where $\mathbb{C}^s$ denotes the space of s-dimensional complex numbers with the inner product, $\langle \alpha, \beta \rangle_{\mathbb{C}^s} = \sum_{i=1}^{s} \alpha_i \beta_i^*$, with z^* denoting the conjugate of the complex number z. (Although Rahimi and Recht [RR08] also define real-valued feature maps for real-valued kernels, our technical exposition is simplified by adopting the generality of complex-valued features.) The mapping $\hat{\Psi}(\cdot)$ is now applied to each of the data points to obtain a randomized feature representation of the data. We then apply a simple linear method to these random features. That is, if our data is $\{(x^i, y^i)\}_{i=1}^n$ we learn on $\{(z^i, y^i)\}_{i=1}^n$, where $z^i = \hat{\Psi}(x^i)$. As long as s is sufficiently smaller than n, this leads to more scalable solutions, e.g., for regression we get back to $O(ns^2)$ training and $O(sd)$ prediction time, with $O(ns)$ memory requirements. This technique is immensely successful, and has been used in recent years to obtain state-of-the-art accuracies for some classical data sets [Hua+14; AS15].

The starting point of [RR08] is a celebrated result that characterizes the class of positive definite functions.

Definition 8.1. *A function $g : \mathbb{R}^d \mapsto \mathbb{C}$ is a* positive definite function *if for any set of n points $x^1, \ldots, x^n \in \mathbb{R}^d$, the $n \times n$ matrix A defined by $A_{ij} = g(x^i - x^j)$ is positive semidefinite.*

Theorem 8.2 (Bochner [Boc33])**.** *A complex-valued function $g : \mathbb{R}^d \mapsto \mathbb{C}$ is positive definite if and only if it is the Fourier Transform of a finite non-negative Borel measure μ on $\mathbb{R}^d$, i.e.,*

$$g(x) = \hat{\mu}(x) = \int_{\mathbb{R}^d} e^{-ix^T w} d\mu(w), \quad \forall x \in \mathbb{R}^d.$$

Without loss of generality, we assume henceforth that $\mu(\cdot)$ is a probability measure with associated probability density function $p(\cdot)$.

A kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{C}$ on $\mathbb{R}^d$ is called *shift-invariant* if $k(x, z) = g(x - z)$ for some positive definite function $g : \mathbb{R}^d \mapsto \mathbb{C}$. Bochner's theorem implies that a scaled

1 In fact, $\mathcal{X}$ can be a rather general set. However, here it is restricted to being a subset of $\mathbb{R}^d$.

shift-invariant kernel can therefore be put into one-to-one correspondence with a density $p(\cdot)$ such that

$$k(x,z) = g(x-z) = \int_{\mathbb{R}^d} e^{-i(x-z)^T w} p(w) dw. \tag{8.2}$$

For the most notable member of the shift-invariant family of kernels, the Gaussian kernel

$$k(x,z) = e^{-\frac{\|x-z\|_2^2}{2\sigma^2}},$$

the associated density is again Gaussian $\mathcal{N}(0, \sigma^{-2} I_d)$.

The integral representation of kernel (8.2) may be approximated as follows:

$$\begin{aligned} k(x,z) &= \int_{\mathbb{R}^d} e^{-i(x-z)^T w} p(w) dw \\ &\approx \frac{1}{s} \sum_{j=1}^{s} e^{-i(x-z)^T w^s} \\ &= \langle \hat{\Psi}_S(x), \hat{\Psi}_S(z) \rangle_{\mathbb{C}^s} \end{aligned}$$

through the feature map

$$\hat{\Psi}_S(x) = \frac{1}{\sqrt{s}} \left[e^{-ix^T w^1} \dots e^{-ix^T w^s} \right] \in \mathbb{C}^s. \tag{8.3}$$

The subscript S denotes dependence of the feature map on the sequence $S = \{w^1, \dots, w^s\}$.

The goal of this work is to improve the convergence behavior of this approximation, so that a smaller s can be used to get the same quality of approximation to the kernel function. This is motivated by recent work showing that in order to obtain state-of-the-art accuracy on some important data sets, a very large number of random features is needed [Hua+14; AS15].

Our point of departure from the work of Rahimi and Recht [RR08] is the simple observation that when $w^1, \dots, w^s$ are drawn from the distribution defined by the density function $p(\cdot)$, the approximation (8.3) may be viewed as a standard *Monte Carlo* (MC) approximation to the integral representation of the kernel. Instead of using plain MC approximation, we propose to use the low-discrepancy properties of *Quasi-Monte Carlo* (QMC) sequences to reduce the integration error in approximations of the form (8.3). A self-contained overview of Quasi-Monte Carlo techniques for high-dimensional integration problems is provided in the next section.

8.2 QUASI-MONTE CARLO TECHNIQUES: AN OVERVIEW

In this section we provide a self-contained overview of Quasi-Monte Carlo (QMC) techniques. For brevity, we restrict our discussion to background that is necessary for understanding subsequent sections. We refer the interested reader to the excellent reviews by Caflisch [Caf98] and Dick, Kuo, and Sloan [DKS13], and the recent book Leobacher and Pillichschammer [LP14] for a much more detailed exposition.

Consider the task of computing an approximation of the integral

$$I_d[f] = \int_{[0,1]^d} f(x) dx. \tag{8.4}$$

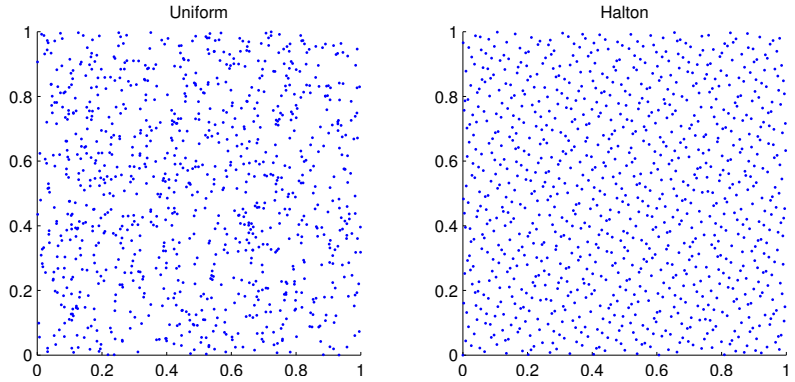


Figure 34: Comparison of MC and QMC sequences.

One can observe that if x is a random vector uniformly distributed over $[0,1]^d$ then $I_d[f] = \mathbb{E}[f(x)]$. An empirical approximation to the expected value can be computed by drawing a random point set $S = \{w^1, \ldots, w^s\}$ independently from $[0,1]^d$ and computing

$$I_S[f] = \frac{1}{s} \sum_{w \in S} f(w).$$

This is the Monte Carlo (MC) method.

Define the integration error with respect to the point set S as

$$\epsilon_S[f] = |I_d(f) - I_S(f)|.$$

When S is drawn randomly, the Central Limit Theorem asserts that if $s = |S|$ is large enough then $\epsilon_S[f] \approx \sigma[f] s^{-1/2} \nu$, where ν is a standard normal random variable and $\sigma[f]$ is the square-root of the variance of f; that is,

$$\sigma^2[f] = \int_{[0,1]^d} (f(x) - I_d(f))^2 \, dx.$$

In other words, the root mean square error of the Monte Carlo method is

$$\left(\mathbb{E}_S \left[\epsilon_S[f]^2 \right] \right)^{1/2} \approx \sigma[f] s^{-1/2}. \tag{8.5}$$

Therefore, the Monte Carlo method converges at a rate of $O(s^{-1/2})$.

The aim of QMC methods is to improve the convergence rate by using a deterministic *low-discrepancy sequence* to construct S, instead of randomly sampling points. The underlying intuition is illustrated in Figure 34, where we plot a set of 1000 two-dimensional random points (left graph), and a set of 1000 two-dimensional points from a quasi-random sequence (Halton sequence; right graph). In the random sequence we see that there is an undesired clustering of points, and as a consequence empty spaces. Clusters add little to the approximation of the integral in those regions, while in the empty spaces the integrand is not sampled. This lack of uniformity is due to the fact that Monte Carlo samples are independent of each other. By carefully designing a sequence of correlated points to avoid such clustering effects, QMC attempts to avoid this phenomenon, and thus provide faster convergence to the integral.

The theoretical apparatus for designing such sequences are inequalities of the form

$$\epsilon_S(f) \leq D(S)V(f),$$

in which $V(f)$ is a measure of the variation or difficulty of integrating $f(\cdot)$ and $D(S)$ is a sequence-dependent term that typically measures the *discrepancy*, or degree of deviation from uniformity, of the sequence S. For example, the expected Monte Carlo integration error decouples into a variance term, and $s^{-1/2}$ as in (8.5).

A prototypical inequality of this sort is the following remarkable and classical result.

Theorem 8.3 (Koksma-Hlawka inequality)**.** *For any function f with bounded variation, and sequence $S = \{w^1, \ldots, w^s\}$, the integration error is bounded above as follows,*

$$\epsilon_S[f] \leq D^\star(S)V_{HK}[f],$$

where V_{HK} is the Hardy-Krause variation of f *(see Niederreiter [Nie92]), which is defined in terms of the partial derivatives*

$$V_{HK}[f] = \sum_{I \subset [d], I \neq \emptyset} \int_{[0,1]^{|I|}} \left| \frac{\partial f}{\partial u_I} \right|_{u_j = 1, j \notin I} \left| du_I, \right. \tag{8.6}$$

and $D^\star$ is the star discrepancy *defined by*

$$D^\star(S) = \sup_{x \in [0,1]^d} |\mathrm{disr}_S(x)|,$$

where disr_S *is the* local discrepancy function

$$\mathrm{disr}_S(x) = \mathrm{Vol}(J_x) - \frac{|\{i \,:\, w^i \in J_x\}|}{s}$$

and $J_x = [0, x_1) \times [0, x_2) \times \cdots \times [0, x_d)$ with $\mathrm{Vol}(J_x) = \prod_{j=1}^d x_j$.

Given x, the second term in $\mathrm{disr}_S(x)$ is an estimate of the volume of J_x, which will be accurate if the points in S are uniform enough. $D^\star(S)$ measures the maximum difference between the actual volume of the subregion J_x and its estimate for all x in $[0,1]^d$.

An infinite sequence $w^1, w^2, \ldots$ is defined to be a *low-discrepancy sequence* if, as a function of s, $D^\star(\{w^1, \ldots, w^s\}) = O((\log s)^d/s)$. Several constructions are known to be low-discrepancy sequences. One notable example is the *Halton sequences*, which are defined as follows. Let $p_1, \ldots, p_d$ be the first d prime numbers. The Halton sequence $w^1, w^2, \ldots$ of dimension d is defined by

$$w^i = (\phi_{p_1}(i), \ldots, \phi_{p_d}(i)),$$

where for integers $i \geq 0$ and $b \geq 2$ we have

$$\phi_b(i) = \sum_{a=1}^{\infty} i_a b^{-a},$$

in which $i_0, i_1, \cdots \in \{0, 1, \ldots, b-1\}$ is given by the unique decomposition

$$i = \sum_{a=1}^{\infty} i_a b^{a-1}.$$

It is outside the scope to describe all these constructions in detail. However, we mention that in addition to the Halton sequences, other notable members are *Sobol' sequences*, *Faure sequences*, *Niederreiter sequences*, and more (see Dick, Kuo, and Sloan [DKS13], Section 2). We also mention that it is conjectured that the $O((\log s)^d/s)$ rate for star discrepancy decay is optimal.

The classical QMC theory, which is based on the Koksma-Hlawka inequality and low discrepancy sequences, thus achieves a convergence rate of $O((\log s)^d/s)$. While this is asymptotically superior to $O(s^{-1/2})$ for a fixed d, it requires s to be exponential in d for the improvement to manifest. As such, in the past QMC methods were dismissed as unsuitable for very high-dimensional integration.

However, several authors noticed that QMC methods perform better than MC even for very high-dimensional integration [DKS13].[2] Contemporary QMC literature explains and expands on these empirical observations, by leveraging the structure of the space in which the integrand function lives, to derive more refined bounds and discrepancy measures, even when classical measures of variation such as (8.6) are unbounded. This literature has evolved along at least two directions: one, where worst-case analysis is provided under the assumption that the integrands live in a Reproducing Kernel Hilbert Space (RKHS) of sufficiently smooth and well-behaved functions (see Dick, Kuo, and Sloan [DKS13], Section 3) and second, where the analysis is done in terms of *average-case* error, under an assumed probability distribution over the integrands, instead of worst-case error [Woz91; TW94]. We refrain from more details, as these are essentially the paths that the analysis in Section 8.3.2 follows for our specific setting.

8.3 MAIN ALGORITHM AND MAIN THEORETICAL RESULTS

8.3.1 *QMC Feature Maps: Our Algorithm*

We assume that the density function in (8.2) can be written as $p(x) = \prod_{j=1}^{d} p_j(x_j)$, where $p_j(\cdot)$ is a univariate density function. The density functions associated with many shift-invariant kernels, e.g., Gaussian, Laplacian and Cauchy, admit such a form.

The QMC method is generally applicable to integrals over a unit cube. Thus, integrals of the form (8.2) are typically handled by generating a low discrepancy sequence $t^1, \ldots, t^s \in [0,1]^d$ and transforming it into a sequence $w^1, \ldots, w^s$ in $\mathbb{R}^d$, instead of drawing the elements of the sequence from $p(\cdot)$ as in the MC method.

To convert (8.2) to an integral over the unit cube, a simple change of variables suffices. For $t \in \mathbb{R}^d$, define

$$\Phi^{-1}(t) = \left(\Phi_1^{-1}(t_1), \ldots, \Phi_d^{-1}(t_d)\right) \in \mathbb{R}^d, \tag{8.7}$$

where $\Phi_j(\cdot)$ is the cumulative distribution function (CDF) of $p_j(\cdot)$, for $j = 1, \ldots, d$. By setting $w = \Phi^{-1}(t)$, then we can write (8.2) as

$$\int_{\mathbb{R}^d} e^{-i(x-z)^T w} p(w) dw = \int_{[0,1]^d} e^{-i(x-z)^T \Phi^{-1}(t)} dt.$$

2 Also see: "On the unreasonable effectiveness of QMC", I. H. Sloan https://mcqmc.mimuw.edu.pl/Presentations/sloan.pdf

Algorithm 12 Quasi-Random Fourier features

1: **Input:** Shift-invariant kernel k, size s.
2: **Output:** Feature map $\hat{\Psi}(x) : \mathbb{R}^d \mapsto \mathbb{C}^s$.
3: Find p, the inverse Fourier transform of k.
4: Generate a low discrepancy sequence $t^1, \ldots, t^s$.
5: Transform the sequence: $w^i = \Phi^{-1}(t^i)$ by (8.7).
6: Set $\hat{\Psi}(x) = \sqrt{\frac{1}{s}} \left[e^{-ix^T w^1}, \ldots, e^{-ix^T w^s} \right]$.
7: **Return** $\hat{\Psi}(x)$.

Thus, a low discrepancy sequence $t^1, \ldots, t^s \in [0,1]^d$ can be transformed using $w^i = \Phi^{-1}(t^i)$, which is inserted into (8.3) to yield the QMC feature map. This simple procedure is summarized in Algorithm 12. QMC feature maps are analyzed in the next section.

8.3.2 *Theoretical Analysis and Average Case Error Bounds*

The proofs for assertions made in this section and the next can be found in the Appendix D. Before diving into the details, we give the notation. We use i both for subscript and for denoting $\sqrt{-1}$, relying on the context to distinguish between the two. Given $x^1, \ldots, x^n$, the Gram matrix is defined as $K \in \mathbb{R}^{n \times n}$ where $K_{ij} = k(x^i, x^j)$ for $i, j = 1, \ldots, n$. We denote the error function by $\mathrm{erf}(\cdot)$, i.e., $\mathrm{erf}(z) = \int_0^z e^{-z^2} dz$ for $z \in \mathbb{C}$; see Weideman [Wei94] for more details.

In "*MC sequence*" we mean points drawn randomly either from the unit cube or certain distribution that will be clear from the text. For "*QMC sequence*" we mean a deterministic sequence designed to reduce the integration error. Typically, it will be a low-discrepancy sequence on the unit cube.

It is also useful to recall the definition of Reproducing Kernel Hilbert Space (RKHS).

Definition 8.4 (Reproducing kernel Hilbert space [BTA04])**.** *A* reproducing kernel Hilbert space (RKHS) *is a Hilbert Space $\mathcal{H} : \mathcal{X} \to \mathbb{C}$ that possesses a reproducing kernel, i.e., a function $h : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$ for which the following hold for all $x \in \mathcal{X}$ and $f \in \mathcal{H}$:*

- $h(x, \cdot) \in \mathcal{H}$
- $\langle f, h(x, \cdot) \rangle_{\mathcal{H}} = f(x)$ *(*Reproducing Property*)*

Equivalently, RKHSs are Hilbert spaces with bounded, continuous evaluation functionals. Informally, they are Hilbert spaces with the nice property that if two functions $f, g \in \mathcal{H}$ are close in the sense of the distance derived from the norm in $\mathcal{H}$ (i.e., $\|f - g\|_{\mathcal{H}}$ is small), then their values $f(x), g(x)$ are also close for all $x \in \mathcal{X}$; in other words, the norm controls the pointwise behavior of functions in $\mathcal{H}$ [BTA04].

The goal of this section is to develop a framework for analyzing the approximation quality of the QMC feature maps described in the previous section (Algorithm 12). We need to develop such a framework since the classical Koksma-Hlawka inequality cannot be applied to our setting, as the following proposition shows:

Proposition 8.5. *For any $p(x) = \prod_{j=1}^d p_j(x_j)$, where $p_j(\cdot)$ is a univariate density function, let*

$$\Phi^{-1}(t) = \left(\Phi_1^{-1}(t_1), \ldots, \Phi_d^{-1}(t_d) \right).$$

For a fixed $u \in \mathbb{R}^d$, consider $f_u(t) = e^{-iu^T \Phi^{-1}(t)}$, $t \in [0,1]^d$. The Hardy-Krause variation of $f_u(\cdot)$ is unbounded. That is, one of the integrals in the sum (8.6) is unbounded.

Our framework is based on a new discrepancy measure, *box discrepancy*, that characterizes integration error for the set of integrals defined with respect to the underlying data domain. Throughout this section we use the convention that $S = \{w^1, \ldots, w^s\}$, and the notation $\bar{\mathcal{X}} = \{x - z \mid x, z \in \mathcal{X}\}$.

Given a probability density function $p(\cdot)$ and S, we define the integration error $\epsilon_{S,p}[f]$ of a function $f(\cdot)$ with respect to $p(\cdot)$ and the s samples as,

$$\epsilon_{S,p}[f] = \left| \int_{\mathbb{R}^d} f(x)p(x)dx - \frac{1}{s}\sum_{i=1}^{s} f(w^i) \right|.$$

We are interested in characterizing the behavior of $\epsilon_{S,p}[f]$ on $f \in \mathcal{F}_{\bar{\mathcal{X}}}$ where

$$\mathcal{F}_{\bar{\mathcal{X}}} = \left\{ f_u(x) = e^{-iu^T x},\ u \in \bar{\mathcal{X}} \right\}.$$

As is common in modern QMC analysis [LP14; DKS13], our analysis is based on setting up a Reproducing Kernel Hilbert Space of "nice" functions that is related to integrands that we are interested in, and using properties of the RKHS to derive bounds on the integration error. In particular, the integration error of integrands in an RKHS can be bounded using the following proposition.

Proposition 8.6 (Integration error in an RKHS)**.** *Let $\mathcal{H}$ be an RKHS with kernel $h(\cdot,\cdot)$. Assume that $\kappa = \sup_{x \in \mathbb{R}^d} h(x,x) < \infty$. Then, for all $f \in \mathcal{H}$ we have,*

$$\epsilon_{S,p}[f] \leq \|f\|_{\mathcal{H}} D_{h,p}(S), \tag{8.8}$$

where

$$\begin{aligned} D_{h,p}(S)^2 &= \left\| \int_{\mathbb{R}^d} h(\omega,\cdot)p(\omega)d\omega - \frac{1}{s}\sum_{l=1}^{s} h(w^l,\cdot) \right\|_{\mathcal{H}}^2 \\ &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi - \frac{2}{s}\sum_{l=1}^{s}\int_{\mathbb{R}^d} h(w^l,\omega)p(\omega)d\omega \\ &\quad + \frac{1}{s^2}\sum_{l=1}^{s}\sum_{j=1}^{s} h(w^l,w^j). \end{aligned} \tag{8.9}$$

Remark 10. *In the theory of RKHS embeddings of probability distributions [Sri+10], the function*

$$\mu_{h,p}(x) = \int_{\mathbb{R}^d} h(\omega,x)p(\omega)d\omega$$

is known as the kernel mean *embedding of $p(\cdot)$. The function*

$$\hat{\mu}_{h,p,S}(x) = \frac{1}{s}\sum_{l=1}^{s} h(w^l,x)$$

is then the empirical mean map.

The RKHS we use is as follows. For a vector $b \in \mathbb{R}^d$, let us define $\square b = \{u \in \mathbb{R}^d \mid |u_j| \leq b_j\}$. Let

$$\mathcal{F}_{\square b} = \left\{ f_u(x) = e^{-iu^T x},\ u \in \square b \right\},$$

and consider the space of functions that admit an integral representation over $\mathcal{F}_{\square b}$ of the form

$$f(x) = \int_{u \in \square b} \hat{f}(u) e^{-iu^T x} du \text{ where } \hat{f}(u) \in L_2(\square b). \tag{8.10}$$

This space is associated with bandlimited functions, i.e., functions with compactly-supported inverse Fourier transforms, which are of fundamental importance in the Shannon-Nyquist sampling theory. Under a natural choice of inner product, these spaces are called *Paley-Wiener spaces* and they constitute an RKHS [BTA04; Yao67].

Proposition 8.7 (Kernel of Paley-Wiener RKHS)**.** *By PW_b, denote the space of functions which admit the representation in (8.10), with the inner product $\langle f, g \rangle_{PW_b} = (2\pi)^{2d} \langle \hat{f}, \hat{g} \rangle_{L_2(\square b)}$. PW_b is an RKHS with kernel function,*

$$\operatorname{sinc}_b(u, v) = \pi^{-d} \prod_{j=1}^{d} \frac{\sin\left(b_j(u_j - v_j)\right)}{u_j - v_j}.$$

For notational convenience, in the above we define $\sin(b \cdot 0)/0$ to be b. Furthermore, $\langle f, g \rangle_{PW_b} = \langle f, g \rangle_{L_2(\square b)}$.

If $b_j = \sup_{u \in \tilde{\mathcal{X}}} |u_j|$ then $\tilde{\mathcal{X}} \subset \square b$, so $F_{\tilde{\mathcal{X}}} \subset F_{\square b}$. Since we wish to bound the integration error on functions in $F_{\tilde{\mathcal{X}}}$, it suffices to bound the integration error on $\mathcal{F}_{\square b}$. Unfortunately, while $\mathcal{F}_{\square b}$ defines PW_b, the functions in it, being not square integrable, are *not* members of PW_b, so analyzing the integration error in PW_b do not directly apply to them. However, damped approximations of $f_u(\cdot)$ of the form $\tilde{f}_u(x) = e^{-iu^T x} \operatorname{sinc}(Tx)$ are members of PW_b with $\|\tilde{f}\|_{PW_b} = \frac{1}{\sqrt{T}}$. Hence, we expect the analysis of the integration error in PW_b to provide provide a discrepancy measure for integrating functions in $\mathcal{F}_{\square b}$.

For PW_b the discrepancy measure $D_{h,S}$ in Proposition 8.6 can be written explicitly.

Theorem 8.8 (Discrepancy in PW_b)**.** *Suppose that $p(\cdot)$ is a probability density function, and that we can write $p(x) = \prod_{j=1}^{d} p_j(x_j)$ where each $p_j(\cdot)$ is a univariate probability density function as well. Let $\varphi_j(\cdot)$ be the characteristic function associated with $p_j(\cdot)$. Then,*

$$\begin{aligned} D_{\operatorname{sinc}_b, p}(S)^2 \quad = \quad & \pi^{-d} \prod_{j=1}^{d} \int_{-b_j}^{b_j} |\varphi_j(\beta)|^2 d\beta - \\ & \frac{2(2\pi)^{-d}}{s} \sum_{l=1}^{s} \prod_{j=1}^{d} \int_{-b_j}^{b_j} \varphi_j(\beta) e^{i w_j^l \beta} d\beta + \\ & \frac{1}{s^2} \sum_{l=1}^{s} \sum_{j=1}^{s} \operatorname{sinc}_b(w^l, w^j). \end{aligned} \tag{8.11}$$

This naturally leads to the definition of the *box discrepancy*, analogous to the star discrepancy described in Theorem 8.3.

Definition 8.9 (Box discrepancy)**.** *The box discrepancy of a sequence S with respect to $p(\cdot)$ is defined as,*

$$D_p^{\square b}(S) = D_{\operatorname{sinc}_b, p}(S).$$

For notational convenience, we generally omit the b from $D_p^{\square b}(S)$ as long as it is clear from the context.

The worse-case integration error bound for Paley-Wiener spaces is stated in the following as a corollary of Proposition 8.6. As explained earlier, this result not yet apply to functions in $\mathcal{F}_{\square b}$ because these functions are not part of PW_b. Nevertheless, we state it here for completeness.

Corollary 8.10 (Integration error in PW_b)**.** *For $f \in PW_b$ we have*

$$\epsilon_{S,p}[f] \leq \|f\|_{PW_b} D_p^{\square}(S).$$

Our main result shows that the expected square error of an integrand drawn from a uniform distribution over $\mathcal{F}_{\square b}$ is proportional to the square discrepancy measure $D_p^{\square}(S)$. This result is in the spirit of similar average case analysis in the QMC literature [Woz91; TW94].

Theorem 8.11 (Average case error)**.** *Let $\mathcal{U}(\mathcal{F}_{\square b})$ denote the uniform distribution on $\mathcal{F}_{\square b}$. That is, $f \sim \mathcal{U}(\mathcal{F}_{\square b})$ denotes $f = f_u$ where $f_u(x) = e^{-iu^T x}$ and u is randomly drawn from a uniform distribution on $\square b$. We have,*

$$\mathbb{E}_{f \sim \mathcal{U}(\mathcal{F}_{\square b})} \left[\epsilon_{S,p}[f]^2 \right] = \frac{\pi^d}{\prod_{j=1}^d b_j} D_p^{\square}(S)^2.$$

We now give an explicit formula for $D_p^{\square}(S)$ for the case that $p(\cdot)$ is the density function of the multivariate Gaussian distribution with zero mean and independent components. This is an important special case since this is the density function that is relevant for the Gaussian kernel.

Corollary 8.12 (Discrepancy for Gaussian distribution)**.** *Let $p(\cdot)$ be the d-dimensional multivariate Gaussian density function with zero mean and covariance matrix equal to* $\mathrm{diag}(\sigma_1^{-2}, \ldots, \sigma_d^{-2})$. *We have,*

$$\begin{aligned} D_p^{\square}(S)^2 \quad = \quad & \frac{1}{s^2} \sum_{l=1}^{s} \sum_{j=1}^{s} \mathrm{sinc}_b(w^l, w^j) - \\ & \frac{2}{s} \sum_{l=1}^{s} \prod_{j=1}^{d} c_{lj} \, \mathrm{Re} \left(\mathrm{erf} \left(\frac{b_j}{\sigma_j \sqrt{2}} - i \frac{\sigma_j w_j^l}{\sqrt{2}} \right) \right) + \\ & \prod_{j=1}^{d} \frac{\sigma_j}{2\sqrt{\pi}} \, \mathrm{erf} \left(\frac{b_j}{\sigma_j} \right), \end{aligned} \tag{8.12}$$

where

$$c_{lj} = \left(\frac{\sigma_j}{\sqrt{2\pi}} \right) e^{-\frac{\sigma_j^2 (w_j^l)^2}{2}}.$$

Intuitively, the box discrepancy of the Gaussian kernel can be interpreted as follows. The function $\mathrm{sinc}(x) = \sin(x)/x$ achieves its maximum at $x = 0$ and minimizes at discrete values of x decaying to 0 as $|x|$ goes to ∞. Hence the first term in (8.12) tends to be minimized when the pairwise distance between w^j are sufficiently separated. Due to the shape of cumulative distribution function of Gaussian distribution, the values of $t^j = \Phi(w^j)$ $(j = 1, \ldots, s)$ are driven to be close to the boundary of the unit cube. As for second

term, the original expression is $-\frac{2}{s}\sum_{l=1}^{s}\int_{\mathbb{R}^d} h(w^l,\omega)p(\omega)d\omega$. This term encourages the sequence $\{w^l\}$ to mimic samples from $p(\omega)$. Since $p(\omega)$ concentrates its mass around $\omega = 0$, the w^j also concentrates around 0 to maximize the integral and therefore the values of $t^j = \Phi(w^j)$ $(j = 1,\dots,s)$ are driven closer to the center of the unit cube. Sequences with low box discrepancy therefore optimize a tradeoff between these competing terms.

Two other shift-invariant kernel that have been mentioned in the machine learning literature is the Laplacian kernel [RR08] and Matern kernel [LSS13]. The distribution associated with the Laplacian kernel can be written as a product $p(x) = \prod_{j=1}^{d} p_j(x_j)$, where $p_j(\cdot)$ is density associated with the Cauchy distribution. The characteristic function is simple $(\phi_j(\beta) = e^{-|\beta|/\sigma_j})$ so analytic formulas like (8.12) can be derived. The distribution associated with the Matern kernel, on the other hand, is the multivariate t-distribution, which cannot be written as a product $p(x) = \prod_{j=1}^{d} p_j(x_j)$, so the presented theory does not apply to it.

DISCREPANCY OF MONTE-CARLO SEQUENCES.

We now derive an expression for the expected discrepancy of Monte-Carlo sequences, and show that it decays as $O(s^{-1/2})$. This is useful since via an averaging argument we are guaranteed that there exists sets for which the discrepancy behaves $O(s^{-1/2})$.

Corollary 8.13. *Suppose $t^1,\dots,t^s$ are chosen uniformly from $[0,1]^d$. Let $w^i = \Phi^{-1}(t^i)$, for $i = 1,\dots,s$. Assume that $\kappa = \sup_{x\in\mathbb{R}^d} h(x,x) < \infty$. Then*

$$\mathbb{E}\left[D_{h,p}(S)^2\right] = \frac{1}{s}\int_{\mathbb{R}^d} h(\omega,\omega)p(\omega)d\omega - \frac{1}{s}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi.$$

Again, we can derive specific formulas for the Gaussian density. The following is straightforward from Corollary 8.13. We omit the proof.

