# RANDOMIZED ALGORITHMS FOR LARGE-SCALE STRONGLY OVER-DETERMINED LINEAR REGRESSION PROBLEMS

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

Xiangrui Meng
June 2014



This dissertation is online at: http://purl.stanford.edu/zn852mp0462

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.

**Margot Gerritsen**

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.*

# Abstract

In the era of big data, distributed systems built on top of clusters of commodity hardware provide cheap and reliable storage and scalable data processing. With cheap storage, instead of storing only currently relevant data, most people choose to store data as much as possible, expecting that its value can be extracted later. In this way, exabytes ($10^{18}$) of data are being created on a daily basis. However, extracting value from big data requires scalable implementation of advanced analytical algorithms beyond simple data processing, e.g., regression analysis and optimization. Many traditional methods are designed to minimize floating-point operations, which is the dominant cost of in-memory computation on a single machine. In a distributed environment, load balancing and communication including disk and network I/O can easily dominate computation. These factors greatly increase the complexity and challenge the way of thinking in the design of distributed algorithms.

Randomized methods for big data analysis have received a great deal of attention in recent years because they are generally faster and simpler to implement than traditional methods, it is easier to distribute the work, and they sometimes have theoretically provable performance. In this work, we are most interested in random projection and random sampling algorithms for $\ell_2$ regression and its robust alternative, $\ell_1$ regression, with strongly rectangular data. Random projection and random sampling are used to create preconditioned systems that are easier to solve or sub-sampled problems that provide relative-error approximations. Our main result shows that in near input-sparsity time and only a few passes through the data we can obtain a good approximate solution, with high probability. Our theory holds for general $p \in [1, 2]$, and thus we formulate our results in $\ell_p$.

In the first chapter, we introduce $\ell_p$ regression problems and $\ell_p$-norm conditioning, as well as traditional solvers for $\ell_p$ regression problems and how they are affected by the condition number. The second chapter describes the solution framework, where we discuss how ellipsoidal rounding and subspace embedding are connected to $\ell_p$ regression and develop faster rounding and embedding algorithms via random projection and random sampling. Chapter 3 describes a parallel solver named `LSRN` for strongly over- or under-determined linear least squares ($\ell_2$ regression), and Chapter 4 establishes the theory for $\ell_p$ subspace embedding and its application to $\ell_p$ regression.

# Acknowledgments

This thesis summarizes my work at Stanford University as a Ph.D. student at Institute for Computational and Mathematical Engineering (ICME). It couldn't be done without the financial support and the academic advice and assistance I received during my Ph.D. study.

The very first funding came from my parents, who had been supporting my education since my kindergarten. The money was well spent on the one way ticket that substantially changed my life. My first year was generously supported by ICME's department fellowship. In my second and third years, I worked under the Computational Approaches to Digital Stewardship (CADS) grant from the Library of Congress. Support for my fourth and fifth years was provided by the U.S. Army Research Laboratory, through the Army High Performance Computing Research Center, Cooperative Agreement W911NF-07-0027 and by NSF grant DMS-1009005.

I'd like to thank the members of my thesis committees. Art Owen, Walter Murray, Margot Gerritsen, Michael Mahoney (MM), and Michael Saunders (MS) attended my defense and gave me useful feedback that helped improve my thesis and extend the work. Margot, MM, and MS proofread drafts of this thesis and provided valuable input, from thesis organization to sentence punctuation. I'm very grateful for the time each member spent to make this thesis better. It is really fortunate to have MS as my adviser and MM as my co-adviser for the thesis work. If there were an alternative title for my thesis, it would be "A Tale of Two Michaels".

Graduating from the ICME program won't be possible for me without Brian Tempero and Indira Choudhury. Brian ensures that every desktop and server is running without any problem, while Indira constantly checks that every student is on track with the program.

Amin Saberi and Margot led the CADS program, a collaboration between ICME and the Library of Congress, where I had chance to apply my knowledge to some real-world datasets. I really enjoyed the time I spent on digging into the American Memory archive. What made the mining process more enjoyable was the companion of several friends from Stanford – Farnaz Ronaghi, Ying Wang, and David Gleich. With them, generating new ideas is never a difficult task. The summer times at Washington D.C. are always good to remember, apparently not because of the weather.

David and Paul Constantine organized the first ICME MapReduce workshop. At that one-day workshop, I implemented a sampling-based $\ell_1$ regression solver, which was my first hands-on experience with MapReduce and the start of the numerical part of this thesis. Now, developing and implementing scalable algorithms for massive datasets becomes my main interest and daily work. I

feel very fortunate to have been introduced into this world of big data.

I want to thank my family members, especially my parents and my wife, for their endless support and encouragement. I also want to thank some of my friends at ICME: Huang-Wei Chang, Youngsoo Choi, Sohan Dharmaraja, Xiaoye Jiang, Mikhail Kapralov, Sang-Yun Oh, Ying Wang, Xianyi Zeng, and `icme-sharedmem.stanford.edu`, the shared memory machine that ran my numerical experiments fast and accurately.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Analyzing data sets of billions of records has now become a regular task in many companies and institutions, where regression problems are ubiquitous, and the fast computation of their solutions on big data platforms is of interest in many large-scale applications. Most traditional algorithms are designed to run on a single machine, focusing on minimizing the number of floating-point operations (FLOPs). However, in a distributed environment, communication cost may become dominant. What is more, if the data cannot fit into memory, we have to scan the records from secondary storage, e.g., hard disk, which makes each pass through the data associate with huge I/O cost. In this work, we are most interested in the $\ell_2$ regression problem and its robust alternative, the $\ell_1$ regression problem, with strongly rectangular data. We show that, with randomized algorithms, it is possible to compute a good approximate solution in only a few passes. Our theory holds for general $p \in [1, 2]$, and thus we formulate our results in $\ell_p$.

## 1.1 Notation conventions

We briefly list the notation conventions we follow in this work:

- We use uppercase letters to denote matrices and constants, e.g., $A$, $R$, $C$, etc.
- We use lowercase letters to denote vectors and scalars, e.g., $x$, $b$, $p$, $m$, $n$, etc.
- We use $\|\cdot\|_p$ to denote the $\ell_p$ 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 $\ell_p$ norm of a matrix.
- We use uppercase calligraphic letters to denote point sets, e.g., $\mathcal{A}$ for the linear subspace spanned by $A$'s columns, $\mathcal{C}$ for a convex set, and $\mathcal{E}$ for an ellipsoid, except that $\mathcal{O}$ is used for big O-notation.
- The "$\tilde{}$" accent is used for sketches of matrices, e.g., $\tilde{A}$, the "$^*$" superscript is used for indicating optimal solutions, e.g., $x^*$, and the "$\hat{}$" accent is used for estimates of solutions, e.g., $\hat{x}$.

## 1.2 $\ell_p$ regression problems

In this work, a parameterized family of linear regression problems that is of particular interest is the $\ell_p$ *regression* problem.

**Definition 1** ($\ell_p$ regression)**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$, a vector $b \in \mathbb{R}^m$, and $p \in [1,\infty]$, the* $\ell_p$ regression problem *specified by $A$, $b$, and $p$ is the following optimization problem:*

$$\text{minimize}_{x\in\mathbb{R}^n} \ \|Ax-b\|_p, \tag{1.1}$$

*where the $\ell_p$ norm of a vector $x$ is $\|x\|_p = (\sum_i |x_i|^p)^{1/p}$, defined to be $\max_i |x_i|$ for $p=\infty$. We call the problem* strongly over-determined *if $m \gg n$, and* strongly under-determined *if $m \ll n$.*

Special cases include the $\ell_2$ regression problem, also known as linear least squares (LS), and the $\ell_1$ regression problem, also known as least absolute deviations (LAD) or least absolute errors (LAE). 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.

In Chapter 3, we consider high-precision solving of LS problems that are strongly over- or under-determined, and possibly rank-deficient. In particular, we wish to develop randomized algorithms to accurately solve the LS problem

$$\text{minimize}_{x\in\mathbb{R}^n} \quad \|Ax-b\|_2. \tag{1.2}$$

If we let $r = \text{rank}(A) \le \min(m,n)$, recall that if $r < n$ (the LS problem is under-determined or rank-deficient), then (1.2) has an infinite number of minimizers. In that case, the set of all minimizers is convex and hence has a unique element having minimum length. On the other hand, if $r = n$ so the problem has full rank, there exists only one minimizer to (1.2) and hence it must have the minimum length. In either case, we denote this unique min-length solution to (1.2) by $x^*$, and we are interested in computing $x^*$ in this work. That is,

$$x^* = \arg\min \|x\|_2 \quad \text{subject to} \quad x \in \arg\min_z \|Az-b\|_2. \tag{1.3}$$

For general $p \in [1,\infty]$, denote $\mathcal{X}^*$ the set of optimal solutions to (1.1). Let $x^* \in \mathcal{X}^*$ be an arbitrary optimal solution, and $f^* = \|Ax^*-b\|_p$ be the optimal objective value. In Chapter 4, we focus on finding a *relative-error approximation*, in terms of the objective value, to the general $\ell_p$ regression problem (1.1).

**Definition 2** (Relative-error approximation)**.** *Given an error parameter $\epsilon > 0$, $\hat{x} \in \mathbb{R}^n$ is a* $(1+\epsilon)$-approximate solution *to the $\ell_p$ regression problem* (1.1) *if and only if*

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

In order to make our theory simpler and our algorithms more concise, we use an equivalent

| | m | n |
|---|---|---|
| SNP | num. of SNPs ($10^7$) | num. of subjects ($10^3$) |
| TinyImages | num. of images ($10^8$) | num. of pixels in each image ($10^3$) |
| PDE | num. of degrees of freedom | num. of time steps |
| sensor network | size of sensing data | num. of sensors |
| NLP | num. of words and $n$-grams | num. of documents |
| tick data | num. of ticks | num. of stocks |

Table 1.1: Stongly rectangular datasets

formulation of (1.1) in our analysis:

$$\begin{array}{ll} \text{minimize}_{x\in\mathbb{R}^n} & \|Ax\|_p \\ \text{subject to} & c^T x = 1. \end{array} \tag{1.4}$$

This formulation of $\ell_p$ regression, which consists of a homogeneous objective and an affine constraint, can be shown to be equivalent to the formulation of (1.1). In particular, the "new" $A$ is $A$ concatenated with $-b$, and $c$ is a vector with a 1 at the last coordinate and zeros elsewhere to force the last element of any feasible solution to be 1. We note that the same formulation is also used by [57] for solving unconstrained convex problems in relative scale.

### 1.2.1 Strongly rectangular data

Strongly rectangular data arises in many fields. Table 1.1 lists a few examples:

- In genetics, single nucleotide polymorphisms (SNPs) are very important in the study of human health. There are roughly 10 known million SNPs in the human genome.[1] However, there are at most a few thousand subjects for a study of a certain type of disease, due to the high cost of determination of genotypes and limited number of target subjects.
- The continuous growth of Internet brings us more strongly rectangular datasets. In Chapter 3, we use an image dataset called TinyImages [66], which contains 80 million images of size $32\times32$, collected from Internet.
- In spatial discretization of high-dimensional partial differential equations (PDEs), the number of degrees of freedom grows exponentially as dimension increases. For 3D problems, it is common that the number of degrees of freedom reaches $10^9$, for example, by having a $1000 \times 1000 \times 1000$ discretization of a cubic domain. However, for a time-dependent problem, time stays one-dimensional. Though depending on spatial discretization (e.g., the Courant-Friedrichs-Lewy condition for hyperbolic PDEs), the number of time steps is usually much less than the number of degrees of freedoms in spatial discretization.

[1] `http://ghr.nlm.nih.gov/handbook/genomicresearch/snp`

- In geophysical applications, especially in seismology, the number of sensors is much less than the number of data points each sensor collects. For example, Werner-Allen et al. [70] deployed three wireless sensors to monitor volcanic eruptions. In 54 hours, each sensor sent back approximately 20 million packets.
- In natural language processing (NLP), the number of documents is much less than the number of $n$-grams, which grows geometrically as $n$ increases.
- In high-frequency trading, the number of relevant stocks is much less than the number of ticks, changes to the best bid and ask.

There are certainly many other examples, but those above should be adequate to demonstrate a broad range of applications involving strongly rectangular data.

### 1.2.2 $\ell_p$-norm condition number

$\ell_p$ regression problems are closely related to the concept of *condition number*. For linear systems and least squares problems, the $\ell_2$-norm condition number is already a well-established term.

**Definition 3** ($\ell_2$-norm condition number)**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ 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* $\ell_2$-norm condition number *of $A$ is defined as $\kappa_2(A) = \sigma_2^{\max}(A)/\sigma_2^{\min}(A)$. For simplicity, we use $\kappa_2$, $\sigma_2^{\min}$, and $\sigma_2^{\max}$ when the underlying matrix is clear from context.*

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

**Definition 4** ($\ell_p$-norm condition number (Clarkson et al. [20]))**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ and $p \in [1,\infty]$, let*

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

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

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

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

**Definition 5** ($(\alpha,\beta,p)$-conditioning (Dasgupta et al. [24]))**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ and $p \in [1,\infty]$, let $\|\cdot\|_q$ be the dual norm of $\|\cdot\|_p$. Then $A$ is* $(\alpha,\beta,p)$-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 $(\alpha,\beta,p)$-condition number of $A$, as the minimum value of $\alpha\beta$ such that $A$ is $(\alpha,\beta,p)$-conditioned. We use $\bar{\kappa}_p$ for simplicity if the underlying matrix is clear.*

**Lemma 1** (Equivalence of $\kappa_p$ and $\bar{\kappa}_p$ (Clarkson et al. [20]))**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ and $p \in [1,\infty]$, we always have*

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

*Proof.* To see the connection, recall that

$$|A|_p = \left(\sum_{j=1}^{n} \|Ae_j\|_p^p\right)^{1/p} \leq \left(\sum_{j=1}^{n} (\sigma_p^{\max}\|e_j\|_2)^p\right)^{1/p} = n^{1/p}\sigma_p^{\max},$$

and that

$$\|Ax\|_p \geq \sigma_p^{\min}\|x\|_2 \geq n^{\min\{1/p-1/2,0\}}\sigma_p^{\min}\|x\|_q, \quad \forall x \in \mathbb{R}^n .$$

Thus, $A$ is $(n^{1/p}\sigma_p^{\max}, 1/(n^{\min\{1/p-1/2,0\}}\sigma_p^{\min}), p)$-conditioned and $\bar{\kappa}_p(A) \leq n^{\max\{1/2,1/p\}}\kappa_p(A)$. On the other hand, if $A$ is $(\alpha, \beta, p)$-conditioned, we have, for all $x \in \mathbb{R}^n$,

$$\|Ax\|_p \leq |A|_p\|x\|_q \leq n^{\max\{1/2-1/p,0\}}\alpha \cdot \|x\|_2,$$

and

$$\|Ax\|_p \geq \|x\|_q/\beta \geq n^{\min\{1/2-1/p,0\}}/\beta \cdot \|x\|_2.$$

Thus, $\kappa_p(A) \leq n^{|1/p-1/2|}\alpha\beta$. □

The $\ell_p$-norm condition number of a matrix can be arbitrarily large. Given the equivalence established by Lemma 1, we say that $A$ is *well-conditioned in the $\ell_p$ norm* if $\kappa_p$ or $\bar{\kappa}_p = \mathcal{O}(\text{poly}(n))$, independent of $m$. We see in the following sections that the condition number plays a very important part in the analysis of traditional algorithms.

## 1.3 Traditional solvers

### 1.3.1 Solvers for linear least squares

Least squares is a classic problem in linear algebra. It has a long history, tracing back to Gauss, and it arises in numerous applications. A detailed survey of numerical algorithms for least squares is certainly beyond the scope of this work. In this section, we briefly describe some well-known direct methods and iterative methods that compute the min-length solution to a possibly rank-deficient least squares problem, and refer readers to Björck [9] for additional details.

#### Direct methods

It is well known that the min-length solution of a least squares problem can be computed using the singular value decomposition (SVD). Let $A = U\Sigma V^T$ be the compact SVD, where $U \in \mathbb{R}^{m\times r}$, $\Sigma \in \mathbb{R}^{r\times r}$, and $V \in \mathbb{R}^{n\times r}$, i.e., only singular vectors corresponding to the non-zero singular values are calculated. We have $x^* = V\Sigma^{-1}U^Tb$. The matrix $V\Sigma^{-1}U^T$ is the Moore-Penrose pseudoinverse of $A$, denoted by $A^\dagger$, which is defined and unique for any matrix. Hence we can simply write $x^* = A^\dagger b$. The SVD approach is accurate and robust to rank-deficiency.

Another way to solve a least squares problem is using complete orthogonal factorization. If we can find orthonormal matrices $Q \in \mathbb{R}^{m\times r}$ and $Z \in \mathbb{R}^{n\times r}$, and a matrix $T \in \mathbb{R}^{r\times r}$, such that

$A = QTZ^T$, then the min-length solution is given by $x^* = ZT^{-1}Q^Tb$. We can treat SVD as a special case of complete orthogonal factorization. In practice, complete orthogonal factorization is usually computed via rank-revealing QR factorizations, making $T$ a triangular matrix. The QR approach is less expensive than SVD, but it is slightly less robust at determining the rank.

A third way to solve a least squares problem is by computing the min-length solution to the normal equation $A^TAx = A^Tb$, namely

$$x^* = (A^TA)^\dagger A^Tb = A^T(AA^T)^\dagger b. \tag{1.5}$$

It is easy to verify the correctness of (1.5) by replacing $A$ by its compact SVD $U\Sigma V^T$. If $r = \min(m, n)$, a Cholesky factorization of either $A^TA$ (if $m \geq n$) or $AA^T$ (if $m \leq n$) solves (1.5) nicely. If $r < \min(m, n)$, we need the eigensystem of $A^TA$ or $AA^T$ to compute $x^*$. The normal equation approach is the least expensive among the three direct approaches we have mentioned, but it is also the least accurate one, especially on ill-conditioned problems. See Chapter 5 of Golub and Van Loan [33] for a detailed analysis.

For sparse least squares problems, by pivoting $A$'s columns and rows, we may find a sparse factorization of $A$, which is preferred to a dense factorization for more efficient storage. For sparse direct methods, we refer readers to Davis [26].

### Iterative methods

Instead of direct methods, we can use iterative methods to solve (1.2). If all the iterates $\{x^{(k)}\}$ are in range$(A^T)$ and if $\{x^{(k)}\}$ converges to a minimizer, it must be the minimizer having minimum length, i.e., the solution to (1.3). This is the case when we use a Krylov subspace method starting with a zero vector. For example, the conjugate gradient (CG) method on the normal equation leads to the min-length solution (see Paige and Saunders [60]). In practice, CGLS [38], LSQR [61] are preferable because they are equivalent to applying CG to the normal equation in exact arithmetic but they are numerically more stable. Other Krylov subspace methods such as LSMR [32] can also solve (1.2) as well. The Chebyshev semi-iterative method [34] can also be modified to solve LS problems.

Importantly, however, it is in general hard to predict the number of iterations for CG-like methods. The convergence rate is affected by the condition number of $A^TA$. A classical result [46, p.187] states that

$$\frac{\|x^{(k)} - x^*\|_{A^TA}}{\|x^{(0)} - x^*\|_{A^TA}} \leq 2\left(\frac{\sqrt{\kappa(A^TA)} - 1}{\sqrt{\kappa(A^TA)} + 1}\right)^k, \tag{1.6}$$

where $\|z\|_{A^TA} = z^TA^TAz = \|Az\|^2$ for any $z \in \mathbb{R}^n$, and where $\kappa(A^TA)$ is the condition number of $A^TA$ under the 2-norm. Estimating $\kappa(A^TA)$ is generally as hard as solving the LS problem itself, and in practice the bound does not hold in any case unless reorthogonalization is used. Thus, the computational cost of CG-like methods remains unpredictable in general, except when $A^TA$ is very well-conditioned and the condition number can be well estimated.

### 1.3.2 Solvers for $\ell_p$ regression

While $\ell_2$ regression can be solved with direct methods such as SVD and QR, the solution of general $\ell_p$ regression has to rely on iterative methods due to the lack of analytical solution. $\ell_1$ and $\ell_\infty$ regression problems can be formulated as linear programs and solved by linear programming solvers, and general $\ell_p$ regression problems can be formulated as convex programs and hence solved by general convex solvers. This, however, comes at the cost of increased complexity, compared to the $\ell_2$ case. For example, it is easy to see that all $\ell_p$ regression problems are convex due to the convexity of vector norms. Therefore, standard convex solvers, e.g., gradient-based methods [54], interior-point methods (IPMs) [71], and interior-point cutting-plane methods (IPCPMs)[52] can be used to solve $\ell_p$ regression problems. Discussing those convex solvers is beyond the scope of the work. We refer readers to the monographs mentioned above or Boyd and Vandenberghe [12] for a general introduction.

When $p = 1$ or $\infty$, the problem is still convex but not smooth. Subgradient methods [19] or gradient methods with smoothing [55] can be used to handle non-smoothness, while another solution is via linear programming. An $\ell_1$ regression problem specified by $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ is equivalent to the following linear program:

$$\begin{aligned} \text{minimize} \quad & \mathbf{1}_m^T y_+ + \mathbf{1}_m^T y_- \\ \text{subject to} \quad & Ax - b = y_+ - y_-, \\ & y_+, y_- \geq 0, \quad y_+, y_- \in \mathbb{R}^m, \quad x \in \mathbb{R}^n, \end{aligned}$$

and an $\ell_\infty$ regression problem specified by $A$ and $b$ is equivalent to the following:

$$\begin{aligned} \text{minimize} \quad & y \\ \text{subject to} \quad & -y \leq Ax - b \leq y, \\ & y \in \mathbb{R}, \quad x \in \mathbb{R}^n, \end{aligned}$$

where $\mathbf{1}_m \in \mathbb{R}^m$ indicates a vector of length $m$ with all ones. We omit the proofs for brevity, as they can be found in standard textbooks of convex optimization. As a linear programming problem, an $\ell_1$ or $\ell_\infty$ regression problem can be solved by any linear programming solver, using the simplex method [23] or IPMs.

Similar to the case for least squares, the condition number affects the performance of $\ell_p$ regression solvers, e.g., on the convergence rate for subgradient [19] or gradient method [56], on the search of an initial feasible point for IPMs [68], and on the initial search region for ellipsoid methods and IPCPMs [52]. Generally speaking, a smaller conditioning number makes the problem easier to solve.

Another popular way to solve $\ell_p$ regression problems is via iteratively re-weighted least squares (IRLS) [39], which solves a sequence of weighted least squares problems and makes the solutions converge to an optimal solution of the original $\ell_p$ regression problem. At step $k$, it solves the following

weighted least squares problem:

$$x^{(k+1)} = \arg\min_{x\in\mathbb{R}^n} \|W^{(k)}(Ax-b)\|_2,$$

where $W^{(k)}$ is a diagonal matrix with positive diagonals $w_i^{(k)}$, $i = 1,\ldots,m$. Let $W^{(0)}$ be an identity matrix and choose

$$w_i^{(k)} = |a_i^T x^{(k)} - b_i|^{p-2}, \quad i = 1,\ldots,m, \quad k = 1,\ldots.$$

until $\{x^{(k)}\}$ converges. The choice of $w_i^{(k)}$ is often smoothed to avoid dividing by zero in practice. It is not hard to show that if $\{x^{(k)}\}$ converges, it converges to an optimal solution of the $\ell_p$ regression problem. However, the convergence theory of IRLS only exists under certain assumptions and the convergence rate is much harder to derive. See Burrus [15] for a survey of related work.

## 1.4 Preconditioning

Although for an arbitrary matrix $A \in \mathbb{R}^{m\times n}$ with full column rank, its condition numbers $\kappa_p(A)$ and $\bar{\kappa}_p(A)$ can be arbitrarily large, we can often find a matrix $R \in \mathbb{R}^{n\times n}$ such that $AR^{-1}$ is well-conditioned. Then the $\ell_p$ regression problem (1.4) is equivalent to the following well-conditioned problem:

$$\begin{array}{ll} \text{minimize}_{y\in\mathbb{R}^n} & \|AR^{-1}y\|_p, \\ \text{subject to} & c^T R^{-1} y = 1. \end{array} \tag{1.7}$$

It is easy to see that if $y^*$ is an optimal solution to (1.7), $x^* = R^{-1}y$ is an optimal solution to (1.4), and vice versa. However, (1.7) may be easier to solve than (1.4) because of better conditioning. This procedure is called *preconditioning*.

Since we want to reduce the condition number of the problem via preconditioning, it is natural to ask what the best outcome would be in theory. Recall that we define two condition numbers in Section 1.2.2. For the $\ell_p$-norm condition number $\kappa_p$, we have the following existence result.

**Lemma 2.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ with full column rank and $p \in [1,\infty]$, there exist a matrix $R \in \mathbb{R}^{n\times n}$ such that $\kappa_p(AR^{-1}) \le n^{1/2}$.*

This is a direct consequence of John's theorem [41] on ellipsoidal rounding of centrally symmetric convex sets. We discuss ellipsoidal rounding in more details in Section 2.1, where we show the connection between ellipsoidal rounding and preconditioning. For the $(\alpha,\beta,p)$-condition number $\bar{\kappa}_p$, we have the following lemma.

**Lemma 3.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ with full column rank and $p \in [1,\infty]$, there exist a matrix $R \in \mathbb{R}^{n\times n}$ such that $\bar{\kappa}_p(AR^{-1}) \le n$.*

*Proof.* This is derived from Auerbach's lemma.

**Lemma 4.** *(Auerbach's lemma [6]) Let $(\mathcal{A}, \|\cdot\|)$ be a $n$-dimensional normed vector space. There exists a basis $\{e_1, \ldots, e_n\}$ of $\mathcal{A}$, called Auerbach basis, such that $\|e_k\| = 1$ and $\|e^k\|^* = 1$ for $k = 1, \ldots, n$, where $\{e^1, \ldots, e^n\}$ is a basis of $\mathcal{A}^*$ dual to $\{e_1, \ldots, e_n\}$.*

Let $R$ be the matrix such that $U = AR^{-1}$ is an Auerbach basis of $\mathcal{A}_p = (\mathcal{A}, \|\cdot\|_p)$, the subspace spanned by $A$'s columns paired with the $\ell_p$ norm. By definition,

$$|U|_p = \left( \sum_{j=1}^{n} \|u_j\|_p^p \right)^{1/p} = n^{1/p},$$

where $u_j$ is the $j$-th column of $U$. Let $U^*$ be the matrix whose columns $u_1^*, \ldots, u_n^*$ form the dual Auerbach basis to $\{u_1, \ldots, u_n\}$. By definition, $\|u_j^*\|_q = 1$, $j = 1, \ldots, n$, and $|U^*|_q = n^{1/q}$, where $\|\cdot\|_q$ is the dual norm of $\|\cdot\|_p$, i.e., $1/p + 1/q = 1$. Then, for any $z \in \mathbb{R}^n$, we have

$$\frac{\|Uz\|_p}{\|z\|_q} = \max_{w \in \mathbb{R}^n} \frac{|\langle U^* w, Uz \rangle|}{\|U^* w\|_q \|z\|_q} \geq \frac{1}{|U^*|_q} \max_{w \in \mathbb{R}^n} \frac{|\langle w, z \rangle|}{\|w\|_p \|z\|_q} = n^{-1/q},$$

where the equality $\langle U^* w, Uz \rangle = \langle w, z \rangle$ comes from the duality. Therefore, we have $\alpha \leq n^{1/p}$, $\beta \leq n^{1/q}$, and hence $\bar{\kappa}_p(AR^{-1}) \leq \alpha\beta \leq n^{1/p+1/q} = n$. □

Lemmas 2 and 3 are both existence results. Unfortunately, no polynomial-time algorithm is known that can provide such preconditioning for general matrices (except the case when $p = 2$). In the next chapter, we discuss two practical approaches for $\ell_p$-norm preconditioning: via ellipsoidal rounding and via subspace embedding, as well as subspace-preserving sampling algorithms built on top of them.

# Chapter 2

# Rounding and Embedding

For any matrix $A \in \mathbb{R}^{m\times n}$ with full column rank, Theorems 2 and 3 show that there always exist a preconditioner matrix $R \in \mathbb{R}^{n\times n}$ such that $AR^{-1}$ is well-conditioned. In this chapter, we discuss practical algorithms to find such a matrix and show the trade-offs between speed and conditioning quality. The algorithms fall into two families: ellipsoidal rounding and subspace embedding. And we present them roughly in the order of speed, from slow ones to fast ones. Note that we assume $m \gg \text{poly}(n)$ and hence $mn^2 \gg mn + \text{poly}(n)$. If $A$ is sparse, we assume that $mn \gg \text{nnz}(A)$. We also show that, bypassing preconditioning, subspace embedding directly relates to $\ell_p$ regression, when the $\ell_p$ norms of the entire subspace of vectors can be well preserved.

## 2.1 Ellipsoidal rounding

In this section, we are interested in the *ellipsoidal rounding* of a centrally symmetric convex set and its application to $\ell_p$-norm preconditioning.

**Definition 6** (Ellipsoidal rounding)**.** *Let $\mathcal{C} \subseteq \mathbb{R}^n$ be a convex set that is full-dimensional, closed, bounded, and centrally symmetric with respect to the origin. An ellipsoid $\mathcal{E}(0, E) = \{x \in \mathbb{R}^n \mid \|Ex\|_2 \leq 1\}$ is a* $\kappa$-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 $1/\kappa$.*

Finding an ellipsoidal rounding with a small $\kappa$ factor for a given convex set has many applications such as computational geometry [8], convex optimization [45], and computer graphics [11]. The $\ell_p$-norm condition number $\kappa_p$ naturally connects to ellipsoidal rounding. Let $\mathcal{C} = \{x \in \mathbb{R}^n \mid \|Ax\|_p \leq 1\}$ and assume that we have a $\kappa$-rounding of $\mathcal{C}$: $\mathcal{E} = \{x \mid \|Rx\|_2 \leq 1\}$. This implies

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

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

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

| | $\kappa$ | time | num. passes |
|---|---|---|---|
| [19, 24] | $(n(n+1))^{1/2}$ | $\mathcal{O}(mn^5 \log m)$ | $\mathcal{O}(n^3 L)$ |
| [20] | $2n$ | $\mathcal{O}(mn^3 \log m)$ | $\mathcal{O}(nL)$ |
| [51] | $2n^{\lvert 2/p-1\rvert+1}$ | $\mathcal{O}(mn^2 \log m)$ | 1 |

Table 2.1: $\ell_p$ conditioning via ellipsoidal rounding

Therefore, we have $\kappa_p(AR^{-1}) \leq \kappa$. So a $\kappa$-rounding of $\mathcal{C}$ leads to a $\kappa$-conditioning of $A$. Recall the well-known result due to John [41] that for a centrally symmetric convex set $\mathcal{C}$ there exists a $n^{1/2}$-rounding, where the 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 $\mathcal{C}$, which leads to Lemma 2 in the previous chapter. Unfortunately, finding an $n^{1/2}$-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 [45]. For polynomial-time algorithms, the best $\kappa$ that we are aware of is $(n(n+1))^{1/2}$, while polynomial algorithms with better $\kappa$ have been proposed for special convex sets, for example, the convex hull of a finite point set [43] and the convex set specified by the matrix $\ell_\infty$ norm [56]. 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 $(n(n+1))^{1/2}$-rounding in polynomial time, in particular, in $O(n^4 \log L)$ calls to the oracle; see Lovász [45, Theorem 2.4.1]. This result was used by Clarkson [19] and by Dasgupta *et al.* [24] for $\ell_p$ regression. In these work, only $\mathcal{O}(n)$-rounding is actually needed instead of $(n(n+1))^{1/2}$-rounding. In Section 2.4, we follow the same construction as in the proof of Lovász [45], but we show that it is much faster (in $\mathcal{O}(n^2 \log L)$ calls to the oracle) to find a slightly worse $2n$-rounding of a centrally symmetric convex set in $\mathbb{R}^n$ that is described by a separation oracle. The trade-offs between rounding quality and running time are also discussed in Section 2.4.

## 2.2 Subspace embedding

Denote $\mathcal{A} \subset \mathbb{R}^m$ the subspace spanned by $A$'s columns. 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. In this section, 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 $\ell_p$ norm $\|\cdot\|_p$.

**Definition 7** ($\ell_p$ subspace embedding)**.** *Given a matrix $A \in \mathbb{R}^{m\times n}$ and $p \in [1, \infty]$, $\Phi \in \mathbb{R}^{s\times m}$ is a low-distortion embedding of $\mathcal{A}_p$ if $s = \mathcal{O}(\text{poly}(n))$, independent of $m$, and there exist $\sigma_\Phi^{\min} > 0$ and $\kappa_\Phi > 0$ such that*

$$\sigma_\Phi \cdot \|Ax\|_p \leq \|\Phi Ax\|_p \leq \kappa_\Phi \sigma_\Phi \cdot \|Ax\|_p, \quad \forall x \in \mathbb{R}^n .$$

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

Low-distortion subspace embeddings can be used for $\ell_p$-norm preconditioning. For example,

given a low-distortion embedding matrix $\Phi$ of $\mathcal{A}_p$ with distortion $\kappa_\Phi$, let $R$ be the "R" matrix from the QR decomposition of $\Phi A$. Then, the matrix $AR^{-1}$ is well-conditioned in the $\ell_p$ norm. To see this, note that we have

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

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

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

Hence, by combining these results, we have $\kappa_p(AR^{-1}) \leq \kappa_\Phi s^{|1/p-1/2|} = \mathcal{O}(\text{poly}(n))$, i.e., the matrix $AR^{-1}$ is well-conditioned in the $\ell_p$ norm.

Instead of QR factorization, ellipsoidal rounding can be used to obtain the preconditioner matrix $R$. To see this, let $R$ be the matrix obtained by applying Corollary 1 to $\Phi A$. We have

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

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

$$\|AR^{-1}x\|_p \geq \sigma_\Phi \|\Phi AR^{-1}x\|_p \geq \sigma_\Phi \|x\|_2, \quad \forall x \in \mathbb{R}^n\,.$$

Hence $\kappa_p(AR^{-1}) \leq 2n\kappa_\Phi = \mathcal{O}(\text{poly}(n))$ and $AR^{-1}$ is well-conditioned.

Comparing the QR approach and the ellipsoidal rounding approach of obtaining the preconditioner matrix $R$, we see there are trade-offs between running times and conditioning quality. QR takes $\mathcal{O}(sn^2)$ time, which is faster than fast ellipsoidal rounding that takes $\mathcal{O}(sn^3 \log s)$ time. However, the latter approach might provide better conditioning quality when $2n < s^{|1/p-1/2|}$. We note that those trade-offs are not important in theory as long as both take $\mathcal{O}(\text{poly}(n))$ time and provide $\mathcal{O}(\text{poly}(n))$ conditioning, independent of $m$, but they do affect the performance in practice.

In this section, we only show that how subspace embedding connects to preconditioning assuming that an embedding matrix is given. Because subspace embedding is the main ingredient of this work, we devote Section 2.5 to a detailed study of subspace embedding algorithms.

## 2.3 Subspace-preserving embedding

Rather than preconditioning, a special family of subspace embedding has a more direct connection with $\ell_p$ regression, called subspace-preserving embedding.

**Definition 8** (Subspace-preserving embedding)**.** *Given an $n$-dimensional subspace $\mathcal{A}_p \subset \mathbb{R}^m$ with*

*$p \in [1, \infty]$ and $\epsilon \in (0, 1)$, an embedding $\Pi : \mathbb{R}^m \hookrightarrow \mathbb{R}^s$ is a subspace-preserving embedding of $\mathcal{A}_p$ if*

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

Given a subspace-preserving embedding with distortion $(1 \pm \epsilon)$, Clarkson et al. [20] show that it is straightforward to compute a $\frac{1+\epsilon}{1-\epsilon}$-approximate solution to an $\ell_p$ regression problem using the constrained formulation (1.4).