Corollary 8.14. *Let $p(\cdot)$ be the d-dimensional multivariate Gaussian density function with zero mean and covariance matrix equal to $\mathrm{diag}(\sigma_1^{-2},\dots,\sigma_d^{-2})$. Suppose $t^1,\dots,t^s$ are chosen uniformly from $[0,1]^d$. Let $w^i = \Phi^{-1}(t^i)$, for $i = 1,\dots,s$. Then,*

$$\mathbb{E}\left[D_p^{\square}(S)^2\right] = \frac{1}{s}\left(\pi^{-d}\prod_{j=1}^{d} b_j - \prod_{j=1}^{d}\frac{\sigma_j}{2\sqrt{\pi}}\,\mathrm{erf}\left(\frac{b_j}{\sigma_j}\right)\right). \tag{8.13}$$

8.4 LEARNING ADAPTIVE QMC SEQUENCES

For simplicity, in this section we assume that $p(\cdot)$ is the density function of Gaussian distribution with zero mean. We also omit the subscript p from $D_p^{\square}$. Similar analysis and equations can be derived for other density functions.

Error characterization via discrepancy measures like (8.12) is typically used in the QMC literature to prescribe sequences whose discrepancy behaves favorably. It is clear that for the box discrepancy, a meticulous design is needed for a high quality sequence and we leave this to future work. Instead, in this work, we use the fact that unlike the star discrepancy (8.3), the box discrepancy is a smooth function of the sequence with a closed-form formula. This allows us to both evaluate various candidate sequences, and select the one with the lowest discrepancy, as well as to *adaptively learn* a QMC sequence that is specialized for our problem setting via numerical optimization. The basis is the following proposition, which gives an expression for the gradient of $D^{\square}(S)$.

Proposition 8.15 (Gradient of box discrepancy). *Define the following scalar functions and variables,*

$$\begin{aligned}
\operatorname{sinc}'(z) &= \frac{\cos(z)}{z} - \frac{\sin(z)}{z^2}, \;\; \operatorname{sinc}'_b(z) = \frac{b}{\pi}\operatorname{sinc}'(bz)\,; \\
c_j &= \left(\frac{\sigma_j}{\sqrt{2\pi}}\right), j = 1, \ldots, d\,; \\
g_j(x) &= c_j e^{-\frac{\sigma_j^2}{2}x^2} \operatorname{Re}\left(\operatorname{erf}\left[\frac{b_j}{\sigma_j\sqrt{2}} - i\frac{\sigma_j x}{\sqrt{2}}\right]\right)\,; \\
g'_j(x) &= -\sigma_j^2 x g_j(x) + \sqrt{\frac{2}{\pi}} c_j \sigma_j e^{-\frac{b_j^2}{2\sigma_j^2}} \sin(b_j x).
\end{aligned}$$

In the above, we define $\operatorname{sinc}'(0)$ *to be* 0*. Then, the elements of the gradient vector of* $D^{\square}$ *are given by,*

$$\begin{aligned}
\frac{\partial D^{\square}}{\partial w_j^l} = \frac{2}{s^2}\sum_{\substack{m=1\\m\neq l}}^{s}\left(b_j \operatorname{sinc}'_{b_j}(w_j^l, w_j^m)\prod_{q\neq j}\operatorname{sinc}_{b_q}(w_q^l, w_q^m)\right) - \\
\frac{2}{s}g'_j(w_l^j)\left(\prod_{q\neq j} g_q(w_q^l)\right).
\end{aligned} \tag{8.14}$$

We explore two possible approaches for finding sequences based on optimizing the box discrepancy, namely *global optimization* and *greedy optimization*. The latter is closely connected to *herding* algorithms [Welo9]. In addition, we also consider *weighted sequences*.

8.4.1 *Global Adaptive Sequences*

The task is posed in terms minimization of the box discrepancy function (8.12) over the space of sequences of s vectors in $\mathbb{R}^d$:

$$S^* = \operatorname{argmin}_{S=(w^1\ldots w^s)\in\mathbb{R}^{ds}} D^{\square}(S).$$

The gradient can be plugged into any first order numerical solver for non-convex optimization. We use nonlinear conjugate gradient in our experiments (Section 8.5.2).

The above learning mechanism can be extended in various directions. For example, QMC sequences for n-point rank-one Lattice Rules [DKS13] are integral fractions of a lattice defined by a single generating vector v. This generating vector may be learnt via local minimization of the box discrepancy.

8.4.2 *Greedy Adaptive Sequences*

Starting with $S_0 = \emptyset$, for $t \geq 1$, let $S_t = \{w^1, \ldots, w^t\}$. At step $t+1$, we solve the following optimization problem,

$$w^{t+1} = \operatorname{argmin}_{w\in\mathbb{R}^d} D^{\square}(S_t \cup \{w\}). \tag{8.15}$$

Set $S_{t+1} = S_t \cup \{w^{t+1}\}$ and repeat the above procedure. The gradient of the above objective is also given in (8.14). Again, we use nonlinear conjugate gradient in our experiments (Section 8.5.2).

The greedy adaptive procedure is closely related to the herding algorithm, recently presented by Welling [Welo9]. Applying the herding algorithm to PW_b and $p(\cdot)$, and using our notation, the points $w^1, w^2, \ldots$ are generated using the following iteration

$$\begin{aligned} w^{t+1} &\in \arg\max_{w \in \mathbb{R}^d} \langle z_t(\cdot), h(w,\cdot)\rangle_{PW_b} \\ z^{t+1}(x) &\equiv z^t(x) + \mu_{h,p}(x) - h(w,x). \end{aligned}$$

In the above, $z^0, z^1, \ldots$ is a series of functions in PW_b. The literature is not specific on the initial value of z^0, with both $z^0 = 0$ and $z^0 = \mu_{h,p}$ suggested. Either way, it is always the case that $z^t = z^0 + t(\mu_{h,p} - \hat{\mu}_{h,p,S_t})$ where $S_t = \{w^1, \ldots, w^t\}$.

Chen, Welling, and Smola [CWS10] showed that under some additional assumptions, the herding algorithm, when applied to a RKHS $\mathcal{H}$, greedily minimizes $\left\|\mu_{h,p} - \hat{\mu}_{h,p,S_t}\right\|_{\mathcal{H}}^2$, which, recall, is equal to $D_{h,p}(S_t)$. Thus, under certain assumptions, herding and (8.15) are equivalent. Chen, Welling, and Smola [CWS10] also showed that under certain restrictions on the RKHS, herding will reduce the discrepancy in a ratio of $O(1/t)$. However, it is unclear whether those restrictions hold for PW_b and $p(\cdot)$. Indeed, Bach, Lacoste-Julien, and Obozinski [BLO12] recently shown that these restrictions never hold for infinite-dimensional RKHS, as long as the domain is compact. This result does not immediately apply to our case since $\mathbb{R}^d$ is not compact.

8.4.3 *Weighted Sequences*

Classically, Monte-Carlo and Quasi-Monte Carlo approximations of integrals are unweighted, or more precisely, have a uniform weights. However, it is quite plausible to weight the approximations, i.e. approximate $I_d[f] = \int_{[0,1]^d} f(x)dx$ using

$$I_{S,\Xi}[f] = \sum_{i=1}^{n} \xi_i f(w^i), \tag{8.16}$$

where $\Xi = \{\xi_1, \ldots, \xi_s\} \subset \mathbb{R}$ is a set of weights. This lead to the feature map

$$\hat{\Psi}_S(x) = \left[\sqrt{\xi_1} e^{-ix^T w^1} \ldots \sqrt{\xi_s} e^{-ix^T w^s}\right].$$

This construction requires $\xi_i \geq 0$ for $i = 1, \ldots, s$, although we note that (8.16) itself does not preclude negative weights. We do not require the weights to be normalized, that is it is possible that $\sum_{i=1}^{s} \xi_i \neq 1$.

One can easily generalize the result of the previous section to derive the following discrepancy measure that takes into consideration the weights

$$\begin{aligned} D_p^{\square b}(S,\Xi)^2 &= \pi^{-d}\prod_{j=1}^{d}\int_{-b_j}^{b_j}|\varphi_j(\beta)|^2 d\beta - \\ &\quad 2(2\pi)^{-d}\sum_{l=1}^{s}\xi_l\prod_{j=1}^{d}\int_{-b_j}^{b_j}\varphi_j(\beta)e^{iw_j^l\beta}d\beta + \\ &\quad \sum_{l=1}^{s}\sum_{j=1}^{s}\xi_l\xi_j\,\mathrm{sinc}_b(w^l,w^j). \end{aligned}$$

Using this discrepancy measure, global adaptive and greedy adaptive sequences of points and weights can be found.

However, we note that if we fix the points, then optimizing just the weights is a simple convex optimization problem. The box discrepancy can be written as

$$D_p^{\square b}(S,\Xi)^2 = \pi^{-d}\prod_{j=1}^{d}\int_{-b_j}^{b_j}|\varphi_j(\beta)|^2 d\beta - 2v^T\xi + \xi^T H\xi\,,$$

where $\xi \in \mathbb{R}^s$ has entry i equal to ξ_i, $v \in \mathbb{R}^s$ and $H \in \mathbb{R}^{s\times s}$ are defined by

$$H_{ij} = \mathrm{sinc}_b(w_l, w_j)$$

$$v_i = (2\pi)^{-d}\prod_{j=1}^{d}\int_{-b_j}^{b_j}\varphi_j(\beta)e^{iw_j^l\beta}d\beta\,.$$

The ξ that minimizes $D_p^{\square b}(S,\Xi)^2$ is equal to $H^{-1}v$, but there is no guarantee that $\xi_i \geq 0$ for all i. We need to explicitly impose these conditions. Thus, the optimal weights can be found by solving the following convex optimization problem

$$\Xi^* = \mathrm{argmin}_{\xi\in\mathbb{R}^s}\,\xi^T H\xi - 2v^T\xi \quad \text{s.t. } \xi \geq 0. \tag{8.17}$$

Selecting the weights in such a way is closely connected to the so-called *Bayesian Monte Carlo* (BMC) method, originally suggested by Ghahramani and Rasmussen [GR03]. In BMC, a Bayesian approach is utilized in which the function is a assumed to be random with a prior that is a Gaussian Process. Combining with the observations, a posterior is obtained, which naturally leads to the selection of weights. Huszár and Duvenaud [HD12] subsequently pointed out the connection between this approach and the herding algorithm discussed earlier.

We remark that as long as all the weights are positive, the hypothesis space of functions induced by the feature map (that is, $\{g_w(x) = \hat{\Psi}_S^T(x)w,\ w \in \mathbb{R}^s\}$) will not change in terms of the set of functions in it. However, the norms will be affected (that is, the norm of a function in that set also depends on the weights), which in turn affects the regularization.

8.5 EMPIRICAL EVALUATION

In this section we report experiments with both classical QMC sequences and adaptive sequences learnt from box discrepancy minimization.

8.5.1 *Experiments With Classical QMC Sequences*

We examine the behavior of classical low-discrepancy sequences when compared to random Fourier features (i.e., MC). We consider four sequences: Halton, Sobol', Lattice Rules, and Digital Nets. For Halton and Sobol', we use the implementation available in MATLAB.[3] For Lattice Rules and Digital Nets, we use publicly available implementations.[4] For all four low-discrepancy sequences, we use scrambling and shifting techniques recommended in the QMC literature (see Dick, Kuo, and Sloan [DKS13] for details). For Sobol', Lattice Rules and Digital Nets, scrambling introduces randomization and hence variance. For Halton sequence, scrambling is deterministic, and there is no variance. The generation of these sequences is extremely fast, and quite negligible when compared to the time for any reasonable downstream use. For example, for `census` data set with size 18,000 by 119, if we choose the number of random features $s = 2000$, the running time for performing kernel ridge regression model is more than 2 minutes, while the time of generating the QMC sequences is only around 0.2 seconds (Digital Nets sequence takes longer, but not much longer) and that of MC sequence is around 0.01 seconds. Therefore, we do not report running times as these are essentially the same across methods.

In all experiments, we work with a Gaussian kernel. For learning, we use regularized least square classification on the feature mapped data set, which can be thought of as a form of approximate kernel ridge regression. For each data set, we performed 5-fold cross-validation when using random Fourier features (MC sequence) to set the bandwidth σ, and then used the same σ for all other sequences.

8.5.1.1 *Quality of Kernel Approximation*

In our setting, the most natural and fundamental metric for comparison is the quality of approximation of the Gram matrix. We examine how close $\tilde{K}$ (defined by $\tilde{K}_{ij} = \tilde{k}(x^i, x^j)$ where $\tilde{k}(\cdot,\cdot) = \langle \hat{\Psi}_S(\cdot), \hat{\Psi}_S(\cdot) \rangle$ is the kernel approximation) is to the Gram matrix K of the exact kernel.

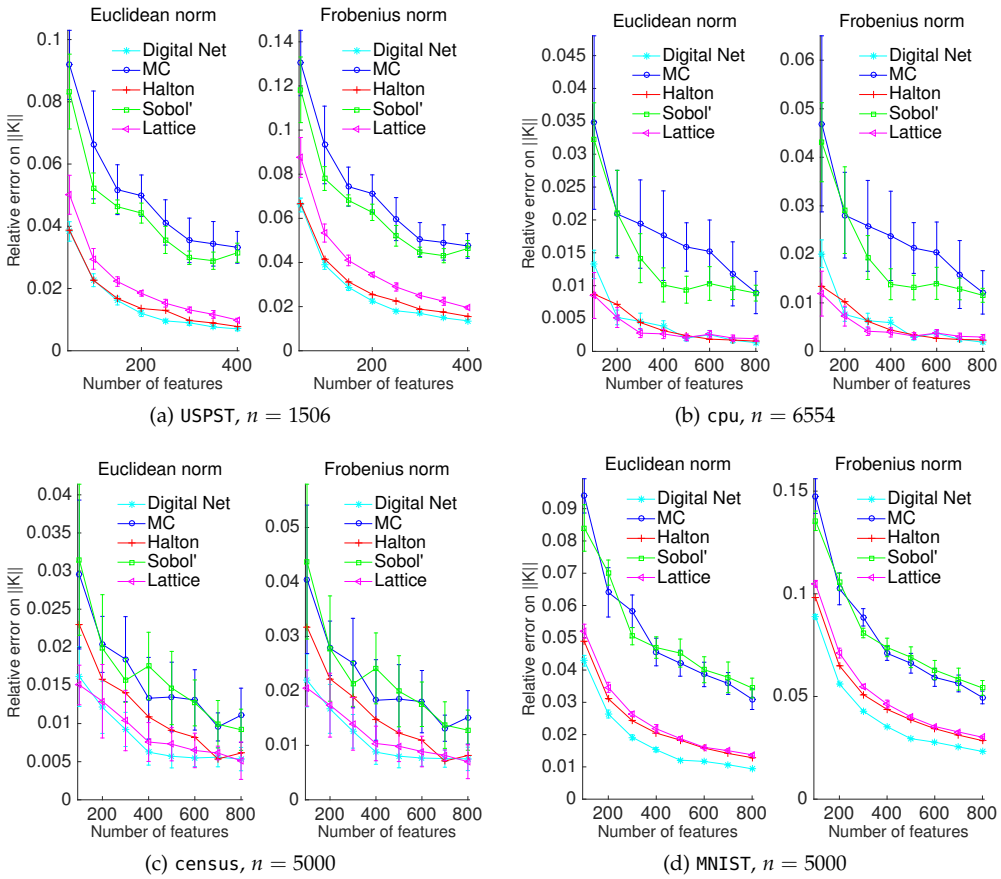
We examine four data sets: `cpu` (6554 examples, 21 dimensions), `census` (a subset chosen randomly with 5,000 examples, 119 dimensions), `USPST` (1,506 examples, 250 dimensions after PCA) and `MNIST` (a subset chosen randomly with 5,000 examples, 250 dimensions after PCA). The reason we do subsampling on large data sets is to be able to compute the full exact Gram matrix for comparison purposes. The reason we use dimensionality reduction on `MNIST` is that the maximum dimension supported by the Lattice Rules implementation we use is 250.

To measure the quality of approximation we use both $\|K - \tilde{K}\|_2 / \|K\|_2$ and $\|K - \tilde{K}\|_F / \|K\|_F$. The plots are shown in Figure 35.

We can clearly see that except Sobol' sequences classical low-discrepancy sequences consistently produce better approximations to the Gram matrix than the approximations produced using MC sequences. Among the four classical QMC sequences, the Digital Nets, Lattice Rules and Halton sequences yield much lower error. Similar results were observed for other data sets (not reported here). Although using scrambled variants of QMC sequences may incur some variance, the variance is quite small compared to that of the MC random features.

Scrambled (whether deterministic or randomized) QMC sequences tend to yield higher accuracies than non-scambled QMC sequences. In Figure 36, we show the ratio between the relative errors achieved by using both scrambled and non-scrambled QMC sequences.

3 http://www.mathworks.com/help/stats/quasi-random-numbers.html
4 http://people.cs.kuleuven.be/ dirk.nuyens/qmc-generators/



(a) USPST, $n = 1506$ (b) cpu, $n = 6554$

(c) census, $n = 5000$ (d) MNIST, $n = 5000$

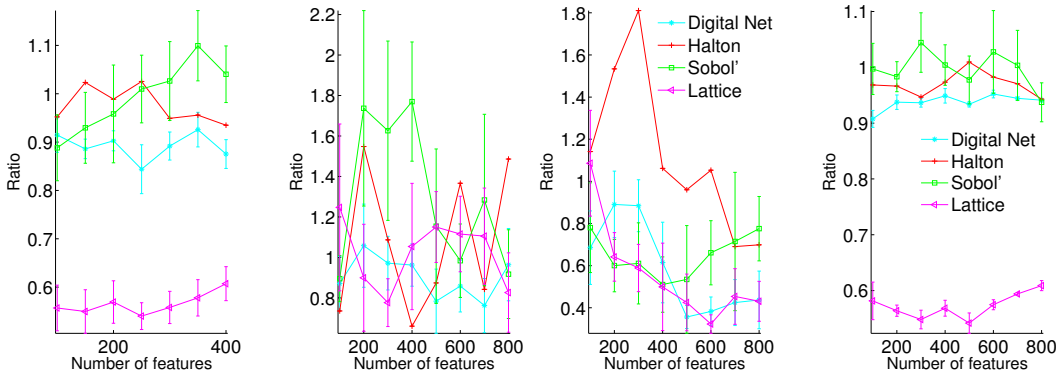
Figure 35: Relative error on approximating the Gram matrix measured in Euclidean norm and Frobenius norm, i.e., $\|K - \tilde{K}\|_2 / \|K\|_2$ and $\|K - \tilde{K}\|_F / \|K\|_F$, for various s. For each kind of random feature and s, 10 independent trials are executed, and the mean and standard deviation are plotted.

As can be seen, scrambled QMC sequences provide more accurate approximations in most cases as the ratio value tends to be less than one. In particular, scrambled Lattice sequence outperforms the non-scrambled one across all the cases for larger values of s. Therefore, in the rest of the experiments we use scrambled sequences.

8.5.1.2 *Generalization Error*

We consider two regression data sets, cpu and census, and use (approximate) kernel ridge regression to build a regression model. The ridge parameter is set by the optimal value we obtain via 5-fold cross-validation on the training set by using the MC sequence. Table 24 summarizes the results.

As we see, for cpu, all the sequences behave similarly, with the Halton sequence yielding the lowest test error. For census, the advantage of using Halton sequence is significant (almost 20% reduction in generalization error) followed by Digital Nets and Sobol'. In addition, MC sequence tends to generate higher variance across all the sampling size. Overall, QMC sequences, especially Halton, outperform MC sequences on these data sets.



(a) USPST (b) CPU (c) CENSUS (d) MNIST

Figure 36: Ratio between relative errors on approximating the Gram matrix using both the scrambled and non-scambled version of the same QMC sequence for various s. The lower the ratio value is, the more accurate the scrambled QMC approximation is. For each kind of QMC sequences and s, 10 independent trails are executed, and the mean and standard deviation are plotted.

Table 24: Regression error, i.e., $\|\hat{y} - y\|_2 / \|y\|_2$ where $\hat{y}$ is the predicted value and y is the ground truth. For each kind of random feature and s, 10 independent trials are executed, and the mean and standard deviation are listed.

	s	HALTON	SOBOL'	LATTICE	DIGIT	MC
cpu	100	**0.0367**	0.0383	0.0374	0.0376	0.0383
		(0)	(0.0015)	(0.0010)	(0.0010)	(0.0013)
	500	**0.0339**	0.0344	0.0348	0.0343	0.0349
		(0)	(0.0005)	(0.0007)	(0.0005)	(0.0009)
	1000	**0.0334**	0.0339	0.0337	0.0335	0.0338
		(0)	(0.0007)	(0.0004)	(0.0003)	(0.0005)
census	400	**0.0529**	0.0747	0.0801	0.0755	0.0791
		(0)	(0.0138)	(0.0206)	(0.0080)	(0.0180)
	1200	**0.0553**	0.0588	0.0694	0.0587	0.0670
		(0)	(0.0080)	(0.0188)	(0.0067)	(0.0078)
	1800	**0.0498**	0.0613	0.0608	0.0583	0.0600
		(0)	(0.0084)	(0.0129)	(0.0100)	(0.0113)

When performed on classification data sets by using the same learning model, with a moderate range of s, e.g., less than 2000, the QMC sequences do not yield accuracy improvements over the MC sequence with the same consistency as in the regression case. The connection between kernel approximation and the performance in downstream applications is outside the scope of the current paper. Worth mentioning in this regard, is the recent work by Bach [Bac13], which analyses the connection between Nyström approximations of the Gram matrix, and the regression error, and the work of El Alaoui and Mahoney [EAM15] on kernel methods with statistical guarantees.

8.5.1.3 *Behavior of Box Discrepancy*

Next, we examine if $D^{\square}$ is predictive of the quality of approximation. We compute the normalized square box discrepancy values (i.e., $\pi^d(\prod_{j=1}^{d} b_j)^{-1} D^{\square}(S)^2$) as well as Gram matrix approximation error for the different sequences with different sample sizes s. The expected normalized square box discrepancy values for MC are computed using (8.13).

Our experiments revealed that using the full $\square b$ does not yield box discrepancy values that are very useful. Either the values were not predictive of the kernel approximation, or they tended to stay constant. Recall, that while the bounding box $\square b$ is set based on observed ranges of feature values in the data set, the actual distribution of points $\bar{\mathcal{X}}$ encountered inside that box might be far from uniform. This lead us to consider the discrepancy measure when measured on the central part of the bounding box (i.e., $\square b/2$ instead of $\square b$), which is equal to the integration error averaged over that part of the bounding box. Presumably, points from $\bar{x}$ concentrate in that region, and they may be more relevant for downstream predictive task.

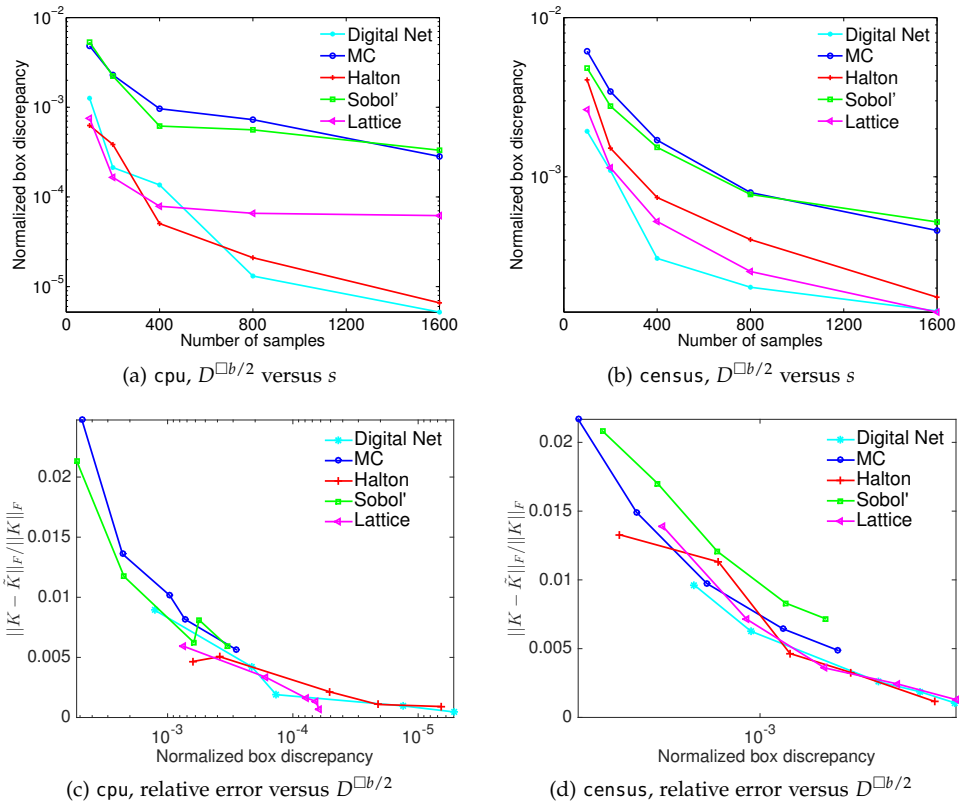
The results are shown in Figure 37. In the top graphs we can see, as expected, increasing number of features in the sequence leads to a lower box discrepancy value. In the bottom graphs, which compare $\|K - \tilde{K}\|_F / \|K\|_F$ to $D^{\square b/2}$, we can see a strong correlation between the quality of approximation and the discrepancy value.

8.5.2 *Experiments With Adaptive QMC Sequences*

The goal of this subsection is to provide a proof-of-concept for learning adaptive QMC sequences, using the three schemes described in Section 8.4. We demonstrate that QMC sequences can be improved to produce better approximation to the Gram matrix, and that can sometimes lead to improved generalization error.

Note that the running time of learning the adaptive sequences is less relevant in our experimental setting for the following reasons. Given the values of s, d, b and σ the optimization of a sequence needs only to be done once. There is some flexibility in these parameters: d can be adjusted by adding zero features or by doing PCA on the input; one can use longer or shorter sequences; and the data can be forced to a fit a particular bounding box using (possibly non-equal) scaling of the features (this, in turn, affects the choice of the σ) . Since designing adaptive QMC sequences is data-independent with applicability to a variety of downstream applications of kernel methods, it is quite conceivable to generate many point sets in advance and to use them for many learning tasks. Furthermore, the total size of the sequences ($s \times d$) is independent of the number of examples n, which is the dominant term in large scale learning settings.

We name the three sequences as *Global Adaptive*, *Greedy Adaptive* and *Weighted* respectively. For *Global Adaptive*, the Halton sequence is used as the initial setting of the optimization variables S. For *Greedy Adaptive*, when optimizing for w_t, the t-th point in the Halton



(a) cpu, $D^{\square b/2}$ versus s

(b) census, $D^{\square b/2}$ versus s

(c) cpu, relative error versus $D^{\square b/2}$

(d) census, relative error versus $D^{\square b/2}$

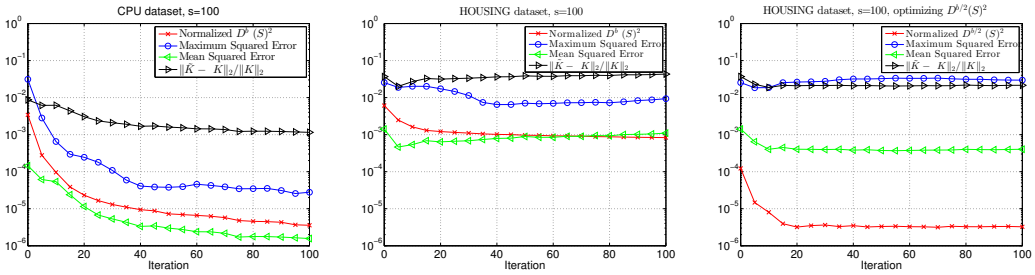
Figure 37: Discrepancy values ($D^{\square b/2}$) for the different sequences on cpu and census. We measure the discrepancy on the central part of the bounding box (we use $\square b/2$ instead of $\square b$ as the domain in the box discrepancy).

sequence is used as the initial point. In both cases, we use nonlinear conjugate gradient to perform numerical optimization. For *Weighted*, the initial features are generated using the Halton sequence and we optimize for the weights. We used CVX [GB14] to compute the sequence (solve (8.17)).

8.5.2.1 *Quality of Kernel Approximation*

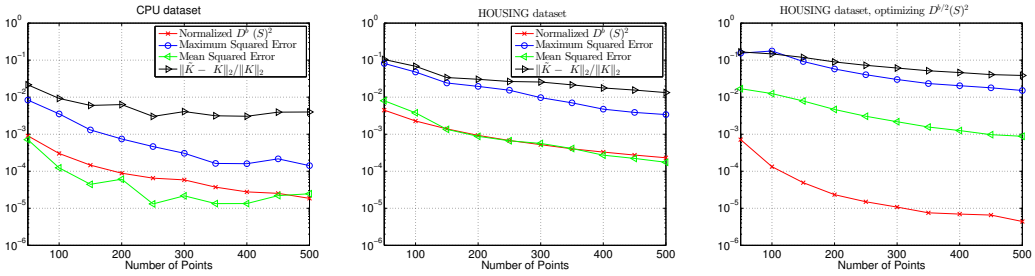
In Figure 38 and Figure 39 we examine how various metrics (discrepancy, maximum squared error, mean squared error, norm of the error) on the Gram matrix approximation evolve during the optimization process for both adaptive sequences. Since learning the adaptive sequences on data set with low dimensional features is more affordable, the experiment is performed on two such data sets, namely, cpu and housing.

For *Global Adaptive*, we fixed $s = 100$ and examine how the performance evolves as the number of iterations grows. In Figure 38a we examine the behavior on cpu. We see that all metrics go down as the iteration progresses. This supports our hypothesis that by optimizing the box discrepancy we can improve the approximation of the Gram matrix. Figure 38b, which examines the same metrics on the scaled version of the housing data set, has some interesting behaviors. Initially all metrics go down, but eventually all the metrics except the box-discrepancy start to go up; the box-discrepancy continues to go



(a) (b) (c)

Figure 38: Examining the behavior of learning *Global Adaptive* sequences. Various metrics on the Gram matrix approximation are plotted.



(a) (b) (c)

Figure 39: Examining the behavior of learning *Greedy Adaptive* sequences. Various metrics on the Gram matrix approximation are plotted.

down. One plausible explanation is that the integrands are not uniformly distributed in the bounding box, and that by optimizing the expectation over the entire box we start to overfit it, thereby increasing the error in those regions of the box where integrands actually concentrate. One possible way to handle this is to optimize closer to the center of the box (e.g., on $\square b/2$), under the assumption that integrands concentrate there. In Figure 38c we try this on the `housing` data set. We see that now the mean error and the norm error are much improved, which supports the interpretation above. But the maximum error eventually goes up. This is quite reasonable as the outer parts of the bounding box are harder to approximate, so the maximum error is likely to originate from there. Subsequently, we stop the adaptive learning of the QMC sequences early, to avoid the actual error from going up due to averaging.

For *Greedy Adaptive*, we examine its behavior as the number of points increases. In Figure 39a and Figure 39b, as expected, as the number of points in the sequence increases, the box discrepancy goes down. This is also translated to non-monotonic decrease in the other metrics of Gram matrix approximation. However, unlike the global case, we see in Figure 39c, when the points are generated by optimizing on a smaller box $\square b/2$, the resulting metrics become higher for a fixed number of points. Although the *Greedy Adaptive* sequence can be computed faster than the adaptive sequence, potentially it might need a large number of points to achieve certain low magnitude of discrepancy. Hence, as shown in the plots, when the number of points is below 500, the quality of the optimization is not good enough to provide a good approximation the Gram matrix. For example, one

can check when the number of points is 100, the discrepancy value of the *Greedy Adaptive* sequence is higher than that of the *Global Adaptive* sequence with more than 10 iterations.

Table 25: Discrepancy values, measured on the full bounding box and its central part, i.e., $D^{\square b}$ and $D^{\square b/4}$.

		$D^{\square b}$				$D^{\square b/4}$			
	s	HALTON	GLOBAL$_b$	GREEDY$_b$	WEIGHTED$_b$	HALTON	GLOBAL$_{b/4}$	GREEDY$_{b/4}$	WEIGHTED$_{b/4}$
cpu	100	3.41e-3	1.29e-6	3.02e-4	7.84e-5	9.44e-5	5.57e-8	2.62e-5	1.67e-8
	300	8.09e-4	5.14e-6	5.85e-5	1.45e-6	2.57e-5	1.06e-7	3.08e-6	2.93e-9
	500	2.39e-4	2.83e-6	1.86e-5	3.39e-7	7.91e-6	2.62e-8	1.04e-6	2.43e-9
census	400	2.61e-3	9.32e-4	7.47e-4	8.83e-4	5.73e-4	2.79e-5	2.20e-5	2.45e-5
	800	1.21e-3	5.02e-4	3.33e-4	4.91e-4	2.21e-4	1.12e-5	8.04e-6	5.46e-6
	1200	8.27e-4	3.41e-4	2.06e-4	3.39e-4	1.39e-4	8.15e-6	4.23e-6	2.29e-6
	1800	5.31e-4	2.17e-4	1.27e-4	2.31e-4	3.79e-5	5.59e-6	2.63e-6	8.37e-7
	2200	4.33e-4	1.73e-4	1.01e-4	1.87e-4	2.34e-5	3.35e-6	1.95e-6	4.93e-7

Table 25 also shows the discrepancy values of various sequences on `cpu` and `census`. Using adaptive sequences improves the discrepancy values by orders-of-magnitude. We note that a significant reduction in terms of discrepancy values can be achieved using only weights, sometimes yielding discrepancy values that are better than the hard-to-compute global or greedy sequences.

8.5.2.2 *Generalization Error*

We use the three algorithms for learning adaptive sequences as described in the previous subsections, and use them for doing approximate kernel ridge regression. The ridge parameter is set by the value which is near-optimal for both sequences in 5-fold cross-validation on the training set. Table 26 summarizes the results.

Table 26: Regression error, i.e., $\|\hat{y} - y\|_2 / \|y\|_2$ where $\hat{y}$ is the predicted value and y is the ground truth.

	s	HALTON	GLBOBAL$_b$	GLOBAL$_{b/4}$	GREEDY$_b$	GREEDY$_{b/4}$	WEIGHTED$_b$	WEIGHTED$_{b/4}$
cpu	100	0.0304	0.0315	**0.0296**	0.0307	**0.0296**	0.0366	0.0305
	300	0.0303	0.0278	0.0293	0.0274	**0.0269**	0.0290	0.0302
	500	0.0348	0.0347	0.0348	0.0328	**0.0291**	0.0342	0.0347
census	400	0.0529	0.1034	0.0997	0.0598	0.0655	**0.0512**	0.0926
	800	0.0545	0.0702	0.0581	0.0522	0.0501	**0.0476**	0.0487
	1200	0.0553	0.0639	**0.0481**	0.0525	0.0498	0.0496	0.0501
	1800	0.0498	0.0568	**0.0476**	0.0685	0.0548	0.0498	0.0491
	2200	0.0519	**0.0487**	0.0515	0.0694	0.0504	0.0529	0.0499

For both `cpu` and `census`, at least one of the adaptive sequences sequences can yield lower test error for each sampling size (since the test error is already low, around 3% or 5%, such improvement in accuracy is not trivial). For `cpu`, greedy approach seems to give slightly better results. When $s = 500$ or even larger (not reported here), the performance of the sequences are very close. For `census`, the weighted sequence yields the lowest generalization error when $s = 400, 800$. Afterwards we can see global adaptive

sequence outperforms the rest of the sequences, even though it has better discrepancy values. In some cases, adaptive sequences sometimes produce errors that are bigger than the unoptimized sequences.

In most cases, the adaptive sequence on the central part of the bounding box outperforms the adaptive sequence on the entire box. This is likely due to the non-uniformity phenomena discussed earlier.

9 RANDOM LAPLACE FEATURE MAPS FOR SEMIGROUP KERNELS ON HISTOGRAMS

A wide spectrum of statistical learning problems in computer vision have been elegantly framed within the framework of kernel methods; see Section 8.1 for introduction. Kernel methods are highly generalizable and versatile: it leads to nonlinear algorithms for supervised image classification and object detection, unsupervised visual feature extraction, image denoising, action recognition in videos, integration of multiple descriptors [GN09] , and many other tasks [Lam09].

In the face of "big data" in computer vision [Den+09; TFF08], the scalability of kernel methods, which is typically super-linear in the number of data points, is well-recognized as a valid concern. In recent years, as introduced in Section 8.1, approximations to kernel functions via explicit low-dimensional feature maps [RR08; VZ12; LIS10; MB09; Hua+14] have emerged as an appealing strategy to turn the complexity of learning nonlinear kernel methods back to that of training linear models, which typically scale linearly in the number of data points in a variety of settings such as regression, classification [Joa06] and principal component analysis. Importantly, storage requirements and test-time prediction speed can also be dramatically improved.

In this work, we propose new randomized approximate feature maps for kernels based on the semigroup structure of $\mathbb{R}^d_+$. These *semigroup kernels* are characterized via extensions of Bochner's theorem (Theorem 8.2) developed in the theory of harmonic analysis for general algebraic structures such as groups and semigroups. The Laplace transform assumes the role of the Fourier transform in our setting. Our proposed technique is therefore termed as *random Laplace features*, analogous to random Fourier features for shift-invariant kernels on $\mathbb{R}^d$. We provide theoretical results on the uniform convergence of random Laplace features. Empirical analyses on image classification and surveillance event detection tasks demonstrate the attractiveness of using random Laplace features relative to several other feature maps proposed in the literature.

Construction of random Laplace features is presented in Section 9.1. Theoretical results are stated in Section 9.2. Finally, empirical results are shown in Section 9.3. The material in this chapter appears in Yang et al. [Yan+14b].

Note that in this chapter, as in Chapter 8, in a sequence of vectors, we use w^i to denote the i-th element of the sequence and use w^i_j to denote the j-th coordinate of vector w^i. Given a vector x, we use x_i to denote the i-th coordinate of x.

9.1 MAIN ALGORITHM

The starting point of this work is the observation that the natural algebraic structure on the space of histograms and other non-negative descriptors, is that of an *abelian semi-group*.

Definition 9.1 (Abelian semigroup)**.** *A semigroup $(S, \circ)$ is a nonempty set S equipped with an associative composition $\circ$, i.e. for any $x, y, z \in S : x \circ (y \circ z) = (x \circ y) \circ z$ and a neutral/identity element e, i.e., for any $x \in S : x \circ e = x$. For an abelian semigroup, the composition is commutative, i.e., for any $x, y \in S : x \circ y = y \circ x$.*

Algorithm 13 Random Laplace features

1: **Input:** Characteristic kernel k on $(\mathbb{R}^d_+, +)$, size s.
2: **Output:** Feature map $\hat{\Psi}(x) : \mathbb{R}^d \mapsto \mathbb{R}^s$.
3: Find p, the inverse Laplace transform of k.
4: Draw sequence $w^1, \ldots, w^s$ from p.
5: Set $\hat{\Psi}(x) = \sqrt{\frac{1}{s}} \left[e^{-x^T w^1}, \ldots, e^{-x^T w^s} \right]$.
6: **Return** $\hat{\Psi}(x)$.

In particular, $(\mathbb{R}^d_+, +)$ forms an abelian semigroup with $0 \in \mathbb{R}^d_+$ as the identity element. This basic definition is sufficient to introduce the concept of kernels on semigroups.

Definition 9.2 (Kernels on Abelian semigroups [BCR84]). *A function $k : S \times S \mapsto \mathbb{R}$ is a positive definite kernel function on an abelian semigroup $(S, \circ)$ if $k(s,t) = \phi(s \circ t)$ where $\phi : S \mapsto \mathbb{R}$ is a positive definite function, i.e., for any $s_1 \ldots s_n \in S$, and any real-valued scalars $c_1 \ldots c_n$, the following holds*[1]*: $\sum_{i,j=1}^n c_i c_j \phi(s_i \circ s_j) \geq 0$.*

As per Definition 9.2, kernels respecting the algebraic structure of the semigroup $(R^d_+, +)$ can be written as $k(x,z) = \phi(x+z)$ where $\phi : \mathbb{R}^d_+ \mapsto \mathbb{R}$ is a positive definite function in the sense of satisfying $\sum_{i=1}^n c_i c_j \phi(x^i + x^j) \geq 0$ for any set of n non-negative vectors $x^1 \ldots x^n$ and choice of real-valued scalars $c_1 \ldots c_n$. The key observation is that positive-definite functions on R^d_+ are characterized by a theorem similar to Bochner's Theorem (Theorem 8.2), in which the Laplace transform replaces the Fourier transform.

Theorem 9.3 ([BCR84]). *A bounded continuous kernel function $k(x,z) \equiv \phi(x+z)$ on the Abelian semigroup $(\mathbb{R}^d_+, +)$ is positive definite if and only if it is the Laplace transform of a unique non-negative measure on $\mathbb{R}^d_+$. That is, for any $x, z \in \mathbb{R}^d_+$,*

$$k(x,z) = \int_{\mathbb{R}^d_+} e^{-(x+z)^T w} p(w) dw = \mathbb{E}_{w \sim p} \left[e^{-(x+z)^T w} \right].$$

This theorem should be contrasted with the Bochner's characterization for shift-invariant kernels on $\mathbb{R}^d$. As We assume without loss of generality that the non-negative measure above is a probability measure with associated density p. This result establishes one-to-one correspondence between semigroup kernels and probability densities on $\mathbb{R}^d$, via the Laplace transform. Exactly analogous to the random Fourier construction, we can now develop random Laplace feature maps via a Monte Carlo approximation,

$$k(x,z) \approx \frac{1}{s} \sum_{j=1}^{s} e^{-x^T w^j} e^{-z^T w^j} = \langle \hat{\Psi}(x), \hat{\Psi}(z) \rangle, \tag{9.1}$$

where the points w^j are drawn from p, yielding a feature map of the form

$$\hat{\Psi}(x) = \frac{1}{\sqrt{s}} [e^{-x^T w^1} \ldots e^{-x^T w^s}]. \tag{9.2}$$

This simple algorithm is summarized in the Algorithm 13.

1 For semigroups with involution operator $*$, the condition is $\sum_{i,j=1}^n c_i c_j \phi(s_i^* \circ s_j)$; see Berg, Christensen, and Ressel [BCR84].

When the density p corresponds to Lévy or Exponential distributions, the Laplace transform provides the associated *Exponential-Semigroup* and *Reciprocal-Semigroup* kernels, respectively. The exact form of these kernels is given in Table 27. These kernels have also been studied in the context of injective Reproducing Kernel Hilbert Space (RKHS) embeddings of probability distributions on groups and semigroups [Fuk+08; DGS10]. Through random Laplace features, one can expect to approximate these kernels well and deploy them for large-scale applications.

Indeed, beyond shift-invariant kernels on $\mathbb{R}^d$, several recent papers have attempted to develop explicit low-dimensional feature maps to approximate specific kernels that excel in computer vision applications [VZ12; LIS10; MB09]. These kernels are typically much better adapted to data representations in the form of finite probability distributions or normalized histograms, that common descriptors such as bag of visual words [Csu+04] and spatial pyramids [GD05] assume. Vedaldi and Zisserman [VZ12] suggested approximate feature maps for the additive family of Intersection, Hellinger's, χ^2 and Jensen-Shannon kernels whose feature spaces respectively induce well-known divergence measures on finite probability distributions. Li, Ionescu, and Sminchisescu [LIS10] suggested approximate feature maps for "skewed" multiplicative variants of the Intersection and χ^2 kernels, in an attempt to match the empirical performance of the exponentiated-χ^2 kernel, considered state-of-the-art [Cha99] for histogram descriptors. Table 27 catalogues these kernels and their associated approximate feature maps obtained through a randomized or deterministic sampling process.

Table 27: Summary of kernels and associated approximate feature maps. Above, $\bigoplus_{j=1}^{s} x^j = (x^1, \ldots, x^s)$, $i = \sqrt{-1}$, $h(x,z) = \frac{x}{2}\log_2\frac{x+z}{2} + \frac{y}{2}\log_2\frac{x+z}{z}$, $\log(x) = [\log(x_1)\ldots\log(x_d)]$. The feature map for Exponentiated-χ^2 is a composition of feature maps for the χ^2 and Gaussian kernels.

KERNEL	$k(x,z)$	$\hat{\Psi}(x)$	SAMPLING	REFERENCE
Gaussian	$e^{\frac{\|x-z\|_2^2}{2\sigma^2}}$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-ix^Tw^j}$	$\frac{1}{\sqrt{(2\pi\sigma^2)}}e^{-\frac{\sigma^2w^2}{2}}$ (Normal)	[RR08]
Laplacian	$e^{\frac{\|x-z\|_1}{\sigma}}$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-x^Tw^j}$	$\frac{\sigma}{\pi(1+\sigma^2w^2)}$ (Cauchy)	[RR08]
Hellinger	$\sum_{j=1}^{d}\sqrt{x_jz_j}$	$\bigoplus_{j=1}^{d}\sqrt{x_j}$	–	[VZ12]
χ^2	$2\sum_{j=1}^{d}\frac{x_jz_j}{x_j+z_j}$	$\bigoplus_{j=1,k}^{d} e^{iw_k\log x_j}\sqrt{x_j\mathrm{sech}(\pi w_k)}$	$w_k = kL, -r \le k \le r$	[VZ12]
Intersection	$\sum_{j=1}^{s}\min(x_j,z_j)$	$\bigoplus_{j=1,k}^{d} e^{iw_k\log x_j}\sqrt{\frac{2x_j}{\pi(1+4w_k^2)}}$	$w_k = kL, -r \le k \le r$	[VZ12]
Jensen-Shannon	$\sum_{j=1}^{d}h(x_j,z_j)$	$\bigoplus_{j=1,k}^{d} e^{iw_k\log x_j}\sqrt{\frac{2x_j\mathrm{sech}(\pi w^k)}{\log 4(1+4w_k^2)}}$	$w_k = kL, -r \le k \le r$	[VZ12]
Exponentiated-χ^2	$e^{-\sigma^{-2}\sum_{j=1}^{d}\frac{(x_j-z_j)^2}{x_j+z_j}}$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-i\hat{\Psi}_{\chi^2}(x)^Tw^j}$	$\frac{1}{\sqrt{(2\pi\sigma^2)}}e^{-\frac{\sigma^2w^2}{2}}$ (Normal)	[VZ12]
Jensen-Shannon	$\sum_{j=1}^{d}h(x_j,z_j)$	$\bigoplus_{j=1,k}^{d} e^{iw_k\log x_j}\sqrt{\frac{2x_j\mathrm{sech}(\pi w_k)}{\log 4(1+4w_k^2)}}$	$w_k = kL, -r \le k \le r$	[VZ12]
Skewed-χ^2	$\prod_{j=1}^{d}\frac{2\sqrt{x_j+\epsilon}\sqrt{z_j+\epsilon}}{x_j+z_j+2\epsilon}$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-i\log(x+\epsilon)^Tw^j}$	$\mathrm{sech}(\pi w)$ (Hyperbolic-secant)	[LIS10]
Skewed-Interection	$\prod_{j=1}^{d}\min\left(\sqrt{\frac{x_j+\epsilon}{z_j+\epsilon}},\sqrt{\frac{z_j+\epsilon}{x_j+\epsilon}}\right)$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-i\log(x+\epsilon)^Tw^j}$	$\frac{2}{\pi(1+4w^2)}$ (Cauchy)	[LIS10]
Exponential-Semigroup [Fuk+08]	$e^{-\beta\sum_{j=1}^{d}\sqrt{x_j+z_j}}$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-x^Tw^j}$	$\frac{\beta}{2\sqrt{\pi}}w^{-\frac{3}{2}}e^{\frac{-\beta^2}{4w}}$ (Lévy)	**This Paper**
Reciprocal-Semigroup [Fuk+08]	$\prod_{j=1}^{d}\frac{\lambda}{x_j+z_j+\lambda}$	$\bigoplus_{j=1}^{s}\sqrt{\frac{1}{s}}e^{-x^Tw^j}$	$\lambda e^{-\lambda w}$ (Exponential)	**This Paper**

In the next section we bound the approximation error $|k(x,z) - \langle\hat{\Psi}(x), \hat{\Psi}(z)\rangle|$ for inputs x, z drawn from a bounded domain in $\mathbb{R}_+^d$; we then study the empirical behaviour of the random Laplace feature map with respect to predictive tasks at hand, and benchmark its performance against several alternative approximate feature maps detailed in Table 27.

9.2 MAIN THEORETICAL RESULTS

In this section, we assume that the density function can be written as $p(w) = \prod_{j=1}^{d} q(w_j)$, where $q(\cdot)$ is a univariate density function. The kernel function can be written as $k(x,z) = \phi(x+z)$. We assume that ϕ is a differentiable function in $\mathbb{R}_d^+ / \{0\}$. Proofs for all the assertions are provided in Appendix E.

The following is a general result characterizing the error in approximating the kernel function using random Laplace features (Algorithm 13). The proof follows a similar strategy to the one used by Rahimi and Recht [RRo8].

Theorem 9.4. *Let $\mathcal{M}$ be the set consisting of all the points in $\mathbb{R}^d$ satisfying $\|x\|_2 \leq R$ and $x_i \geq r \geq 0$, $i = 1, \ldots, d$. Then, provided that $L_{q,r} \equiv \mathbb{E}_{w\sim q}\left[e^{-2wr}w^2\right] < \infty$ and $L_{q,r}R > \epsilon$, then for the mapping $\hat{\Psi}$ defined in Algorithm 13, we have*

$$\mathbb{P}\left[\sup_{x,z\in\mathcal{M}} |\langle\hat{\Psi}(x), \hat{\Psi}(z)\rangle - k(x,z)| \geq \epsilon\right] \leq 2^6 \left(\frac{dR^2 L_{q,r}}{\epsilon^2}\right) \exp\left(-\frac{s\epsilon^2}{d+2}\right), \tag{9.3}$$

Furthermore,

$$\sup_{x,z\in\mathcal{M}} |\langle\hat{\Psi}(x), \hat{\Psi}(z)\rangle - k(x,z)| < \epsilon$$

with any constant probability when $s = \Omega\left(\frac{d}{\epsilon^2}\log\frac{R^2 L}{\epsilon^2}\right)$.

Note that the quantity $L_{q,r}$ depends on the specific choice of kernel. We now give explicit expression for $L_{q,r}$ for q corresponding to two popular semigroup kernels.

The following proposition gives an explicit expression for the Exponential-Semigroup kernel.

Proposition 9.5. *Let $\beta > 0$, and let $q(w) = \frac{\beta}{2\sqrt{\pi}} w^{-3/2} e^{-\beta^2/4w}$ (this corresponds to the kernel $k(x,z) = e^{-\beta\sum_{i=1}^{d}\sqrt{x_i+z_i}}$). For $r > 0$ we have $L_{q,r} = \frac{\beta}{4}\frac{\sqrt{2r}\beta+1}{(2r)^{\frac{3}{2}}e^{\sqrt{2r}\beta}}$.*

In the above we require all the coordinates of x and z to be positive. Furthermore, if β is fixed, $L_{q,r}$ will go to infinity as r approaches zero. To get an approximate feature map with finite error bound for the Exponential-Semigroup kernel even when some coordinates of x or z are zero, it is natural to consider building a feature map on perturbed dataset, i.e., on $x+\delta$ and $z+\delta$ for some small δ (the addition here denotes a component-wise addition of the scalar).

Let $x' = x+\delta$ and $z' = z+\delta$. For any approximate kernel $c(\cdot,\cdot)$, by the triangle inequality we have,

$$\begin{aligned} |k(x,z) - c(x',z')| \quad \leq \quad & |k(x,z) - k(x',z')| + \\ & |k(x',z') - c(x',z')|. \end{aligned} \tag{9.4}$$

So, if the kernel function is sufficiently smooth and $c(x+\delta, z+\delta)$ approximates $k(x+\delta, z+\delta)$ well, then $c(x+\delta, z+\delta)$ will approximate $k(x,z)$ well. In particular, for the Exponential-Semigroup kernel we have the following proposition.

Proposition 9.6. *Let $\mathcal{M}$ be the set consisting of all the points in $\mathbb{R}^d$ satisfying $\|x\|_2 \leq R$ and $x_i \geq 0, i = 1, \ldots, d$. Let $\beta > 0$, and let $q(w) = \frac{\beta}{2\sqrt{\pi}} w^{-3/2} e^{-\beta^2/4w}$ (this corresponds to the kernel $k(x,z) = e^{-\beta \sum_{i=1}^{d} \sqrt{x_i + z_i}}$). For $\delta = \frac{\epsilon^2}{4d^2}$ we have*

$$\mathbb{P}\left[\sup_{x,z\in\mathcal{M}} |k(x,z) - \langle \hat{\Psi}(x+\delta), \hat{\Psi}(z+\delta)\rangle| \geq \epsilon\right] \leq 1 - 2^6 \left(\frac{d(R+\delta)^2 L_{q,\delta}}{\epsilon^2/4}\right) \exp\left(-\frac{s\epsilon^2/4}{d+2}\right), \tag{9.5}$$

where $L_{q,\delta} = \frac{\beta}{4} \frac{\sqrt{2\delta}\beta + 1}{(2\delta)^{\frac{3}{2}} e^{\sqrt{2\delta}\beta}}$.

The following proposition gives an explicit expression for the Reciprocal-Semigroup kernel.

Proposition 9.7. *Let $\lambda > 0$, $r \geq 0$, and let $q(w) = \lambda e^{-\lambda w}$ (this corresponds to the kernel $k(x,z) = \prod_{i=1}^{d} \left(\frac{\lambda}{x_i + z_i + \lambda}\right)$). We have, $L_{q,r} = \frac{2\lambda}{(\lambda + 2r)^3}$.*

9.3 EMPIRICAL EVALUATION

The number of random features controls the kernel approximation quality and the computational cost of solving a downstream task such as classification. Several empirical questions are of interest: For the same number of random features, how well do random Laplace features perform relative to other alternative feature maps on a given predictive task? How well is the underlying exact semigroup kernel approximated? Do histogram-based kernels outperform common shift-invariant kernels on $\mathbb{R}^d$ for problems of interest?

In the results reported in this section, we use the kernel name to denote the associated feature map. For example, by "Exp-Semigroup", we mean the use of random Laplace features to approximate the Exponential-Semigroup kernel. We report experiments on an image classification task (Caltech-101 [LFP03]) and a surveillance event detection task (TRECVID SED).

9.3.1 *Caltech-101*

For this dataset, we evaluate how well inner products in the Euclidean space induced by random Laplace features approximates the true semigroup kernel; we compare random Laplace features with other approximate feature maps on the predictive problem of classifying the 102 (101 + background) classes of the Caltech-101 benchmark dataset [LFP03]. Our data preparation follows the one used by Vedaldi and Zisserman [VZ12]. In particular, our results are competitive with the state of the art on this dataset for methods that use a single but strong image feature (multi-scale dense SIFT). We use the *phow_caltech* function of VLFeat[2] which rescales images to have a largest side of 480 pixels; dense SIFT features are extracted every four pixels at four scales and quantized into a 200 visual words dictionary estimated using k-means. Each image is described by a 4200-dimensional histogram of visual words with $1 \times 1, 2 \times 2$ and 4×4 spatial subdivisions.

QUALITY OF GRAM MATRIX APPROXIMATION

Given data points $\{x^i\}_{i=1}^n$, the Gram matrix $K \in \mathbb{R}^{n \times n}$ is defined as $K_{ij} = k(x^i, x^j)$.

2 http://www.vlfeat.org

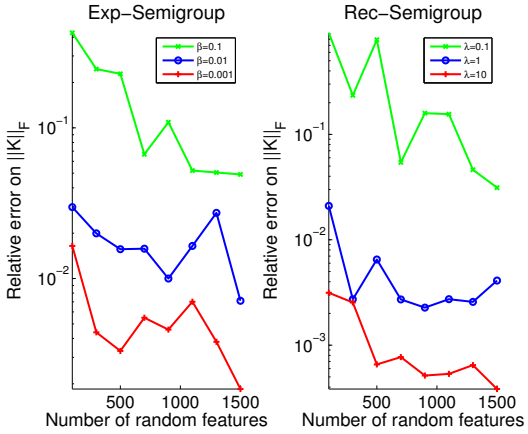


Figure 40: Gram matrix approximation relative error on Caltech-101 by using random Laplace features with s ranging from 100 to 1500. The two plots correspond to Exponential-Semigroup and Reciprocal-Semigroup kernel, respectively (see Table 27). Each curve corresponds to one value of β or λ. For the fixed pair of parameter, ten independent trials are executed and the mean is reported.

Suppose $Z \in \mathbb{R}^{n \times s}$ is the data matrix in the induced feature space, with the i-th row $Z^{(i)} = \hat{\Psi}(x^i)$ where $\hat{\Psi}(x^i)$ is the random Laplace feature of x^i generated from Algorithm 13. We will evaluate relative error in terms of Frobenius norm, $\|K - ZZ^T\|_F / \|K\|_F$, as a function of s, as this provides an overall measure of approximation quality over the entire set of points. We consider both Exp-Semigroup and Rec-Semigroup kernels.

In Figure 40, we show the relative error of random Laplace feature with the number of random features s ranging from 100 to 1500 on the full Caltech-101 dataset comprising of $n = 3060$ samples. We can see that as s grows, random Laplace feature converges to the exact kernel quickly, meaning the relative approximation error goes to zero fast in practice. As expected, the rate of convergence depends on the choice of the kernel parameters.

CLASSIFICATION PERFORMANCE

Next, we compare SVM classification accuracies on Caltech-101 by using approximate feature maps associated with different kernels as described in Table 27. We use the usual training-test splitting protocol with 15-images per class in the training set and an equal number in the test set. SVM parameters are tuned with cross-validation. Figure 41 reports mean test set accuracy as a function of the number of random features s. We include the original input features (i.e., using linear kernel) as a baseline. Among the semigroup kernels, the Exponential-Semigroup significantly outperforms the reciprocal semigroup whose performance is below baseline levels, and hence results for the latter are ommited.

Among the seven feature maps compared in Figure 41, the random Laplace features for the Exponential-Semigroup ($\beta = 0.01$) consistently yield the highest accuracy, outperforming the Exponentiated-χ^2 kernel which is widely considered state of the art on histogram descriptors.

The Gaussian kernel does not improve over the linear kernel baseline, while kernels such as the skewed-Intersection kernel [LIS10] perform significantly better, confirming the need to design kernels better adapted to histogram-like data.

The computation time for generating random features and solving the resulting classification problem is near-identical for all feature maps shown in Figure 41, except for the

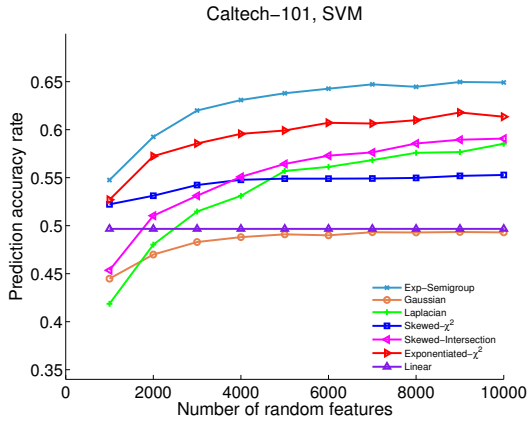


Figure 41: Prediction accuracy on Caltech-101 by using random feature maps associated to different kernels with s ranging from 1000 to 10000. The results are generated by using SVM. For a feature map and an s, five independent trials are executed and the mean is reported. For comparisons with χ^2, Intersection and Jensen-Shannon, see Table 28.

Exponentiated-χ^2. While Exponentiated-χ^2 yields the second highest accuracy, its running time is six times higher than the rest, since it generates a much higher dimensional intermediate χ^2 feature map, which is then composed with the feature map for the Gaussian kernel. For the feature maps of the homogeneous additive kernels [VZ12] which include χ^2, the number of random features that can be generated is of the form $(2r+1)d$ where d is the dimension of the original feature and r is a parameter. For high-dimensional input spaces, the resulting feature maps can be very high-dimensional and hence costly for downstream processing. For the Exponentiated-χ^2 feature map, we used $r = 3$ which corresponds to 29400 intermediate χ^2 features. The comparison with homogeneous additive kernels [VZ12] is reported in Table 28 for $s = 12600, 21000, 29400$ corresponding to $r = 1, 2, 3$. Again, the proposed random Laplace feature maps for the Exponential-Semigroup kernel are significantly better.

KERNEL	$s = 12600$	$s = 21000$	$s = 29400$
Exp-Semigroup	0.6536	0.6627	0.6643
χ^2	0.6510	0.6497	0.6471
Intersection	0.6399	0.6392	0.635
Jensen-Shannon	0.6510	0.6477	0.6477

Table 28: Caltech 101 SVM accuracies: Comparison against χ^2, Intersection and Jensen-Shannon can only be done for $s = (2r+1)d$. Here, $r = 1, 2, 3$. For a feature map and an s, five independent trials are executed and the mean is reported.