**Lemma 5.** *Given an $\ell_p$ regression problem specified by $A \in \mathbb{R}^{m \times n}$ and $p \in [1, \infty)$ using the constrained formulation (1.4), let $\Phi$ be a $(1 \pm \epsilon)$-distortion embedding of $\mathcal{A}_p$, and $\hat{x}$ be an optimal solution to the reduced-sized problem $\min_{c^T x = 1} \|\Phi A x\|_p$. Then $\hat{x}$ is a $\frac{1+\epsilon}{1-\epsilon}$-approximate solution to the original problem.*

*Proof.* It is easy to see that $\hat{x}$ is a feasible solution to the original problem and

$$\|A\hat{x}\|_p \leq (1 + \epsilon)\|\Phi A\hat{x}\|_p \leq (1 + \epsilon)\|\Phi A x^*\|_p \leq \frac{1 + \epsilon}{1 - \epsilon}\|A x^*\|_p,$$

where the first and third inequalities are due to the distortion of the subspace embedding and the second inequality is due to the optimality of $\hat{x}$. Hence, $\hat{x}$ is a $\frac{1+\epsilon}{1-\epsilon}$-approximate solution to the original $\ell_p$ regression problem. □

Therefore, an $\ell_p$ regression problem can be approximated by solving a reduced-sized $\ell_p$ regression problem obtained from subspace-preserving embedding. We show in later sections that such embeddings can be obtained directly for $\ell_2$ and via random sampling for general $\ell_p$.

## 2.4 Fast ellipsoidal rounding

We follow the same construction as in the proof of Lovász [45], but we show that it is much faster to find a (slightly worse) $2n$-rounding of a centrally symmetric convex set in $\mathbb{R}^n$ that is described by a separation oracle.

**Theorem 1** (Fast ellipsoidal rounding)**.** *Given a centrally symmetric convex set $\mathcal{C} \subseteq \mathbb{R}^n$, which is centered at the origin and described by a separation oracle, and an ellipsoid $\mathcal{E}_0$ centered at the origin such that $\mathcal{E}_0/L \subseteq \mathcal{C} \subseteq \mathcal{E}_0$ for some $L \geq 1$, it takes at most $3.15n^2 \log L$ calls to the oracle and additional $O(n^4 \log L)$ time to find a $2n$-rounding of $\mathcal{C}$.*

*Proof.* For completeness, we state the following lemma, which is from [65] and which we use in the proof of this theorem.

**Lemma 6.** *(Todd [65]) Given an ellipsoid $\mathcal{E} = \{u \in \mathbb{R}^n \,|\, u^T E^{-1} u \leq 1\}$ where $E \in \mathbb{R}^{n \times n}$ is symmetric positive-definite and $\mathcal{K} = \{u \in \mathbb{R}^n \mid -\beta(g^T E g)^{1/2} \leq g^T u \leq \beta(g^T E g)^{1/2}\}$ for some*

*$g \in \mathbb{R}^n$, the minimum-volume ellipsoid that contains $\mathcal{E} \cap \mathcal{K}$ is given by*

$$\mathcal{E}_+ = \begin{cases} \mathcal{E} & \text{if } \beta \geq n^{-1/2} \\ \{u \in \mathbb{R}^n \mid u^T E_+^{-1} u \leq 1\} & \text{if } 0 < \beta < n^{-1/2}, \end{cases}$$

*where*

$$E_+ = \delta \left( E - \sigma \frac{(Eg)(Eg)^T}{g^T E g} \right),$$
$$\delta = \frac{n(1-\beta^2)}{n-1}, \quad \sigma = \frac{1 - n\beta^2}{1-\beta^2}.$$

*When $\beta < n^{-1/2}$, we have*

$$\frac{|\mathcal{E}_+|}{|\mathcal{E}|} = n^{1/2} \left( \frac{n}{n-1} \right)^{(n-1)/2} \beta (1-\beta^2)^{(n-1)/2}.$$

Now we proceed with the main part of the proof. We construct a sequence of ellipsoids $\mathcal{E}_1, \mathcal{E}_2, \ldots$, all centered at the origin, such that $\mathcal{E}_k \supseteq \mathcal{C}$ and $|\mathcal{E}_k|/|\mathcal{E}_{k-1}| < e^{3/8}/2,\ k = 1, 2, \ldots$, and thus this sequence must terminate in

$$\log(L^{-n}) / \log(e^{3/8}/2) < 3.15 n \log L$$

steps. Suppose we have $\mathcal{E}_k \supseteq \mathcal{C}$ centered at the origin. Determine all the extreme points of $\mathcal{E}_k$ along its axes. Let these points be $\pm x_{k,i},\ i = 1, \ldots, n$, and then check whether $\frac{1}{2\sqrt{n}} x_{k,i} \in \mathcal{C}$ for $i = 1, \ldots, n$. If all these points are in $\mathcal{C}$, so is their convex hull, denoted by $\mathcal{H}$. Apparently, $\frac{1}{2\sqrt{n}} \mathcal{E}_k$ is the LJ ellipsoid of $\mathcal{H}$, and hence shrinking $\frac{1}{2\sqrt{n}} \mathcal{E}_k$ by a factor $\frac{1}{\sqrt{n}}$ makes it contained in $\mathcal{H} \subseteq \mathcal{C}$. We have $\frac{1}{2n} \mathcal{E}_k \subseteq \mathcal{C} \subseteq \mathcal{E}_k$. Now suppose that $\frac{1}{2\sqrt{n}} x_{k,i_k} \notin \mathcal{C}$ for some $i_k$ and the separation oracle returns $\mathcal{K}_k = \{x \in \mathbb{R}^n \mid -1 \leq g_k^T x \leq 1\}$ such that $\mathcal{C} \subseteq \mathcal{K}_k$ but $\frac{1}{2\sqrt{n}} x_{k,i_k} \notin \mathcal{K}_k$. Let $\mathcal{E}_{k+1}$ be the LJ ellipsoid of $\mathcal{E}_k \cap \mathcal{K}_k \supseteq \mathcal{C}$, which must be centered at the origin. Lemma 6 gives analytic formulas of $\mathcal{E}_{k+1}$ and $|\mathcal{E}_{k+1}|/|\mathcal{E}_k|$. Adopting the notation from Lemma 6, let $\mathcal{E}_k = \{x \in \mathbb{R}^n \mid x^T E_k^{-1} x \leq 1\}$ and we have

$$\begin{aligned} (g_k^T E_k g_k)^{1/2} &= \left[ g_k^T \left( \sum_{i=1}^n x_{k,i} x_{k,i}^T \right) g_k \right]^{1/2} \\ &\geq |g_k^T x_{k,i_k}| > 2\sqrt{n}. \end{aligned}$$

The last inequality comes from the fact that $\frac{1}{2\sqrt{n}} x_{k,i_k} \notin \mathcal{K}_k$. Therefore, we have $\beta = (g_k^T E_k g_k)^{-1/2} < \frac{1}{2\sqrt{n}}$, and

$$\frac{|\mathcal{E}_{k+1}|}{|\mathcal{E}_k|} < \frac{1}{2} \left( 1 + \frac{3}{4n-4} \right)^{(n-1)/2} < e^{3/8}/2.$$

Thus, our construction is valid. For each step, it takes at most $n$ calls to the separation oracle. Therefore, we need at most $3.15n^2 \log L$ calls to find a $2n$-rounding of $\mathcal{C}$. Computing the extreme

points of $\mathcal{E}_k$ requires an eigendecomposition, which takes $\mathcal{O}(n^3)$ time. Hence the total cost to find a $2n$-rounding is $3.15n^2 \log L$ calls and additional $\mathcal{O}(n^4 \log L)$ time. We note that rank-one updates can be used for computing the eigendecomposition of $\mathcal{E}_k$ for efficiency. See Gu and Eisenstat [35]. □

Applying Theorem 1 to the convex set $\mathcal{C} = \{x \mid \|Ax\|_p \leq 1\}$, with the separation oracle described via a subgradient of $\|Ax\|_p$ and the initial rounding provided by the "$R$" matrix from the QR decomposition of $A$, we improve the running time of the algorithm used by Clarkson [19] and by Dasgupta *et al.* [24] from $\mathcal{O}(mn^5 \log m)$ to $\mathcal{O}(mn^3 \log m)$ while maintaining an $\mathcal{O}(n)$-conditioning.

**Corollary 1.** *Given a matrix $A \in \mathbb{R}^{m \times n}$ with full column rank, it takes at most $\mathcal{O}(mn^3 \log m)$ time to find a matrix $R \in \mathbb{R}^{n \times n}$ such that $\kappa_p(AR^{-1}) \leq 2n$.*

*Proof.* This is a direct consequence of Theorem 1. We present the proof for the case $p < 2$. The proof for the case $p > 2$ is similar. Let $\mathcal{C} = \{x \in \mathbb{R}^n \mid \|Ax\|_p \leq 1\}$. For any $z \notin \mathcal{C}$, define $\mathcal{K}(z) = \{x \in \mathbb{R}^n \mid -1 \leq g(z)^T x \leq 1\}$. We have $\mathcal{K}(z) \supseteq \mathcal{C}$ and $z \notin \mathcal{K}(z)$, which gives the separation oracle. Let $A = QR_0$ be $A$'s QR factorization. We have,

$$\|R_0 x\|_2 = \|Ax\|_2 \leq \|Ax\|_p \leq n^{1/p-1/2}\|Ax\|_2 = n^{1/p-1/2}\|R_0 x\|_2, \quad \forall x \in \mathbb{R}^n,$$

which means $\mathcal{E}_0 = \mathcal{E}(0, R_0^{-1})$ gives an $n^{1/p-1/2}$-rounding of $\mathcal{C}$. Applying Theorem 1, we can find a $2n$-rounding of $\mathcal{C}$ in at most $3.15n^2 \log(m^{1/p-1/2})$ calls to the separation oracle. Let $\mathcal{E} = \mathcal{E}(0, E)$ be the ellipsoid that gives such rounding. We have

$$\|y\|_2 \leq \|AEy\|_p \leq 2n\|y\|_2, \quad \forall y \in \mathbb{R}^n .$$

The QR factorization takes $O(mn^2)$ time. Each call to the separation oracle takes $O(mn)$ time. Computing the extreme points of an ellipsoid takes $O(n^3)$ time. In total, we need $\mathcal{O}(mn^3 \log m)$ time. □

The algorithm of Corollary 1 for computing a $2n$-conditioning is not immediately applicable to very large matrices, which are usually distributively stored on secondary storage, since each call to the oracle requires a pass through the data. We can group $n$ calls together within a single pass, but we would still need $\mathcal{O}(n \log m)$ passes. We present a deterministic single-pass conditioning algorithm that balances the cost-performance trade-off to provide a $2n^{|2/p-1|+1}$-conditioning of $A$. See Algorithm 1. Our main result for Algorithm 1 is given in the following lemma.

**Lemma 7.** *Algorithm 1 is a $2n^{|2/p-1|+1}$-conditioning algorithm and it runs in $\mathcal{O}((mn^2 + n^4) \log m)$ time.*

*Proof.* We show the proof for $p \in [1, 2)$, while the proof for $p \in [2, \infty]$ follows similar arguments. The idea is to use block-wise reduction in $\ell_2$ norm and apply fast rounding to a small problem. The tool we need is simply the equivalence of vector norms. Let $\mathcal{C} = \{x \in \mathbb{R}^n \mid \|Ax\|_p \leq 1\}$, which is convex, full-dimensional, bounded, and centrally symmetric. Adopting notation from Algorithm 1,

**Algorithm 1** A single-pass conditioning algorithm.

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

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

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

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

4: Compute $\tilde{A}$'s SVD: $\tilde{A} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$.
5: Let $\mathcal{E}_0 = \mathcal{E}(0, E_0)$ where $E_0 = n^{\max\{1/p-1/2,0\}}\tilde{V}\tilde{\Sigma}^{-1}$.
6: Compute an ellipsoid $\mathcal{E} = \mathcal{E}(0, E)$ that gives a $2n$-rounding of $\tilde{\mathcal{C}}$ starting from $\mathcal{E}_0$ that gives an $(Mn^2)^{|1/p-1/2|}$-rounding of $\tilde{\mathcal{C}}$.
7: Return $n^{\min\{1/p-1/2,0\}}E$.

we first have

$$n^{1-2/p}\tilde{\mathcal{C}} \subseteq \mathcal{C} \subseteq \tilde{\mathcal{C}}$$

because for all $x \in \mathbb{R}^n$,

$$\|Ax\|_p^p = \sum_{i=1}^{M} \|A_i x\|_p^p \le n^{2-p} \sum_{i=1}^{M} \|A_i x\|_2^p = n^{2-p} \sum_{i=1}^{M} \|\tilde{A}_i x\|_2^p$$

and

$$\|Ax\|_p^p = \sum_{i=1}^{M} \|A_i x\|_p^p \ge \sum_{i=1}^{M} \|A_i x\|_2^p = \sum_{i=1}^{M} \|\tilde{A}_i x\|_2^p.$$

Next we prove that $\mathcal{E}_0$ gives an $(Mn^2)^{1/p-1/2}$-rounding of $\tilde{C}$. For all $x \in \mathbb{R}^n$, we have

$$\sum_{i=1}^{M} \|\tilde{A}_i x\|_2^p \le \sum_{i=1}^{M} \|\tilde{A}_i x\|_p^p = \|\tilde{A}x\|_p^p \le (Mn)^{1-p/2}\|\tilde{A}x\|_2^p = (Mn)^{1-p/2}\|\tilde{\Sigma}\tilde{V}^T x\|_2^p,$$

and

$$\sum_{i=1}^{M} \|\tilde{A}_i x\|_2^p \ge n^{p/2-1} \sum_{i=1}^{M} \|\tilde{A}_i x\|_p^p = n^{p/2-1}\|\tilde{A}x\|_p^p \ge n^{p/2-1}\|\tilde{A}x\|_2^p = n^{p/2-1}\|\tilde{\Sigma}\tilde{V}^T x\|_2^p.$$

Then by choosing $E_0 = n^{1/p-1/2}\tilde{V}\tilde{\Sigma}^{-1}$, we get

$$\|E_0^{-1}x\|_2 \le (\sum_{i=1}^{M} \|\tilde{A}_i x\|_2^p)^{1/p} \le (Mn^2)^{1/p-1/2}\|E_0^{-1}x\|_2$$

for all $x \in \mathbb{R}^n$ and hence $\mathcal{E}_0$ gives an $(Mn^2)^{1/p-1/2}$-rounding of $\tilde{\mathcal{C}}$. Since $n^{1-2/p}\tilde{\mathcal{C}} \subseteq \mathcal{C} \subseteq \tilde{\mathcal{C}}$, we

know that any $2n$-rounding of $\tilde{\mathcal{C}}$ is a $2n \cdot n^{2/p-1} = 2n^{2/p}$-rounding of $\mathcal{C}$. Therefore, Algorithm 1 computes a $2n^{2/p}$-conditioning of $A$. Note that the rounding procedure is applied to a problem of size $Mn \times n \approx m/n \times n$. Therefore, Algorithm 1 only needs a single pass through the data, with $\mathcal{O}((mn^2 + n^4) \log m)$ FLOPs. □

**Remark.** Solving the rounding problem of size $m/n \times n$ requires $\mathcal{O}(m)$ RAM, which might be too much for large-scale problems. In such cases, we can increase the block size from $n^2$ to, for example, $n^3$. This gives us a $2n^{|3/p-3/2|+1}$-conditioning algorithm that only needs $\mathcal{O}(m/n)$ RAM and $\mathcal{O}((mn+n^4) \log m)$ FLOPs for the rounding problem. The proof follows similar arguments. We can also replace SVD by the fast $\ell_2$ subspace embedding we introduce in next section, which reduces the overall running time to $\mathcal{O}((mn + n^4) \log(mn))$. However, it would lead to a non-deterministic result. How to balance those trade-offs depends on the underlying application.

## 2.5 Fast subspace embedding

Subspace embedding is the main ingredient of this work. Subspace embedding algorithms are critical building blocks for developing improved random sampling and random projection algorithms for common linear algebra problems. Usually, a subspace embedding algorithm is a randomized algorithm with a certain chance of failure. By "failure", we mean that it fails to produce an embedding with expected properties. There are many properties of interest in the analysis of subspace embedding algorithms. For example, given a subspace embedding algorithm, we may want to know: a) whether it is *oblivious* (independent of the input subspace), b) the time and storage it needs to construct an embedding, c) the time and storage to apply the embedding to an input matrix, d) the failure rate, e) the embedding dimension, f) the distortion of the embedding, as well as how to balance the trade-offs among those properties. We note that some of them may not be important for theoretical analysis but still affect practical performance.

Because $\ell_2$ space has some nice theoretical properties compared to other $\ell_p$ spaces, e.g., self-dual, which lead to better embedding algorithms and simpler analysis, we discuss $\ell_2$ subspace embeddings first, followed by general $\ell_p$ subspace embeddings and subspace-preserving sampling. At the end of this section, we show that instead of using subspace embedding for preconditioning, we can use them for solving $\ell_p$ regression problems via random sampling.

### 2.5.1 $\ell_2$ subspace embeddings

$\ell_2$ subspace embedding is closely related to the Johnson-Lindenstrauss (J-L) lemma.

**Lemma 8** (Johnson-Lindenstrauss lemma [42])**.** *Given $\epsilon \in (0,1)$, a point set $\mathcal{X}$ of $N$ points in $\mathbb{R}^m$, there is a linear map $\phi : \mathbb{R}^m \hookrightarrow \mathbb{R}^s$ with $s = C \log N/\epsilon^2$, where $C > 0$ is a global constant, such that*

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

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

The original proof of the J-L lemma is done by constructing a projection from $\mathbb{R}^m$ to a randomly chosen $s$-dimensional subspace. Basically, the projection is represented by a random orthonormal matrix in $\mathbb{R}^{s\times m}$. In [40], Indyk and Motwani show 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, which simplifies the construction of a J-L transform. Later, Achlioptas [1] show 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, though still at the same order as the random normal projection. The breakthrough is from Ailon and Chazelle [2]. They construct so-called fast Johnson-Lindenstrauss transform (FJLT), which is a product of three matrices $\Phi = PHD$, where $P \in \mathbb{R}^{s\times m}$ is a sparse J-L transform with approximately $\mathcal{O}(s\log^2 N)$ nonzeros, $H \in \mathbb{R}^{m\times m}$ is a normalized Walsh-Hadamard matrix, and $D \in \mathbb{R}^{m\times m}$ is a diagonal matrix with its diagonals drawn independently from $\{-1, 1\}$ with probability 1/2. Because multiplying $H$ with a vector can be done in $\mathcal{O}(m\log m)$ time using a FFT-like algorithm, it reduces the projection time from $\mathcal{O}(sm)$ to $\mathcal{O}(m\log m)$. This FJLT construction is further simplified by Ailon and Liberty [3, 4].

The J-L lemma applies to an arbitrary set of $N$ vectors in $\mathbb{R}^m$. By using an $\epsilon$-net argument and triangle inequality, Sarlós [63] shows that a J-L transform can also preserve the Euclidean geometry of an entire $n$-dimensional subspace of vectors in $\mathbb{R}^m$, with embedding dimension $\mathcal{O}(n\log(n/\epsilon)/\epsilon^2)$.