9.3.2 *Surveillance Event Detection (SED)*

There are seven target events in the TRECVED Surveillance Event Detection (SED) dataset, i.e., *CellToEar*, *Embrace*, *ObjectPut*, *Pointing*, *PeopleMeet*, *PeopleSplitUp* and *PersonRuns*. The dataset was captured in five locations at a busy airport. Many confounding issues exist in this dataset such as high activity levels, camera view changes, large variances in how

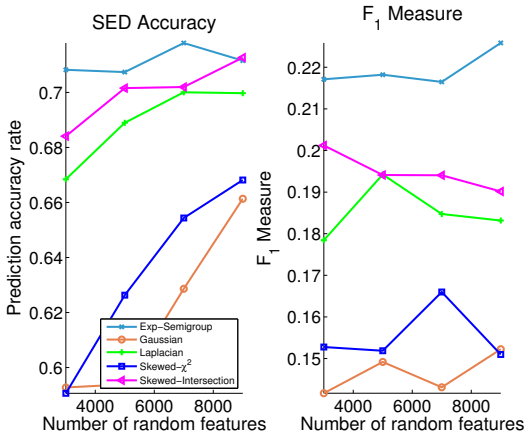


Figure 42: Prediction accuracy and F_1 score on SED by using random feature maps associated to different kernels with s ranging from 3000 to 9000. The two subplots are results of Prediction accuracy and F_1 score respectively. For a feature map and an s, five independent trials are executed and the mean is reported.

events play out (e.g., "PeopleMeet") and small objects carrying predictive signals (i.e., "CellToEar"). The development set consists of 100 hours of video and the evaluation set has an additional 50 hours of data. The annotations of the dataset only include temporal extents and event labels, and no localization information is provided for events. We used the development set in our experiments, and divided it into two equal part for validation.

The training size of SED is 7024, containing approximately even number of event and non-event samples. The test size is 22437, containing 5381 events instances. By event sample, we mean an observation coming from any of the seven events described above. We use bag of visual words on motion-SIFT features resulting in final dimensionality of $d = 24000$.

Due to the high dimensionality of the SED dataset, we do not generate the homogeneous additive and Exponentiated-χ^2 kernels in this case. We report the results for a regularized least squares classification model (SVMs perform similarly) with parameters tuned using cross-validation. The test accuracies and F_1 scores are shown in Figure 42. As before, random Laplace feature maps for the exponential semigroup kernel ($\beta = 0.001$) outperform other feature maps in this task.

10 SUMMARY AND DISCUSSION

This thesis mainly illustrates how randomized linear algebra can be useful in big data applications. Theoretically, as advanced algorithms with better complexities for common matrix problems are desired, we show how techniques such as rounding, embedding, and preconditioning discussed in Chapter 2 can be used to derive improved algorithms for large-scale regression and optimization problems. Practically, as distributed systems built on top of clusters of commodity hardware provide cheap and reliable storage, scalable implementations of these algorithms on cluster computing platforms are essential. We present computational results of implementing several RLA algorithms in parallel and distributed environments showing that these algorithms successfully minimize not only floating-point operations but also communication cost. We also show that RLA algorithms are useful in real-world large-scale applications for better interpretability.

10.1 THEORETICAL RESULTS

Part I focuses on the underlying theory of RLA. We show that subspace embeddings can be used in one of two related ways: to construct sub-sampled problems that can be solved with traditional numerical methods; or to construct preconditioned versions of the original full problem that are easier to solve with iterative algorithms. These ideas are manifested in Chapter 3 and Chapter 4. In particular, in Chapter 3 we develop a fast sampling algorithm for quantile regression. As a key step, our algorithm computes a low-distortion subspace-preserving embedding with respect to the loss function of quantile regression to obtain a subproblem whose size depends on the low dimension only. This subspace-preserving embedding is constructed based on data-aware ℓ_1 subspace-preserving embedding techniques discussed in Section 2.3.2.4 because of the close relationship between quantile regression and ℓ_1 regression. In Chapter 4, we present a hybrid algorithm for ℓ_p regression that combines RLA and SGD. Here embeddings are used to construct a preconditioner for the linear system, with which the underlying SGD solver converges much more quickly and has a complexity that depends on the low-dimension of the linear system only. This permits us to obtain a higher-precision solution. Finally, in Chapter 5, we develop a class of randomized Newton-type algorithms for optimization problems whose objective is $\sum_i^n f_i(w) + R(w)$. As the challenge of applying Newton's method to these optimization problems is the Hessian computation, which scales linearly in n, by noticing that forming the Hessian is essentially equivalent to a matrix-matrix multiplication problem, we exploit two data-aware sampling techniques in RLA to expedite the Hessian computation without losing too much convergence speed.

We only studied the optimization performance of these algorithms. That is, our theory is about the closeness between the approximate solution, $\hat{w}$ provided by our algorithms, and the optimal solution of the optimization problem being solved. However, in many applications such as statistical learning, it is often assumed that the data matrix is generated from a statistical model parameterized by a true parameter vector w^*. Attention has been drawn to the investigation of the statistical performance of RLA algorithms, namely closeness between $\hat{w}$ and w^*. Several recent works [MMY15; Che+15; WYS16] have shown

promising results for linear regression problems. A more comprehensive study of the statistical property of RLA algorithms remains an exciting research area.

Furthermore, most of our theory provides worst-case theoretical guarantees. That is, these bounds are "pessimistic" in the sense that they hold for arbitrary inputs. Two lines of research are of interest. The first is the study of complexity lower bounds. For example, Section 6.3 in [Woo14] discusses complexity lower bounds for several typical matrix problems in the streaming setting. The second is to study the performance of the algorithm when the input matrix satisfies certain structure. Whether computationally and statistically improved algorithms can be derived for specific inputs is left to explore in the future.

10.2 IMPLEMENTATIONS AND APPLICATIONS

Part II focuses on implementations and applications. In Chapter 6 we present the computational results of implementing RLA algorithms for ℓ_2 regression, quantile regression and CX decomposition problems in parallel and distributed environment with terabyte-sized inputs using Apache Spark and Apache Hadoop. These algorithms are described in Section 2.4.1, Chapter 3 and Section 2.4.2, respectively. Empirical results indicate these RLA algorithms are scalable to large-scale datasets and are amenable to parallel and distributed computing environments. Next, in Chapter 7, we present the results of applying CX decomposition to problems in bioimaging. More specifically, we show that CX decomposition successfully identifies a few important actual rows and columns of the original data matrix, i.e., pixels and ions of a complex biological sample, for nearly optimal reconstruction.

As Spark has been developed for industrial applications and commodity data center hardware, a more extensive study of the linear algebraic computation performance of Spark at scale is necessary. Recently, we [Git+16b] took on the important task of testing nontrivial linear algebra and matrix factorization computations in Spark, and compare and contrast its performance with C+MPI implementations on High Performance Computing (HPC) hardware. These comparisons have revealed a number of opportunities for improving Spark performance. For example, the current end-to-end performance gap is 2x-25x (Spark is slower), and 10x-40x without I/O. This is mainly because Spark performance overheads associated with scheduling, stragglers, result serialization, and task deserialization dominate the runtime by an order of magnitude. Improvements can be considered from these perspectives. In addition, in order for Spark to leverage existing high-performance linear algebra libraries, it may be worthwhile to investigate better mechanisms for integrating and interfacing with MPI-based runtimes with Spark.

Part IV

PROOFS

A

PROOFS OF RESULTS IN CHAPTER 3

Throughout the proofs in this chapter, for simplicity, we use κ to denote $\bar{\kappa}_1$. Also, by Step x below, we mean Step $x+2$ in the corresponding algorithm description, as the first two steps are used for illustrating inputs and outputs.

A.1 PROOF OF LEMMA 3.2

By Lemma 2.17, in Step 1, Π_1 is a low-distortion embedding satisfying (2.2) with $\kappa_\Phi = \mathcal{O}(d^3 \log^3 d)$, and $r_1 = \mathcal{O}(d^5 \log^5 d)$. In fact, AR^{-1} in Step 2 is a well-conditioned basis with $\kappa = \mathcal{O}(d^{\frac{13}{2}} \log^{\frac{11}{2}} d)$ according to (2.3). In Step 3, by Lemma 2.22, the sampling complexity required for obtaining a $(1 \pm 1/2)$-distortion sampling matrix is $\tilde{s} = \mathcal{O}(d^{\frac{15}{2}} \log^{\frac{11}{2}} d)$. Finally, if we view $\tilde{S}$ as a low-distortion embedding matrix with $r = \tilde{s}$ and $\kappa_\Phi = 3$, then the resulting R in Step 4 will ensure that AR^{-1} is a well-conditioned basis with $\kappa = \mathcal{O}(d^{\frac{19}{4}} \log^{\frac{11}{4}} d)$.

For the running time, it takes $\mathcal{O}(\mathtt{nnz}(A))$ time for completing Step 1. In Step 2, the running time is $r_1 d^2 = \mathtt{poly}(d)$. As pointed out in Section 2.3.2.4, the running time for constructing $\tilde{S}$ in Step 3 is $\mathcal{O}(\mathtt{nnz}(A) \cdot \log n)$. Since the large dimension of $\tilde{S}A$ is a low-degree polynomial of d, the QR factorization of it costs $\tilde{s}d^2 = \mathtt{poly}(d)$ time in Step 4. Overall, the running time of Algorithm 6 is $\mathcal{O}(\mathtt{nnz}(A) \cdot \log n)$.

A.2 PROOF OF LEMMA 3.3

Although $\rho_\tau(\cdot)$, defined in (3.2), is not a norm, since the loss function does not have the positive linearity, it satisfies some "good" properties, as stated in the following lemma:

Lemma A.1. *Suppose that $\tau \geq \frac{1}{2}$. Then, for any $x, y \in \mathbb{R}^d, a \geq 0$, the following hold:*

1. $\rho_\tau(x+y) \leq \rho_\tau(x) + \rho_\tau(y)$;
2. $(1-\tau)\|x\|_1 \leq \rho_\tau(x) \leq \tau\|x\|_1$;
3. $\rho_\tau(ax) = a\rho_\tau(x)$;
4. $|\rho_\tau(x) - \rho_\tau(y)| \leq \tau\|x-y\|_1$.

Proof. It is trivial to prove every equality or inequality for x, y in one dimension. Then by the definition of $\rho_\tau(\cdot)$ for vectors, the inequalities and equalities hold for general x and y. □

Two important ingredients for proving subspace preservation are γ-nets and tail inequalities. Suppose that Z is a point set and $\|\cdot\|$ is a metric on Z. A subset Z_γ is called a γ-net for some $\gamma > 0$ if for every $x \in Z$ there is a $y \in Z_\gamma$ such that $\|x-y\| \leq \gamma$. It is well known that the unit ball of a d-dimensional subspace has a γ-net with size at most $(3/\gamma)^d$ [BLM89]. Also, we use the standard Bernstein inequality to prove concentration results for the sum of independent random variables.

Lemma A.2 (Bernstein inequality). *Let $X_1, \ldots, X_n$ be independent random variables with zero-mean. Suppose that $|X_i| \le M$, for $i \in [n]$; then for any positive number t, we have*

$$\Pr\left[\sum_{i\in[n]} X_i > t\right] \le \exp\left(-\frac{t^2/2}{\sum_{i\in[n]} \mathbb{E}\left[X_j\right]^2 + Mt/3}\right).$$

Since U is a well-conditioned basis for the range space of A, to prove (3.5) it is equivalent to prove the following holds for all $y \in \mathbb{R}^d$:

$$(1-\varepsilon)\rho_\tau(Uy) \le \rho_\tau(SUy) \le (1+\varepsilon)\rho_\tau(Uy). \tag{A.1}$$

To prove that (A.1) holds for any $y \in \mathbb{R}^d$, first we show that (A.1) holds for any fixed $y \in \mathbb{R}^d$, and then we apply a standard γ-net argument to show that (A.1) holds for every $y \in \mathbb{R}^d$.

Assume that U is (α, β)-conditioned with $\kappa = \alpha\beta$. For $i \in [n]$, let $v_i = U_{(i)}y$. Then $\rho_\tau(SUy) = \sum_{i\in[n]} \rho_\tau(S_{ii}v_i) = \sum_{i\in[n]} S_{ii}\rho_\tau(v_i)$ since $S_{ii} \ge 0$. Let $w_i = S_{ii}\rho_\tau(v_i) - \rho_\tau(v_i)$ be a random variable, where

$$w_i = \begin{cases} (\frac{1}{\hat{p}_i} - 1)\rho_\tau(v_i), & \text{with probability } \hat{p}_i; \\ -\rho_\tau(v_i), & \text{with probability } 1-\hat{p}_i. \end{cases}$$

Therefore, $\mathbb{E}[w_i] = 0, \mathbf{Var}[w_i] = (\frac{1}{\hat{p}_i} - 1)\rho_\tau(v_i)^2, |w_i| \le \frac{1}{\hat{p}_i}\rho_\tau(v_i)$. Note here we only consider i such that $\|U_{(i)}\|_1/|U|_1 < 1$ because otherwise we have $\hat{p}_i = 1$ and the corresponding term will not contribute to the variance. According to our definition, $\hat{p}_i \ge s \cdot \|U_{(i)}\|_1/|U|_1 = s \cdot t_i$. Consider

$$\rho_\tau(v_i) = \rho_\tau(U_{(i)}y) \le \tau\|U_{(i)}y\|_1 \le \tau\|(U_{(i)})\|_1\|y\|_\infty.$$

Hence,

$$\begin{aligned} |w_i| &\le \frac{1}{\hat{p}_i}\rho_\tau(v_i) \le \frac{1}{\hat{p}_i}\tau\|U_{(i)}\|_1\|y\|_\infty \le \frac{\tau}{s}|U|_1\|y\|_\infty \\ &\le \frac{1}{s}\frac{\tau}{1-\tau}\alpha\beta\rho_\tau(Uy) := M. \end{aligned}$$

Also,

$$\sum_{i\in[n]} \mathbf{Var}[w_i] \le \sum_{i\in[n]} \frac{1}{\hat{p}_i}\rho_\tau(v_i)^2 \le M\rho_\tau(Uy).$$

Applying the Bernstein inequality to the zero-mean random variables w_i gives

$$\Pr\left[\left|\sum_{i\in[n]} w_i\right| > \varepsilon\right] \le 2\exp\left(\frac{-\varepsilon^2}{2\sum_i \mathbf{Var}[w_i] + \frac{2}{3}M\epsilon}\right).$$

Since $\sum_{i\in[n]} w_i = \rho_\tau(SUy) - \rho_\tau(Uy)$, setting ε to $\varepsilon\rho_\tau(Uy)$ and substituting the results we derive above gives

$$\Pr\left[|\rho_\tau(SUy) - \rho_\tau(Uy)| > \varepsilon\rho_\tau(Uy)\right] \le 2\exp\left(\frac{-\varepsilon^2\rho_\tau^2(Uy)}{2M\rho_\tau(Uy) + \frac{2\varepsilon}{3}M\rho_\tau(Uy)}\right).$$

Let's simplify the exponential term on the right-hand side of the above expression:

$$\frac{-\varepsilon^2\rho_\tau^2(Uy)}{2M\rho_\tau(Uy) + \frac{2\varepsilon}{3}M\rho_\tau(Uy)} = \frac{-s\varepsilon^2}{\alpha\beta}\frac{1-\tau}{\tau}\frac{1}{2+\frac{2\varepsilon}{3}} \le \frac{-s\varepsilon^2}{3\alpha\beta}\frac{1-\tau}{\tau}.$$

Then, when $s \ge \frac{\tau}{1-\tau}\frac{27\alpha\beta}{\epsilon^2}\left(d\log\left(\frac{3}{\gamma}\right) + \log\left(\frac{4}{\delta}\right)\right)$, with probability at least $1 - (\gamma/3)^d\delta/2$,

$$(1-\epsilon/3)\rho_\tau(Uy) \le \rho_\tau(SUy) \le (1+\epsilon/3)\rho_\tau(Uy), \tag{A.2}$$

where γ is specified later.

We show that, for all $z \in \text{range}(U)$,

$$(1-\epsilon)\rho_\tau(z) \le \rho_\tau(Sz) \le (1+\epsilon)\rho_\tau(z). \tag{A.3}$$

By the positive linearity of $\rho_\tau(\cdot)$, it suffices to show (A.3) holds for all z with $\|z\|_1 = 1$.

Next, let $Z = \{z \in \text{range}(U) \mid \|z\|_1 \le 1\}$ and construct a γ-net of Z, denoted by Z_γ, such that for any $z \in Z$, there exists a $z_\gamma \in Z_\gamma$ that satisfies $\|z - z_\gamma\|_1 \le \gamma$. By [BLM89], the number of elements in Z_γ is at most $(3/\gamma)^d$. Hence, with probability at least $1 - \delta/2$, (A.2) holds for all $z_\gamma \in Z_\gamma$.

We claim that with suitable choice γ, with probability at least $1 - \delta/2$, S will be a $(1 \pm 2/3)$-distortion embedding matrix of $(\mathcal{A}, \|\cdot\|_1)$. To show this, first we state a similar result for $\|\cdot\|_1$ from Theorem 6 in [Das+09] with $p = 1$ as follows.

Lemma A.3 (ℓ_1 subspace-preserving sampling lemma)**.** *Given $A \in \mathbb{R}^{n\times d}$, let $U \in \mathbb{R}^{n\times d}$ be an (α, β)-conditioned basis for $\mathcal{A}$. For $s > 0$, define*

$$\hat{p}_i \ge \min\{1, s\cdot\|U_{(i)}\|_1/|U|_1\}$$

and let $S \in \mathbb{R}^{n\times n}$ be a random diagonal matrix with $S_{ii} = 1/\hat{p}_i$ with probability $\hat{p}_i$, and 0 otherwise. Then when $\epsilon < 1/2$ and

$$s \ge \frac{32\alpha\beta}{\epsilon^2}\left(d\log\left(\frac{12}{\epsilon}\right) + \log\left(\frac{2}{\delta}\right)\right),$$

with probability at least $1 - \delta$, for every $x \in \mathbb{R}^d$,

$$(1-\varepsilon)\|Ax\|_1 \le \|SAx\|_1 \le (1+\varepsilon)\|Ax\|_1. \tag{A.4}$$

Note here we change the constraint $\epsilon \le 1/7$ and the original theorem to $\epsilon \le 1/2$ above. One can easily show that the result still holds with such setting. If we set $\epsilon = 2/3$ and the failure probability to be at most $\delta/2$, the construction of S satisfies the conditions of Lemma A.3 when the expected sampling complexity $s \ge \bar{s} := 72\alpha\beta\left(d\log(18) + \log\left(\frac{4}{\delta}\right)\right)$. Then our claim for S holds. Hence we only need to make sure with suitable choice of γ that $s \ge \bar{s}$.

For any z with $\|z\|_1 = 1$, we have

$$\begin{aligned}
|\rho_\tau(Sz) - \rho_\tau(z)| &\leq |\rho_\tau(Sz) - \rho_\tau(Sz_\gamma)| + |\rho_\tau(Sz_\gamma) - \rho_\tau(z_\gamma)| + |\rho_\tau(z_\gamma) - \rho_\tau(z)| \\
&\leq \tau\|S(z - z_\gamma)\|_1 + (\epsilon/3)\rho_\tau(z_\gamma) + \tau\|z_\gamma - z\|_1 \\
&\leq \tau|\|S(z - z_\gamma)\|_1 - \|(z - z_\gamma)\|_1| + (\epsilon/3)\rho_\tau(z) + (\epsilon/3)\rho_\tau(z_\gamma - z) \\
&\quad +2\tau\|z_\gamma - z\|_1 \\
&\leq 2\tau/3\|z - z_\gamma\|_1 + (\epsilon/3)\rho_\tau(z) + \tau(\epsilon/3)\|z_\gamma - z\|_1 + 2\tau\|z_\gamma - z\|_1 \\
&\leq (\epsilon/3)\rho_\tau(z) + \tau\gamma(2/3 + \epsilon/3 + 2) \\
&\leq \left(\epsilon/3 + \frac{\tau}{1-\tau}\gamma(2/3 + \epsilon/3 + 2)\right)\rho_\tau(z) \\
&\leq \epsilon\rho_\tau(z),
\end{aligned}$$

where we take $\gamma = \frac{1-\tau}{6\tau}\epsilon$, and the expected sampling size becomes

$$s = \frac{\tau}{1-\tau}\frac{27\alpha\beta}{\epsilon^2}\left(d\log\left(\frac{\tau}{1-\tau}\frac{18}{\epsilon}\right) + \log\left(\frac{4}{\delta}\right)\right).$$

When $\epsilon < 1/2$, we will have $s > \bar{s}$. Hence the claim for S holds and (A.3) holds for every $z \in \text{range}(U)$.

Since the proof involves two random events with failure probability at most $\delta/2$, by a simple union bound, (A.1) holds with probability at least $1 - \delta$. Our results follows because $\kappa = \alpha\beta$.

A.3 PROOF OF LEMMA 3.4

In this lemma, slightly different from the previous notation, we use s and $\hat{s}$ to denote the actual number of rows selected and the input parameter for defining the sampling probability, respectively. From Lemma 3.3, a $(1 \pm \epsilon)$-distortion sampling matrix S could be constructed by calculating the ℓ_1 norms of the rows of AR^{-1}. Indeed, we will estimate these row norms and adjust the sampling complexity s. According to Lemma 12 in [Cla+13], with probability at least 0.95, the $\lambda_i, i \in [n]$ we compute in Steps 1-3 of Algorithm 7 satisfy

$$\frac{1}{2}\|U_{(i)}\|_1 \leq \lambda_i \leq \frac{3}{2}\|U_{(i)}\|_1,$$

where $U = AR^{-1}$. Conditioned on this high probability event, we set

$$\hat{p}_i \geq \min\left\{1, \hat{s} \cdot \frac{\lambda_i}{\sum_{i\in[n]}\lambda_i}\right\}.$$

Then we will have $\hat{p}_i \geq \min\left\{1, \frac{\hat{s}}{3} \cdot \frac{\|U_{(i)}\|_1}{|U|_1}\right\}$. Since $\hat{s}/3$ satisfies the sampling complexity required in Lemma 3.3 with $\delta = 0.05$, the corresponding sampling matrix S is constructed as desired. These are done in Step 4 and Step 5. Since the algorithm involves two random events, by a simple union bound, with probability at least 0.9, S is a $(1 \pm \epsilon)$-distortion sampling matrix.

By the definition of sampling probabilities, $\mathbb{E}[s] = \sum_{i\in[n]}\hat{p}_i \leq \hat{s}$. Note here that s is the sum of some random variables and it is tightly concentrated around its expectation. By

a standard Bernstein bound, with probability $1 - o(1)$, $s \leq 2\hat{s} = \mathcal{O}\left(\mu\kappa d \log\left(\mu/\epsilon\right)/\epsilon^2\right)$, where $\mu = \frac{\tau}{1-\tau}$, as claimed.

Now let's compute the running time in Algorithm 7. The main computational cost comes from Steps 2, 3 and 5. The running time in other steps will be dominated by it. It takes $d^2 r_2$ time to compute $R^{-1}\Pi_2$; then it takes $\mathcal{O}(\mathtt{nnz}(A) \cdot r_2)$ time to compute $AR^{-1}\Pi_2$; and finally it takes $\mathcal{O}(n)$ time to compute all the λ_i and construct S. Since $r_2 = \mathcal{O}(\log n)$, the total running time is $\mathcal{O}((d^2 + \mathtt{nnz}(A)) \log n + n) = \mathcal{O}(\mathtt{nnz}(A) \cdot \log n)$.

A.4 PROOF OF THEOREM 3.5

In Step 1, by Lemma 3.1, the matrix $R \in \mathbb{R}^{d\times d}$ computed by Algorithm 5 satisfies that with probability at least 0.9, AR^{-1} is a well-condition basis for $\mathcal{A}$ with $\kappa = 6d^2$. The probability bound can be attained by setting the corresponding constants sufficiently large. In Step 2, when we apply Algorithm 7 to construct the sampling matrix S, by Lemma 3.4, with probability at least 0.9, S will be a $(1 \pm \epsilon)$-distortion sampling matrix of $(\mathcal{A}, \rho_\tau(\cdot))$. Solving the subproblem $\min_{x\in\mathcal{C}} \rho_\tau(SAx)$ gives a $(1+\epsilon)/(1-\epsilon)$ solution to the original problem (3.3). This is because

$$\rho_\tau(A\hat{x}) \leq \frac{1}{1-\varepsilon}\rho_\tau(SA\hat{x}) \leq \frac{1}{1-\varepsilon}\rho_\tau(SAx^*) \leq \frac{1+\varepsilon}{1-\varepsilon}\rho_\tau(Ax^*), \tag{A.5}$$

where the first and third inequalities come from (3.6) and the second inequality comes from the fact that $\hat{x}$ is the minimizer of the subproblem. Hence the solution $\hat{x}$ returned by Step 3 satisfies our claim. The whole algorithm involves two random events, and the overall success probability is at least 0.8.

Now let's compute the running time for Algorithm 8. In Step 1, by Lemma 3.1, the running time for Algorithm 5 to compute R is $\mathcal{O}(\mathtt{nnz}\, A)$. By Lemma 3.4, the running time for Step 2 is $\mathcal{O}(\mathtt{nnz}(A) \cdot \log n)$. Furthermore, as stated in Lemma 3.4 because $\kappa(AR^{-1}) = 2d^2$, with probability $1 - o(1)$, the actual sampling complexity is $\mathcal{O}\left(\mu d^3 \log\left(\mu/\epsilon\right)/\epsilon^2\right)$, where $\mu = \tau/(1-\tau)$, and it takes $\phi\left(\mathcal{O}\left(\mu d^3 \log\left(\mu/\epsilon\right)/\epsilon^2\right), d\right)$ time to solve the subproblem in Step 3. This implies the overall running time of Algorithm 8 as claimed.

B

PROOFS OF RESULTS IN CHAPTER 4

B.1 PROOF OF THEOREM 4.2

The proof of this theorem is structured as follows. First we reformulate the problem using Lemma 4.1. Second we show that the sequence of solution vector estimates $\{x_t\}_{t=1}^T$ in Algorithm 9 is equivalent to the solution vector estimates $\{y_t\}_{t=1}^T$ obtained by running SGD on the equivalent problem. Third, we analyze the convergence rate of $\{y_t\}_{t=1}^T$ and conclude the error bound analysis.

PROBLEM REFORMULATION Suppose U is an ℓ_p well-conditioned basis for the range space of A and $A = UR$ for some nonsingular matrix R. Let P be the distribution based on the estimation of the corresponding leverage scores. That is, for $i \in [n]$,

$$p_i = \frac{\lambda_i}{\sum_{j=1}^n \lambda_j},$$

where λ_i is an estimation of $\|U_i\|_p^p$ satisfying

$$(1-\gamma)\|U_{(i)}\|_p^p \leq \lambda_i \leq (1+\gamma)\|U_{(i)}\|_p^p.$$

This implies

$$\frac{1-\gamma}{1+\gamma}\frac{\|U_{(i)}\|_p^p}{|U|_p^p} \leq p_i \leq \frac{1+\gamma}{1-\gamma}\frac{\|U_{(i)}\|_p^p}{|U|_p^p}. \tag{B.1}$$

From Lemma 4.1, recall that for any nonsingular matrix $F \in \mathbb{R}^{(d+1)\times(d+1)}$, the constrained ℓ_p regression problem

$$\min_{x\in\mathcal{Z}} f(x) := \|Ax-b\|_p^p \tag{B.2}$$

can be equivalently written as the stochastic optimization problem

$$\min_{y\in\mathcal{Y}} h(y) = \|URFy-b\|_p^p = \mathbb{E}_{\xi\sim P}\left[|U_{(\xi)}RFy - b_\xi|^p/p_\xi\right]. \tag{B.3}$$

Note that by comparing to the objective function defined in (4.1) with $f(x) = \|Ax-b\|_p$, we rewrite $f(x)$ in the form of the sum of subfunctions, i.e., $f(x) = \|Ax-b\|_p^p$, so that SGD can be applied.

EQUIVALENCE OF SEQUENCES By using the following linear transformation, one notices that the sequence $\{x_t\}_{t=1}^T$ obtained by (4.8) in Algorithm 9 has a one-to-one correspondence to the sequence $\{y_t\}_{t=1}^T$ obtained by running SGD on problem (B.3):

$$\begin{aligned} Fy_t &= x_t, \\ F\bar{y} &= \bar{x}, \\ Fy^* &= x^*. \end{aligned} \tag{B.4}$$

Thus with condition (B.4), immediately the objective function value has the following equivalence as well:

$$\begin{aligned} h(y_t) &= f(x_t), \\ h(\bar{y}) &= f(\bar{x}), \\ h(y^*) &= f(x^*), \end{aligned} \tag{B.5}$$

where $\bar{x} = \frac{1}{T}\sum_{i=1}^{T} x_t$, $\bar{y} = \frac{1}{T}\sum_{i=1}^{T} y_t$, and x^* and y^* are the optimal points to optimization problem (B.2) and (B.3) respectively.

Now we prove (B.4) by induction. By defining $Fy_0 = x_0$, one immediately shows that the equivalence condition holds at the base case ($t = 0$). Now as induction hypothesis, assume (B.4) holds for case $t = k$. Now for $t = k+1$, we show that x_{k+1} returned by Algorithm 9 and y_{k+1} returned by the update rule of SGD satisfy (B.4).

For simplicity, assume that at k-th iteration, the i-th row is picked. For subfunction $h_k(y) = |U_{(i)}RFy|^p - b_i/p_i$, its (sub)gradient is

$$g_k(y) = p \cdot \operatorname{sgn}(U_{(i)}RFy - b_i) \cdot (U_{(i)}RFy - b_i)^{p-1} \cdot U_{(i)}RF/p_i,$$

for which the SGD update rule becomes

$$y_{k+1} = \arg\min_{y \in \mathcal{Y}} \eta \langle y - y_k, c_k U_{(i)}RF \rangle + \frac{1}{2}\|y_k - y\|_2^2, \tag{B.6}$$

where $c_k = p \cdot \operatorname{sgn}(U_{(i)}RFy - b_i) \cdot (U_{(i)}RFy - b_i)^{p-1}/p_i$ is the corresponding (sub)gradient. Recall that with the linear transformation $Fy_k = x_k$, feasible set $\mathcal{Y} = \{y \in \mathbb{R}^k | y = F^{-1}x, x \in \mathcal{Z}\}$ and input matrix $A_i = U_{(i)}R$, the update rule (B.6) becomes

$$x_{k+1} = \arg\min_{x \in \mathcal{Z}} \eta c_k A_{(i)} x + \frac{1}{2}\|F^{-1}(x_k - x)\|_2^2. \tag{B.7}$$

This equation is exactly the update performed in (4.8). In particular, when $\mathcal{Z} = \mathbb{R}^d$, i.e., in the unconstrained case, (B.7) has a closed-form solution as shown in (4.8). From the above analysis on the equivalence between (B.6) and (B.7), one notices x_{k+1} and y_{k+1} satisfy the relationship defined in (B.4), i.e., the induction hypothesis holds at $t = k+1$.

By induction, we just showed that condition (B.4), and therefore condition (B.5), hold for any t.

CONVERGENCE RATE Based on the equivalence condition in (B.5), it is sufficient to analyze the performance of sequence $\{y_t\}_{t=1}^T$. When $p = 1$, the objective function is non-differentiable. Thus by substituting the subgradient of an ℓ_1 objective function into the update in (B.6), one notices that the SA method simply reduces to stochastic subgradient descent. We now analyze the convergence rate of running stochastic subgradient descent on problem (B.3) with $p = 1$.

Suppose the i-th row is picked at the t-th iteration. Recall that the (sub)gradient of the sample objective $|U_{(i)}RFy - b_i|/p_i$ in (B.6) is expressed as

$$g_t(y) = \operatorname{sgn}(U_{(i)}RFy - b_i) \cdot U_{(i)}RF/p_i.$$

Hence, by inequality (B.1), the norm of $g_t(y)$ is upper-bounded as follows:

$$\begin{aligned}\|g_t(y)\|_1 &= \|U_{(i)}RF \cdot \operatorname{sgn}(U_{(i)}RFy - b_i)\|_1 / p_i \\ &\leq |RF|_1 \|U_i\|_1 \frac{1+\gamma}{1-\gamma} \cdot \frac{|U|_1}{\|U_{(i)}\|_1} \leq \alpha |RF|_1 \frac{1+\gamma}{1-\gamma}.\end{aligned}$$

Here, we use the property of the well-conditioned basis U. Furthermore by Proposition 17 in [Yan+16b] and the equivalence condition in (B.5), for $H = \left(FF^T\right)^{-1}$ we have

$$\begin{aligned}\mathbb{E}\left[f(\bar{x})\right] - f(x^*) &= \mathbb{E}\left[h(\bar{y})\right] - h(y^*) \\ &\leq \frac{1}{2\eta(T+1)}\|y^* - y_0\|_2^2 + \frac{\eta}{2}\left(\alpha |RF|_1 \frac{1+\gamma}{1-\gamma}\right)^2 \\ &= \frac{1}{2\eta(T+1)}\|x^* - x_0\|_H^2 + \frac{\eta}{2}\left(\alpha |RF|_1 \frac{1+\gamma}{1-\gamma}\right)^2.\end{aligned}$$

In particular, when the step-size is

$$\eta = \frac{\|y^* - y_0\|_2}{\alpha |RF|_1 \sqrt{T+1}} \frac{1-\gamma}{1+\gamma},$$

the expected error bound is given by

$$\mathbb{E}\left[h(\bar{y})\right] - h(y^*) \leq \alpha |RF|_1 \frac{\|y^* - y_0\|_2}{\sqrt{T+1}} \frac{1+\gamma}{1-\gamma}. \tag{B.8}$$

By simple algebraic manipulations, we have

$$\begin{aligned}\frac{1}{\sqrt{d}}\|y^*\|_2 &\leq \|y^*\|_\infty = \|(RF)^{-1}RFy^*\|_\infty \leq |(RF)^{-1}|_1 \|RFy^*\|_\infty \\ &\leq \beta |(RF)^{-1}|_1 \|URFy^*\|_1 = c_3 \beta |(RF)^{-1}|_1 h(y^*),\end{aligned}$$

where $c_3 = \|URFy^*\|_1 / h(y^*)$ and we use the property of the well-conditioned basis U.

Furthermore from inequality (B.8) and the equivalence condition in (B.5), the expected relative error bound can be upper-bounded by

$$\begin{aligned}\frac{\mathbb{E}\left[f(\bar{x})\right] - f(x^*)}{f(x^*)} &= \frac{\mathbb{E}\left[h(\bar{y})\right] - h(y^*)}{h(y^*)} \\ &\leq \frac{c_3\sqrt{d}\beta |(RF)^{-1}|_1}{\|y^*\|_2}\left(\alpha |RF|_1 \frac{\|y^* - y_0\|_2}{\sqrt{T+1}} \frac{1+\gamma}{1-\gamma}\right) \\ &\leq |RF|_1 |(RF)^{-1}|_1 \frac{\|y^* - y_0\|_2}{\|y^*\|_2}\left(\frac{c_3\sqrt{d}\alpha\beta}{\sqrt{T+1}} \frac{1+\gamma}{1-\gamma}\right).\end{aligned}$$

Since the right-hand side of the last inequality is a function of stopping time $T > 0$, for any arbitrarily given error bound threshold $\epsilon > 0$, by setting the right-hand side to be ϵ, one obtains the following stopping condition:

$$\frac{\sqrt{d}\alpha\beta}{\sqrt{T+1}} = \frac{\epsilon}{c_1 c_3 \sqrt{c_2} |RF|_1 |(RF)^{-1}|_1},$$

where the constants are given by

$$c_1 = \frac{1+\gamma}{1-\gamma},\ c_2 = \frac{\|x_0 - x^*\|_H^2}{\|x^*\|_H^2} = \frac{\|y_0 - y^*\|_2^2}{\|y^*\|_2^2}.$$

Rearranging the above terms we know that after

$$T \geq \frac{d\alpha^2\beta^2 c_1^2 c_2 c_3^2}{\epsilon^2}|RF|_1^2|(RF)^{-1}|_1^2$$

iterations, the relative expected error is upper-bounded by $\epsilon > 0$, i.e.,

$$\frac{\mathbb{E}\left[f(\bar{x})\right] - f(x^*)}{f(x^*)} \leq \epsilon.$$

B.2 PROOF OF THEOREM 4.3

Similar to the proof of Theorem 4.2, the proof of this theorem is split into three parts: **Problem reformulation**, **Equivalence of sequences** and **Convergence rates**. From the proof of Theorem 4.2, one notices that the proofs in **Problem reformulation** and **Equivalence of sequences** hold for general p, and thus the proofs hold for the case when $p = 2$ as well. Now we proceed to the proof of the convergence rate. Again by the equivalence condition, we can show the convergence rate of solution vector estimate $\{x_t\}$ by showing the convergence rate achieved by the sequence $\{y_t\}$, i.e., the convergence rate of SGD of problem (B.3) for $p = 2$.

Throughout the rest of the proof, we denote

$$f(x) = \|Ax - b\|_2^2, \quad h(y) = \|AFy - b\|_2^2.$$

Denote by $H = \left(FF^T\right)^{-1}$ the weights of the ellipsoidal norm. Also recall that when the leverage scores satisfy the error condition in (4.3), we have the condition

$$\frac{1-\gamma}{1+\gamma}\frac{\|U_{(i)}\|_2^2}{\|U\|_F^2} \leq p_i \leq \frac{1+\gamma}{1-\gamma}\frac{\|U_{(i)}\|_2^2}{\|U\|_F^2}.$$

Also, we assume that U is (α, β)-conditioned with $\bar{\kappa}_2(U) = \alpha\beta$. Based on Definition 2.4, we have

$$\begin{aligned} \alpha^2 &= \|U\|_F^2, \\ \beta^2 &= \|(U^TU)^{-1}\|_2, \end{aligned}$$

and thus

$$\bar{\kappa}_2^2(U) = \|(U^TU)^{-1}\|_2 \cdot \|U\|_F^2 = \alpha^2\beta^2.$$

Before deriving the convergence rate, we compute a few constants.

$$\mu = 2\sigma_{\min}^2(AF) = \frac{2}{\left\|((URF)^TURF)^{-1}\right\|_2^2} \geq \frac{2}{\left\|(U^TU)^{-1}\right\|_2 \cdot \|(RF)^{-1}\|_2^2} = \frac{2}{\beta^2 \cdot \|(RF)^{-1}\|_2^2}, \tag{B.9}$$

and

$$\sup_i L_i = \sup_i \frac{2\|A_{(i)}F\|_2^2}{p_i} = \sup_i \frac{2\|U_{(i)}RF\|_2^2}{p_i} \leq 2c_1\|U\|_F^2 \cdot \|RF\|_2^2 = 2c_1\alpha^2 \cdot \|RF\|_2^2, \quad \text{(B.10)}$$

and

$$\begin{aligned}
\sigma^2 = \mathbb{E}_{i\sim\mathcal{D}}\left[\|g_i(y^*)\|^2\right] &= 4\sum_{i=1}^n (A_{(i)}Fy^* - b_i)^2\|A_{(i)}F\|^2/p_i \\
&= 4\sum_{i=1}^n (U_{(i)}RFy^* - b_i)^2\|U_{(i)}RF\|^2/p_i \\
&\leq 4c_1\|RF\|_2^2\|U\|_F^2\left(\sum_{i=1}^n (U_{(i)}RFy^* - b_i)^2\right) \\
&= 4c_1\|U\|_F^2 \cdot \|RF\|_2^2 \cdot h(y^*) \\
&= 4c_1\alpha^2 \cdot \|RF\|_2^2 \cdot h(y^*). \qquad \text{(B.11)}
\end{aligned}$$

Equipped with these constants and from Proposition 18 in [Yan+16b], we have the following error bound of the solution vector estimate $\{y_t\}_{t=1}^T$ generated by the weighted SGD algorithm:

$$\begin{aligned}
&\mathbb{E}\left[\|x_T - x^*\|_H^2\right] \\
=\ &\mathbb{E}\left[\|y_T - y^*\|_2^2\right] \\
\leq\ &\left(1 - 4\eta\sigma_{\min}^2(AF)\left(1 - \eta\sup_i \frac{2\|A_{(i)}F\|_2^2}{p_i}\right)\right)^T \|y_0 - y^*\|_2^2 \\
&+ \frac{2\eta\sum_{i=1}^n (A_iFy^* - b_i)^2\|A_{(i)}F\|_2^2/p_i}{\sigma_{\min}^2(AF)(1 - \eta\sup_i \frac{2\|A_{(i)}F\|_2^2}{p_i})} \\
=\ &\left(1 - 4\eta\sigma_{\min}^2(AF)\left(1 - \eta\sup_i \frac{2\|A_{(i)}F\|_2^2}{p_i}\right)\right)^T \|x_0 - x^*\|_H^2 \\
&+ \frac{2\eta\sum_{i=1}^n (A_{(i)}Fy^* - b_i)^2\|A_{(i)}F\|_2^2/p_i}{\sigma_{\min}^2(AF)(1 - \eta\sup_i \frac{2\|A_{(i)}F\|_2^2}{p_i})} \\
\leq\ &\left(\frac{1 - 4\eta\left(1 - 2\eta c_1\alpha^2\|RF\|_2^2\right)}{\beta^2\|(RF)^{-1}\|_2^2}\right)^T \|x_0 - x^*\|_H^2 + \frac{2c_1\eta\bar{\kappa}_2^2(U)\kappa^2(RF)h(y^*)}{1 - 2\eta c_1\alpha^2\|RF\|_2^2}.
\end{aligned}$$

Notice that the above equalities follow from the equivalence condition in (B.5).

Before moving on, we show a useful inequality:

$$c_3h(y^*) = c_3f(x^*) = \|Ax^*\|_2^2 = \|URFy^*\|_2^2 \geq \mu\|y^*\|_2^2/2. \quad \text{(B.12)}$$

Also recall the following constants defined in the statement of proposition:

$$c_1 = \frac{1+\gamma}{1-\gamma}, \quad c_2 = \frac{\|y_0 - y^*\|_2^2}{\|y^*\|_2^2} = \frac{\|x_0 - x^*\|_H^2}{\|x^*\|_H^2}, \quad c_3 = \frac{\|Ax^*\|_2^2}{f(x^*)}.$$

Now we show the first part. For an arbitrary target error $\epsilon > 0$, using (B.9), (B.10), (B.11) and setting

$$\frac{c_3\epsilon \cdot h(y^*)}{\|AF\|_2^2} \to \epsilon$$

in Corollary 19 in [Yan+16b] we have that when the step-size is set to be

$$\eta = \frac{1}{4} \frac{c_3\epsilon \cdot \sigma_{\min}^2(AF) \cdot h(y^*)/\|AF\|_2^2}{\sum_{i=1}^n (A_{(i)}Fy^* - b_i)^2 \|A_iF\|_2^2/p_i + c_3\left(\epsilon \cdot h(y^*)/\|AF\|_2^2\right)\sigma_{\min}^2(AF)\sup_i \frac{\|A_{(i)}F\|_2^2}{p_i}},$$

then after

$$\begin{aligned} & \log\left(\frac{2\|y_0 - y^*\|_2^2}{c_3\epsilon \cdot h(y^*)/(\|U\|_2^2\|RF\|_2^2)}\right) && \text{(B.13)} \\ & \cdot \left(c_1\alpha^2\beta^2\|RF\|_2^2\|(RF)^{-1}\|_2^2 + \frac{c_1\alpha^2\beta^4\|U\|_2^2\|RF\|_2^4\|(RF)^{-1}\|_2^4}{c_3\epsilon}\right) \\ \leq\ & \log\left(\frac{2\|U\|_2^2\|RF\|_2^2 \cdot \|y_0 - y^*\|_2^2}{c_3\epsilon \cdot h(y^*)}\right)\left(c_1\bar{\kappa}_2^2(U)\kappa^2(RF) + \frac{c_1\bar{\kappa}_2^2(U)\kappa^2(U)\kappa^4(RF)}{c_3\epsilon}\right) \\ \leq\ & \log\left(\frac{2c_2\kappa^2(U)\kappa^2(RF))}{\epsilon}\right)\left(c_1\bar{\kappa}_2^2(U)\kappa^2(RF)\right)\left(1 + \frac{\kappa^2(U)\kappa^2(RF)}{c_3\epsilon}\right) && \text{(B.14)} \end{aligned}$$

iterations, the sequence $\{y_t\}_{k=1}^T$ generated by running weighted SGD algorithm satisfies the error bound

$$\|y_T - y^*\|_2^2 \leq \frac{c_3\epsilon \cdot h(y^*)}{\|AF\|_2^2}.$$

Note that in (B.13) we used (B.12). From this, we have

$$\begin{aligned} \|A(x_T - x^*)\|_2^2 &= \|AFF^{-1}(x_T - x^*)\|_2^2 \\ &\leq \|AF\|_2^2 \cdot \|x_T - x^*\|_H^2 \\ &= \|AF\|_2^2 \cdot \|y_T - y^*\|_2^2 \\ &= c_3\epsilon \cdot h(y^*) \\ &= \epsilon\|Ax^*\|_2^2. \end{aligned}$$

For the second part, we show the result for general choice of F. The proof is basically the same as that of the first part except that we set

$$\frac{2\epsilon h(y^*)}{\|AF\|_2^2} \to \epsilon$$

in Corollary 19 in [Yan+16b]. The resulting step-size η and number of iterations required T become

$$\eta = \frac{1}{4} \frac{2\epsilon \cdot \sigma_{\min}^2(AF) \cdot h(y^*)/\|AF\|_2^2}{\sum_{i=1}^n (A_{(i)}Fy^* - b_i)^2 \|A_{(i)}F\|_2^2/p_i + \left(2\epsilon \cdot h(y^*)/\|AF\|_2^2\right)\sigma_{\min}^2(AF)\sup_i \frac{\|A_{(i)}F\|_2^2}{p_i}}$$

and

$$T = \log\left(\frac{c_2\kappa^2(U)\kappa^2(RF))}{\epsilon}\right)\left(c_1\bar{\kappa}_2^2(U)\kappa^2(RF)\right)\left(1+\frac{\kappa^2(U)\kappa^2(RF)}{2\epsilon}\right).$$

Setting $F = R^{-1}$ recovers the value of T shown in Theorem 4.3. The sequence $\{y_t\}_{k=1}^T$ generated by running weighted SGD algorithm satisfies the error bound

$$\|y_T - y^*\|_2^2 \leq \frac{2\epsilon h(y^*)}{\|AF\|_2^2}.$$

Note that when the problem is unconstrained, by smoothness of the objective $h(y)$ we have

$$h(y_T) - h(y^*) \leq \|AF\|_2^2 \cdot \|y_T - y^*\|_2^2 \leq 2\epsilon h(y^*).$$

Then by (B.5) we have

$$f(x_T) \leq (1+2\epsilon)f(x^*) \leq (1+2\epsilon+\epsilon^2)f(x^*).$$

This implies

$$\sqrt{f(x_T)} \leq (1+\epsilon)\sqrt{f(x^*)}.$$

This completes the proof because $\sqrt{f(x)} = \|Ax-b\|_2$.

B.3 PROOF OF THEOREM 4.4

Consider the following three events:

- $\mathcal{E}_1$: Compute a matrix R such that $U = AR^{-1}$ has condition number $\bar{\kappa}_p$, and then compute $F = R^{-1}$ and $H = \left(FF^T\right)^{-1}$.
- $\mathcal{E}_2$: Given R^{-1}, compute $\{\lambda_i\}_{i=1}^n$ as an estimation of row norms of AR^{-1} satisfying (4.3) with $\gamma = 0.5$.
- $\mathcal{E}_3$: For a given basis U with condition number $\bar{\kappa}_p(U)$ and $\{\lambda_i\}_{i=1}^n$ with approximation quality γ, pwSGD returns a solution with the desired accuracy with iterations $10T$, where T is specified in Theorem 4.2 or Theorem 4.3.

Since each preconditioning method shown in Table 2 succeeds with constant probability, $\mathcal{E}_1$ holds with a constant probability. Also, as introduced in Section 2.3.2.3, $\mathcal{E}_2$ has a constant failure probability. Finally, by Markov inequality, we know that $\mathcal{E}_3$ holds with probability at least 0.9. As setting the failure probability of $\mathcal{E}_1$ and $\mathcal{E}_2$ to be arbitrarily small will not alter the results in big-O notation, we can ensure that, with constant probability, $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3$ holds.

Conditioned on the fact that $\mathcal{E}_1 \cap \mathcal{E}_2 \cap \mathcal{E}_3$ holds, to converge to the desired solution for ℓ_1 regression, pwSGD runs in $\mathcal{O}(d\bar{\kappa}_1(U)/\epsilon^2)$ iterations. Since all the preconditioning methods in Table 2 provide $\kappa(U) = \mathcal{O}(1)$ and $\bar{\kappa}_2(U) = \mathcal{O}(\sqrt{d})$, for unconstrained ℓ_2 regression it runs in $\mathcal{O}(d\log(1/\epsilon)/\epsilon)$ iterations. For constrained ℓ_2 regression, since an ϵ-approximate solution in terms of the solution vector measured in the prediction norm implies an $\sqrt{\epsilon}$-approximate solution on the objective, it runs in $\mathcal{O}(d\log(1/\epsilon)/\epsilon^2)$ iterations to return an ϵ-solution in the objective value.

The overall complexity is the sum of the complexity needed in each of the above events. For $\mathcal{E}_1$, it is $time(R)$ because the time for computing F and H is $\mathcal{O}(d^3)$, which can absorbed

into $time(R)$ and they only have to be computed once. For $\mathcal{E}_2$, it is $\mathcal{O}(\texttt{nnz}(A) \cdot \log n)$. Finally, for $\mathcal{E}_3$, when the problem is unconstrained, $time_{update} = \mathcal{O}(d^2)$; when the problem is constrained, $time_{update} = \texttt{poly}(d)$. Combining these, we get the complexities shown in the statement.

B.4 PROOF OF THEOREM 4.7

Let G_f consist of m_f copies of g_f and $G = \bigcup_{f \in \mathcal{F}} G_f$. We may view the sampling step in Algorithm 10 as follows. Sample s items uniformly from G independently with replacement and denote the corresponding subset of samples by S. Then rescale every function in S by $M(\mathcal{F})/s$ and obtain $\mathcal{D}$.

By Theorem 4.1 in [FL11], we know that if the above intermediate set S is an $(\epsilon \cdot n/M(\mathcal{F}))$-approximation of the set G, then the resulting set $\mathcal{D}$ is a desired ϵ-coreset for $\mathcal{F}$. Indeed, S is such a set according to Theorem 6.10 in [FL11].

B.5 PROOF OF PROPOSITION 4.8

We use A to denote $\bar{A}$ for the sake of simplicity. Also define the sensitivity at row index $i \in [n]$ as

$$s_i = n \cdot \sup_{x \in \mathcal{C}} \frac{|A_{(i)}x|^p}{\sum_{j=1}^n |A_{(j)}x|^p}. \tag{B.15}$$

Suppose $U \in \mathbb{R}^{n \times k}$ is an (α, β) well-conditioned basis of the range space of A satisfying $A = UR$, where $k = \text{rank}(A)$ and $R \in \mathbb{R}^{k \times (d+1)}$. Then from (B.15), we have

$$\begin{aligned} \frac{s_i}{n} &= \sup_{x \in \mathcal{C}} \frac{|A_{(i)}x|^p}{\|Ax\|_p^p} = \sup_{x \in \mathcal{C}} \frac{|U_{(i)}Rx|^p}{\|URx\|_p^p} = \sup_{y \in \mathcal{C}'} \frac{|U_{(i)}y|^p}{\|Uy\|_p^p} \\ &\leq \sup_{y \in \mathcal{C}'} \frac{\|U_{(i)}\|_p^p \|y\|_q^p}{\|y\|_q^p / \beta^p} = \beta^p \|U_{(i)}\|_p^p = \beta^p \cdot \lambda_i, \end{aligned}$$

where $\mathcal{C}' = \{y \in \mathbb{R}^d | y = Rx, x \in \mathcal{C}\}$ is a one-to-one mapping. The first inequality follows from Hölder's inequality with $\frac{1}{p} + \frac{1}{q} = 1$ and the properties of well-conditioned bases. According to the definition of sensitivity $m(f_i) = \lfloor s_i \rfloor + 1$, the above property implies

$$m(f_i) \leq n\beta^p \lambda_i + 1,$$

which implies $M(\mathcal{F}) = \sum_{i=1}^n s_i \leq (n\beta^p \sum_{i=1}^n \lambda_i) + n = n((\alpha\beta)^p + 1)$, and completes the proof.

B.6 PROOF OF PROPOSITION 4.9

According to Definition 4.6, we only have to show that for any arbitrary constant n and set of points $G = \{a_1, \ldots, a_n\} \subseteq \mathbb{R}^d$, the following condition holds:

$$|\{\textbf{Range}(G, x, r) | x \in \mathcal{X}, r \geq 0\}| \leq n^{d+1},$$

where $\textbf{Range}(G, x, r) = \{a_i | |a_i^T x|^p \leq r\}$ is the region located in the p-norm ellipsoid $|a_i^T x|^p = r$. Since the condition $\{a_i | |a_i^T x|^p \leq r\} = \{a_i | |a_i^T x| \leq r^{\frac{1}{p}}\}$ holds and the constant r is non-negative and arbitrary. Without loss of generality, we assume $p = 1$ in the above definition, i.e., $\textbf{Range}(G, x, r) = \{a_i | |a_i^T x| \leq r\}$.

Notice that for every x and r, $\textbf{Range}(G, x, r)$ is a subset of G. Hence, we may view it as a binary classifier on G, denoted by $c_{x,r}$. Given $x \in \mathcal{X}$ and $r \geq 0$, for any $a_i \in G$ we have

$$c_{x,r}(a_i) = \begin{cases} 1, & \text{if } |a_i^T x| \leq r; \\ 0, & \text{otherwise.} \end{cases}$$

Therefore, one immediately sees that $|\{\textbf{Range}(G, x, r) | x \in \mathcal{X}, r \geq 0\}|$ is the shattering coefficient of $C := \{c_{x,r} | x \in \mathcal{X}, r \geq 0\}$ on n points, denoted by $s(C, n)$. To bound the shattering coefficient of C, we provide an upper bound based on its VC dimension.

We claim that the VC dimension of C is at most $d + 1$. By contradiction, suppose there exist $n + 2$ points such that any labeling on these $n + 2$ points can be shattered by C. By Radon's Theorem [Cla+93], we can partition these points into two disjoint subsets, namely V and W with size n_1 and n_2 respectively, where the intersection of their convex hulls is nonempty. Let b be a point located in the intersection of the convex hulls of V and W, which in general can be written as

$$b = \sum_{i=1}^{n_1} \lambda_i v_i = \sum_{i=1}^{n_2} \sigma_i w_i, \tag{B.16}$$

where $\lambda_i \geq 0$, $\sigma_i \geq 0$ and $\sum_{i=1}^{n_1} \lambda_i = \sum_{i=1}^{n_2} \sigma_i = 1$.

By the above assumption, we can find vector $x \in \mathbb{R}^n$ and nonnegative constant r such that

$$-r \leq x^T v_i \leq r, \quad i = 1, \ldots, n_1; \tag{B.17}$$

$$x^T w_i > r \text{ or } x^T w_i < -r, \quad i = 1, \ldots, n_2. \tag{B.18}$$

By combining conditions (B.16), (B.17) and (B.18), we further obtain inequalities

$$-r \leq b^T x \leq r$$

and

$$b^T x < -r \text{ or } b^T x > r,$$

which is clearly paradoxical! This confirms that the VC dimension of C is less than or equal to $d + 1$. Furthermore, by Sauer's Lemma [Sau72], for $n \geq 2$ the shattering coefficient $s(C, n) = |\{\textbf{Range}(G, x, r) | x \in \mathcal{X}, r \geq 0\}|$ is less than n^{d+1}, which completes the proof of this proposition.

B.7 PROOF OF PROPOSITION 4.10

Without loss of generality, assume the low dimension d is even (because if d is odd, we can always add an extra arbitrary row to input matrix A and upper bound the size of the original total sensitivity set by the same analysis). Let $a_i \in [0, 1]^d$ be a vector with exactly

$d/2$ elements to be 1. For each $i \in [n]$, let $B_i = \{j | a_{ij} = 1\}$, where a_{ij} denotes the j-th element of vector a_i. For fixed i, define x as,

$$x_j = \begin{cases} 2/d, & \text{if} \quad j \in B_i, \\ -d, & \text{otherwise.} \end{cases}$$

One immediately notices that $x^T a_i = 1$. Thus for $j \neq i$, $a_j \neq a_i$, there exists an index $k \in [d]$ such that $a_{jk} = 1$ but $a_{ik} = 0$. Furthermore,

$$x^\top a_j = \sum_{l=1}^{d} x_l a_{jl} = \sum_{l \in B_j, l \neq k}^{d} x_l a_{jl} + \sum_{l \neq B_j}^{d} x_l a_{jl} + x_k a_{jk} \leq (d/2 - 1)(2/d) - d < 0,$$

which further implies $f_j(x) = x^T a_j = 0$; Therefore, the i-th sensitivity becomes

$$s_i = \sup_x \frac{f_i(x)}{\sum_{i=j}^{n} f_j(x)} \geq 1.$$

Since the above condition holds for arbitrary index $i \in [n]$, and we have $\binom{d}{d/2}$ vectors a_i, i.e., $n = \binom{d}{d/2}$, this confirms that the size of the total sensitivity set is at least $\binom{d}{d/2} \approx 2^d$.

C

PROOFS OF RESULTS IN CHAPTER 5

C.1 PROOF OF LEMMA 5.6

The proof has two parts. **Part 1** is to prove the case where the subproblem is solved exactly and **Part 2** is for the case when the subproblem is solved approximately. In **Part 1**, there are also two cases corresponding to condition (**C1**) and condition (**C2**) respectively.

Throughout the proof, we use $\preceq$ to denote the partial order defined on the cone $\{B \in \mathbb{R}^{d\times d} \mid w^T B w \geq 0 \ \forall w \in \mathcal{K}\}$. With $\Delta_t := w_t - w^*$, then based on Assumptions 1 and 2, we have

$$\|\nabla^2 F(w_t) - \nabla^2 F(w^*)\|_2 \leq L\|\Delta_t\|_2, \quad \mu I \preceq H^* \preceq \nu I.$$

Therefore we can get

$$(\mu - L\|\Delta_t\|_2)I \preceq \nabla^2 F(w_t) \preceq (\nu + L\|\Delta_t\|_2)I.$$

Since $\|\Delta_t\|_2 \leq \frac{\mu}{4L}$, then

$$\frac{3}{4}\mu I \preceq \nabla^2 F(w_t) \preceq \frac{5}{4}\nu I. \tag{C.1}$$

Let

$$\Psi_t(w) := \frac{1}{2}(w - w_t)^T \tilde{H}_t (w - w_t) + (w - w_t)^T \nabla F(w_t).$$

Part 1 In this part, we consider the case where the subproblem is solved exactly in every iteration of Algorithm 11. We show the following two results:

(A) Under condition (**C1**), the error recursion (5.10) holds with factors in (5.12).

(B) Under condition (**C2**), the error recursion (5.10) holds with factors in (5.13).

If the subproblem $\min_{w\in\mathcal{C}} \Psi_t(w)$ is solved exactly in Algorithm 11, namely

$$w_{t+1} = \arg\min_{w\in\mathcal{C}} \Psi_t(w),$$

then $\Psi_t(w_{t+1}) \leq \Psi_t(w^*)$. By expanding both sides, we have

$$\frac{1}{2}\Delta_{t+1}^T \tilde{H}_t \Delta_{t+1} \leq \Delta_t^T \tilde{H}_t \Delta_{t+1} - \nabla F(w_t)^T \Delta_{t+1} + \nabla F(w^*)^T \Delta_{t+1}. \tag{C.2}$$

The third term on the right-hand side satisfies $\nabla F(w^*)^T \Delta_{t+1} \geq 0$ because of the optimality condition.