**Lemma 9** (Sarlós [63])**.** *Let $\mathcal{A}$ be an arbitrary $n$-dimensional subspace of $\mathbb{R}^m$ and $0 \le \epsilon, \delta < 1$. If $\Phi$ is a J-L transform from $\mathbb{R}^m$ to $\mathcal{O}(n\log(n/\epsilon)/\epsilon^2 \cdot f(\delta))$ dimensions for some function $f$. Then*

$$\Pr(\forall x \in \mathcal{A} : |\|x\|_2 - \|\Phi x\|_2| \le \epsilon\|x\|_2) \ge 1 - \delta.$$

This result applies to any J-L transform. However, for some J-L transforms, we are able to obtain more refined results by bounding the spectral norm of $(\Phi U)^T(\Phi U) - I$, where $U$ is an orthonormal basis of $\mathcal{A}_2$. For example, if $\|(\Phi U)^T(\Phi U) - I\| \le \epsilon$, for any $x \in \mathcal{A}_2$, we have

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

and hence

$$|\|\Phi x\|_2 - \|x\|_2| \le \frac{\epsilon\|x\|_2^2}{\|\Phi x\|_2 + \|x\|_2} \le \epsilon\|x\|_2.$$

We show some results following this approach. First consider the a random normal matrix, which has the following concentration result on its extreme singular values.

**Lemma 10.** (Davidson and Szarek [25]) *Consider an $s \times n$ random matrix $G$ with $s > n$, whose entries are independent random variables following the standard normal distribution. Let the singular values be $\sigma_1 \ge \cdots \ge \sigma_n$. Then for any $t > 0$,*

$$\max\left\{\Pr(\sigma_1 \ge \sqrt{s} + \sqrt{n} + t), \Pr(\sigma_n \le \sqrt{s} - \sqrt{n} - t)\right\} < e^{-t^2/2}. \tag{2.1}$$

Using this concentration result, we can easily present a better analysis of random normal projection than in Lemma 9.

**Lemma 11.** *Given an $n$-dimensional subspace $\mathcal{A}_2 \in \mathbb{R}^m$ and $\epsilon, \delta \in (0,1)$, let $G \in \mathbb{R}^{s\times m}$ be a random matrix whose entries are independently drawn from the standard normal distribution. There exist $s = \mathcal{O}((\sqrt{n} + \log(1/\delta))^2/\epsilon^2)$ such that, with probability at least $1-\delta$, we have*

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

*Proof.* Let $U$ be an orthonormal basis of $\mathcal{A}_2$. Because $G$'s entries are i.i.d. normal variables, $G_1 = GU \in \mathbb{R}^{s\times n}$ is also a random matrix with i.i.d. normal entries. Applying Lemma 10, we get for any $t > 0$,

$$\max\left\{\Pr(\sigma_1(G_1) \geq \sqrt{s} + \sqrt{n} + t),\ \Pr(\sigma_n(G_1) \leq \sqrt{s} - \sqrt{n} - t)\right\} < e^{-t^2/2}.$$

It is easy to derive that if $s \geq \frac{6(\sqrt{n}+t)^2}{\epsilon^2}$ and $t = \sqrt{2}\log(2/\delta)$ for some $\delta \in (0,1)$,

$$\Pr(\|s^{-1}\cdot G_1^T G_1 - I\|_2 > \epsilon) < \delta,$$

which concludes the proof because $\|s^{-1}\cdot G_1^T G_1 - I\|_2 \leq \epsilon$ implies that $G$ preserves the $\ell_2$ norms of the entire subspace of vectors with distortion $1 \pm \epsilon$. □

Another example is from Tropp [67], who proves that a variant of the FJLT, named subsampled randomized Hadamard transform (SRHT) can preserve the geometry of an entire $\ell_2$ subspace of vectors by using a matrix Chernoff inequality to bound $\|(\Phi U)^T(\Phi U) - I\|_2$.

**Definition 9** (SRHT, Tropp [67])**.** *An SRHT is an $s \times m$ matrix of the form*

$$\Phi = \sqrt{\frac{m}{s}} RHD,$$

*where*

- *$D \in \mathbb{R}^{m\times m}$ is a diagonal matrix whose entries are independent random signs,*
- *$H \in \mathbb{R}^{m\times m}$ is a Walsh-Hadamard matrix scaled by $m^{-1/2}$,*
- *$R \in \mathbb{R}^{s\times m}$ restricts an $n$-dimensional vector to $s$ coordinates, chosen uniformly at random.*

**Lemma 12** (Tropp [67])**.** *Fix an $m \times n$ matrix $U$ with orthonormal columns, and draw a random $s \times m$ SRHT matrix $\Phi$ where the embedding dimension $s$ satisfies*

$$4\left(\sqrt{n} + \sqrt{8\log(mn)}\right)^2 \log(n) \leq s \leq m.$$

*Then, except with probability $\mathcal{O}(n^{-1})$,*

$$0.40 \leq \sigma_n(\Phi U) \text{ and } \sigma_1(\Phi U) \leq 1.48.$$

Dense J-L transforms, e.g., random normal projection and its variants, use matrix-vector multiplication for the embedding. Given a matrix $A \in \mathbb{R}^{m\times n}$, computing $\tilde{A} = \Phi A$ takes $\mathcal{O}(\text{nnz}(A) \cdot s)$ time when $\Phi$ is a dense matrix of size $s \times m$ and $\text{nnz}(A)$ is the number of non-zero elements in $A$. Fast J-L transforms, e.g, FJLT and SRHT, use FFT-like algorithms for the embedding, which lead to $\mathcal{O}(m \log m)$ time for each projection. Hence, computing $\tilde{A} = \Phi A$ takes $\mathcal{O}(mn \log m)$ time when $\Phi$ is a fast J-L transform. Though called "fast", a fast transform might be slower than a dense transform when $\text{nnz}(A) = \mathcal{O}(m)$ or $A$'s columns are distributively stored that slows down FFT-like algorithms. We explore more along this direction in Chapter 3.

Given the long history of development, it is a significant result when Clarkson and Woodruff [21] developed an algorithm for the $\ell_2$ subspace embedding that runs in *input-sparsity* time, i.e., in $\mathcal{O}(\text{nnz}(A))$ time. Surprisingly, their construction is exactly the CountSketch matrix in the data stream literature [18], which is an extremely simple and sparse matrix. It can be written as the product of two matrices $\Phi = SD \in \mathbb{R}^{s\times m}$, where $S \in \mathbb{R}^{s\times m}$ has each column chosen independently and uniformly from the $s$ standard basis vectors of $\mathbb{R}^s$ and $D \in \mathbb{R}^{m\times m}$ is a diagonal matrix with diagonal entries chosen independently and uniformly from $\pm 1$.

**Theorem 2** (Clarkson and Woodruff [21])**.** *Given an $n$-dimensional subspace $\mathcal{A} \subset \mathbb{R}^m$ and $\epsilon \in (0, 1)$, there is $s = \mathcal{O}((n/\epsilon)^4 \log^2(n/\epsilon))$ such that with probability at least 0.9,*

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

*where $\Phi$ is the CountSketch matrix described above.*

Their proof is done by decoupling $\mathcal{A}$ into two orthogonal subspaces, called "heavy" and "light" based on the row norms of $U$, an orthonormal basis of $\mathcal{A}$. Therefore any $x \in \mathcal{A}$ can be written as the sum of two vectors $x = x^H + x^L$, where $x^H$ belongs to the heavy subspace and $x^L$ belongs to the light. For any $x \in \mathcal{A}$, we have

$$\|\Phi x\|_2^2 = \|\Phi x^H\|_2^2 + \|\Phi x^L\|_2^2 + 2(\Phi x^H)^T \Phi x^L.$$

They show that each term can be bounded separately to get the $1 \pm \epsilon$ distortion on the entire subspace of vectors. We refer readers to [21] for more details of the proof. Here, we present an improved result for the input-sparsity time $(1 \pm \epsilon)$-distortion embedding of [21]. In particular, for the same embedding procedure, we obtain improved bounds for the embedding dimension with a much simpler analysis. Our proof is similar to the proofs of Lemmas 11 and 12. See also Nelson and Nguyen [53] for a similar result with a slightly better constant.

**Theorem 3** (Input-sparsity time embedding for $\ell_2$)**.** *Given an $n$-dimensional subspace $\mathcal{A} \subset \mathbb{R}^m$ and any $\delta \in (0, 1)$, let $s = (n^2 + n)/(\epsilon^2\delta)$. Then, with probability at least $1 - \delta$,*

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

*where $\Phi$ is the CountSketch matrix.*

*Proof.* Let the $m \times n$ matrix $U$ be an orthonormal basis for the range of the $m \times n$ matrix $A$. Rather than proving the theorem by establishing that

$$(1-\epsilon)\|Uz\|_2 \leq \|\Phi Uz\|_2 \leq (1+\epsilon)\|Uz\|_2$$

holds for all $z \in \mathbb{R}^n$, as is essentially done in, e.g., [30] and [21], we note that $U^T U = I_n$, and we directly bound the extent to which the embedding process perturbs this product. To do so, define

$$X = (\Phi U)^T(\Phi U) = U^T D^T S^T S D U.$$

That is,

$$x_{kl} = \sum_{i=1}^{s}\left(\sum_{j=1}^{m} s_{ij}d_j u_{jk}\right)\left(\sum_{j=1}^{m} s_{ij}d_j u_{jl}\right), \quad k,l \in \{1,\ldots,n\},$$

where $s_{ij}$ is the $(i,j)$-th element of $S$, $d_j$ is the $j$-th diagonal element of $D$, and $u_{jk}$ is the $(j,k)$-th element of $U$. We use the following facts in the proof:

$$\mathbf{E}[d_{j_1}d_{j_2}] = \delta_{j_1 j_2},$$

$$\mathbf{E}[s_{i_1 j_1}s_{i_2 j_2}] = \begin{cases} \frac{1}{s^2} & \text{if } j_1 \neq j_2, \\ \frac{1}{s} & \text{if } i_1 = i_2, j_1 = j_2, \\ 0 & \text{if } i_1 \neq i_2, j_1 = j_2. \end{cases}$$

We have,

$$\mathbf{E}[x_{kl}] = \sum_i \sum_{j_1,j_2} \mathbf{E}[s_{ij_1}d_{j_1}u_{j_1 k} \cdot s_{ij_2}d_{j_2}u_{j_2 l}] = \sum_i \sum_j \mathbf{E}[s_{ij}u_{jk}u_{jl}] = \sum_j u_{jk}u_{jl} = \delta_{kl},$$

and we also have

$$\begin{aligned}
\mathbf{E}[x_{kl}^2] &= \mathbf{E}\left[\left(\sum_i\left(\sum_j s_{ij}d_j u_{jk}\right)\left(\sum_j s_{ij}d_j u_{jl}\right)\right)^2\right] \\
&= \sum_{i_1,i_2}\mathbf{E}\left[\left(\sum_j s_{i_1 j}d_j u_{jk}\right)\left(\sum_j s_{i_1 j}d_j u_{jl}\right)\left(\sum_j s_{i_2 j}d_j u_{jk}\right)\left(\sum_j s_{i_2 j}d_j u_{jl}\right)\right] \\
&= \sum_{i_1,i_2}\sum_{j_1,j_2,j_3,j_4}\mathbf{E}[s_{i_1 j_1}d_{j_1}u_{j_1 k} \cdot s_{i_1 j_2}d_{j_2}u_{j_2 l} \cdot s_{i_2 j_3}d_{j_3}u_{j_3 k} \cdot s_{i_2 j_4}d_{j_4}u_{j_4 l}]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i_1,i_2} \left( \sum_j \mathbf{E}[s_{i_1 j} u_{jk} \cdot s_{i_1 j} u_{jl} \cdot s_{i_2 j} u_{jk} \cdot s_{i_2 j} u_{jl}] \right. \\
&\qquad + \sum_{j_1 \neq j_2} \mathbf{E}[s_{i_1 j_1} u_{j_1 k} \cdot s_{i_1 j_1} u_{j_1 l} \cdot s_{i_2 j_2} u_{j_2 k} \cdot s_{i_2 j_2} u_{j_2 l}] \\
&\qquad + \sum_{j_1 \neq j_2} \mathbf{E}[s_{i_1 j_1} u_{j_1 k} \cdot s_{i_1 j_2} u_{j_2 l} \cdot s_{i_2 j_1} u_{j_1 k} \cdot s_{i_2 j_2} u_{j_2 l}] \\
&\qquad \left. + \sum_{j_1 \neq j_2} \mathbf{E}[s_{i_1 j_1} u_{j_1 k} \cdot s_{i_1 j_2} u_{j_2 l} \cdot s_{i_2 j_2} u_{j_2 k} \cdot s_{i_2 j_1} u_{j_1 l}] \right) \\
&= \sum_j u_{jk}^2 u_{jl}^2 + \sum_{j_1 \neq j_2} u_{j_1 k} u_{j_1 l} u_{j_2 k} u_{j_2 l} + \frac{1}{s} \sum_{j_1 \neq j_2} u_{j_1 k}^2 u_{j_2 l}^2 + \frac{1}{s} \sum_{j_1 \neq j_2} u_{j_1 k} u_{j_2 l} u_{j_2 k} u_{j_1 l} \\
&= \left( \sum_j u_{jk} u_{jl} \right)^2 + \frac{1}{s} \left( \left( \sum_j u_{jk}^2 \right) \left( \sum_j u_{jl}^2 \right) + \left( \sum_j u_{jk} u_{jl} \right)^2 - 2 \sum_j u_{jk}^2 u_{jl}^2 \right) \\
&= \begin{cases} 1 + \frac{2}{s}(1 - \|U_{*k}\|_4^4) & \text{if } k = l, \\ \frac{1}{s}(1 - 2\langle U_{*k}^2, U_{*l}^2 \rangle) & \text{if } k \neq l. \end{cases}
\end{aligned}
$$

Given these results, it is easy to obtain that

$$
\begin{aligned}
\mathbf{E}[\|X - I\|_F^2] = \sum_{k,l} \mathbf{E}[(x_{kl} - \delta_{kl})^2] &= \frac{2}{s} \left( \sum_k (1 - \|U_{*k}\|_4^4) + \sum_{k<l} (1 - 2\langle U_{*k}^2, U_{*l}^2 \rangle) \right) \\
&\leq \frac{n^2 + n}{s}.
\end{aligned}
$$

For any $\delta \in (0, 1)$, set $s = (n^2 + n)/(\epsilon^2 \delta)$. Then, by Markov's inequality,

$$
\mathbf{Pr}[\|X - I\|_F \geq \epsilon] = \mathbf{Pr}[\|X - I\|_F^2 \geq \epsilon^2] \leq \frac{n^2 + n}{\epsilon^2 s} = \delta.
$$

Therefore, with probability at least $1 - \delta$, we have $\|X - I\|_2 \leq \|X - I\|_F \leq \epsilon$, which implies

$$
(1 - \epsilon)\|Uz\|_2 \leq \|\Phi U z\|_2 \leq (1 + \epsilon)\|Uz\|_2, \quad \forall z \in \mathbb{R}^n,
$$

which is equivalent to the result stated in the theorem. □

The construction of $\Phi$ in this theorem is the same as the construction in Clarkson and Woodruff [21]. For them, $s = \mathcal{O}((n/\epsilon)^4 \log^2(n/\epsilon))$ in order to achieve $(1 \pm \epsilon)$ distortion with a constant probability. Theorem 3 shows that it actually suffices to set $s = \mathcal{O}((n^2 + n)/\epsilon^2)$, and, surprisingly, the proof is rather simple. We note that the improved result of Theorem 3 propagate to related applications, e.g., to the $\ell_2$ regression problem, the low-rank matrix approximation problem, and the problem of computing approximations to the $\ell_2$ leverage scores as considered in [21].

**Remark.** It is easy to see that computing $\Phi A$ takes $\mathcal{O}(\text{nnz}(A))$ time. The $\mathcal{O}(\text{nnz}(A))$ running time is indeed optimal, up to constant factors, for general inputs. Consider the case when $A$ has an important row $a_i$ such that $A$ becomes rank-deficient without it. Thus, we have to observe $a_i$ in order to compute a low-distortion embedding. However, without any prior knowledge, we have to scan at least a constant portion of the input to guarantee that $a_i$ is observed with a constant probability, which takes $\mathcal{O}(\text{nnz}(A))$ time. Also note that this optimality result applies to general $\ell_p$ norms.

### 2.5.2 Low-distortion $\ell_1$ subspace embeddings

General $\ell_p$ subspace embedding is quite different from $\ell_2$ subspace embedding. We elaborate the differences in Chapter 4, while in this section we briefly introduce some existing results on $\ell_1$ subspace embedding. For $\ell_1$, the first question to ask is whether there exist a J-L transform equivalent. This question is answered by Charikar and Sahai [17]. However, the answer is negative.

**Lemma 13** (Charikar and Sahai [17])**.** *There exists a set of $\mathcal{O}(m)$ points in $\ell_1^m$ such that any linear embedding into $\ell_1^s$ has distortion at least $\sqrt{m/s}$. The trade-off between dimension and distortion for linear embeddings is tight up to a logarithmic factor. There exists a linear embedding of any set of $N$ points in $\ell_1^m$ to $\ell_1^{s'}$ where $s' = \mathcal{O}(s \log N)$ and the distortion is $\mathcal{O}(\sqrt{m/s})$.*

This result shows that linear embeddings are particularly bad in $\ell_1$, compared to the J-L lemma for $\ell_2$. To obtain a constant distortion, we need $s \geq Cm$ for some constant $C$. So the embedding dimension cannot be independent of $m$. However, the negative result is obtained by considering arbitrary point sets. In many applications, we are dealing with structured point sets, e.g., vectors from a low-dimensional subspace. Sohler and Woodruff [64] give the first linear oblivious embedding of a $n$-dimensional subspace of $\ell_1^m$ into $\ell_1^{\mathcal{O}(n \log n)}$ with distortion $\mathcal{O}(n \log n)$, where both the embedding dimension and the distortion are independent of $m$.