Note that

$$\begin{aligned} RHS &= \Delta_t^T \tilde{H}_t \Delta_{t+1} - \nabla F(w_t)^T \Delta_{t+1} + \nabla F(w^*)^T \Delta_{t+1} \\ &= \underbrace{\Delta_t^T \nabla^2 F(w_t)\Delta_{t+1} - (\nabla F(w_t) - \nabla F(w^*))^T \Delta_{t+1}}_{T_1} + \Delta_t^T (\tilde{H}_t - \nabla^2 F(w_t))\Delta_{t+1}, \end{aligned}$$

where,

$$\begin{aligned} T_1 &= \Delta_t^T \nabla^2 F(w_t)\Delta_{t+1} - \Delta_t^T \left(\int_0^1 \nabla^2 F(w^* + \delta(w_t - w^*))d\delta \right) \Delta_{t+1} \\ &\leq \Delta_t^T \left(\int_0^1 \left(\nabla^2 F(w^* + \delta(w_t - w^*)) - \nabla^2 F(w_t) \right) d\delta \right) \Delta_{t+1} \\ &\leq \|\Delta_t\|_2 \cdot \int_0^1 L\delta \|\Delta_t\|_2 d\delta \cdot \|\Delta_{t+1}\| \\ &\leq \frac{L}{2} \cdot \|\Delta_t\|^2 \cdot \|\Delta_{t+1}\|. \end{aligned}$$

Substituting it into T_1 and rewriting (C.2), we get

$$1 \leq \frac{L \cdot \|\Delta_t\|^2 \cdot \|\Delta_{t+1}\|}{\Delta_{t+1}^T \tilde{H}_t \Delta_{t+1}} + 2\frac{\Delta_t^T (\tilde{H}_t - \nabla^2 F(w_t))\Delta_{t+1}}{\Delta_{t+1}^T \tilde{H}_t \Delta_{t+1}}. \tag{C.3}$$

(A) Under condition (C1), we have

$$\begin{aligned} \lambda_{\min}^{\mathcal{K}}(\tilde{H}_t) &\geq \lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t)) - \epsilon \cdot \lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w_t)) \\ &= (1 - \epsilon\kappa(\nabla^2 F(w_t))) \cdot \lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t)) \\ &\geq (1 - 2\epsilon\kappa) \cdot \lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t)). \end{aligned}$$

Then

$$\Delta_t^T (\tilde{H}_t - \nabla^2 F(w_t))\Delta_{t+1} \geq [\lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t)) - \epsilon \cdot \lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w_t))] \cdot \|\Delta_{t+1}\|^2.$$

Therefore

$$\begin{aligned} 1 \quad \leq \quad & \frac{L \cdot \|\Delta_t\|^2 \cdot \|\Delta_{t+1}\|}{[\lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t)) - \epsilon \cdot \lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w_t))] \cdot \|\Delta_{t+1}\|^2} \\ & + 2\frac{\epsilon \cdot \lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w_t)) \cdot \|\Delta_t\| \cdot \|\Delta_{t+1}\|}{[\lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t)) - \epsilon \cdot \lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w_t))] \cdot \|\Delta_{t+1}\|^2}. \end{aligned}$$

Reorganizing it and combining (C.1), we get

$$\begin{aligned} \|\Delta_{t+1}\| &\leq \frac{L}{3\mu/4 - 5\epsilon\nu/4} \cdot \|\Delta_t\|^2 + \frac{5\epsilon\nu/2}{3\mu/4 - 5\epsilon\nu/4} \cdot \|\Delta_t\| \\ &\leq \frac{2L}{(1 - 2\kappa\epsilon)\mu} \cdot \|\Delta_t\|^2 + \frac{4\kappa\epsilon}{1 - 2\kappa\epsilon} \cdot \|\Delta_t\|. \end{aligned}$$

(B) Under condition (C2), we have

$$\begin{aligned} \Delta_{t+1}^T \tilde{H}_t \Delta_{t+1} \quad &\geq \quad \Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1} - \epsilon \cdot \Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1} \\ &= \quad (1 - \epsilon) \cdot \Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1} \end{aligned}$$

and

$$\Delta_t^T (\tilde{H}_t - \nabla^2 F(w_t))\Delta_{t+1} \leq \epsilon \cdot \sqrt{\Delta_t^T \nabla^2 F(w_t)\Delta_t} \cdot \sqrt{\Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1}}.$$

Substituting into (C.3) gives

$$\begin{aligned}
1 &\leq \frac{L \cdot \|\Delta_t\|^2 \cdot \|\Delta_{t+1}\|}{(1-\epsilon)\cdot \Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1}} + 2\frac{\epsilon \cdot \sqrt{\Delta_t^T \nabla^2 F(w_t)\Delta_t}\cdot \sqrt{\Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1}}}{(1-\epsilon)\cdot \Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1}} \\
&\leq \frac{L \cdot \|\Delta_t\|^2 \cdot \|\Delta_{t+1}\|}{(1-\epsilon)\cdot \lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t))\cdot \|\Delta_{t+1}\|^2} + \frac{2\epsilon \cdot \sqrt{\Delta_t^T \nabla^2 F(w_t)\Delta_t}}{(1-\epsilon)\cdot \sqrt{\Delta_{t+1}^T \nabla^2 F(w_t)\Delta_{t+1}}} \\
&\leq \frac{L}{(1-\epsilon)\cdot \lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t))} \cdot \frac{\|\Delta_t\|^2}{\|\Delta_{t+1}\|} + \frac{2\epsilon}{1-\epsilon}\cdot \sqrt{\frac{\lambda_{\max}^{\mathcal{K}}(\nabla^2 F(w_t))\cdot \|\Delta_t\|^2}{\lambda_{\min}^{\mathcal{K}}(\nabla^2 F(w_t))\cdot \|\Delta_{t+1}\|^2}}.
\end{aligned}$$

Reorganizing and combining (C.1) gives

$$\begin{aligned}
\|\Delta_{t+1}\| &\leq \frac{L}{(1-\epsilon)\cdot 3\mu/4}\cdot \|\Delta_t\|^2 + \frac{2\epsilon}{1-\epsilon}\cdot \sqrt{\frac{5\nu/4}{3\mu/4}}\cdot \|\Delta_t\| \\
&\leq \frac{2L}{(1-\epsilon)\mu}\cdot \|\Delta_t\|^2 + \frac{3\epsilon}{1-\epsilon}\cdot \sqrt{\kappa}\cdot \|\Delta_t\|.
\end{aligned}$$

Part 2 Now we prove the case where the subproblem is solved approximately. We want to show that under condition (5.9), the error recursion (5.11) holds with C_l, C_q being the same as in the case where the problem is solved exactly.

Consider iteration t in Algorithm 11. First, $w_{t+1}^* = \operatorname{argmin}_{w\in\mathcal{C}} \Psi_t(w)$ (note that w_{t+1} is not the minimizer any more). Based on the proof in **Part 1**, we have

$$\|w_{t+1}^* - w^*\| \leq C_q \cdot \|w_t - w^*\|^2 + C_l \cdot \|w_t - w^*\|.$$

Then

$$\begin{aligned}
\|w_{t+1} - w^*\| &\leq \|w_{t+1} - w_{t+1}^*\| + \|w_{t+1}^* - w^*\| \\
&\leq \epsilon_0 \cdot \|w_{t+1}^* - w_t\| + \|w_{t+1}^* - w^*\| \\
&\leq \epsilon_0 \cdot \|w_{t+1}^* - w^*\| + \epsilon_0 \cdot \|w_t - w^*\| + \|w_{t+1}^* - w^*\| \\
&= (1+\epsilon_0)\cdot \|w_{t+1}^* - w^*\| + \epsilon_0 \cdot \|w_t - w^*\| \\
&\leq (1+\epsilon_0)C_q \cdot \|w_t - w^*\|^2 + (\epsilon_0 + (1+\epsilon_0)C_l)\cdot \|w_t - w^*\|.
\end{aligned}$$

C.2 PROOF OF THEOREM 5.7

From Assumption 3, it takes $\mathcal{O}(\mathtt{nnz}(A))$ time to construct the augmented matrix $A \in \mathbb{R}^{nk\times d}$. Here we apply a variant of the algorithm in [Dri+12], which uses the sparse subspace embedding [Coh16] as the underlying sketching method. One can show that, with high probability, it takes $\mathcal{O}(\mathtt{nnz}(A))d\log(nk+d)) = \mathcal{O}(\mathtt{nnz}(A)\log n)$ time to compute a set of approximate leverage scores with constant approximation factor, under the assumption $n \geq d^3 \log d$. This completes the proof.

C.3 PROOF OF THEOREM 5.8

Before we show the proof of Theorem 5.8, we present our main results for the new proposed leverage scores, namely block partial leverage scores, for approximating a matrix of the form $A^TA + Q$.

Theorem C.1. *Given $A \in \mathbb{R}^{N\times d}$ with n blocks, $Q \in \mathbb{R}^{d\times d}$ satisfying $Q \succeq 0$ and $\epsilon \in (0,1)$, let $\{\tau_i^Q(A)\}_{i=1}^n$ be its block partial leverage scores. Let $p_i = \min\{1, s\cdot \frac{\tilde{\tau}_i}{\sum_{j=1}^n \tilde{\tau}_j}\}$ with $\tilde{\tau}_i \geq \tau_i^Q(A)$. Construct SA by sampling the i-th block of A with probability p_i and rescaling it by $1/\sqrt{p_i}$. Then if*

$$s \geq 2\left(\sum_{i=1}^n \tilde{\tau}_i\right)\cdot\left(\|U^TDU\| + \frac{\epsilon}{3}\right)\cdot \log\left(\frac{4d}{\delta}\right)\cdot\frac{1}{\epsilon^2}, \tag{C.4}$$

where $\mathcal{D}$ is a diagonal matrix with $\mathcal{D}_{ii} = \tau_i^Q(A)/\tilde{\tau}_i$, and U is a matrix satisfying $U^TU \preceq I$, with probability at least $1-\delta$, we have

$$-\epsilon(A^TA+Q) \preceq A^TS^TSA - A^TA \preceq \epsilon(A^TA+Q). \tag{C.5}$$

Proof. Denote $\bar{A} = \begin{pmatrix} A \\ Q^{\frac{1}{2}}\end{pmatrix}$. Let $\bar{A} = \bar{U}R$ where $\bar{U}$ has orthonormal columns. Then define $U = AR^{-1}$ and $U_i = A_iR^{-1}$ for $i = 1,\ldots,n$. By definition, the true partial leverage scores $\tau_i^Q(A)$'s are defined as $\tau_i = tr(U_iU_i^T)$. For simplicity, we use τ_i to denote $\tau_i^Q(A)$.

In the following we bound $\|U^TS^TSU - U^TU\| \leq \epsilon$ with high probability. For $i = 1,\ldots,n$, define

$$X_j = \begin{cases} (\frac{1}{p_i}-1)U_i^TU_i & \text{with probability } p_i; \\ -U_i^TU_i & \text{with probability } 1-p_i. \end{cases}$$

Also define $Y = \sum_{i=1}^n X_i$. We have $\mathbb{E}[X_i] = 0$. In the following we bound $\|Y\|$ using the matrix Bernstein bound.

First, we bound $\|X_i\|$ for $i = 1,\ldots,n$. Let $\mathcal{I} = \{i | p_i < 1\}$. If $i \in \mathcal{I}^c$, then $X_i = 0$ and $\|X_i\| = 0$. Thus we only consider the case where $i \in \mathcal{I}$. In this case $p_i = \frac{s\tilde{\tau}_i}{\sum_j \tilde{\tau}_j}$. Since $p_i < 1$, we have

$$-U_i^TU_i/p_i \preceq X_i \preceq U_i^TU_i/p_i.$$

Moreover,

$$\begin{aligned} \|\mathcal{X}_i\| &\leq \|U_i^TU_i\|/p_i = \|U_i\|^2/p_i \\ &\leq \frac{\left(\sum_{j=1}^n \tilde{\tau}_j\right)\|U_i\|^2}{s\tilde{\tau}_i} = \frac{\left(\sum_{j=1}^n \tilde{\tau}_j\right)\|U_i\|^2}{s\tau_i}\frac{\tau_i}{\tilde{\tau}_i} \\ &\leq \frac{\left(\sum_{j=1}^n \tilde{\tau}_j\right)}{s}\frac{\tau_i}{\tilde{\tau}_i} \leq \frac{\left(\sum_{j=1}^n \tilde{\tau}_j\right)}{s}. \end{aligned} \tag{C.6}$$

The last inequality comes from the fact that $\tilde{\tau}_i \geq \tau_i = tr(U_iU_i^T) \geq \|U_i\|^2$.

Next, we bound $\mathbb{E}\left[Y^2\right] = \sum_{i=1}^n \mathbb{E}\left[X_i^2\right]$. We have

$$\begin{aligned}
\sum_{i=1}^n \mathbb{E}\left[X_i^2\right] &= \sum_{i\in\mathcal{I}} \mathbb{E}\left[X_i^2\right] = \sum_{i=1}^n (p_i \cdot (\frac{1}{p_i} - 1)^2 + (1-p_i)) A_i^T A_i A_i^T A_i \\
&= \sum_{i\in\mathcal{I}} (\frac{1}{p_i} - 1) U_i^T U_i U_i^T U_i \preceq \sum_{i\in\mathcal{I}} \frac{1}{p_i} U_i^T U_i U_i^T U_i \\
&\preceq \sum_{i\in\mathcal{I}} \frac{\sum_{j=1}^n \tilde{\tau}_j}{s} \cdot \frac{\tau_i}{\tilde{\tau}_i} \cdot U_i^T U_i \qquad \text{(C.7)} \\
&\preceq \frac{\sum_{j=1}^n \tilde{\tau}_j}{s} \cdot U^T \mathcal{D} U,
\end{aligned}$$

where $\mathcal{D}$ is a diagonal matrix with $\mathcal{D}_{ii} = \tau_i/\tilde{\tau}_i$. Above, (C.7) holds since by (C.6) we have $U_i^T U_i / p_i \preceq \left(\sum_{j=1}^n \tilde{\tau}_j\right) \cdot \frac{\tau_i}{\tilde{\tau}_i} \frac{1}{s} \cdot I$. Therefore,

$$\|\mathbb{E}\left[Y^2\right]\| \leq \frac{\sum_{j=1}^n \tilde{\tau}_j}{s} \cdot \|U^T D U\|.$$

Since U consists of a subset of rows of $\bar{U}$, one can show that $U^T U \preceq \bar{U}^T \bar{U} = I$.

Given these, by the matrix Bernstein bound [Tro15], we have when

$$s \geq 2\left(\sum_{j=1}^n \tilde{\tau}_j\right)\left(\|U^T D U\| + \frac{\epsilon}{3}\right) \cdot \log\left(\frac{4d}{\delta}\right) \cdot \frac{1}{\epsilon^2},$$

with probability at least $1-\delta$,

$$\|Y\| \leq \epsilon$$

holds. With this one can show that

$$-\epsilon I \preceq U^T S^T S U - U^T U \preceq \epsilon I.$$

Furthermore, we have

$$-\epsilon R^T R \preceq R U^T S^T S U R - R^T U^T U R \preceq \epsilon R^T R.$$

Therefore,

$$-\epsilon \bar{A}^T \bar{A} \preceq A^T S^T S A - A^T A \preceq \epsilon \bar{A}^T \bar{A}.$$

□

Remark 11. *In (C.4), since each element of D is no greater than 1 and $U^T U \preceq I$, we have*

$$U^T \mathcal{D} U \preceq U^T U \preceq I.$$

Hence the sampling size

$$s = 4\left(\sum_{i=1}^n \tilde{\tau}_i\right) \cdot \log\left(\frac{4d}{\delta}\right) \cdot \frac{1}{\epsilon^2}$$

is sufficient to yield (C.5).

Based on Lemma C.2 (stated below), we convert condition (**C2**) to a standard matrix product approximation guarantee. A direct corollary of Theorem C.1 completes the proof.

Lemma C.2. *Given $A \in \mathbb{R}^{N\times d}$ with n blocks, $Q \in \mathbb{R}^{d\times d}$ satisfying $Q \succeq 0$ and $\epsilon \in (0,1)$, and a sampling matrix $S \in \mathbb{R}^{s\times n}$, the following two conditions are equivalent:*

(A)
$$-\epsilon(A^TA + Q) \preceq A^TS^TSA - A^TA \preceq \epsilon(A^TA + Q). \tag{C.8}$$

(B)
$$|x^T(A^TS^TSA - A^TA)y| \le \epsilon \cdot \sqrt{\|Ax\|_2^2 + x^TQx} \cdot \sqrt{\|Ay\|_2^2 + y^TQy}, \ \forall x, y \in \mathbb{R}^d. \tag{C.9}$$

Proof. First, it is straightforward to see (B) $\Rightarrow$ (A) by setting $x = y$ in (C.9). So now we prove the other direction.

Denote $\bar{A} = \begin{pmatrix} A \\ Q^{\frac{1}{2}} \end{pmatrix}$. Let $\bar{A} = \bar{U}R$ where $\bar{U}$ has orthonormal columns. Then define $U = AR^{-1}$ and $U_i = A_iR^{-1}$ for $i = 1, \dots, n$. Then (C.8) is equivalent to

$$-\epsilon\bar{A}^T\bar{A} \preceq A^TS^TSA - A^TA \preceq \epsilon\bar{A}^T\bar{A}.$$

Since R^{-1} is a full-rank matrix,

$$-\epsilon R^{-T}\bar{A}^T\bar{A}R^{-1} \preceq R^{-T}(A^TS^TSA - A^TA)R^{-1} \preceq \epsilon R^{-T}\bar{A}^T\bar{A}R^{-1}.$$

That is,

$$-\epsilon I = -\epsilon\bar{U}^T\bar{U} \preceq U^TS^TSU - U^TU \preceq \epsilon\bar{U}^T\bar{U} = \epsilon I. \quad [\bar{U} \text{ is orthonormal bases}]$$

Therefore

$$\|U^TS^TSU - U^TU\| \le \epsilon.$$

Now consider

$$\begin{aligned}
|x^T(A^TS^TSA - A^TA)y| &= |x^TR^T(U^TS^TSU - U^TU)Ry| \\
&\le \|Rx\|_2 \cdot \|U^TS^TSU - U^TU\|_2 \cdot \|Ry\|_2 \\
&\le \epsilon \cdot \|\bar{U}Rx\|_2 \cdot \|\bar{U}Ry\|_2 \quad [\bar{U} \text{ is orthonormal bases}] \\
&= \epsilon \cdot \|\bar{A}x\|_2 \cdot \|\bar{A}y\|_2 \\
&= \epsilon \cdot \sqrt{\|Ax\|_2^2 + x^TQx} \cdot \sqrt{\|Ay\|_2^2 + y^TQy}.
\end{aligned}$$

□

C.4 PROOF OF COROLLARY 5.12

According to Lemma 5.6, we have the error recursion

$$\begin{aligned}\|w_{t+1}-w^*\| &\leq (1+\epsilon_0)C_q\cdot\|w_t-w^*\|^2+(\epsilon_0+(1+\epsilon_0)C_l)\cdot\|w_t-w^*\|\\ &\leq \left[(1+\epsilon_0)C_q\frac{\mu}{4L}+\epsilon_0+(1+\epsilon_0)C_l\right]\cdot\|w_t-w^*\|.\end{aligned}$$

If leverage scores sampling is used, then

$$C_q=\frac{2L}{(1-\epsilon)\mu},\quad C_l=\frac{3\epsilon\sqrt{\kappa}}{1-\epsilon}.$$

Therefore

$$\begin{aligned}\|w_{t+1}-w^*\| &\leq \left[(1+\epsilon_0)C_q\frac{\mu}{4L}+\epsilon_0+(1+\epsilon_0)C_l\right]\cdot\|w_t-w^*\|\\ &=\left[\frac{1+\epsilon_0}{2(1-\epsilon)}+\epsilon_0+(1+\epsilon_0)\frac{3\epsilon\sqrt{\kappa}}{1-\epsilon}\right]\cdot\|w_t-w^*\|\\ &=\rho\cdot\|w_t-w^*\|.\end{aligned}$$

Now by choosing $\epsilon\leq\min\left\{\frac{1}{10\sqrt{\kappa}},0.1\right\}$ and $\epsilon_0\leq 0.01$, we can get $\rho<0.9$.

Since we use CG, in order to achieve $\epsilon_0\leq 0.01$, we need $\tilde{\mathcal{O}}(sd\sqrt{\kappa})$ according to Table 12 (note that since we are in the local region, $\tilde{\kappa_t}=\Theta(\kappa)$). Meanwhile, since $\epsilon\leq\min\left\{\frac{1}{10\sqrt{\kappa}},0.1\right\}$, then $s=\tilde{\mathcal{O}}(d/\epsilon^2)=\tilde{\mathcal{O}}(d\kappa)$.

Therefore, the complexity per iteration is

$$t_{const}+t_{solve}=\tilde{\mathcal{O}}(\mathsf{nnz}(A))+\tilde{\mathcal{O}}(d^2\kappa^{3/2})=\tilde{\mathcal{O}}(\mathsf{nnz}(A)+d^2\kappa^{3/2}).$$

Similarly, if block norm squares sampling is used, then

$$C_q=\frac{2L}{(1-2\epsilon\kappa)\mu},\quad C_l=\frac{4\epsilon\kappa}{1-2\epsilon\kappa}.$$

Therefore

$$\begin{aligned}\|w_{t+1}-w^*\| &\leq \left[(1+\epsilon_0)C_q\frac{\mu}{4L}+\epsilon_0+(1+\epsilon_0)C_l\right]\cdot\|w_t-w^*\|\\ &=\left[\frac{1+\epsilon_0}{2(1-2\epsilon\kappa)}+\epsilon_0+(1+\epsilon_0)\frac{4\epsilon\kappa}{1-2\epsilon\kappa}\right]\cdot\|w_t-w^*\|\\ &=\rho\cdot\|w_t-w^*\|.\end{aligned}$$

Now by choosing $\epsilon\leq\min\left\{\frac{1}{10\kappa},0.1\right\}$ and $\epsilon_0\leq 0.01$, we can get $\rho<0.9$. As in the case where leverage scores sampling is used, we get the total complexity per iteration as

$$t_{const}+t_{solve}=\tilde{\mathcal{O}}(\mathsf{nnz}(A))+\tilde{\mathcal{O}}(\mathbf{sr}(A)d\kappa^{5/2})=\tilde{\mathcal{O}}(\mathsf{nnz}(A)+\mathbf{sr}(A)d\kappa^{5/2}).$$

C.5 PROOF OF THEOREM 5.13

For $j = 1, \ldots, s$, define

$$X_j = \frac{1}{s}(nA_i^T A_i - A^T A) \text{ with probability } 1/n, \forall i.$$

Let $Y = \sum_{j=1}^{s} X_j = A^T S^T S A - A^T A$. Then $\mathbb{E}\left[X_j\right] = 0$. In the following we bound $\|Y\|$ through matrix Bernstein inequality. For convenience, we denote $K_t := \max_i \|A_i\|^2$.

First,

$$\|X_j\| = \frac{1}{s}\|nA_i^T A_i - A^T A\| \leq 2nK_t/s,$$

and

$$\begin{aligned}
\mathbb{E}\left[X_j^2\right] &= \frac{1}{ns^2}\sum_{i=1}^{n}(nA_i^T A_i - A^T A)^2 \\
&= \frac{1}{ns^2}\sum_{i=1}^{n}\left[n^2(A_i^T A_i)^2 + (A^T A)^2 - 2nA_i^T A_i A^T A\right] \\
&= \frac{1}{s^2}(A^T A)^2 + \frac{1}{s^2}\sum_{i=1}^{n} n(A_i^T A_i)^2 - \frac{2}{s^2}(A^T A)^2 \\
&\preceq \frac{1}{s^2}\sum_{i=1}^{n} n(A_i^T A_i)^2 \\
&\preceq \frac{1}{s^2} nK_t A^T A.
\end{aligned}$$

Therefore,

$$\|\mathbb{E}\left[Y^2\right]\| \leq \|\frac{1}{s} nK_t A^T A\| = \frac{1}{s} nK_t \|A\|^2.$$

By the matrix Bernstein bound [Tro15], we have when

$$s \geq 4nK_t(\|A\|^2 + \frac{\epsilon_0}{3}) \cdot \log(\frac{d}{\delta}) \cdot \frac{1}{\epsilon_0^2},$$

with probability at least $1 - \delta$,

$$\|Y\| \leq \epsilon.$$

Now by setting $\epsilon_0 = \|A\|^2\epsilon$, where $\epsilon \in (0, 1)$, we have when

$$s \geq 4\frac{nK_t}{\|A\|^2} \cdot \log(\frac{d}{\delta}) \cdot \frac{1}{\epsilon^2} = 4n\frac{\max_i \|A_i\|^2}{\|A\|^2} \cdot \log(\frac{d}{\delta}) \cdot \frac{1}{\epsilon^2},$$

with probability at least $1 - \delta$, $\|A^T S^T S A - A^T A\| \leq \epsilon \cdot \|A^T A\|$ holds. Since $Q \succeq 0$, $\|A^T A\| \leq \|A^T A + Q\|$. This implies condition (**C1**).

D

PROOFS OF RESULTS IN CHAPTER 8

D.1 PROOF OF PROPOSITION 8.5

Recall, for any $t \in \mathbb{R}^d$, for $\Phi^{-1}(t)$, we mean $\left(\Phi_1^{-1}(t_1), \ldots, \Phi_d^{-1}(t_d)\right) \in \mathbb{R}^d$, where $\Phi_j(\cdot)$ is the CDF of $p_j(\cdot)$.

From $f_u(t) = e^{-iu^T\Phi^{-1}(t)}$, for any $j = 1, \ldots, d$ we have

$$\frac{\partial f(t)}{\partial t_j} = (-i)\frac{u_j}{p_j(\Phi_j^{-1}(t_j))}e^{-iu^T\Phi_j^{-1}(t)}.$$

Thus,

$$\frac{\partial^d f(t)}{\partial t_1 \cdots \partial t_d} = \prod_{j=1}^{d}\left((-i)\frac{u_j}{p_j(\Phi_j^{-1}(t_j))}\right)e^{-iu^T\Phi_j^{-1}(t)}.$$

In (8.6), when $I = [d]$,

$$\begin{aligned}
\int_{[0,1]^{|I|}}\left|\frac{\partial f}{\partial u_I}\right| dt_I &= \int_{[0,1]^d}\left|\frac{\partial^d f(t)}{\partial t_1 \cdots \partial t_d}\right| dt_1 \cdots dt_d \\
&= \int_{[0,1]^d}\left|\prod_{j=1}^{d}\left((-i)\frac{u_j}{p_j(\Phi_j^{-1}(t_j))}\right)e^{-iu^T\Phi_j^{-1}(t)}\right| dt_1 \cdots dt_d \\
&= \int_{[0,1]^d}\prod_{j=1}^{d}\left|\frac{u_j}{p_j(\Phi_j^{-1}(t_j))}\right| dt_1 \cdots dt_d \\
&= \prod_{j=1}^{d}\left(\int_{[0,1]}\left|\frac{u_j}{p_j(\Phi_j^{-1}(t_j))}\right| dt_j\right). \qquad \text{(D.1)}
\end{aligned}$$

With a change of variable, $\Phi_j(t_j) = v_j$, for $j = 1, \ldots, d$, (D.1) becomes

$$\prod_{j=1}^{d}\left(\int_{[0,1]}\left|\left(\frac{u_j}{p_j(\Phi_j^{-1}(t_j))}\right)\right| dt_j\right) = \prod_{j=1}^{d}\left(\int_{\mathbb{R}}|u_j| dv_j\right) = \infty.$$

As this is a term in (8.6), we know that $V_{HK}[f_u(t)]$ is unbounded.

D.2 PROOF OF PROPOSITION 8.6

We need the following lemmas, across which we share some notation.

Lemma D.1. *Assuming that $\kappa = \sup_{x\in\mathbb{R}^d} h(x,x) < \infty$ if $f \in \mathcal{H}$, where $\mathcal{H}$ is an RKHS with kernel $h(\cdot,\cdot)$, the integral $\int_{\mathbb{R}^d} f(x)p(x)dx$ is finite.*

Proof. For notational convenience, we note that

$$\int_{\mathbb{R}^d} f(x)p(x)dx = \mathbb{E}\left[f(X)\right],$$

where $\mathbb{E}\left[\cdot\right]$ denotes expectation and X is a random variable distributed according to the probability density $p(\cdot)$ on $\mathbb{R}^d$.

Now consider a linear functional T that maps f to $\mathbb{E}\left[f(X)\right]$, i.e.,

$$T[f] = \mathbb{E}\left[f(X)\right]. \tag{D.2}$$

The linear functional T is a bounded linear functional on the RKHS $\mathcal{H}$. To see this:

$$\begin{aligned}
|\mathbb{E}\left[f(X)\right]| &\leq \mathbb{E}\left[|f(X)|\right] \quad \text{(Jensen's Inequality)} \\
&= \mathbb{E}\left[|\langle f, h(X,\cdot)\rangle_{\mathcal{H}}|\right] \quad \text{(Reproducing Property)} \\
&\leq \|f\|_{\mathcal{H}}\mathbb{E}\left[\|h(X,\cdot)\|_{\mathcal{H}}\right] \quad \text{(Cauchy-Schwartz)} \\
&\leq \|f\|_{\mathcal{H}}\mathbb{E}\left[\sqrt{h(X,X)}\right] = \|f\|_{\mathcal{H}}\sqrt{\kappa} < \infty.
\end{aligned}$$

This shows that the integral $\int_{\mathbb{R}^d} f(x)p(x)dx$ exists. □

Lemma D.2. *The mean $\mu_{h,p}(u) = \int_{\mathbb{R}^d} h(u,x)p(x)dx$ is in $\mathcal{H}$. In addition, for any $f \in \mathcal{H}$,*

$$\mathbb{E}\left[f(X)\right] = \int_{\mathbb{R}^d} f(x)p(x)dx = \langle f, \mu_{h,p}\rangle_{\mathcal{H}}. \tag{D.3}$$

Proof. From the Riesz Representation Theorem, every bounded linear functional on $\mathcal{H}$ admits an inner product representation. Therefore, for T defined in (D.2), there exists $\mu_{h,p} \in \mathcal{H}$ such that

$$T[f] = \mathbb{E}\left[f(X)\right] = \langle f, \mu_{h,p}\rangle_{\mathcal{H}}.$$

Therefore we have, $\langle f, \mu_{h,p}\rangle_{\mathcal{H}} = \int f(x)p(x)dx$ for all $f \in \mathcal{H}$. For any z, choosing $f(\cdot) = h(z,\cdot)$, where $h(\cdot,\cdot)$ is the kernel associated with $\mathcal{H}$, and invoking the reproducing property we see that

$$\mu_{h,p}(z) = \langle h(z,\cdot), \mu_{h,p}\rangle_{\mathcal{H}} = \int_{\mathbb{R}^d} h(z,x)p(x)dx.$$

□

The proof of Proposition 8.6 follows from the existence lemmas above, and the following steps.

$$\begin{aligned}
\epsilon_{S,p}[f] &= \left| \int_{\mathbb{R}^d} f(x)p(x)dx - \frac{1}{s}\sum_{l=1}^{s} f(w^l) \right| \\
&= \left| \langle f, \mu_{h,p} \rangle_{\mathcal{H}} - \frac{1}{s}\sum_{l=1}^{s} \langle f, h(w^l, \cdot) \rangle_{\mathcal{H}} \right| \\
&= \left| \langle f, \mu_{h,p} - \frac{1}{s}\sum_{l=1}^{s} h(w^l, \cdot) \rangle_{\mathcal{H}} \right| \\
&\leq \|f\|_{\mathcal{H}} \left\| \mu_{h,p} - \frac{1}{s}\sum_{l=1}^{s} h(w^l, \cdot) \right\|_{\mathcal{H}} = \|f\|_{\mathcal{H}} D_{h,p}(S),
\end{aligned}$$

where $D_{h,p}(S)$ is given by

$$\begin{aligned}
D_{h,p}(S)^2 &= \left\| \mu_{h,p} - \frac{1}{s}\sum_{l=1}^{s} h(w^l, \cdot) \right\|_{\mathcal{H}}^2 \\
&= \langle \mu_{h,p}, \mu_{h,p} \rangle_{\mathcal{H}} - \frac{2}{s}\sum_{l=1}^{s} \langle \mu_{h,p}, h(w^l, \cdot) \rangle_{\mathcal{H}} + \frac{1}{s^2}\sum_{l=1}^{s}\sum_{j=1}^{s} \langle h(w^l, \cdot), h(w^j, \cdot) \rangle_{\mathcal{H}} \\
&= \mathbb{E}\left[\mu_{h,p}(X) \right] - \frac{2}{s}\sum_{l=1}^{s} \mathbb{E}\left[h(w^l, \cdot) \right] + \frac{1}{s^2}\sum_{l=1}^{s}\sum_{j=1}^{s} h(w^l, w^j) \\
&= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega, \phi)p(\omega)p(\phi)d\omega d\phi - \frac{2}{s}\sum_{l=1}^{s}\int_{\mathbb{R}^d} h(w^l, \omega)p(\omega)d\omega \\
&\quad + \frac{1}{s^2}\sum_{l=1}^{s}\sum_{j=1}^{s} h(w^l, w^j).
\end{aligned}$$

D.3 PROOF OF THEOREM 8.8

We apply (8.9) to the particular case of $h = \text{sinc}_b$. We have

$$\begin{aligned}
\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega, \phi)p(\omega)p(\phi)d\omega d\phi &= \pi^{-d}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} \prod_{j=1}^{d} \frac{\sin(b_j(\omega_j - \phi_j))}{\omega_j - \phi_j} p_j(\omega_j)p_j(\phi_j)d\omega d\phi \\
&= \pi^{-d}\prod_{j=1}^{d}\int_{\mathbb{R}}\int_{\mathbb{R}} \frac{\sin(b_j(\omega_j - \phi_j))}{\omega_j - \phi_j} p_j(\omega_j)p_j(\phi_j)d\omega_j d\phi_j,
\end{aligned}$$

and

$$\begin{aligned}
\sum_{l=1}^{s}\int_{\mathbb{R}^d} h(w^l, \omega)p(\omega)d\omega &= \pi^{-d}\sum_{l=1}^{s}\int_{\mathbb{R}^d} \prod_{j=1}^{d} \frac{\sin(b_j(w_j^l - \omega_j))}{w_j^l - \omega_j} p_j(\omega_j)d\omega \\
&= \pi^{-d}\sum_{l=1}^{s}\prod_{j=1}^{d}\int_{\mathbb{R}^d} \frac{\sin(b_j(w_j^l - \omega_j))}{w_j^l - \omega_j} p_j(\omega_j)d\omega_j.
\end{aligned}$$

Thus we can consider each coordinate on its own.