**Lemma 14** (Cauchy transform (CT), Sohler and Woodruff [64])**.** *Let $\mathcal{A}_1$ be an arbitrary $n$-dimensional linear subspace of $\ell_1^m$. Then there is an $s_0 = s_0(n) = \mathcal{O}(n \log n)$ and a sufficiently large constant $C_0 > 0$, such that for any $s$ with $s_0 \leq s \leq n^{\mathcal{O}(1)}$, and any constant $C \geq C_0$, if $\Phi \in \mathbb{R}^{s \times m}$ is a random matrix whose entries are choose independently from the standard Cauchy distribution and are scaled by $C/s$, then with probability at least 0.99,*

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

The proof is by constructing tail inequalities for the sum of half Cauchy random variables. We refer readers to [64] for more details. We note that the construction here is quite similar to the construction of the dense Gaussian embedding in Lemma 11. The differences are that a) Cauchy random variables replace standard normal random variables, b) larger embedding dimension does not always lead to better distortion, c) the failure rate becomes harder to control. We show the underlying connection between Gaussian distribution and Cauchy distributions in Chapter 4.

As CT is the $\ell_1$ counterpart of the dense Gaussian transform, the Fast Cauchy Transform (FCT) proposed by Clarkson et al. [20] is the $\ell_1$ counterpart of FJLT. This FCT construction first pre-processes by a deterministic low-coherence matrix, then rescales by Cauchy random variables, and finally samples linear combinations of the rows. Then, we construct $\Phi$ as

$$\Phi = 4BCH,$$

where

- $B \in \mathbb{R}^{s \times 2m}$ has each column chosen independently and uniformly from the $s$ standard basis vectors for $\mathbb{R}^s$; for $\alpha$ sufficiently large, we set the parameter $s = \alpha n \log(n/\delta)$, where $\delta \in (0,1)$ controls the probability that our algorithm fails and $\alpha$ is a suitably large constant,
- $C \in \mathbb{R}^{2m \times 2m}$ is a diagonal matrix with diagonal entries chosen independently from a Cauchy distribution,
- $H \in \mathbb{R}^{2m \times m}$ is a block-diagonal matrix comprised of $m/t$ blocks along the diagonal. Each block is the $2t \times t$ matrix $G_s = \binom{H_t}{I_t}$, where $I_t$ is the $t \times t$ identity matrix, and $H_t$ is the normalized Hadamard matrix. (For simplicity, we assume $t$ is a power of two and $m/t$ is an integer.)

$$H = \begin{pmatrix} G_s & & & \\ & G_s & & \\ & & \ddots & \\ & & & G_s \end{pmatrix}.$$

Heuristically, the effect of $H$ in the above FCT construction is to spread the weight of a vector, so that $Hy$ has many entries that are not too small. This means that the vector $CHy$ comprises Cauchy random variables with scale factors that are not too small; and finally these variables are summed up by $B$, yielding a vector $BCHy$, whose $\ell_1$ norm won't be too small relative to $\|y\|_1$.

**Lemma 15** (Fast Cauchy Transform (FCT), Clarkson et al. [20])**.** *There is a distribution (given by the above construction) over matrices $\Phi \in \mathbb{R}^{s \times n}$, with $s = \mathcal{O}(n \log n + n \log(1/\delta))$, such that for an arbitrary (but fixed) $A \in \mathbb{R}^{m \times n}$, and for all $x \in \mathbb{R}^n$, the inequalities*

$$\|Ax\|_1 \leq \|\Phi Ax\|_1 \leq \kappa \|Ax\|_1$$

*hold with probability $1 - \delta$, where*

$$\kappa = \mathcal{O}\left(\frac{n\sqrt{t}}{\delta} \log(sn)\right).$$

*Further, for any $y \in \mathbb{R}^n$, the product $\Phi y$ can be computed in $\mathcal{O}(n \log s)$ time.*

Setting $\delta$ to a small constant, since $\sqrt{t} = s^3$ and $s = \mathcal{O}(n \log n)$, it follows that $\kappa = O(n^4 \log^4 n)$ in the above theorem. We note that the 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 at the cost of worse distortion.

In Chapter 4, we follow this line of work and develop low-distortion $\ell_p$ subspace embedding algorithms in input-sparsity time for $p \in [1, 2)$, the $\ell_p$ equivalent of the input-sparsity time $\ell_2$ embedding proposed by Clarkson and Woodruff [21].

## 2.6 Subspace-preserving sampling

All of the linear subspace embedding algorithms mentioned in previous sections are oblivious, i.e., independent of the input subspace. Since oblivious embedding is not a hard constraint for the $\ell_p$ regression problems we are interested in, would non-oblivious embeddings be able to give better performance?

Clarkson [19] first shows that the $\ell_1$ subspace embedding can be done by weighted sampling of the coordinates. The weighted sampling is done after preprocessing, including preconditioning using ellipsoidal rounding. Sampled coordinates are chosen with probabilities that depend on the $\ell_1$ norms of the rows of the preconditioned matrix. Moreover, the resulting sample has each coordinate weighted by the reciprocal of its sampling probability. Different from oblivious $\ell_1$ subspace embeddings, the sampling approach can achieve a much better distortion.

**Lemma 16** (Clarkson [19])**.** *Given an $n$-dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^m$ and $\epsilon, \delta \in (0, 1)$, it takes $\mathcal{O}(mn^5 \log m)$ time to compute a diagonal matrix $S \in \mathbb{R}^{m \times m}$ with $\mathcal{O}(n^{3.5} \log(n/(\delta\epsilon))/\epsilon^2)$ non-zero diagonals such that, with probability at least $1 - \delta$,*

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

Therefore, to estimate the $\ell_1$ norms of any vector from a $n$-dimensional subspace of $\mathbb{R}^m$, we only need to compute the weighted sum of the absolute values of a few coordinates of this vector. This subset of the sampled coordinates is called a *coreset* for the $\ell_1$ norm estimation problem. The existence of coresets has many applications, especially in large-scale data analysis, for example, clustering [14] and set covering [13]. A similar sampling algorithm for the $\ell_2$ norm is developed by Drineas et al. [30]. The algorithm of [30] relies on the estimation of the row norms of an orthonormal basis of the subspace. The results for $\ell_1$ and $\ell_2$ are generalized to $\ell_p$ by Dasgupta et al. [24], where this family of sampling algorithms are called *subspace-preserving sampling*.

**Theorem 4** (Subspace-preserving sampling, Dasgupta et al. [24])**.** *Given an $n$-dimensional subspace $\mathcal{A}_p \subset \ell_p^m$ represented by a matrix $A \in \mathbb{R}^{m \times n}$ and $p \in [1, \infty)$, $\epsilon \in (0, 1/7)$, and $\delta \in (0, 1)$, choose*

$$s \geq 16(2^p + 2)\bar{\kappa}_p^p(A)(n \log(12/\epsilon) + \log(2/\delta))/(p^2\epsilon^2),$$

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

$$s_{ii} = \begin{cases} 1/p_i^{1/p} & \text{with probability } p_i, \\ 0 & \text{otherwise,} \end{cases} \qquad i = 1, \ldots, m,$$

*where*

$$p_i \geq \min\left\{1, s \cdot \|a_i\|_p^p / |A|_p^p\right\}, \quad i = 1, \ldots, m.$$

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

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

If the matrix $A$ is explicitly given, computing the sampling probabilities takes $\mathcal{O}(\text{nnz}(A))$ time, constructing the sampling matrix $S$ takes $\mathcal{O}(m)$ time, and computing $Sy$ for any $y \in \mathcal{A}_p$ only takes $\mathcal{O}(s)$ time (assuming random access to $y$'s elements). The running time seems to be very promising. However, the sampling complexity $s$ depends on $\bar{\kappa}_p(A)$, the condition number of $A$, which could be arbitrarily large. To control the sample size, preconditioning is necessary. That is, finding a matrix $R \in \mathbb{R}^{n\times n}$ such that $\bar{\kappa}_p(AR^{-1}) = \mathcal{O}(\text{poly}(n))$, which could be done by the preconditioning algorithms introduced in Section 1.4. Assume that such a preconditioner matrix $R$ is given and let $U = AR^{-1}$. The sampling matrix obtained by applying Theorem 4 to $U$ also works for $A$ because $A$ and $U$ share the same subspace, while the sample size is now bounded by $\mathcal{O}(\text{poly}(n)\log(1/\epsilon)/\epsilon^2)$. However, computing the row norms of $U$ takes $\mathcal{O}(\text{nnz}(A)\cdot n)$ time because of computing $AR^{-1}$, which is much more expensive than computing the row norms of $A$ that takes $\mathcal{O}(\text{nnz}(A))$ time. Clarkson et al. [20] improve this running time by estimating the row norms of $AR^{-1}$ instead of computing them exactly to define importance sampling probabilities.

**Lemma 17** (Fast subspace-preserving sampling (Clarkson et al. [20]))**.** *Given a matrix* $A \in \mathbb{R}^{m\times n}$*,* $p \in [1, \infty)$*,* $\epsilon > 0$*, and a matrix* $R \in \mathbb{R}^{n\times n}$ *such that* $AR^{-1}$ *is well-conditioned, it takes* $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ *time to compute a sampling matrix* $S \in \mathbb{R}^{s'\times m}$ *(with only one nonzero element per row) with* $s' = \mathcal{O}(\bar{\kappa}_p^p(AR^{-1})n^{|p/2-1|+1}\log(1/\epsilon)/\epsilon^2)$ *such that with a constant probability,*

$$(1-\epsilon)\|Ax\|_p \leq \|SAx\|_p \leq (1+\epsilon)\|Ax\|_p, \quad \forall x \in \mathbb{R}^n\,.$$

*Proof.* Instead of computing $U = AR^{-1}$ explicitly, we compute $\tilde{U} = A(R^{-1}G)$, where $G \in \mathbb{R}^{n\times l}$ is a random matrix whose entries are i.i.d. standard normal variables scaled by $l^{-1/2}$ with $l = \mathcal{O}(\log m)$ such that by the J-L lemma, with a constant probability, e.g., 0.99, we have the following

$$\frac{1}{2}\|u_i\|_2 \leq \|\tilde{u}_i\|_2 \leq \frac{3}{2}\|u_i\|_2, \quad i = 1, \ldots, m. \tag{2.2}$$

Note that computing $\tilde{U}$ takes $\mathcal{O}(\text{nnz}(A)\cdot\log m)$ time. Let $s$ be the sampling complexity required by Theorem 4 for $U = AR^{-1}$ and set $s' = 3^p n^{|p/2-1|}s$. Define sampling probabilities based on the $\ell_2$

row norm estimates:

$$p_i = \min\left\{1, s' \cdot \frac{\|\tilde{u}_i\|_2^p}{\sum_{j=1}^m \|\tilde{u}_i\|_2^p}\right\}, \quad i = 1, \ldots, m.$$

We have $p_i \geq s\|u_i\|_p^p/|U|_p^p$ for $i = 1, \ldots, m$ because

$$\frac{\|\tilde{u}_i\|_2^p}{\sum_{j=1}^m \|\tilde{u}_i\|_2^p} \geq 3^{-p}\frac{\|u_i\|_2^p}{\sum_{j=1}^m \|u_i\|_2^p} \geq 3^{-p}n^{-|p/2-1|}\frac{\|u_i\|_p^p}{\sum_{j=1}^m \|u_i\|_p^p} = 3^{-p}n^{-|p/2-1|}\frac{\|u_i\|_p^p}{|U|_p^p},$$

where the first inequality is due to (2.2) and the second inequality is due to the equivalence of vector norms. Therefore, the sampling probabilities $\{p_i\}$ meet the requirement in Theorem 4 and hence the sampling matrix constructed based on them is a subspace-preserving sampling matrix with a constant probability. □

We note that the speed-up comes at the cost of increased sampling complexity, which does not affect the theory much because the sampling complexity is still $\mathcal{O}(\text{poly}(n)\log(1/\epsilon)/\epsilon^2)$. In practice, it might be worth computing $U = AR^{-1}$ and its row norms explicitly to obtain a smaller sample size. One should be aware of this trade-off when implementing a subspace-preserving sampling algorithm.

## 2.7 Application to $\ell_p$ regression

As mentioned in Section 1.4, subspace embedding algorithms can be used for creating preconditioner matrices for traditional solvers. Indeed, $(1 \pm \epsilon)$-distortion subspace embedding algorithms can be used directly to approximate $\ell_p$ regression problems, and this is the main application of subspace embedding in [19, 30, 24]. In these work, an approximate solution to an $\ell_p$ regression problem specified by $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ is computed in several steps:

1. compute a preconditioner matrix $R$ for $A$,
2. compute a constant-factor approximate solution $x_C$,
3. construct a sampling matrix $S$ based on sampling probabilities depending on $AR^{-1}$ and the residual $b - Ax_C$ such that

$$\|S(Ax - b)\|_p \leq (1+\epsilon)\|Ax - b\|_p, \quad \forall x \in \mathbb{R}^n,$$

4. solve the subsampled problem $\hat{x} = \arg\min_{x\in\mathbb{R}^n} \|S(Ax - b)\|_p$, which is a $(1+\epsilon)$-approximate solution to the original problem because

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

We refer readers to the corresponding work for more details. Actually, with constrained formulation (1.4) and the result from Lemma 5, the step of computing the constant-factor approximate

solution becomes unnecessary and it is straightforward to compute a $\frac{1+\epsilon}{1-\epsilon}$-approximate solution to an $\ell_p$ regression problem. The simplified procedure has the following steps:

1. compute a preconditioner matrix $R$ for $A$,
2. construct a sampling matrix $S$ based on sampling probabilities depending on $AR^{-1}$ such that

$$\|SAx\|_p \leq (1+\epsilon)\|Ax\|_p, \quad \forall x \in \mathbb{R}^n,$$

3. solve the sub-sampled problem $\hat{x} = \arg\min_{c^T x=1} \|SAx\|_p$, which is a $\frac{1+\epsilon}{1-\epsilon}$-approximate solution to the original problem.

Recall that for $\ell_2$, J-L transforms can provide subspace embeddings with distortion $(1 \pm \epsilon)$, and for general $\ell_p$ with $p \in [1, \infty)$, subspace-preserving sampling algorithms can provide subspace embeddings with distortion $(1 \pm \epsilon)$. Therefore, an $\ell_p$ regression problem can be approximated by solving a reduced-sized $\ell_p$ regression problem obtained from subspace-preserving embedding.

## 2.8 Summary

By combining the tools we have introduced in this chapter and traditional solvers in the previous chapter, we can propose several approaches to compute a $(1 \pm \epsilon)$-approximate solution to an $\ell_p$ regression problem, which are reflected in the literature. For $\ell_2$, we might use

1. **a stable direct method**
   e.g., SVD or complete orthogonal factorization [33],
2. **preconditioning + an unstable direct method**
   e.g., preconditioning via Gaussian transform + the normal equation approach [22],
3. **preconditioning + an iterative methods**
   e.g., preconditioning via FJLT + LSQR [7, 62],
4. **subspace-preserving embedding + solving the reduced-sized problem**
   e.g., input-sparsity time embedding + (FJLT + SVD) [21],
5. **preconditioning + subspace-preserving sampling + solving the subsampled problem**
   e.g., preconditioning via FJLT + subspace-preserving sampling + SVD [31].

In the third approach, if we use SRHT (Lemma 12) 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 $\mathcal{O}(mn\log(m/\epsilon) + n^3 \log n)$, where $\mathcal{O}(mn\log(m))$ comes from SRHT, $\mathcal{O}(n^3 \log n)$ from computing the preconditioner matrix, and $\mathcal{O}(mn\log(1/\epsilon))$ from LSQR iterations. Then, in the fourth approach, if we use the input-sparsity time algorithm from Clarkson

and Woodruff [21] for embedding and the SRHT + LSQR approach above to solve the reduced-sized problem, the total running time would be just the number of nonzeros in $A$ plus a low-degree polynomial of $n/\epsilon$.

**Theorem 5** (Fast $\ell_2$ regression)**.** *With a constant probability, a $(1+\epsilon)$-approximate solution to an $\ell_2$ regression problem specified by $A \in \mathbb{R}^{m\times n}$ using the constrained formulation can be computed in $\mathcal{O}(\mathrm{nnz}(A) + n^3 \log(n/\epsilon)/\epsilon^2)$ time.*

*Proof.* The proof is basically combining Theorem 3 with the result of the SRHT + LSQR approach. First compute a $(1 \pm \frac{\epsilon}{4})$-distortion embedding $\Phi \in \mathbb{R}^{s\times m}$ in $\mathcal{O}(\mathrm{nnz}(A))$ time with embedding dimension $s = \mathcal{O}(n^2/\epsilon^2)$, with a small failure rate, then compute a $(1 \pm \frac{\epsilon}{6})$-approximate solution to the reduced-sized problem specified by $\Phi A$ using SRHT + LSQR, which takes $\mathcal{O}(sn \log(s/\epsilon) + n^3 \log n) = \mathcal{O}(n^3 \log(n/\epsilon)/\epsilon^2)$ time with a small failure rate. The solution $\hat{x}$ is a $(1+\epsilon)$-approximate solution to the original problem because $\hat{x}$ is feasible and

$$\begin{aligned}\|A\hat{x}\|_2 &\le \frac{1}{1-\epsilon/4}\|\Phi A\hat{x}\|_2 \le \frac{(1+\epsilon/6)}{1-\epsilon/4}\|\Phi A x^*\|_2 \\ &\le \frac{(1+\epsilon/6)(1+\epsilon/4)}{1-\epsilon/4}\|Ax^*\|_2 \le (1+\epsilon)\|Ax^*\|_2.\end{aligned}$$

Hence $\hat{x}$ is a $(1+\epsilon)$-approximate solution with a constant probability. □

Hence, under the assumption that $m \ge \mathrm{poly}(n)$ and $\epsilon$ is fixed, this particular combination would become the best approach proposed. However, we note that there are various trade-offs among those approaches. For instance, there are trade-offs between running time and conditioning quality in preconditioning, and there are trade-offs between embedding dimension/sample size and failure rate in embedding/sampling. It is indeed hard to tell which approach is the best without a least squares problem given. In Chapter 3, we develop a randomized least squares solver following the third approach using Gaussian transform and LSQR or the Chebyshev semi-iterative method, and we show practical trade-offs on different problem types and computing platforms.

For $\ell_p$, we might use

1. **a second-order method**
   e.g., IPMs,
2. **preconditioning + a first-order method**
   e.g., preconditioning via ellipsoidal rounding + accelerated gradient descent [56],
3. **preconditioning + subspace-preserving sampling + solving the subsampled problem**
   e.g., preconditioning via ellipsoidal rounding + subspace-preserving sampling + an IPM [19, 24], preconditioning via subspace embedding + subspace-preserving sampling + an IPM [64, 20].

If we take the first approach and use a second-order method, the expected running time is $\mathcal{O}(m^{3/2}n^2)$ [58], which is not favorable for strongly over-determined problems due to the possibly very large $m^{3/2}$ factor. If we take the second approach and use a first-order method to solve the preconditioned problem, for each iteration we have to make one pass through the entire matrix data, which might involve huge communication cost when the data is stored distributively on secondary storage. Therefore, reducing the number of iterations becomes crucial for solving large-scale $\ell_p$ regression problems along this approach. Lastly, if we take the third approach, as mentioned in Section 2.7, only a subsampled problem of size $\mathcal{O}(\text{poly}(n/\epsilon)) \times n$ needs to be solved to obtain a $(1+\epsilon)$-approximate solution, which takes $\mathcal{O}(\text{poly}(n/\epsilon))$ time. Conditioning via FCT takes $\mathcal{O}(mn \log m)$ time, and fast subspace-preserving sampling takes $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time. Hence, the best combination we have introduced so far takes $\mathcal{O}(mn \log m + \text{poly}(n/\epsilon))$ time. In Chapter 4, we develop input-sparsity time $\ell_p$ subspace embedding algorithms for $p \in [1, 2)$, and hence reduce the total running time to $\mathcal{O}(\text{nnz}(A) \cdot \log m + \text{poly}(n/\epsilon))$, which is optimal up to a log factor. Again, there are many trade-offs among those approaches. For example, the third approach is good if we only need low-precision solutions. If we need more accurate solutions, we have to solve really large subsampled problems due to the $\text{poly}(1/\epsilon)$ factor in sample size, which might be infeasible in practice.

# Chapter 3

# $\ell_2$ Regression

In this chapter, we consider high-precision solving of linear least squares (LS) problems that are strongly over- or under-determined, and possibly rank-deficient. In particular, given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$, where $m \gg n$ or $m \ll n$ and we do not assume that $A$ has full rank, we wish to develop randomized algorithms to accurately solve the problem (same as (1.2)):

$$\text{minimize}_{x \in \mathbb{R}^n} \quad \|Ax - b\|_2.$$

If we let $r = \text{rank}(A) \leq \min(m, n)$, then recall that if $r < n$ (the LS problem is under-determined or rank-deficient), then (1.2) has an infinite number of minimizers. In that case, the set of all minimizers is convex and hence has a unique element having minimum length. On the other hand, if $r = n$ so the problem has full rank, there exists only one minimizer to (1.2) and hence it must have the minimum length. In either case, we denote this unique min-length solution to (1.2) by $x^*$, and we are interested in computing $x^*$ in this work. That is,

$$x^* = \arg\min \|x\|_2 \quad \text{subject to} \quad x \in \arg\min_z \|Az - b\|_2. \tag{3.1}$$

LS problems arise in numerous applications, and the demand for faster LS solvers will continue to grow in light of new data applications and as problem scales become larger and larger.

We describe an LS solver called `LSRN` for these strongly over- or under-determined, and possibly rank-deficient, systems. `LSRN` uses random normal projections to compute a preconditioner matrix such that the preconditioned system is provably extremely well-conditioned. Importantly for large-scale applications, the preconditioning process is embarrassingly parallel, and it automatically speeds up with sparse matrices and fast linear operators. LSQR [61] or the Chebyshev semi-iterative (CS) method [34] can be used at the iterative step to compute the min-length solution within just a few iterations. We show that the latter method is preferred on clusters with high communication cost.

Because of its provably-good conditioning properties, `LSRN` has a fully predictable run-time performance, just like direct solvers, and it scales well in parallel environments. On large dense systems, `LSRN` is competitive with LAPACK's DGELSD for strongly over-determined problems, and it is much

faster for strongly under-determined problems, although solvers using fast random projections, like Blendenpik [7], are still slightly faster in both cases. On sparse systems without simple sparsity patterns, `LSRN` runs significantly faster than competing solvers, for both the strongly over- or under-determined cases.

In section 1.3.1 we describe existing deterministic LS solvers and recent randomized algorithms for the LS problem. In section 3.2 we show how to do preconditioning correctly for rank-deficient LS problems, and in section 3.3 we introduce `LSRN` and discuss its properties. Section 3.4 describes how `LSRN` can handle Tikhonov regularization for both over- and under-determined systems, and in section 3.5 we provide a detailed empirical evaluation illustrating the behavior of `LSRN`.

## 3.1 Randomized methods

In 2007, Drineas, Mahoney, Muthukrishnan, and Sarlós [31] introduced two randomized algorithms for the LS problem, each of which computes an approximate solution $\hat{x}$ in $\mathcal{O}(mn\log(n/\epsilon)+n^3/\epsilon)$ time such that $\|A\hat{x}-b\|_2 \leq (1+\epsilon)\|Ax^*-b\|_2$ given $\epsilon > 0$. Both of these algorithms apply a randomized Hadamard transform to the columns of $A$, thereby generating a problem of smaller size, one using uniformly random sampling and the other using a sparse random projection. They proved that, in both cases, the solution to the smaller problem leads to relative-error approximations of the original problem. The algorithms are suitable when low accuracy is acceptable, but the $\epsilon$ dependence quickly becomes the bottleneck otherwise. Using those algorithms as preconditioners was also mentioned in [31]. This work laid the ground for later algorithms and implementations.

Later, in 2008, Rokhlin and Tygert [62] described a related randomized algorithm for over-determined systems. They used a randomized transform named SRFT that consists of $m$ random Givens rotations, a random diagonal scaling, a discrete Fourier transform, and a random sampling. They considered using their method as a preconditioning method, and they showed that to get relative precision $\epsilon$, only $\mathcal{O}(n\log(1/\epsilon))$ samples are needed. In addition, they proved that if the sample size is greater than $4n^2$, the condition number of the preconditioned system is bounded above by a constant. Although choosing this many samples would adversely affect the running time of their solver, they also illustrated examples of input matrices for which the $4n^2$ sample bound was weak and for which many fewer samples sufficed.

Then, in 2010, Avron, Maymounkov, and Toledo [7] implemented a high-precision LS solver, called Blendenpik, and compared it to LAPACK's DGELS and to LSQR without preconditioning. Blendenpik uses a Walsh-Hadamard transform, a discrete cosine transform, or a discrete Hartley transform for blending the rows/columns, followed by a random sampling, to generate a problem of smaller size. The $R$ factor from the QR factorization of the smaller matrix is used as the preconditioner for LSQR. Based on their analysis, the condition number of the preconditioned system depends on the coherence or statistical leverage scores of $A$, i.e., the maximal row norm of $U$, where $U$ is an orthonormal basis of range($A$). We note that a solver for under-determined problems is also included in the Blendenpik package.

In 2011, Coakley, Rokhlin, and Tygert [22] described an algorithm that is also based on random

normal projections. It computes the orthogonal projection of any vector $b$ onto the null space of $A$ or onto the row space of $A$ via a preconditioned normal equation. The algorithm solves the over-determined LS problem as an intermediate step. They show that the normal equation is well-conditioned and hence the solution is reliable. Unfortunately, no implementation was provided. For an over-determined problem of size $m \times n$, the algorithm requires applying $A$ or $A^T$ $3n + 6$ times, while `LSRN` needs approximately $2n + 200$ matrix-vector multiplications under the default setting. Asymptotically, `LSRN` becomes faster as $n$ increases beyond several hundred. See section 3.3.4 for further complexity analysis of `LSRN`.

All the approaches mentioned in this section assume that $A$ has full rank, and for those based on iterative solvers, none provides a tight upper bound on the condition number of the preconditioned system with $\mathcal{O}(n)$ sample size (and hence the number of iterations). For `LSRN`, Theorem 6 ensures that the min-length solution is preserved, independent of the rank, and Theorems 8 and 9 provide bounds on the condition number and number of iterations, independent of the spectrum of $A$. In addition to handling rank-deficiency well, `LSRN` can even take advantage of it, resulting in a smaller condition number and fewer iterations.

Some prior work on the LS problem has explored "fast" randomized transforms that run in roughly $\mathcal{O}(mn \log m)$ time on a dense matrix $A$, while the random normal projection we use in `LSRN` takes $\mathcal{O}(mn^2)$ time. Although this could be an issue for some applications, the use of random normal projections comes with several advantages. First, if $A$ is a sparse matrix or a linear operator, which is common in large-scale applications, then the FFT-based fast transforms are no longer "fast". Second, the random normal projection is easy to implement using threads or MPI, and it scales well in parallel environments. Third, the strong symmetry of the standard normal distribution helps give the strong high probability bounds on the condition number in terms of sample size. These bounds depend on nothing but $s/r$, where $s$ is the sample size. For example, if $s = 4r$, Theorem 8 ensures that, with high probability, the condition number of the preconditioned system is less than 3.

This last property about the condition number of the preconditioned system makes the number of iterations and thus the running time of `LSRN` fully predictable like for a direct method. It also enables use of the CS method, which needs only one level-1 and two level-2 BLAS operations per iteration, and is particularly suitable for clusters with high communication cost because it doesn't have vector inner products that require synchronization between nodes. Although the CS method has the same theoretical upper bound on the convergence rate as CG-like methods, it requires accurate bounds on the singular values in order to work efficiently. Such bounds are generally hard to come by, limiting the popularity of the CS method in practice, but they are provided for the preconditioned system by our Theorem 8, and we do achieve high efficiency in our experiments.

## 3.2 Preconditioning for linear least squares

In light of (1.6), much effort has been made to transform a linear system into an equivalent system with reduced condition number. This *preconditioning*, for a square linear system $Bx = d$ of full

rank, usually takes one of the following forms:

$$\begin{aligned} \text{left preconditioning} \quad & M^T B x = M^T d, \\ \text{right preconditioning} \quad & BNy = d,\ x = Ny, \\ \text{left and right preconditioning} \quad & M^T BNy = M^T d,\ x = Ny. \end{aligned}$$

Clearly, the preconditioned system is consistent with the original one, i.e., has the same $x^*$ as the unique solution, if the preconditioners $M$ and $N$ are nonsingular.

For the general LS problem (1.3), more care should be taken so that the preconditioned system has the same min-length solution as the original one. For example, if we apply left preconditioning to the LS problem $\min_x \|Ax - b\|_2$, the preconditioned system becomes $\min_x \|M^T Ax - M^T b\|_2$, and its min-length solution is given by

$$x^*_{\text{left}} = (M^T A)^\dagger M^T b.$$

Similarly, the min-length solution to the right preconditioned system is given by

$$x^*_{\text{right}} = N(AN)^\dagger b.$$

The following lemma states the necessary and sufficient conditions for $A^\dagger = N(AN)^\dagger$ or $A^\dagger = (M^T A)^\dagger M^T$ to hold. Note that these conditions holding certainly imply that $x^*_{\text{right}} = x^*$ and $x^*_{\text{left}} = x^*$, respectively.

**Lemma 18.** *Given $A \in \mathbb{R}^{m\times n}$, $N \in \mathbb{R}^{n\times p}$ and $M \in \mathbb{R}^{m\times q}$, we have*

1. *$A^\dagger = N(AN)^\dagger$ if and only if range$(NN^T A^T)$ = range$(A^T)$,*
2. *$A^\dagger = (M^T A)^\dagger M^T$ if and only if range$(MM^T A)$ = range$(A)$.*

*Proof.* Let $r = \text{rank}(A)$ and $U\Sigma V^T$ be $A$'s compact SVD as in section 1.3.1, with $A^\dagger = V\Sigma^{-1}U^T$. Before continuing our proof, we reference the following facts about the pseudoinverse:

1. $B^\dagger = B^T(BB^T)^\dagger$ for any matrix $B$,
2. For any matrices $B$ and $C$ such that $BC$ is defined, $(BC)^\dagger = C^\dagger B^\dagger$ if (i) $B^T B = I$ or (ii) $CC^T = I$ or (iii) $B$ has full column rank and $C$ has full row rank.

Now let's prove the "if" part of the first statement. If $\text{range}(NN^T A^T) = \text{range}(A^T) = \text{range}(V)$, we can find a matrix $Z$ with full row rank such that $NN^T A^T = VZ$. Then,

$$\begin{aligned} N(AN)^\dagger &= N(AN)^T(AN(AN)^T)^\dagger = NN^T A^T(ANN^T A^T)^\dagger \\ &= VZ(U\Sigma V^T VZ)^\dagger = VZ(U\Sigma Z)^\dagger = VZZ^\dagger\Sigma^{-1}U^T = V\Sigma^{-1}U^T = A^\dagger. \end{aligned}$$

Conversely, if $N(AN)^\dagger = A^\dagger$, we know that $\text{range}(N(AN)^\dagger) = \text{range}(A^\dagger) = \text{range}(V)$ and hence $\text{range}(V) \subseteq \text{range}(N)$. Then we can decompose $N$ as $(V\ V_c)\begin{pmatrix} Z \\ Z_c \end{pmatrix} = VZ + V_c Z_c$, where $V_c$ is

orthonormal, $V^T V_c = 0$, and $\begin{pmatrix} Z \\ Z_c \end{pmatrix}$ has full row rank. Then,

$$\begin{aligned} 0 &= N(AN)^\dagger - A^\dagger = (VZ + V_c Z_c)(U\Sigma V^T (VZ + V_c Z_c))^\dagger - V\Sigma^{-1}U^T \\ &= (VZ + V_c Z_c)(U\Sigma Z)^\dagger - V\Sigma^{-1}U^T \\ &= (VZ + V_c Z_c)Z^\dagger \Sigma^{-1} U^T - V\Sigma^{-1}U^T = V_c Z_c Z^\dagger \Sigma^{-1} U^T. \end{aligned}$$

Multiplying by $V_c^T$ on the left and $U\Sigma$ on the right, we get $Z_c Z^\dagger = 0$, which is equivalent to $Z_c Z^T = 0$. Therefore,

$$\begin{aligned} \text{range}(NN^T A^T) &= \text{range}((VZ + V_c Z_c)(VZ + V_c Z_c)^T V\Sigma U^T) \\ &= \text{range}((VZZ^T V^T + V_c Z_c Z_c^T V_c^T) V\Sigma U^T) \\ &= \text{range}(VZZ^T \Sigma U^T) \\ &= \text{range}(V) = \text{range}(A^T), \end{aligned}$$

where we used the facts that $Z$ has full row rank and hence $ZZ^T$ is nonsingular, $\Sigma$ is nonsingular, and $U$ has full column rank.

To prove the second statement, let us take $B = A^T$. By the first statement, we know $B^\dagger = M(BM)^\dagger$ if and only if $\text{range}(MM^T B^T) = \text{range}(B^T)$, which is equivalent to saying $A^\dagger = (M^T A)^\dagger M^T$ if and only if $\text{range}(MM^T A) = \text{range}(A)$. □

Although Lemma 18 gives the necessary and sufficient condition, it does not serve as a practical guide for preconditioning LS problems. In this work, we are more interested in a sufficient condition that can help us build preconditioners. To that end, we provide the following theorem.

**Theorem 6.** *Given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, $N \in \mathbb{R}^{n\times p}$, and $M \in \mathbb{R}^{m\times q}$, let $x^*$ be the min-length solution to the LS problem $\min_x \|Ax - b\|_2$, $x^*_{right} = Ny^*$ where $y^*$ is the min-length solution to $\min_y \|ANy - b\|_2$, and $x^*_{left}$ be the min-length solution to $\min_x \|M^T Ax - M^T b\|_2$. Then,*

1. *$x^*_{right} = x^*$ if range$(N)$ = range$(A^T)$,*
2. *$x^*_{left} = x^*$ if range$(M)$ = range$(A)$.*

*Proof.* Let $r = \text{rank}(A)$, and let $U\Sigma V^T$ be $A$'s compact SVD. If $\text{range}(N) = \text{range}(A^T) = \text{range}(V)$, we can write $N$ as $VZ$, where $Z$ has full row rank. Therefore,

$$\begin{aligned} \text{range}(NN^T A^T) &= \text{range}(VZZ^T V^T V\Sigma U^T) = \text{range}(VZZ^T \Sigma U^T) \\ &= \text{range}(V) = \text{range}(A^T). \end{aligned}$$

By Lemma 18, $A^\dagger = N(AN)^\dagger$ and hence $x^*_{\text{left}} = x^*$. The second statement can be proved by similar arguments. □

## 3.3 Algorithm LSRN

In this section we present LSRN, an iterative solver for solving strongly over- or under-determined systems, based on "random normal projection". To construct a preconditioner we apply a transformation matrix whose entries are independent random variables drawn from the standard normal distribution. We prove that the preconditioned system is almost surely consistent with the original system, i.e., both have the same min-length solution. At least as importantly, we prove that the spectrum of the preconditioned system is independent of the spectrum of the original system; and we provide a strong concentration result on the extreme singular values of the preconditioned system. This concentration result enables us to predict the number of iterations for CG-like methods, and it also enables use of the CS method, which requires an accurate bound on the singular values to work efficiently.

### 3.3.1 The algorithm

Algorithm 2 shows the detailed procedure of LSRN to compute the min-length solution to a strongly over-determined problem, and Algorithm 3 shows the detailed procedure for a strongly under-determined problem. We refer to these two algorithms together as LSRN. Note that they only use the input matrix $A$ for matrix-vector and matrix-matrix multiplications, and thus $A$ can be a dense matrix, a sparse matrix, or a linear operator. In the remainder of this section we focus on analysis of the over-determined case. We emphasize that analysis of the under-determined case is quite analogous.

**Algorithm 2** LSRN (computes $\hat{x} \approx A^{\dagger}b$ when $m \gg n$)

1: Choose an oversampling factor $\gamma > 1$ and set $s = \lceil \gamma n \rceil$.
2: Generate $G = \text{randn}(s, m)$, i.e., an $s$-by-$m$ random matrix whose entries are independent random variables following the standard normal distribution.
3: Compute $\tilde{A} = GA$.
4: Compute $\tilde{A}$'s compact SVD $\tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $r = \text{rank}(\tilde{A})$, $\tilde{U} \in \mathbb{R}^{s\times r}$, $\tilde{\Sigma} \in \mathbb{R}^{r\times r}$, $\tilde{V} \in \mathbb{R}^{n\times r}$, and only $\tilde{\Sigma}$ and $\tilde{V}$ are needed.
5: Let $N = \tilde{V}\tilde{\Sigma}^{-1}$.
6: Compute the min-length solution to $\min_y \|ANy - b\|_2$ using an iterative method. Denote the solution by $\hat{y}$.
7: Return $\hat{x} = N\hat{y}$.