For fixed j, we have

$$\begin{aligned}\int_{\mathbb{R}} \frac{\sin(b_j x)}{x} p_j(x)dx &= \int_{\mathbb{R}} \int_0^{b_j} \cos(\beta x) p_j(x) d\beta dx \\ &= \frac{1}{2}\int_{-b_j}^{b_j} \int_{\mathbb{R}} e^{i\beta x} p(x) dx d\beta \\ &= \frac{1}{2}\int_{-b_j}^{b_j} \varphi_j(\beta) d\beta.\end{aligned}$$

The interchange in the second line is allowed because the $p_j(x)$ makes the function integrable (with respect to x).

Now fix $w \in \mathbb{R}$ as well. Let $h_j(x,y) = \sin(b_j(x-y))/\pi(x-y)$. We have

$$\begin{aligned}\int_{\mathbb{R}} h_j(\omega, w) p_j(\omega) d\omega &= \pi^{-1} \int_{\mathbb{R}} \frac{\sin(b_j(\omega - w))}{\omega - w} p_j(\omega) d\omega \\ &= \pi^{-1} \int_{\mathbb{R}} \frac{\sin(b_j x)}{x} p_j(x+w) dx \\ &= (2\pi)^{-1} \int_{-b_j}^{b_j} \varphi_j(\beta) e^{iw\beta} d\beta,\end{aligned}$$

where the last equality follows from first noticing the characteristic function associated with the density function $x \mapsto p_j(x+w)$ is $\beta \mapsto \varphi(\beta)e^{iw\beta}$, and then applying the previous inequality.

We also have

$$\begin{aligned}\int_{\mathbb{R}}\int_{\mathbb{R}} \frac{\sin(b_j(x-y))}{x-y} p_j(x)p_j(y)dxdy &= \int_{\mathbb{R}}\int_{\mathbb{R}}\int_0^{b_j} \cos(\beta(x-y)) p_j(x)p_j(y) d\beta dx dy \\ &= \frac{1}{2}\int_{\mathbb{R}}\int_{\mathbb{R}}\int_{-b_j}^{b_j} e^{i\beta(x-y)} p_j(x)p_j(y) d\beta dx dy \\ &= \frac{1}{2}\int_{-b_j}^{b_j}\int_{\mathbb{R}}\int_{\mathbb{R}} e^{i\beta(x-y)} p_j(x)p_j(y) dx dy d\beta \\ &= \frac{1}{2}\int_{-b_j}^{b_j} \left(\int_{\mathbb{R}} e^{i\beta x} p_j(x) dx\right)\left(\int_{\mathbb{R}} e^{-i\beta y} p_j(y) dy\right) d\beta \\ &= \frac{1}{2}\int_{-b_j}^{b_j} \varphi_j(\beta)\varphi_j(\beta)^* d\beta \\ &= \frac{1}{2}\int_{-b_j}^{b_j} |\varphi_j(\beta)|^2 d\beta.\end{aligned}$$

The interchange at the third line is allowed because of $p_j(x)p_j(y)$. In the last line we use the fact that $\varphi_j(\cdot)$ is Hermitian.

D.4 PROOF OF THEOREM 8.11

Let $b > 0$ be a scalar, and let $u \in [-b, b]$ and $z \in \mathbb{R}$. We have

$$\begin{aligned}\int_{-\infty}^{\infty} e^{-iux} \frac{\sin(b(x-z))}{\pi(x-z)} dx &= e^{-iuz} \int_{-\infty}^{\infty} e^{-i2\pi \frac{u}{2b} y} \frac{\sin(\pi y)}{\pi y} dy \\ &= e^{-iuz} \operatorname{rect}(u/2b) \\ &= e^{-iuz},\end{aligned}$$

where rect is the function that is 1 on $[-1/2, 1/2]$ and zero elsewhere.

The last equality implies that for every $u \in \square b$ and every $x \in \mathbb{R}^d$ we have

$$f_u(x) = \int_{\mathbb{R}^d} f_u(y) \operatorname{sinc}_b(y, x) dy.$$

We now have for every $u \in \square b$,

$$\begin{aligned}\epsilon_{S,p}[f_u] &= \left| \int_{\mathbb{R}^d} f_u(x) p(x) dx - \frac{1}{s} \sum_{i=1}^{s} f(w^i) \right| \\ &= \left| \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} f_u(y) \operatorname{sinc}_b(y, x) dy p(x) dx - \frac{1}{s} \sum_{i=1}^{s} \int_{\mathbb{R}^d} f_u(y) \operatorname{sinc}_b(y, w^i) dy \right| \\ &= \left| \int_{\mathbb{R}^d} f_u(y) \left[\int_{\mathbb{R}^d} \operatorname{sinc}_b(y, x) p(x) dx - \frac{1}{s} \sum_{i=1}^{s} \operatorname{sinc}_b(y, w^i) \right] dy \right|.\end{aligned}$$

Let us denote

$$r_S(y) = \int_{\mathbb{R}^d} \operatorname{sinc}_b(y, x) p(x) dx - \frac{1}{s} \sum_{i=1}^{s} \operatorname{sinc}_b(y, w^i).$$

Then

$$\epsilon_{S,p}[f_u] = \left| \int_{\mathbb{R}^d} f_u(y) r_S(y) dy \right|.$$

Since the function $r_S(\cdot)$ is square-integrable, it has a Fourier transform $\hat{r}_S(\cdot)$, and this formula is exactly the value of $\hat{r}_S(u)$. That is,

$$\epsilon_{S,p}[f_u] = |\hat{r}_S(u)|.$$

Now,

$$\begin{aligned}
\mathbb{E}_{f\sim\mathcal{U}(\mathcal{F}_{\square b})}\left[\epsilon_{S,p}[f]^2\right] &= \mathbb{E}_{u\sim\mathcal{U}(\square b)}\left[\epsilon_{S,p}[f_u]^2\right] \\
&= \int_{u\in\square b} |\hat{r}_S(u)|^2 \left(\prod_{j=1}^{d} 2b_j\right)^{-1} du \\
&= \left(\prod_{j=1}^{d} 2b_j\right)^{-1} \|\hat{r}_S\|_{L2}^2 \\
&= \frac{(2\pi)^d}{\prod_{j=1}^{d} 2b_j} \|r_S\|_{PW_b}^2 \\
&= \frac{\pi^d}{\prod_{j=1}^{d} b_j} D_p^{\square}(S)^2.
\end{aligned}$$

The equality before the last follows from Plancherel formula and the equality of the norm in PW_b to the $L2$-norm. The last equality follows from the fact that r_S is exactly the expression used in the proof of Proposition 8.6 to derive $D_p^{\square}$.

D.5 PROOF OF COROLLARY 8.12

In this case, $p(x) = \prod_{j=1}^{d} p_j(x_j)$ where $p_j(\cdot)$ is the density function of $\mathcal{N}(0, 1/\sigma_j)$. The characteristic function associated with $p_j(\cdot)$ is $\varphi_j(\beta) = e^{-\frac{\beta^2}{2\sigma_j^2}}$. We apply (8.11) directly.

For the first term, since

$$\begin{aligned}
\int_0^{b_j} |\varphi_j(\beta)|^2 d\beta &= \int_0^{b_j} e^{-\frac{\beta^2}{\sigma_j^2}} d\beta \\
&= \sigma_j \int_0^{b_j/\sigma_j} e^{-y^2} dy \\
&= \frac{\sigma_j\sqrt{\pi}}{2} \operatorname{erf}\left(\frac{b_j}{\sigma_j}\right),
\end{aligned}$$

we have

$$\pi^{-d} \prod_{j=1}^{d} \int_0^{b_j} |\varphi_j(\beta)|^2 d\beta = \prod_{j=1}^{d} \frac{\sigma_j}{2\sqrt{\pi}} \operatorname{erf}\left(\frac{b_j}{\sigma_j}\right). \tag{D.4}$$

For the second term, since

$$\begin{aligned}
\int_{-b_j}^{b_j} \varphi_j(\beta) e^{iw_j^l\beta} d\beta &= \int_{-b_j}^{b_j} e^{-\frac{\beta_j^2}{2\sigma^2}+iw_j^l\beta} d\beta \\
&= e^{-\frac{\sigma_j w_j^l}{2}} \int_{-b_j}^{b_j} e^{-\left(\frac{\beta}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)^2} d\beta \\
&= \sqrt{2}\sigma_j e^{-\frac{\sigma_j^2(w_j^l)^2}{2}} \int_{-\frac{b_j}{\sqrt{2}\sigma_j}}^{\frac{b_j}{\sqrt{2}\sigma_j}} e^{-\left(y-i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)^2} dy \\
&= \sqrt{2}\sigma_j e^{-\frac{\sigma_j^2(w_j^l)^2}{2}} \int_{-\frac{b_j}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}}^{\frac{b_j}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}} e^{-z^2} dz \\
&= \frac{\sqrt{\pi}\sigma_j}{\sqrt{2}} e^{-\frac{\sigma_j^2(w_j^l)^2}{2}} \left(\operatorname{erf}\left(-\frac{b_j}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)-\operatorname{erf}\left(\frac{b_j}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)\right) \\
&= \sqrt{2\pi}\sigma_j e^{-\frac{\sigma_j^2(w_j^l)^2}{2}} \operatorname{Re}\left(\operatorname{erf}\left(-\frac{b_j}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)\right),
\end{aligned}$$

we have

$$\frac{2}{s}(2\pi)^{-d}\sum_{l=1}^{s}\prod_{j=1}^{d}\int_{-b_j}^{b_j}\varphi_j(\beta)e^{iw_j^l\beta}d\beta = \frac{2}{s}\sum_{l=1}^{s}\prod_{j=1}^{d}\frac{\sigma_j}{\sqrt{2\pi}}e^{-\frac{\sigma_j^2(w_j^l)^2}{2}}\operatorname{Re}\left(\operatorname{erf}\left(-\frac{b_j}{\sqrt{2}\sigma_j}-i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)\right). \tag{D.5}$$

Combining (D.4), (D.5) and (8.11), (8.12) concludes the proof.

D.6 PROOF OF COROLLARY 8.13

The proof is similar to the proof of Theorem 3.6 of [DKS13]. Note that since $\sup_{x\in\mathbb{R}^d} h(x,x) < \infty$, we have $\int_{\mathbb{R}^d} h(x,x)p(x)dx < \infty$. From Lemma D.2 we know that $\int_{\mathbb{R}^d} h(\cdot,y)p(y)dy \in \mathcal{H}$; hence from Lemma D.1, we have $\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(x,y)p(x)p(y)dxdy < \infty$.

By (8.9), we have

$$\begin{aligned}
D_{h,p}(S)^2 &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi \\
&\quad -\frac{2}{s}\sum_{l=1}^{s}\int_{\mathbb{R}^d} h(w^l,\omega)p(\omega)d\omega \\
&\quad +\frac{1}{s^2}\sum_{l=1}^{s} h(w^l,w^l)+\frac{1}{s^2}\sum_{l,j=1,l\neq j}^{s} h(w^l,w^j).
\end{aligned}$$

Then,

$$\mathbb{E}\left[D_{h,p}(S)^2\right] = \int_{[0,1]^d} \cdots \int_{[0,1]^d} \left(\int_{\mathbb{R}^d} \int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi - \frac{2}{s}\sum_{l=1}^{s} \int_{\mathbb{R}^d} h(\Phi^{-1}(t^l),\omega)p(\omega)d\omega + \frac{1}{s^2}\sum_{l=1}^{s} h(\Phi^{-1}(t^l),\Phi^{-1}(t^l)) + \frac{1}{s^2}\sum_{l,j=1,l\neq j}^{s} h(\Phi^{-1}(t^l),\Phi^{-1}(t^j)) \right) dt^1 \cdots dt^s.$$

Obviously, the first is a constant independent of $t^1,\ldots,t^s$. Since all the terms are finite, we can interchange the integral and the sum among rest of the terms. In the second term, for each l, the only dependence on $t^1,\ldots,t^s$ is t^l; hence all the other t^j can be integrated out. That is,

$$\begin{aligned}\int_{[0,1]^d} \cdots \int_{[0,1]^d} \int_{\mathbb{R}^d} h(\Phi^{-1}(t^l),\omega)p(\omega)d\omega dt^1\cdots dt^s &= \int_{[0,1]^d} \int_{\mathbb{R}^d} h(\Phi^{-1}(t^l),\omega)p(\omega)d\omega dt^l \\ &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\phi,\omega)p(\phi)p(\omega)d\phi d\omega.\end{aligned}$$

The last equality comes from a change of variable, i.e., $t^l = (\Phi_1(\phi_1),\ldots,\Phi_d(\phi_d))$.

Similar operations can be done for the third and fourth term. Combining all of these, we have

$$\begin{aligned}\mathbb{E}\left[D_{h,p}(S)^2\right] &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi - 2\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi \\ &\quad + \frac{1}{s}\int_{\mathbb{R}^d} h(\omega,\omega)p(\omega)d\omega + \frac{s-1}{s}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi \\ &= \frac{1}{s}\int_{\mathbb{R}^d} h(\omega,\omega)p(\omega)d\omega - \frac{1}{s}\int_{\mathbb{R}^d}\int_{\mathbb{R}^d} h(\omega,\phi)p(\omega)p(\phi)d\omega d\phi.\end{aligned}$$

D.7 PROOF OF PROPOSITION 8.15

Before we compute the derivative, we prove two auxiliary lemmas.

Lemma D.3. *Let $x \in \mathbb{R}^d$ be a variable and $z \in \mathbb{R}^d$ be fixed vector. Then,*

$$\frac{\partial \operatorname{sinc}_b(x,z)}{\partial x_j} = b_j \operatorname{sinc}'_{b_j}(x_j,z_j)\prod_{q\neq j}\operatorname{sinc}_{b_q}(x_q,z_q). \tag{D.6}$$

We omit the proof as it is a simple computation that follows from the definition of sinc_b.

Lemma D.4. *The derivative of the scalar function $f(x) = \operatorname{Re}\left[e^{-ax^2}\operatorname{erf}(c+idx)\right]$, for real scalars a,c,d is given by*

$$\frac{\partial f}{\partial x} = -2axe^{-ax^2}\operatorname{Re}\left[\operatorname{erf}(c+idx)\right] + \frac{2d}{\sqrt{\pi}}e^{-ax^2}e^{d^2x^2-c^2}\sin(2cdx).$$

Proof. Since

$$\begin{aligned} f(x) &= \frac{1}{2}\left(e^{-ax^2}\operatorname{erf}(c+idx) + \left(e^{-ax^2}\operatorname{erf}(c+idx)\right)^*\right) \\ &= \frac{1}{2}\left(e^{-ax^2}\operatorname{erf}(c+idx) + e^{-ax^2}\operatorname{erf}(c-idx)\right), \end{aligned} \tag{D.7}$$

it suffices to compute the the derivative $g(x) = e^{-ax^2}\operatorname{erf}(c+idx)$.

Let $k(x) = \operatorname{erf}(c+idx)$. We have

$$g'(x) = -2axe^{-ax^2}k(x) + e^{-ax^2}k'(x). \tag{D.8}$$

Since

$$\begin{aligned} k(x) &= \operatorname{erf}(c+idx) \\ &= \frac{2}{\sqrt{\pi}}\int_0^{c+idx} e^{-z^2}dz \\ &= \frac{2}{\sqrt{\pi}}\left(\int_0^{c} e^{-z^2}dz + \int_c^{c+idx} e^{-z^2}dz\right) \\ &= \frac{2}{\sqrt{\pi}}\left(\int_0^{c} e^{-y^2}dy + (id)\int_0^{x} e^{-(c+idt)^2}dt\right), \end{aligned} \tag{D.9}$$

we have

$$k'(x) = \frac{2}{\sqrt{\pi}}e^{-(c+idx)^2} = \frac{2d}{\sqrt{\pi}}e^{d^2x^2-c^2}(\sin(2cdx) + i\cos(2cdx)). \tag{D.10}$$

We now have

$$\begin{aligned} f'(x) &= \frac{1}{2}\left(g'(x) + (g^*(x))'\right) \\ &= \frac{1}{2}\left(g'(x) + (g'(x))^*\right) \\ &= \frac{1}{2}\left(-2axe^{-ax^2}(k(x) + k^*(x)) + e^{-ax^2}(k'(x) + (k'(x))^*)\right) \\ &= \frac{1}{2}\left(-4axe^{-ax^2}\operatorname{Re}\left[\operatorname{erf}(c+idx)\right] + e^{-ax^2}\frac{4d}{\sqrt{\pi}}e^{d^2x^2-c^2}\sin(2cdx)\right) \\ &= -2axe^{-ax^2}\operatorname{Re}\left[\operatorname{erf}(c+idx)\right] + \frac{2d}{\sqrt{\pi}}e^{-ax^2}e^{d^2x^2-c^2}\sin(2cdx). \end{aligned} \tag{D.11}$$

□

For the first term in (8.12), that is $\frac{1}{s^2}\sum_{m=1}^{s}\sum_{r=1}^{s}\operatorname{sinc}_b(w^m, w^r)$, to compute the partial derivative of w_j^l, we only have to consider when at least m or r is equal to l. If $m = j = l$, by definition, the corresponding term in the summation is one. Hence, we only have to consider the case when $m \neq r$. By symmetry, it is equivalent to compute the partial derivative of the function $\frac{2}{s^2}\sum_{m=1,m\neq l}^{s}\operatorname{sinc}_b(w^l, w^m)$. Applying Lemma D.3, we get the first term in (8.14).

Next, for the last term in (8.12), we only have consider the term associated with one in the summation and the term associated with j in the product. Since

$\left(\frac{\sigma_j}{\sqrt{2\pi}}\right) e^{-\frac{\sigma_j^2 (w_j^l)^2}{2}} \operatorname{Re}\left(\operatorname{erf}\left(\frac{b_j}{\sigma_j\sqrt{2}} - i\frac{\sigma_j w_j^l}{\sqrt{2}}\right)\right)$ satisfies the formulation in Lemma D.4, we can simply apply Lemma D.4 and get its derivative with respect to w_j^l.

Equation (8.14) follows by combining these terms.

E

PROOFS OF RESULTS IN CHAPTER 9

E.1 PROOF OF THEOREM 9.4

Let $c(\sigma) = \frac{1}{s}\sum_{i=1}^{s} g_i(\sigma)$ where $g_i(\sigma) = e^{-\sigma^T w^i}$. It is easy to see that for any $x + z = \sigma$ we have $c(\sigma) = \langle \hat{\Psi}(x), \hat{\Psi}(z) \rangle$. The function k is additive-invariant, so we can abuse notation and write $k(\sigma) = k(x, z)$ for $x + z = \sigma$. Now, let us denote $f(\sigma) = c(\sigma) - k(\sigma)$.

Let $\mathcal{M}_\sigma = \{x + z \mid x, z \in \mathcal{M}\}$. It is easy to see that $\mathcal{M}_\sigma$ is a subset of $\mathcal{T} = \{\sigma \mid \|\sigma\| \leq 2R \text{ and } \sigma_i \geq r, i = 1, \ldots, d\}$. Our goal is to show that with probability specified in (9.3),

$$|f(\sigma)| < \epsilon, \quad \forall \sigma \in \mathcal{T}. \tag{E.1}$$

Since $\text{diam}(\mathcal{T}) \leq 4R$ and it is closed in $\mathbb{R}^d$, $\mathcal{T}$ is compact. We can construct an ϵ-net over $\mathcal{T}$ using less than $J = (4\text{diam}(\mathcal{T})/\gamma)^d$ balls with radius γ [CS02]. Denote the anchors of the net by $\{\eta^i\}_{i=1}^{J}$. We bound $|f(\sigma)|$ uniformly (with high probability) by showing that the following two claims hold with high probability:

- For $i = 1, \ldots, J$, $|f(\eta^i)| < \epsilon/2$;
- For $\forall \sigma \in \mathcal{M}_\sigma$, $\|\nabla f(\sigma)\| < \epsilon/2\gamma$.

These are sufficient because for any $\sigma \in \mathcal{T}$, let η^i be the nearest anchor to σ. It satisfies $\|\sigma - \eta^i\| < \gamma$, and hence

$$\begin{aligned} \|f(\sigma)\| &= \|f(\eta^i) + f(\sigma) - f(\eta^i)\| \\ &\leq \|f(\eta^i)\| + \|f(\sigma) - f(\eta^i)\| \\ &= \|f(\eta^i)\| + \|\nabla f(\xi)^T(\sigma - \eta^i)\| \\ &\leq \|f(\eta^i)\| + \|\nabla f(\xi)\| \cdot \|\sigma - \eta^i\| \\ &\leq \epsilon/2 + \gamma \cdot \epsilon/2\gamma = \epsilon \end{aligned} \tag{E.2}$$

for some ξ on the line connecting σ and η^i (note that $\mathcal{T}$ is convex, so the line is contained in it). The second equality uses the mean-value theorem.

To show the first claim, it is sufficient to show $|f(\sigma)| < \epsilon$ holds with high probability for any (fixed) σ. The claim then follows from a union bound. For any $\sigma \in \mathcal{T}$, we have

$$f(\sigma) = c(\sigma) - k(\sigma) = \frac{1}{s}\sum_{i=1}^{s} \left(g_i(\sigma) - k(\sigma)\right). \tag{E.3}$$

From Theorem 9.3, it is not hard to show that $\mathbb{E}[w_i]\, g_i(\sigma) = k(\sigma)$, for $i = 1, \ldots, s$. Also, $|g_i(\sigma)| \leq 1$. By the Hoeffding inequality, we have

$$\mathbb{P}\left\{\left|\frac{1}{s}\sum_{i=1}^{s} g_i(\sigma) - k(\sigma)\right| > \epsilon\right\} \leq 2e^{-2s\epsilon^2}. \tag{E.4}$$

By a union bound, the first claim holds with probability at least $1 - 2Je^{-s\epsilon^2/2}$.

To show the second claim, we need to bound the Lipschitz constant of f uniformly. Let $\lambda = \arg\max_{\sigma \in \mathcal{T}} \|\nabla f(\sigma)\|$. Since $\mathbb{E}[c(\lambda)] = k(\lambda)$, we have $\mathbb{E}[\nabla c(\lambda)] = \nabla k(\lambda)$, as

$$\begin{aligned}\frac{\partial k(\lambda)}{\partial \lambda_i} &= \frac{\partial}{\partial \lambda_i} \int_{\mathbb{R}^d_+} e^{-\lambda^T w} p(w) dw \\ &= \int_{\mathbb{R}^d_+} w_i e^{-\lambda^T w} p(w) dw \\ &= \mathbb{E}\left[w_i e^{-\lambda^T w}\right] = \mathbb{E}\left[\frac{\partial c(\lambda)}{\partial \lambda_i}\right]. \end{aligned} \tag{E.5}$$

The interchange between the integral and derivative is allowed because the functions $w, \lambda \mapsto e^{-\lambda^T w} p(w)$ and $w, \lambda \mapsto w_i e^{-\lambda^T w} p(w)$ are both continuous on λ and w.

Taking the expectation on $\|\nabla f(\lambda)\|^2 = \|\nabla c(\lambda) - \nabla k(\lambda)\|^2$, we have

$$\begin{aligned}\mathbb{E}\left[\|\nabla f(\lambda)\|^2\right] &= \mathbb{E}\left[\|\nabla c(\lambda)\|^2 - 2\nabla c(\lambda)^T \nabla f(\lambda) + \|\nabla k(\lambda)\|^2\right] \\ &= \mathbb{E}\left[\|\nabla c(\lambda)\|^2\right] - \|\nabla k(\lambda)\|^2 \\ &\leq \mathbb{E}\left[\|\nabla c(\lambda)\|^2\right] \\ &= \mathbb{E}\left[\sum_{j=1}^d \left(\frac{\partial c(\lambda)}{\partial \lambda_j}\right)^2\right]. \end{aligned} \tag{E.6}$$

Recall that $c(\lambda) = \frac{1}{s}\sum_{i=1}^s g_i(\lambda)$, so $\frac{\partial c(\lambda)}{\partial \lambda_j} = \frac{1}{s}\sum_{i=1}^s \frac{\partial g_i(\lambda)}{\partial \lambda_i}$. As $g_i(\lambda) = e^{-\lambda^T w^i}$, we have

$$\frac{\partial g_i(\lambda)}{\partial \lambda_j} = -e^{-\lambda^T w^i} w_j^i. \tag{E.7}$$

Hence, continuing (E.6), we have

$$\begin{aligned}\mathbb{E}\left[\|\nabla f(\lambda)\|^2\right] &\leq \frac{1}{s^2}\mathbb{E}\left[\sum_{j=1}^d \left(\sum_{i=1}^s e^{-\lambda^T w^i} w_j^i\right)^2\right] \\ &\leq \frac{1}{s^2}\mathbb{E}\left[\sum_{j=1}^d \left(\sum_{i=1}^s e^{-w_j^i \lambda_j} w_j^i\right)^2\right] \\ &\leq \frac{1}{s^2}\mathbb{E}\left[\sum_{j=1}^d \left(\sum_{i=1}^s e^{-w_j^i r} w_j^i\right)^2\right] \\ &= \frac{1}{s^2}\sum_{j=1}^d \mathbb{E}\left[\left(\sum_{i=1}^s e^{-w_j^i r} w_j^i\right)^2\right], \end{aligned} \tag{E.8}$$

where we used the facts that w_j^i, λ_j are positive and $\lambda_j \geq r$.

By our assumption, w_j^i are i.i.d. variables with density q. Hence the expectations in the summand above are identical. Let ν be a random variable with density function q. We have

$$\mathbb{E}\left[\|\nabla f(\lambda)\|^2\right] \leq \frac{d}{s^2}\mathbb{E}\left[\left(\sum_{i=1}^{s} e^{-\nu_i r}\nu_i\right)^2\right]. \tag{E.9}$$

Expanding the last inequality and using Jensen's inequality, we have

$$\begin{aligned}\mathbb{E}\left[\|\nabla f(\lambda)\|^2\right] &= \frac{d}{s^2}\left(s\mathbb{E}\left[(e^{-\nu r}\nu)^2\right] + (s^2-s)(\mathbb{E}\left[e^{-\nu r}\nu\right])^2\right)\\ &\leq d\cdot\mathbb{E}\left[e^{-2\nu r}\nu^2\right]\\ &= dL_{q,r}.\end{aligned} \tag{E.10}$$

By Markov's inequality, we have

$$\mathbb{P}\{\|\nabla f(\lambda)\|^2 > \epsilon^2/4\gamma^2\} \leq 4\gamma^2 L_{q,r}/\epsilon^2. \tag{E.11}$$

Overall, with probability at least $1 - 2\left(\frac{16R}{\gamma}\right)^d e^{-s\epsilon^2/2} - 4\gamma^2 L_{q,r}/\epsilon^2$, the two events will hold simultaneously. With $\gamma := (16R)^{\frac{d}{d+2}}(2e^{-s\epsilon^2/2}\gamma^2 L_{q,r})^{\frac{1}{d+2}}$, and since $L_{q,r}R > \epsilon$, the success probability is at least

$$1 - 2^6\left(\frac{R^2 L_{q,r}}{\epsilon^2}\right)^2 e^{-\frac{s\epsilon^2}{d+2}}. \tag{E.12}$$

The second part of the theorem follows by fixing the failure probability and solving for s.

E.2 PROOF OF PROPOSITION 9.5

By the definition of $L_{q,r}$, we have

$$\begin{aligned}L_{q,r} &= \mathbb{E}\left[e^{-2wr}w^2\right]\\ &= \frac{\beta}{2\sqrt{\pi}}\int_0^\infty e^{-2wr}w^2 w^{-\frac{3}{2}}e^{-\beta^2/4w}dw\\ &= \frac{\beta}{2\sqrt{\pi}}\int_0^\infty e^{-2wr-\beta^2/4w}w^{\frac{1}{2}}dw\\ &= \frac{\beta}{2\sqrt{\pi}}c^{\frac{3}{2}}\int_0^\infty e^{-(2rc)(w+1/w)}w^{\frac{1}{2}}dw\\ &= \frac{\beta}{4}\frac{\sqrt{2r}\beta+1}{(2r)^{\frac{3}{2}}e^{\sqrt{2r}\beta}},\end{aligned} \tag{E.13}$$

where $c = \frac{\beta}{2\sqrt{2\sigma_i}}$ and we use the fact that

$$\begin{aligned}
&\int e^{-a(x+1/x)} x^{1/2} dx \\
&= -\frac{\sqrt{\pi} e^{-2a}}{4a^{3/2}} \left((2a+1) erf(\sqrt{a}(1/\sqrt{x} - \sqrt{x})\right) - \\
&\quad \frac{\sqrt{\pi} e^{-2a}}{4a^{3/2}} \left((2a-1) e^{4a} erf(\sqrt{a}(1/\sqrt{x} + \sqrt{x}))\right) - \\
&\quad \frac{\sqrt{x} e^{-a(x+1/x)}}{a} + C.
\end{aligned} \tag{E.14}$$

E.3 PROOF OF PROPOSITION 9.6

Let $\phi(\sigma) = e^{-\beta \sum_{i=1}^{d} \sqrt{\sigma_i}}$. It is easy to verify that for $\rho = \epsilon^2/d^2$, we have $|\phi(\sigma+\rho) - \phi(\sigma)| < \epsilon$ for $\sigma \in \mathbb{R}_+^d$. Hence, for $\delta = \frac{\epsilon^2}{4d^2}$ we have $|k(x,z) - k(x+\delta, z+\delta)| < \epsilon/2$. Now, applying Theorem 9.4 to the set $\mathcal{M} + \delta$ shows that with the specified probability, $|k(x+\delta, z+\delta) - c(x+\delta, z+\delta)| < \epsilon/2$. The bound now follows from (9.4).