### 3.3.2 Theoretical properties

The use of random normal projection offers LSRN some nice theoretical properties. We start with consistency.

**Theorem 7.** *In Algorithm 2, we have $\hat{x} = A^{\dagger}b$ almost surely.*

**Algorithm 3** `LSRN` (computes $\hat{x} \approx A^\dagger b$ when $m \ll n$)

1: Choose an oversampling $\gamma > 1$ and set $s = \lceil \gamma m \rceil$.
2: Generate $G = \text{randn}(n, s)$, i.e., an $n$-by-$s$ random matrix whose entries are independent random variables following the standard normal distribution.
3: Compute $\tilde{A} = AG$.
4: Compute $\tilde{A}$'s compact SVD $\tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $r = \text{rank}(\tilde{A})$, $\tilde{U} \in \mathbb{R}^{n\times r}$, $\tilde{\Sigma} \in \mathbb{R}^{r\times r}$, $\tilde{V} \in \mathbb{R}^{s\times r}$, and only $\tilde{U}$ and $\tilde{\Sigma}$ are needed.
5: Let $M = \tilde{U}\tilde{\Sigma}^{-1}$.
6: Compute the min-length solution to $\min_x \|M^T Ax - M^T b\|_2$ using an iterative method, denoted by $\hat{x}$.
7: Return $\hat{x}$.

*Proof.* Let $r = \text{rank}(A)$ and $U\Sigma V^T$ be $A$'s compact SVD. We have

$$\begin{aligned} \text{range}(N) &= \text{range}(\tilde{V}\tilde{\Sigma}^{-1}) = \text{range}(\tilde{V}) \\ &= \text{range}(\tilde{A}^T) = \text{range}(A^T G^T) = \text{range}(V\Sigma(GU)^T). \end{aligned}$$

Define $G_1 = GU \in \mathbb{R}^{s\times r}$. Since $G$'s entries are independent random variables following the standard normal distribution and $U$ is orthonormal, $G_1$'s entries are also independent random variables following the standard normal distribution. Then given $s \geq \gamma n > n \geq r$, we know $G_1$ has full column rank $r$ with probability 1. Therefore,

$$\text{range}(N) = \text{range}(V\Sigma G_1^T) = \text{range}(V) = \text{range}(A^T),$$

and hence by Theorem 6 we have $\hat{x} = A^\dagger b$ almost surely. □

A more interesting property of `LSRN` is that the spectrum (the set of singular values) of the preconditioned system is solely associated with a random matrix of size $s \times r$, independent of the spectrum of the original system.

**Lemma 19.** *In Algorithm 2, the spectrum of $AN$ is the same as the spectrum of $G_1^\dagger = (GU)^\dagger$, independent of $A$'s spectrum.*

*Proof.* Continue to use the notation from the proof of Theorem 7. Let $G_1 = U_1\Sigma_1 V_1^T$ be $G_1$'s compact SVD, where $U_1 \in \mathbb{R}^{s\times r}$, $\Sigma_1 \in \mathbb{R}^{r\times r}$, and $V_1 \in \mathbb{R}^{r\times r}$. Since $\text{range}(\tilde{U}) = \text{range}(GA) = \text{range}(GU) = \text{range}(U_1)$ and both $\tilde{U}$ and $U_1$ are orthonormal matrices, there exists an orthonormal matrix $Q_1 \in \mathbb{R}^{r\times r}$ such that $U_1 = \tilde{U}Q_1$. As a result,

$$\tilde{U}\tilde{\Sigma}\tilde{V}^T = \tilde{A} = GU\Sigma V^T = U_1\Sigma_1 V_1^T \Sigma V^T = \tilde{U}Q_1\Sigma_1 V_1^T \Sigma V^T.$$

Multiplying by $\tilde{U}^T$ on the left of each side, we get $\tilde{\Sigma}\tilde{V}^T = Q_1\Sigma_1 V_1^T \Sigma V^T$. Taking the pseudoinverse gives $N = \tilde{V}\tilde{\Sigma}^{-1} = V\Sigma^{-1}V_1\Sigma_1^{-1}Q_1^T$. Thus,

$$AN = U\Sigma V^T V\Sigma^{-1}V_1\Sigma_1^{-1}Q_1^T = UV_1\Sigma_1^{-1}Q_1^T,$$

which gives $AN$'s SVD. Therefore, $AN$'s singular values are $\mathrm{diag}(\Sigma_1^{-1})$, the same as $G_1^\dagger$'s spectrum, but independent of $A$'s. □

We know that $G_1 = GU$ is a random matrix whose entries are independent random variables following the standard normal distribution. The spectrum of $G_1$ is a well-studied problem in random matrix theory, and in particular the properties of extreme singular values have been studied. See Lemma 10. With the aid of Lemma 10, it is straightforward to obtain the concentration result of $\sigma_1(AN)$, $\sigma_r(AN)$, and $\kappa(AN)$ as follows.

**Theorem 8.** *In Algorithm 2, for any $\alpha \in (0, 1 - \sqrt{r/s})$, we have*

$$\max\left\{\mathcal{P}\left(\sigma_1(AN) \geq \tfrac{1}{(1-\alpha)\sqrt{s}-\sqrt{r}}\right), \mathcal{P}\left(\sigma_r(AN) \leq \tfrac{1}{(1+\alpha)\sqrt{s}+\sqrt{r}}\right)\right\} < e^{-\alpha^2 s/2} \tag{3.2}$$

*and*

$$\mathcal{P}\left(\kappa(AN) = \frac{\sigma_1(AN)}{\sigma_r(AN)} \leq \frac{1+\alpha+\sqrt{r/s}}{1-\alpha-\sqrt{r/s}}\right) \geq 1 - 2e^{-\alpha^2 s/2}. \tag{3.3}$$

*Proof.* Set $t = \alpha\sqrt{s}$ in Lemma 10. □

In order to estimate the number of iterations for CG-like methods, we can now combine (1.6) and (3.3).

**Theorem 9.** *In exact arithmetic, given a tolerance $\epsilon > 0$, a CG-like method applied to the preconditioned system $\min_y \|ANy - b\|_2$ with $y^{(0)} = 0$ converges within $(\log\epsilon - \log 2)/\log(\alpha + \sqrt{r/s})$ iterations in the sense that*

$$\|\hat{y}_{CG} - y^*\|_{(AN)^T(AN)} \leq \epsilon \|y^*\|_{(AN)^T(AN)} \tag{3.4}$$

*holds with probability at least $1 - 2e^{-\alpha^2 s/2}$ for any $\alpha \in (0, 1 - \sqrt{r/s})$, where $\hat{y}_{CG}$ is the approximate solution returned by the CG-like solver and $y^* = (AN)^\dagger b$. Let $\hat{x}_{CG} = N\hat{y}_{CG}$ be the approximate solution to the original problem. Since $x^* = Ny^*$, (3.4) is equivalent to*

$$\|\hat{x}_{CG} - x^*\|_{A^TA} \leq \epsilon \|x^*\|_{A^TA}, \tag{3.5}$$

*or in terms of residuals,*

$$\|\hat{r}_{CG} - r^*\|_2 \leq \epsilon \|b - r^*\|_2, \tag{3.6}$$

*where $\hat{r}_{CG} = b - A\hat{x}_{CG}$ and $r^* = b - Ax^*$.*

If $n$ is large and thus $s$ is large, we can set $\alpha$ very small but still make $1 - 2e^{-\alpha^2 s/2}$ very close to 1. Moreover, the bounds in (3.3) and (1.6) are not tight. These facts allow us to ignore $\alpha$ in a practical setting when we solve a large-scale problem. For example, to reach precision $\epsilon = 10^{-14}$, it is safe in practice to set the maximum number of iterations to $(\log\epsilon - \log 2)/\log\sqrt{r/s} \approx 66/\log(s/r)$ for a numerically stable CG-like method, e.g., LSQR. We verify this claim in section 3.5.2.

In addition to allowing us to bound the number of iterations for CG-like methods, the result given by (3.2) also allows us to use the CS method. This method needs only one level-1 and two level-2 BLAS operations per iteration; and, importantly, because it doesn't have vector inner products that require synchronization between nodes, this method is suitable for clusters with high communication cost. It does need an explicit bound on the singular values, but once that bound is tight, the CS method has the same theoretical upper bound on the convergence rate as other CG-like methods. Unfortunately, in many cases, it is hard to obtain such an accurate bound, which prevents the CS method becoming popular in practice. In our case, however, (3.2) provides a probabilistic bound with very high confidence. Hence, we can employ the CS method without difficulty. For completeness, Algorithm 4 describes the CS method we implemented for solving LS problems. For discussion of its variations, see Gutknecht and Rollin [36].

**Algorithm 4** Chebyshev semi-iterative (CS) method (computes $x \approx A^{\dagger}b$)

1: Given $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and a tolerance $\epsilon > 0$, choose $0 < \sigma_L \leq \sigma_U$ such that all non-zero singular values of $A$ are in $[\sigma_L, \sigma_U]$ and let $d = (\sigma_U^2 + \sigma_L^2)/2$ and $c = (\sigma_U^2 - \sigma_L^2)/2$.
2: Let $x = \mathbf{0}$, $v = \mathbf{0}$, and $r = b$.
3: **for** $k = 0, 1, \ldots, \left\lceil (\log\epsilon - \log 2) \,/\, \log\frac{\sigma_U-\sigma_L}{\sigma_U+\sigma_L} \right\rceil$ **do**
4: $\beta \leftarrow \begin{cases} 0 & \text{if } k = 0, \\ \frac{1}{2}(c/d)^2 & \text{if } k = 1, \\ (\alpha c/2)^2 & \text{otherwise,} \end{cases} \quad \alpha \leftarrow \begin{cases} 1/d & \text{if } k = 0, \\ d - c^2/(2d) & \text{if } k = 1, \\ 1/(d - \alpha c^2/4) & \text{otherwise.} \end{cases}$
5: $v \leftarrow \beta v + A^T r$
6: $x \leftarrow x + \alpha v$
7: $r \leftarrow r - \alpha A v$
8: **end for**

### 3.3.3 Approximate rank-deficiency

Our theory above assumes exact arithmetic. For rank-deficient problems stored with limited precision, it is common for $A$ to be approximately but not exactly rank-deficient, i.e., to have small but not exactly zero singular values. A common practice to handle approximate rank-deficiency is to set a threshold and treat as zero any singular values smaller than the threshold. (This is called truncated SVD.) In LAPACK, the threshold is the largest singular value of $A$ multiplied by a user-supplied constant, called RCOND. Let $A \in \mathbb{R}^{m\times n}$ with $m \gg n$ be an approximately rank-$k$ matrix that can be written as $A_k + E$, where $k < n$ and $A_k$ is the best rank-$k$ approximation to $A$. For simplicity, we assume that a constant $c > 0$ is known such that $\sigma_1 \geq \sigma_k \gg c\sigma_1 \gg \sigma_{k+1} = \|E\|_2$. If we take the truncated SVD approach, $c$ can be used to determine the effective rank of $A$, and the solution becomes $x^* = A_k^{\dagger}b$. In `LSRN`, we can perform a truncated SVD on $\tilde{A} = GA$, where the constant $c$ is used to determine the effective rank of $\tilde{A}$, denoted by $\tilde{k}$. The rest of the algorithm remains the same. In this section, we present a sufficient condition for $\tilde{k} = k$ and analyze the approximation error of $\hat{x}$, the solution from `LSRN`. For simplicity, we assume exact arithmetic and exact solving of

the preconditioned system in our analysis. Recall that in LSRN we have

$$\tilde{A} = GA = GA_k + GE,$$

where $G \in \mathbb{R}^{s\times m}$ is a random normal projection. If $\gamma = s/n$ is sufficiently large, e.g., 2.0, and $n$ is not too small, Lemma 10 implies that there exist $0 < q_1 < q_2$ such that, with high probability, $G$ has full rank and

$$q_1\|Gw\|_2 \le \|w\|_2 \le q_2\|Gw\|_2, \quad \forall w \in \text{range}(A). \tag{3.7}$$

**Theorem 10.** *If* (3.7) *holds,* $\sigma_{k+1} < c\sigma_1 q_1/q_2$*, and* $\sigma_k > c\sigma_1(1 + q_2/q_1)$*, we have* $\tilde{k} = k$*, so that* LSRN *determines the effective rank of* $A$ *correctly, and*

$$\|\hat{x} - x^*\|_2 \le \frac{6\|b\|_2}{(\sigma_k q_1/q_2)(\sigma_k q_1/q_2 - \sigma_{k+1})} \cdot \sigma_{k+1}.$$

*Proof.* (3.7) implies

$$\sigma_1(\tilde{A}) = \max_{\|x\|_2\le 1} \|\tilde{A}x\|_2 \le \max_{\|x\|_2\le 1} \|Ax\|_2/q_1 \le \sigma_1/q_1, \tag{3.8}$$

$$\sigma_1(\tilde{A}) = \max_{\|x\|_2\le 1} \|\tilde{A}x\|_2 \ge \max_{\|x\|_2\le 1} \|Ax\|_2/q_2 \ge \sigma_1/q_2, \tag{3.9}$$

and hence $\sigma_1/q_2 \le \sigma_1(\tilde{A}) \le \sigma_1/q_1$. Similarly, we have $\|GE\|_2 = \sigma_1(GE) \le \sigma_{k+1}/q_1$. Let $\tilde{A}_k$ be $\tilde{A}$'s best rank-$k$ approximation and $\tilde{U}_k\tilde{\Sigma}_k\tilde{V}_k^T$ be its SVD. We have

$$\sigma_{k+1}(\tilde{A}) = \|\tilde{A}_k - \tilde{A}\|_2 \le \|GA_k - \tilde{A}\|_2 = \|GE\|_2 \le \sigma_{k+1}/q_1. \tag{3.10}$$

Note that $\text{range}(A_k) \subset \text{range}(A)$. We have

$$\sigma_k(GA_k) = \min_{v\in\text{range}((GA_k)^T),\|v\|_2=1} \|GA_kv\|_2 \ge \min_{v\in\text{range}(A_k^T),\|v\|_2=1} \|A_kv\|_2/q_2 = \sigma_k/q_2,$$

where $\text{range}((GA_k)^T) = \text{range}(A_k^T)$ because $G$ has full rank. Therefore,

$$\sigma_k(\tilde{A}) = \sigma_k(\tilde{A}_k) \ge \sigma_k(GA_k) - \|\tilde{A}_k - GA_k\|_2 \ge \sigma_k/q_2 - \sigma_{k+1}/q_1. \tag{3.11}$$

Using the assumptions $\sigma_{k+1} < c\sigma_1 q_1/q_2$ and $\sigma_k > c\sigma_1(1 + q_2/q_1)$, and the bounds (3.8)–(3.11), we get

$$c\sigma_1(\tilde{A}) \ge c\sigma_1/q_2 > \sigma_{k+1}/q_1 \ge \sigma_{k+1}(\tilde{A}),$$

and

$$\sigma_k(\tilde{A}) \ge \sigma_k/q_2 - \sigma_{k+1}/q_1 > \sigma_k/q_2 - c\sigma_1/q_2 \ge c\sigma_1/q_1 \ge c\sigma_1(\tilde{A}).$$

Thus if we use $c$ to determine the effective rank of $\tilde{A}$, the result would be $k$, the same as the effective rank of $A$.

Following the LSRN algorithm, the preconditioner matrix is $N = \tilde{V}_k \tilde{\Sigma}_k^{-1}$. Note that $AN$ has full rank because $k = \text{rank}(N) \geq \text{rank}(AN) \geq \text{rank}(GAN) = \text{rank}(\tilde{U}_k) = k$. Therefore, LSRN's solution can be written as

$$\hat{x} = N(AN)^\dagger b = (ANN^\dagger)^\dagger b = (A\tilde{V}_k\tilde{V}_k^T)^\dagger b.$$

For the matrix $A\tilde{V}_k\tilde{V}_k^T$, we have the following bound on its $k$-th singular value:

$$\begin{aligned}\sigma_k(A\tilde{V}_k\tilde{V}_k^T) &= \min_{v\in\text{range}(\tilde{V}_k),\|v\|_2=1} \|A\tilde{V}_k\tilde{V}_k^T v\|_2 \geq q_1 \min_{v\in\text{range}(\tilde{A}_k^T),\|v\|_2=1} \|GA\tilde{V}_k\tilde{V}_k^T v\|_2 \\ &\geq q_1 \min_{v\in\text{range}(\tilde{A}_k^T),\|v\|_2=1} \|\tilde{A}_k v\|_2 = \sigma_k q_1/q_2 - \sigma_{k+1}.\end{aligned}$$

The distance between $A_k$ and $A\tilde{V}_k\tilde{V}_k^T$ is also bounded:

$$\begin{aligned}\|A_k - A\tilde{V}_k{\tilde{V}_k}^T\|_2 &\leq q_2\|GA_k - \tilde{A}\tilde{V}_k{\tilde{V}_k}^T\|_2 \leq q_2\left(\|GA_k - \tilde{A}\|_2 + \|\tilde{A} - \tilde{A}\tilde{V}_k\tilde{V}_k^T\|_2\right) \\ &\leq 2q_2\|GA_k - \tilde{A}\|_2 \leq 2q_2\|GE\|_2 \leq 2\sigma_{k+1}q_2/q_1.\end{aligned}$$

For perturbation of a pseudoinverse, Wedin [69] shows that if $\text{rank}(A) = \text{rank}(B)$,

$$\|B^\dagger - A^\dagger\|_2 \leq 3\|A^\dagger\|_2\|B^\dagger\|_2\|A - B\|_2.$$

Applying this result, we get

$$\begin{aligned}\|x^* - \hat{x}\|_2 &\leq \|A_k^\dagger - (A\tilde{V}_k\tilde{V}_k^T)^\dagger\|_2\|b\|_2 \leq 3\|A_k^\dagger\|_2\|(A\tilde{V}_k\tilde{V}_k^T)^\dagger\|_2\|A_k - A\tilde{V}_k\tilde{V}_k^T\|_2\|b\|_2 \\ &\leq \frac{6\|b\|_2}{(\sigma_k q_1/q_2)(\sigma_k q_1/q_2 - \sigma_{k+1})} \cdot \sigma_{k+1}.\end{aligned}$$

Thus, $\hat{x}$ is a good approximation to $x^*$ if $\sigma_{k+1} = \|E\|_2$ is sufficiently small. □

Theorem 10 suggests that, to correctly determine the effective rank, we need $\sigma_k$ and $\sigma_{k+1}$ well-separated with respect to the distortion $q_2/q_1$ introduced by $G$. For LSRN, $q_2/q_1$ is bounded by a small constant with high probability if we choose the oversampling factor $\gamma$ to be a moderately large constant, e.g., 2. We note that the distortion of the random normal projection used in Coakley et al. [22] is around 1000, which reduces the reliability of determining the effective rank of an approximately rank-deficient problem. We verify this claim empirically in section 3.5.9.