E.4 PROOF OF PROPOSITION 9.7

By the definition of $L_{q,r}$, we have

$$\begin{aligned}
L_{q,r} &= \mathbb{E}\left[e^{-2wr} w^2\right] \\
&= \lambda \int_0^\infty e^{-2wr} e^{-\lambda w} w^2 dw \\
&= \lambda \int_0^\infty e^{-(2r+\lambda)w} w^2 dw \\
&= \frac{2\lambda}{(\lambda + 2r)^3},
\end{aligned} \tag{E.15}$$

where we use the fact that

$$\int e^{ax} x^2 dx = \frac{e^{ax}(a^2x^2 - 2ax + 2)}{a^3} + C. \tag{E.16}$$

BIBLIOGRAPHY

[AA16] Y. Aizenbud and A. Averbuch. "Matrix decompositions using sub-Gaussian random matrices." *arXiv preprint arXiv:1602.03360* (2016).

[ABH16] N. Agarwal, B. Bullins, and E. Hazan. "Second order stochastic optimization in linear time." *arXiv preprint arXiv:1602.03943* (2016).

[AC06] N. Ailon and B. Chazelle. "Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform." In: *Symposium on Theory of Computing (STOC)*. 2006.

[AC09] N. Ailon and B. Chazelle. "The fast Johnson-Lindenstrauss transform and approximate nearest neighbors." *SIAM J. Comput.* 39.1 (2009), pp. 302–322.

[Ach01] D. Achlioptas. "Database-friendly random projections." In: *Symposium on Principles of Database Systems (PODS)*. 2001.

[AL09] N. Ailon and E. Liberty. "Fast dimension reduction using Rademacher series on dual BCH codes." *Discrete Comput. Geom.* 42.4 (2009), pp. 615–630.

[AL11] N. Ailon and E. Liberty. "An almost optimal unrestricted fast Johnson-Lindenstrauss transform." In: *Symposium on Discrete Algorithms (SODA)*. 2011.

[Alv+12] B. Alverson, E. Froese, L. Kaplan, and D. Roweth. "Cray XC series network." *Cray Inc. White Paper WP-Aries01-1112* (2012).

[AMT10] H. Avron, P. Maymounkov, and S. Toledo. "Blendenpik: supercharging LAPACK's least-squares solver." *SIAM J. Sci. Comput.* 32.3 (2010), pp. 1217–1236.

[AS15] H. Avron and V. Sindhwani. "High-performance kernel machines with implicit distributed optimization and randomization." *Technometrics* just-accepted (2015), pp. 1–27.

[Avr+16] H. Avron, V. Sindhwani, J. Yang, and M. W. Mahoney. "Quasi-Monte Carlo feature maps for shift-invariant kernels." *J. Mach. Learn. Res.* 17.120 (2016), pp. 1–38.

[Bac13] F. Bach. "Sharp analysis of low-rank kernel matrix approximations." In: *Conference on Learning Theory (COLT)*. 2013.

[BCR84] C. Berg, J. P. R. Christensen, and P. Ressel. *Harmonic Analysis on Semigroups: Theory of Positive-definite and Related Functions*. Springer, 1984.

[BE02] O. Bousquet and A. Elisseeff. "Stability and generalization." *J. Mach. Learn. Res.* 2.Mar (2002), pp. 499–526.

[BGG13] B. Boots, A. Gretton, and G. J. Gordon. "Hilbert space embeddings of predictive state representations." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2013.

[BLM89] J. Bourgain, J. Lindenstrauss, and V. Milman. "Approximation of zonoids by zonotopes." *Acta Math.* 162 (1989), pp. 73–141.

[BLO12] F. Bach, S. Lacoste-Julien, and G. Obozinski. "On the equivalence between herding and conditional gradient algorithms." In: *International Conference on Machine Learning (ICML)*. 2012.

[BNS06] M. Belkin, P. Niyogi, and V. Sindhwani. "Manifold regularization: a geometric framework for learning from labeled and unlabeled examples." *J. Mach. Learn. Res.* 7.Nov (2006), pp. 2399–2434.

[Boc33] S. Bochner. "Monotone funktionen, Stieltjes integrale und harmonische analyse." *Math. Ann.* 108 (1933), pp. 378–410.

[Bot10] L. Bottou. "Large-scale machine learning with stochastic gradient descent." In: *Computational Statistics (COMPSTAT)*. 2010.

[Bra83] R. N. Bracewell. "Discrete Hartley transform." *J. Opt. Soc. Am.* 73.12 (1983), pp. 1832–1835.

[BTA04] A. Berlinet and C. Thomas-Agnan. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Kluwer Academic Publishers, 2004.

[Bub14] S. Bubeck. "Theory of convex optimization for machine learning." *arXiv preprint arXiv:1405.4980* (2014).

[Buc94] M. Buchinsky. "Changes is US wage structure 1963-87: an application of quantile regression." *Econometrica* 62 (1994), pp. 405–408.

[Buh05] I. S. Buhai. "Quantile regression: overview and selected applications." *Ad Astra* 4 (2005).

[Byr+11] R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal. "On the use of stochastic Hessian information in optimization methods for machine learning." *SIAM J. Comput.* 21.3 (2011), pp. 977–995.

[Caf98] R. E. Caflisch. "Monte Carlo and Quasi-Monte Carlo methods." *Acta Numer.* 7 (Jan. 1998), pp. 1–49.

[CBCG04] N. Cesa-Bianchi, A. Conconi, and C. Gentile. "On the generalization ability of on-line learning algorithms." *IEEE Trans. Inform. Theory* 50.9 (2004), pp. 2050–2057.

[CCFC02] M. Charikar, K. Chen, and M. Farach-Colton. "Finding frequent items in data streams." In: *International Colloquium on Automata, Languages, and Programming (ICALP)*. 2002.

[CFG97] R. M. Caprioli, T. B. Farmer, and J. Gile. "Molecular imaging of biological samples: localization of peptides and proteins using MALDI-TOF MS." *Anal. Chem.* 69 (1997), pp. 4751–4760.

[CH10] K. Chughtai and R. M. A. Heeren. "Mass spectrometric imaging for biomedical tissue analysis." *Chem. Rev.* 110 (2010), pp. 3237–3277.

[CH86] S. Chatterjee and A. S. Hadi. "Influential observations, high leverage points, and outliers in linear regression." *Statist. Sci.* 1.3 (1986), pp. 379–393.

[CH88] S. Chatterjee and A. S. Hadi. *Sensitivity Analysis in Linear Regression*. New York: John Wiley & Sons, 1988.

[Cha99] O. Chapelle. "Support vector machines for histogram-based image classification." *IEEE Transactions on Neural Networks* 10 (1999), pp. 1055 –1064.

[Che+15] S. Chen, R. Varma, A. Singh, and J. Kovačević. "A statistical perspective of sampling scores for linear regression." *arXiv preprint arXiv:1507.05870* (2015).

[Cla+13] K. L. Clarkson, P. Drineas, M. Magdon-Ismail, M. W. Mahoney, X. Meng, and D. P. Woodruff. "The Fast Cauchy Transform and faster robust linear regression." In: *Symposium on Discrete Algorithms (SODA)*. 2013.

[Cla+93] K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S. Teng. "Approximating center points with iterated radon points." In: *Symposium on Computational Geometry (SoCG)*. 1993.

[Cla05] K. L. Clarkson. "Subgradient and sampling algorithms for ℓ_1 regression." In: *Symposium on Discrete Algorithms (SODA)*. 2005.

[CMM15] M. B. Cohen, C. Musco, and C. Musco. "Ridge leverage scores for low-rank approximation." *arXiv preprint arXiv:1511.07263* (2015).

[Coh+15] M. B. Cohen, Y. T. Lee, C. Musco, C. Musco, R. Peng, and A. Sidford. "Uniform sampling for matrix approximation." In: *Conference on Innovations in Theoretical Computer Science (ITCS)*. 2015.

[Coh16] M. B. Cohen. "Nearly tight oblivious subspace embeddings by trace inequalities." In: *Symposium on Discrete Algorithms (SODA)*. 2016.

[CP15] M. B. Cohen and R. Peng. "ℓ_p row sampling by Lewis weights." In: *Symposium on Theory of Computing (STOC)*. 2015.

[CR12] E. J. Candes and B. Recht. "Exact matrix completion via convex optimization." *Commun. ACM* 55.6 (2012), pp. 111–119.

[CRT11] E. S. Coakley, V. Rokhlin, and M. Tygert. "A fast randomized algorithm for orthogonal projection." *SIAM J. Sci. Comput.* 33.2 (2011), pp. 849–868.

[CS02] F. Cucker and S. Smale. "On the mathematical foundations of learning." *Bull. Amer. Math. Soc.* 39.1 (2002).

[CSHS11] M. de Carli Silva, N. Harvey, and C. M. Sato. "Sparse sums of positive semidefinite matrices." *arXiv preprint arXiv:1107.0088* (2011).

[Csu+04] G. Csurka, C.R. Dance, L. Dan, J. Williamowski, and C. Bray. "Visual categorization with bags of keypoints." In: *European Conference on Computer Vision (ECCV)*. 2004.

[CW13a] K. L. Clarkson and D. P. Woodruff. "Low rank approximation and regression in input sparsity time." In: *Symposium on Theory of Computing (STOC)*. 2013.

[CW13b] K. L. Clarkson and D. P. Woodruff. "Low rank approximation and regression in input sparsity time." *arXiv preprint arXiv:1207.6365* (2013).

[CWS10] Y. Chen, M. Welling, and A. Smola. "Super-samples from kernel herding." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2010.

[Das+09] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney. "Sampling algorithms and coresets for ℓ_p regression." *SIAM J. Comput.* 38.5 (2009), pp. 2060–2078.

[Den+09] W. J. Deng, R. Dong, L. J. Socher, K. L. Li, and F. Li. "ImageNet: a large-scale hierarchical image database." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[DES82] R. S. Dembo, S. C. Eisenstat, and T. Steihaug. "Inexact Newton methods." *SIAM J. Numer. Anal.* 19.2 (1982), pp. 400–408.

[DGS10] S. Danafar, A. Gretton, and J. Schmidhuber. "Characteristic kernels on structured domains excel in robotics and human action recognition." *Machine Learning and Knowledge Discovery in Databases* (2010), pp. 264–279.

[DHS11] J. C. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." *J. Mach. Learn. Res.* 12 (2011), pp. 2121–2159.

[DKS13] J. Dick, F. Y. Kuo, and I. H. Sloan. "High-dimensional integration: the Quasi-Monte Carlo way." *Acta Numer.* 22 (2013), pp. 133–288.

[DM16] P. Drineas and M. W. Mahoney. "RandNLA: randomized numerical linear algebra." *Commun. ACM* 59.6 (2016), pp. 80–90.

[DMM06] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. "Sampling algorithms for ℓ_2 regression and applications." In: *Symposium on Discrete Algorithms (SODA)*. 2006.

[DMM08] P. Drineas, M. W. Mahoney, and S Muthukrishnan. "Relative-error CUR matrix decompositions." *SIAM J. Matrix Anal. Appl.* 30.2 (2008), pp. 844–881.

[Dri+11] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlos. "Faster least squares approximation." *Numer. Math.* 117 (2011), pp. 219–248.

[Dri+12] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. "Fast approximation of matrix coherence and statistical leverage." *J. Mach. Learn. Res.* 13.1 (2012), pp. 3475–3506.

[DS01] K. R. Davidson and S. J. Szarek. "Local operator theory, random matrices and Banach spaces." In: *Handbook of the Geometry of Banach Spaces*. Vol. 1. North Holland, 2001, pp. 317–366.

[EAM15] A. El Alaoui and M. W. Mahoney. "Fast randomized kernel methods with statistical guarantees." *Neural Information Processing Systems (NIPS)* (2015).

[EM15] M. A Erdogdu and A. Montanari. "Convergence rates of sub-sampled Newton methods." In: *Neural Information Processing Systems (NIPS)*. 2015.

[Faa+12] G. Faanes et al. "Cray cascade: a scalable HPC system based on a Dragonfly network." In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. 2012.

[FHT01] J. Friedman, T. Hastie, and R. Tibshirani. *The Elements of Statistical Learning*. Springer, 2001.

[FL11] D. Feldman and M. Langberg. "A unified framework for approximating and clustering data." In: *Symposium on Theory of Computing (STOC)*. 2011.

[Fuk+08] K. Fukumiuzu, B. Sriperumbudur, A. Gretton, and B. Scholkopf. "Characteristic kernels on groups and semigroups." In: *Neural Information Processing Systems (NIPS)*. 2008.

[GB14] M. Grant and S. Boyd. *CVX: Matlab Software for Disciplined Convex Programming, version 2.1*. http://cvxr.com/cvx. Mar. 2014.

[GD05] K. Grauman and T. Darrel. "The pyramid match kernel: discriminative classification with sets of image features." In: *International Conference on Computer Vision (ICCV)*. 2005.

[GE96] M. Gu and S. C. Eisenstat. "Efficient algorithms for computing a strong rank-revealing QR factorization." *SIAM J. Sci. Comput.* 17.4 (1996), pp. 848–869.

[Git+16a] A. Gittens et al. "A multi-platform evaluation of the randomized CX low-rank matrix factorization in Spark." In: *International Workshop on Parallel and Distributed Computing for Large Scale Machine Learning and Big Data Analytics (ParLearning), at IPDPS*. 2016.

[Git+16b] A. Gittens et al. "Matrix factorizations at scale: a comparison of Spark and MPI using three case studies in scientific data analysis." *arXiv preprint arXiv:1607.01335* (2016).

[GN09] P. V. Gehler and S. Nowozin. "On feature combination methods for multiclass object classification." In: *International Conference on Computer Vision (ICCV)*. 2009.

[GR03] Z. Ghahramani and C. E. Rasmussen. "Bayesian Monte Carlo." In: *Neural Information Processing Systems (NIPS)*. 2003.

[GVL96] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Univ Press, 1996.

[Har+13] Z. Harchaoui, F. Bach, O. Cappe, and E. Moulines. "Kernel-based methods for hypothesis testing: a unified view." *IEEE Signal Processing Magazine* 30.4 (2013), pp. 87–97.

[HD12] F. Huszár and D. Duvenaud. "Optimally-weighted herding is Bayesian quadrature." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2012.

[HI15] J. T. Holodnak and I. C. F. Ipsen. "Randomized approximation of the Gram matrix: exact computation and probabilistic bounds." *SIAM J. Matrix Anal. Appl.* 36.1 (2015), pp. 110–137.

[HMT11] N. Halko, P. Martinsson, and J. Tropp. "Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions." *SIAM Rev.* 53.2 (2011), pp. 217–288.

[HTW15] T. Hastie, R. Tibshirani, and M. Wainwright. *Statistical Learning with Sparsity: The Lasso and Generalizations*. CRC Press, 2015.

[Hua+14] P. Huang, H. Avron, T. Sainath, V. Sindhwani, and B. Ramabhadran. "Kernel methods match Deep Neural Networks on TIMIT." In: *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. 2014.

[HW78] D. C. Hoaglin and R. E. Welsch. "The hat matrix in regression and ANOVA." *Amer. Statist.* 32.1 (1978), pp. 17–22.

[IM98] P. Indyk and R. Motwani. "Approximate nearest neighbors: towards removing the curse of dimensionality." In: *Symposium on Theory of Computing (STOC)*. 1998.

[JL84] W. B. Johnson and J. Lindenstrauss. "Extensions of Lipschitz mappings into a Hilbert space." *Contemp. Math.* 26.189-206 (1984), p. 1.

[Joa06] T. Joachims. "Training linear SVMs in linear time." In: *Conference on Knowledge Discovery and Data Mining (KDD)*. 2006.

[Joh48] F. John. "Extremum problems with inequalities as subsidiary conditions." In: *Studies and Essays presented to R. Courant on his 60th Birthday*. 1948, pp. 187–204.

[Jol86] I. T. Jolliffe. *Principal Component Analysis*. Springer, 1986.

[Jon+12] E. A. Jones, S.-O. Deininger, P. C. W. Hogendoorn, A. M. Deelder, and L. A. McDonnell. "Imaging mass spectrometry statistical analysis." *J. Proteomics* 75 (2012), pp. 4962–4989.

[KB78] R. Koenker and G. Bassett. "Regression quantiles." *Econometrica* 46.1 (1978), pp. 33–50.

[KD93] R. Koenker and V. D'Orey. "Computing regression quantiles." *Statistical Algorithms* 43 (1993), pp. 410–414.

[KH01] R. Koenker and K. Hallock. "Quantile regression." *J. Econ. Perspect.* 15.4 (2001), pp. 143–156.

[KL51] S. Kullback and R. A. Leibler. "On information and sufficiency." *Ann. Math. Statist.* 22.1 (1951), pp. 79–86.

[Kul13] B. Kulis. "Metric Learning: A Survey." *Foundations and Trends® in Machine Learning* 5.4 (2013), pp. 287–364.

[Lam09] C. H. Lampert. "Kernel Methods in Computer Vision." *Foundations and Trends® in Computer Graphics and Vision* 4.3 (2009), pp. 193–285.

[LFP03] F. Li, R. Fergus, and P. Perona. "A Bayesian approach to unsupervised one-shot learning of object categories." In: *International Conference on Computer Vision (ICCV)*. 2003.

[LIS10] F. Li, C. Ionescu, and C. Sminchisescu. "Random Fourier approximations for skewed multiplicative histogram kernels." *Pattern Recognition* 6376 (2010), pp. 262–271.

[LN89] D. C. Liu and J. Nocedal. "On the limited memory BFGS method for large scale optimization." *Math. Prog.* 45 (1989), pp. 503–528.

[Lou+13] K. B. Louie et al. ""Replica-extraction-transfer" nanostructure-initiator mass spectrometry imaging of acoustically printed bacteria." *Anal. Chem.* 85 (2013), pp. 10856–10862.

[Lov86] L. Lovasz. *Algorithmic Theory of Numbers, Graphs, and Convexity*. SIAM, 1986.

[LP14] G. Leobacher and F. Pillichschammer. *Introduction to Quasi-Monte Carlo Integration and Applications*. Springer, 2014.

[LS01] D. D. Lee and H. S. Seung. "Algorithms for non-negative matrix factorization." In: *Neural Information Processing Systems (NIPS)*. 2001.

[LS13] Y. T. Lee and A. Sidford. "Efficient accelerated coordinate descent methods and faster algorithms for solving linear systems." In: *Symposium on Foundations of Computer Science (FOCS)*. 2013.

[LSS13] Q. Le, T. Sarlós, and A. Smola. "Fastfood – Approximating kernel expansions in loglinear time." In: *International Conference on Machine Learning (ICML)*. 2013.

[Mah11] M. W. Mahoney. "Randomized algorithms for matrices and data." *Foundations and Trends® in Machine Learning* 3.2 (2011), pp. 123–224.

[Mar16] P. Martinsson. "Randomized methods for matrix computations and analysis of high dimensional data." *arXiv preprint arXiv:1607.01649* (2016).

[MB09] S. Maji and A.C. Berg. "Max-margin additive classifiers for detection." In: *International Conference on Computer Vision (ICCV)*. 2009.

[MD09] M. W. Mahoney and P. Drineas. "CUR matrix decompositions for improved data analysis." *Proc. Natl. Acad. Sci. USA* 106 (2009), pp. 697–702.

[MH07] L. A. McDonnell and R. M. A. Heeren. "Imaging mass spectrometry." *Mass Spectrom. Rev.* 26 (2007), pp. 606–643.

[MM13a] X. Meng and M. W. Mahoney. "Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression." In: *Symposium on Theory of Computing (STOC)*. 2013.

[MM13b] X. Meng and M. W. Mahoney. "Robust regression on MapReduce." In: *International Conference on Machine Learning (ICML)*. 2013.

[MMY15] P. Ma, M. W. Mahoney, and B. Yu. "A statistical perspective on algorithmic leveraging." *J. Mach. Learn. Res.* 16 (2015), pp. 861–911.

[MRT11] P. Martinsson, V. Rohklin, and M. Tygert. "A randomized algorithm for the decomposition of matrices." *Appl. Comput. Harmon. Anal.* 30 (2011), pp. 47–68.

[MSM14] X. Meng, M. A. Saunders, and M. W. Mahoney. "LSRN: a parallel iterative solver for strongly over- or under-determined systems." *SIAM J. Sci. Comput.* 36.2 (2014), pp. 95–118.

[Nes08] Y. Nesterov. "Rounding of convex sets and efficient gradient methods for linear programming problems." *Optim. Methods Softw.* 23.1 (2008), pp. 109–128.

[Nes13] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer, 2013.

[Nie92] H. Niederreiter. *Random number generation and Quasi-Monte Carlo methods*. SIAM, 1992.

[NN13] J. Nelson and H. L. Nguyen. "OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings." In: *Symposium on Foundations of Computer Science (FOCS)*. 2013.

[NN94] Y. Nesterov and A. Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM, 1994.

[NW06] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[NWS14] D. Needell, R. Ward, and N. Srebro. "Stochastic gradient descent, weighted sampling, and the randomized Kaczmarz algorithm." In: *Neural Information Processing Systems (NIPS)*. 2014.

[Par70] E. Parzen. "Statistical inference on time series by RKHS methods." In: *Biennial Seminar Canadian Mathematical Congress on Time Series and Stochastic Processes: convexity and combinatorics*. 1970.

[Pas+07] P. Paschou et al. "PCA-correlated SNPs for structure identification in world-wide human populations." *PLoS Genetics* 3 (2007), pp. 1672–1686.

[PK97] S. Portnoy and R. Koenker. "The Gaussian hare and the Laplacian tortoise: computability of squared-error versus absolute-error estimators." *Statist. Sci.* 12.4 (1997), pp. 279–300.

[Por97] S. Portnoy. "On computation of regression quantiles: making the Laplacian tortoise faster." *Lecture Notes-Monograph Series, Vol. 31, L_1-Statistical Procedures and Related Topics* (1997), pp. 187–200.

[PS82] C. C. Paige and M. A. Saunders. "LSQR: an algorithm for sparse linear equations and sparse least squares." *ACM Trans. Math. Softw.* 8.1 (1982), pp. 43–71.

[PW15] M. Pilanci and M. J. Wainwright. "Newton sketch: a linear-time optimization algorithm with linear-quadratic convergence." *arXiv preprint arXiv:1505.02250* (2015).

[R̈+13] O. Rübel et al. "OpenMSI: a high-performance web-based platform for mass spectrometry imaging." *Anal. Chem.* 85 (2013), pp. 10354–10361.

[Rei+11] W. Reindl, B. P. Bowen, M. A. Balamotis, J. E. Green, and T. R. Northen. "Multivariate analysis of a 3D mass spectral image for examining tissue heterogeneity." *Integr Biol (Camb)* 3 (2011), pp. 460–467.

[RKM16a] F. Roosta-Khorasani and M. W. Mahoney. "Sub-Sampled Newton Methods I: globally convergent algorithms." *arXiv preprint arXiv:1601.04737* (2016).

[RKM16b] F. Roosta-Khorasani and M. W. Mahoney. "Sub-Sampled Newton Methods II: local convergence rates." *arXiv preprint arXiv:1601.04738* (2016).

[RR08] A. Rahimi and B. Recht. "Random features for large-scale kernel machines." In: *Neural Information Processing Systems (NIPS)*. 2008.

[RSS12] A. Rakhlin, O. Shamir, and K. Sridharan. "Making gradient descent optimal for strongly convex stochastic optimization." In: *International Conference on Machine Learning (ICML)*. 2012.

[RT08] V. Rokhlin and M. Tygert. "A fast randomized algorithm for overdetermined linear least-squares regression." *Proc. Natl. Acad. Sci. USA* 105.36 (2008), pp. 13212–13217.

[Sar06] T. Sarlós. "Improved approximation algorithms for large matrices via random projections." In: *Symposium on Foundations of Computer Science (FOCS)*. 2006.

[Sau72] N. Sauer. "On the density of families of sets." *J. Combin. Theory, Series A* 13.1 (1972), pp. 145–147.

[Son+10] L. Song, B. Boots, S. Siddiqi, G. Gordon, and A. Smola. "Hilbert space embeddings of Hidden Markov Models." In: *International Conference on Machine Learning (ICML)*. 2010.

[Sri+10] B. Sriperumbudur, A. Gretton, K. Fukumizu, B. Scholkopf, and G. Lanckriet. "Hilbert space embeddings and metrics on probability measures." *J. Mach. Learn. Res.* 11 (2010), pp. 1517–1561.

[SS02] B. Schölkopf and A. Smola, eds. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

[SV09] T. Strohmer and R. Vershynin. "A randomized Kaczmarz algorithm with exponential convergence." *J. Fourier Anal. Appl.* 15.2 (2009).

[SW11] C. Sohler and D. P. Woodruff. "Subspace embeddings for the ℓ_1-norm with applications." In: *Symposium on Theory of Computing (STOC)*. 2011.

[TFF08] A. Torralba, R. Fergus, and W. T. Freeman. "80 million tiny images: a large data set for nonparametric object and scene recognition." *IEEE Trans. Pattern Anal. Mach. Intell.* 30.11 (2008), pp. 1958–1970.

[Tib96] R. Tibshirani. "Regression shrinkage and selection via the Lasso." *J. R. Stat. Soc. Ser. B. Stat. Methodol.* (1996), pp. 267–288.

[Tro11] J. A. Tropp. "Improved analysis of the subsampled randomized Hadamard transform." *Adv. Adapt. Data Anal.* 3.1-2 (2011), pp. 115–126.

[Tro15] J. A. Tropp. "An introduction to matrix concentration inequalities." *arXiv preprint arXiv:1501.01571* (2015).

[TW94] J. F. Traub and H. Wozniakowski. "Breaking intractability." *Scientific American* (1994), pp. 102–107.

[VW81] P. F. Velleman and R. E. Welsch. "Efficient computing of regression diagnostics." *Am. Stat.* 35.4 (1981), pp. 234–242.

[VX12] K. Varadarajan and X. Xiao. "On the sensitivity of shape fitting problems." In: *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*. 2012.

[VZ12] A. Vedaldi and A. Zisserman. "Efficient additive kernels via explicit feature maps." *IEEE Trans. Pattern Anal. Mach. Intell.* 3.34 (2012), pp. 480–492.

[Wah90] G. Wahba, ed. *Spline Models for Observational Data*. SIAM, 1990.

[Wei94] J. A. C. Weideman. "Computation of the complex error function." *SIAM J. Numer. Anal.* 31.5 (Oct. 1994), pp. 1497–1518.

[Wel09] M. Welling. "Herding dynamical weights to learn." In: *International Conference on Machine Learning (ICML)*. 2009.

[Woo14] D. P. Woodruff. "Sketching as a tool for numerical linear algebra." *Foundations and Trends® in Theoretical Computer Science* 10.1–2 (2014), pp. 1–157.

[Woz91] H. Wozniakowski. "Average case complexity of multivariate integration." *Bull. Amer. Math. Soc.* 24 (1991), pp. 185–194.

[WYS16] Y. Wang, A. W. Yu, and A. Singh. "Computationally feasible near-optimal subset selection for linear regression under measurement constraints." *arXiv preprint arXiv:1601.02068* (2016).

[WZ13] D. P. Woodruff and Q. Zhang. "Subspace embeddings and ℓ_p-regression using exponential random variables." *Conference on Learning Theory (COLT)* (2013).

[Xu+16] P. Xu, J. Yang, F. Roosta-Khorasani, C. Ré, and M. W. Mahoney. "Sub-sampled Newton methods with non-uniform sampling." In: *Neural Information Processing Systems (NIPS)*. 2016.

[Yan+14a] J. Yang, V. Sindhwani, H. Avron, and M. W. Mahoney. "Quasi-Monte Carlo feature maps for shift-invariant kernels." In: *International Conference on Machine Learning (ICML)*. 2014.

[Yan+14b] J. Yang, V. Sindhwani, Q. Fan, H. Avron, and M. W. Mahoney. "Random Laplace feature maps for semigroup kernels on histograms." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[Yan+15] J. Yang, O. Rübel, Prabhat, M. W. Mahoney, and B. P. Bowen. "Identifying important ions and positions in mass spectrometry imaging data using CUR matrix decompositions." *Anal. Chem.* 87.9 (2015), pp. 4658–4666.

[Yan+16a] J. Yang, Y. Chow, C. Ré, and M. W. Mahoney. "Weighted SGD for ℓ_p regression with randomized preconditioning." In: *Symposium on Discrete Algorithms (SODA)*. 2016.

[Yan+16b] J. Yang, Y. Chow, C. Ré, and M. W. Mahoney. "Weighted SGD for ℓ_p regression with randomized preconditioning." *arXiv preprint arXiv:1502.03571* (2016).

[Yao67] K. Yao. "Applications of Reproducing Kernel Hilbert Spaces - bandlimited signal models." *Inform. Control* 11 (1967), pp. 429–444.

[Yip+14] C.-W. Yip et al. "Objective identification of informative wavelength regions in galaxy spectra." *Astron. J.* 147 (2014).

[YMM13] J. Yang, X. Meng, and M. W. Mahoney. "Quantile regression for large-scale applications." In: *International Conference on Machine Learning (ICML)*. 2013.

[YMM14] J. Yang, X. Meng, and M. W. Mahoney. "Quantile regression for large-scale applications." *SIAM J. Sci. Comput.* 36.5 (2014), S78–S110.

[YMM16] J. Yang, X. Meng, and M. W. Mahoney. "Implementing randomized matrix algorithms in parallel and distributed environments." *Proceedings of the IEEE* 104.1 (2016), pp. 58–92.

[Zha+11] K. Zhang, J. Peters, D. Janzing, and B. Scholkopf. "Kernel based conditional independence test and application in causal discovery." In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2011.

[van69] A. van der Sluis. "Condition numbers and equilibration of matrices." *Numer. Math* 14.1 (1969), pp. 14–23.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

http://code.google.com/p/classicthesis/

Final Version as of August 22, 2016 (`JiyanThesis` version 5.0).