**Remark** Theorem 10 assumes that $G$ has full rank and subspace embedding property (3.7). It is not necessary for $G$ to be a random normal projection matrix. The result also applies to other random projection matrices satisfying this condition, e.g., the randomized discrete cosine transform used in Blendenpik [7].

### 3.3.4 Running time complexity

In this section, we discuss the running time complexity of LSRN. Let's first calculate the computational cost of LSRN (Algorithm 2) in terms of floating-point operations (flops). Note that we need only $\tilde{\Sigma}$ and $\tilde{V}$ but not $\tilde{U}$ or a full SVD of $\tilde{A}$ in step 4 of Algorithm 2. In step 6, we assume that the dominant cost per iteration is the cost of applying $AN$ and $(AN)^T$. Then the total cost is given by

$$\begin{array}{lcl} & sm \times \text{flops}(\text{randn}) & \text{for generating } G \\ + & s \times \text{flops}(A^T u) & \text{for computing } \tilde{A} \\ + & 2sn^2 + 11n^3 & \text{for computing } \tilde{\Sigma} \text{ and } \tilde{V} \text{ [33, p. 254]} \\ + & N_{\text{iter}} \times (\text{flops}(Av) + \text{flops}(A^T u) + 4nr) & \text{for solving } \min_y \|ANy - b\|_2, \end{array}$$

where lower-order terms are ignored. Here, flops(randn) is the average flop count to generate a sample from the standard normal distribution, while flops($Av$) and flops($A^T u$) are the flop counts for the respective matrix-vector products. If $A$ is a dense matrix, then we have flops($Av$) = flops($A^T u$) = $2mn$. Hence, the total cost becomes

$$\text{flops}(\texttt{LSRN}_{\text{dense}}) = sm\,\text{flops}(\text{randn}) + 2smn + 2sn^2 + 11n^3 + N_{\text{iter}} \times (4mn + 4nr).$$

Comparing this with the SVD approach, which uses $2mn^2 + 11n^3$ flops, we find LSRN requires more flops, even if we only consider computing $\tilde{A}$ and its SVD. However, the actual running time is not fully characterized by the number of flops. It is also affected by how efficiently the computers can do the computation. We empirically compare the running time in section 3.5. If $A$ is a sparse matrix, we generally have flops($Av$) and flops($A^T u$) of order $\mathcal{O}(m)$. In this case, LSRN should run considerably faster than the SVD approach. Finally, if $A$ is an operator, it is hard to apply SVD, while LSRN still works without any modification. If we set $\gamma = 2$ and $\epsilon = 10^{-14}$, we know $N_{\text{iter}} \approx 100$ by Theorem 9 and hence LSRN needs approximately $2n + 200$ matrix-vector multiplications. Note that the randomized LS solver proposed by Coakley et al. [22] requires $3n + 6$ matrix-vector multiplications. Thus, it requires more matrix-vector multiplications than LSRN when $n$ is beyond a few hundred.

One advantage of LSRN is that the stages of generating $G$ and computing $\tilde{A} = GA$ are embarrassingly parallel. Thus, it is easy to implement LSRN in parallel. For example, on a shared-memory machine using $p$ cores, the total running time decreases to

$$T_{\texttt{LSRN}}^{\text{mt},p} = T_{\text{randn}}/p + T_{\text{mult}}/p + T_{\text{svd}}^{\text{mt},p} + T_{\text{iter}}/p, \tag{3.12}$$

where $T_{\text{randn}}$, $T_{\text{mult}}$, and $T_{\text{iter}}$ are the running times for the respective stages if LSRN runs on a single core, $T_{\text{svd}}^{\text{mt},p}$ is the running time of SVD using $p$ cores, and communication cost among threads is ignored. Hence, multi-threaded LSRN has very good scalability with near-linear speedup on strongly over- or under-determined problems.

Alternatively, let us consider a cluster of size $p$ using MPI, where each node stores a portion

of rows of $A$ (with $m \gg n$). Each node can generate random samples and do the multiplication independently, and then an MPI_Reduce operation is needed to obtain $\tilde{A}$. Since $n$ is small, the SVD of $\tilde{A}$ and the preconditioner $N$ are computed on a single node and distributed to all the other nodes via an MPI_Bcast operation. If the CS method is chosen as the iterative solver, we need one MPI_Allreduce operation per iteration in order to apply $A^T$. Note that all the MPI operations that `LSRN` uses are collective. If we assume the cluster is homogeneous and has perfect load balancing, the time complexity to perform a collective operation should be $\mathcal{O}(\log p)$. Hence the total running time becomes

$$T_{\texttt{LSRN}}^{\text{mpi},p} = T_{\text{randn}}/p + T_{\text{mult}}/p + T_{\text{svd}} + T_{\text{iter}}/p + (C_1 + C_2 N_{\text{iter}})\mathcal{O}(\log p), \tag{3.13}$$

where $C_1$ corresponds to the cost of computing $\tilde{A}$ and broadcasting $N$, and $C_2$ corresponds to the cost of applying $A^T$ at each iteration. Therefore, the MPI implementation of `LSRN` still has good scalability as long as $T_{\text{svd}}$ is not dominant, i.e., as long as $\tilde{A}$ is not too big. Typical values of $n$ (or $m$ for under-determined problems) in our empirical evaluations are around 1000, and thus this is the case.

## 3.4 Tikhonov regularization

We point out that it is easy to extend `LSRN` to handle certain types of Tikhonov regularization, also known as ridge regression. Recall that Tikhonov regularization involves solving the problem

$$\text{minimize} \quad \frac{1}{2}\|Ax - b\|_2^2 + \frac{1}{2}\|Wx\|_2^2, \tag{3.14}$$

where $W \in \mathbb{R}^{n\times n}$ controls the regularization term. In many cases, $W$ is chosen as $\lambda I_n$ for some value of a regularization parameter $\lambda > 0$. It is easy to see that (3.14) is equivalent to the following LS problem, without any regularization:

$$\text{minimize} \quad \frac{1}{2}\left\| \begin{pmatrix} A \\ W \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2^2. \tag{3.15}$$

This is an over-determined problem of size $(m+n)\times n$. If $m \gg n$, then we certainly have $m+n \gg n$. Therefore, if $m \gg n$, we can directly apply `LSRN` to (3.15) in order to solve (3.14). On the other hand, if $m \ll n$, then although (3.15) is still over-determined, it is "nearly square" in the sense that $m + n$ is only slightly larger than $n$. In this regime, random sampling methods and random projection methods like `LSRN` do not perform well. In order to deal with this regime, note that (3.14) is equivalent to

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|r\|_2^2 + \frac{1}{2}\|Wx\|_2^2 \\ \text{subject to} \quad & Ax + r = b, \end{aligned}$$

where $r = b - Ax$ is the residual vector. (Note that we use $r$ to denote the matrix rank in a scalar context and the residual vector in a vector context.) By introducing $z = Wx$ and assuming that $W$ is non-singular, we can re-write the above problem as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\left\|\begin{pmatrix} z \\ r \end{pmatrix}\right\|_2^2 \\ \text{subject to} \quad & \begin{pmatrix} AW^{-1} & I_m \end{pmatrix}\begin{pmatrix} z \\ r \end{pmatrix} = b, \end{aligned}$$

i.e., as computing the min-length solution to

$$\begin{pmatrix} AW^{-1} & I_m \end{pmatrix}\begin{pmatrix} z \\ r \end{pmatrix} = b. \tag{3.16}$$

Note that (3.16) is an under-determined problem of size $m \times (m+n)$. Hence, if $m \ll n$, we have $m \ll m+n$ and we can use `LSRN` to compute the min-length solution to (3.16), denoted by $\begin{pmatrix} z^* \\ r^* \end{pmatrix}$. The solution to the original problem (3.14) is then given by $x^* = W^{-1}z^*$. Here, we assume that $W^{-1}$ is easy to apply, as is the case when $W = \lambda I_n$, so that $AW^{-1}$ can be treated as an operator. The equivalence between (3.14) and (3.16) was first established by Herman, Lent, and Hurwitz [37].

In most applications of regression analysis, the amount of regularization, e.g., the optimal regularization parameter, is unknown and thus determined by cross-validation. This requires solving a sequence of LS problems where only $W$ differs. For over-determined problems, we only need to perform a random normal projection on $A$ once. The marginal cost to solve for each $W$ is the following: a random normal projection on $W$, an SVD of size $\lceil \gamma n \rceil \times n$, and a predictable number of iterations. Similar results hold for under-determined problems when each $W$ is a multiple of the identity matrix.

## 3.5 Numerical experiments

We implemented our LS solver `LSRN` and compared it with competing solvers: DGELSD/DGELSY from LAPACK [5], spqr_solve (SPQR for short) from SuiteSparseQR [26, 27], and Blendenpik [7]. Table 3.1 summarizes the properties of those solvers. It is impossible to compare `LSRN` with all the LS solvers. We choose solvers from LAPACK and SuiteSparseQR because they are the *de facto* standards for dense and sparse problems, respectively. DGELSD takes the SVD approach, which is accurate and robust to rank deficiency. DGELSY takes the orthogonal factorization approach, which should be almost as robust as the SVD approach but less expensive. SPQR uses multifrontal sparse QR factorization. With the "min2norm" option, it computes min-length solutions to full-rank under-determined LS problems. However, it doesn't compute min-length solutions to rank-deficient problems. Note that the widely used MATLAB's backslash calls LAPACK for dense problems and

Table 3.1: LS solvers and their properties.

| solver | min-len solution to | | taking advantage of | |
|---|---|---|---|---|
| | under-det? | rank-def? | sparse $A$ | operator $A$ |
| DGELSD/DGELSY | yes | yes | no | no |
| SPQR | yes | no | yes | no |
| Blendenpik | yes | no | no | no |
| LSRN | yes | yes | yes | yes |

SuiteSparseQR for sparse problems[1]. But it doesn't call the functions that return min-length solutions to rank-deficient or under-determined systems. We choose Blendenpik out of several recently proposed randomized LS solvers, e.g., [62] and [22], because a high-performance implementation is publicly available and it is easy to adapt it to use multi-threads. Blendenpik assumes that $A$ has full rank.

### 3.5.1 Implementation and system setup

The experiments were performed on either a local shared-memory machine or a virtual cluster hosted on Amazon's Elastic Compute Cloud (EC2). The shared-memory machine has 12 Intel Xeon CPU cores at clock rate 2GHz with 128GB RAM. The virtual cluster consists of 20 m1.large instances configured by a third-party tool called StarCluster[2]. An m1.large instance has 2 virtual cores with 2 EC2 Compute Units[3] each. To attain top performance on the shared-memory machine, we implemented a multi-threaded version of `LSRN` in C, and to make our solver general enough to handle large problems on clusters, we also implemented an MPI version of `LSRN` in Python with NumPy, SciPy, and mpi4py. Both packages are available for download[4]. We use the multi-threaded implementation to compare `LSRN` with other LS solvers and use the MPI implementation to explore scalability and to compare iterative solvers under a cluster environment. To generate values from the standard normal distribution, we adopted the code from Marsaglia and Tsang [48] and modified it to use threads; this can generate a billion samples in less than two seconds on the shared-memory machine. We compiled SuiteSparseQR with Intel Threading Building Blocks (TBB) enabled, as suggested by its author. We also modified Blendenpik to call multi-threaded FFTW routines. Blendenpik's default settings were used. All the solvers were linked against the BLAS and LAPACK libraries shipped with MATLAB R2011b. So, in general, this is a fair setup because all the solvers can use multi-threading automatically and are linked against the same BLAS and LAPACK libraries. The running times were measured in wall-clock times.

[1] As stated by Tim Davis, "SuiteSparseQR is now QR in MATLAB 7.9 and $x = A\backslash b$ when $A$ is sparse and rectangular." `http://www.cise.ufl.edu/research/sparse/SPQR/`

[2] `http://web.mit.edu/stardev/cluster/`

[3] "One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor." `http://aws.amazon.com/ec2/faqs/`

[4] `http://www.stanford.edu/group/SOL/software/lsrn.html`

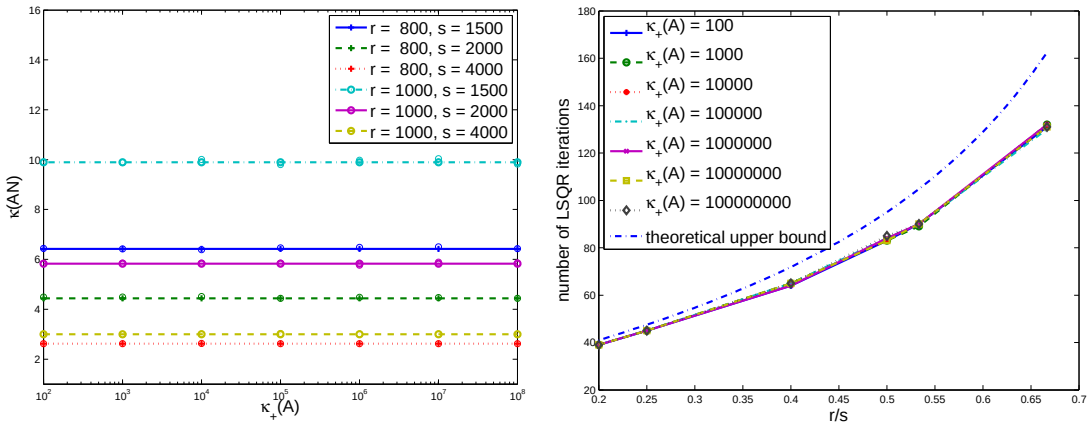


Figure 3.1: Left: $\kappa_+(A)$ vs. $\kappa(AN)$ for different choices of $r$ and $s$. $A \in \mathbb{R}^{10^4 \times 10^3}$ is randomly generated with rank $r \in \{800, 1000\}$ and effective condition number $\kappa_+(A) \in \{10^2, 10^3, \ldots, 10^8\}$. For each $(r, s)$ pair, we take the largest value of $\kappa(AN)$ in 10 independent runs for each $\kappa_+(A)$ and plot them using circle marks. The estimate $(1 + \sqrt{r/s})/(1 - \sqrt{r/s})$ is drawn using a solid line for each $(r, s)$ pair. Right: number of LSQR iterations vs. $r/s$. The number of LSQR iterations is merely a function of $r/s$, independent of the condition number of the original system.

### 3.5.2 $\kappa(AN)$ and number of iterations

Recall that Theorem 8 states that $\kappa(AN)$, the condition number of the preconditioned system, is roughly bounded by $(1 + \sqrt{r/s})/(1 - \sqrt{r/s})$ when $s$ is large enough such that we can ignore $\alpha$ in practice. To verify this statement, we generate random matrices of size $10^4 \times 10^3$ with condition numbers ranged from $10^2$ to $10^8$. The left figure in Figure 3.1 compares $\kappa(AN)$ with $\kappa_+(A)$, the effective condition number of $A$, under different choices of $s$ and $r$. We take the largest value of $\kappa(AN)$ in 10 independent runs as the $\kappa(AN)$ in the plot. For each pair of $s$ and $r$, the corresponding estimate $(1+\sqrt{r/s})/(1-\sqrt{r/s})$ is drawn in a solid line of the same color. We see that $(1+\sqrt{r/s})/(1-\sqrt{r/s})$ is indeed an accurate estimate of the upper bound on $\kappa(AN)$. Moreover, $\kappa(AN)$ is not only independent of $\kappa_+(A)$, but it is also quite small. For example, we have $(1 + \sqrt{r/s})/(1 - \sqrt{r/s}) < 6$ if $s > 2r$, and hence we can expect super fast convergence of CG-like methods. Based on Theorem 9, the number of iterations should be less than $(\log \epsilon - \log 2)/\log \sqrt{r/s}$, where $\epsilon$ is a given tolerance. In order to match the accuracy of direct solvers, we set $\epsilon = 10^{-14}$. The right figure in Figure 3.1 shows the number of LSQR iterations for different combinations of $r/s$ and $\kappa_+(A)$. Again, we take the largest iteration number in 10 independent runs for each pair of $r/s$ and $\kappa_+(A)$. We also draw the theoretical upper bound $(\log \epsilon - \log 2)/\log \sqrt{r/s}$ in a dotted line. We see that the number of iterations is basically a function of $r/s$, independent of $\kappa_+(A)$, and the theoretical upper bound is very good in practice. This confirms that the number of iterations is fully predictable given $\gamma$.

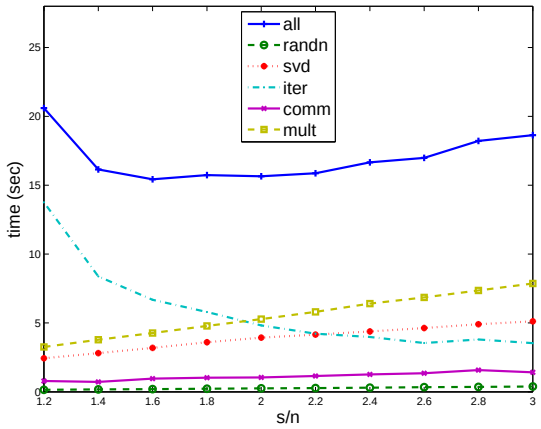


Figure 3.2: The overall running time of `LSRN` and the running time of each `LSRN` stage with different oversampling factor $\gamma$ for a randomly generated problem of size $10^5 \times 10^3$. For this particular problem, the optimal $\gamma$ that minimizes the overall running time lies in $[1.8, 2.2]$.

### 3.5.3 Tuning the oversampling factor $\gamma$

Once we set the tolerance and maximum number of iterations, there is only one parameter left: the oversampling factor $\gamma$. To demonstrate the impact of $\gamma$, we fix problem size to $10^5 \times 10^3$ and condition number to $10^6$, set the tolerance to $10^{-14}$, and then solve the problem with $\gamma$ ranged from 1.2 to 3. Figure 3.2 illustrates how $\gamma$ affects the running times of `LSRN`'s stages: `randn` for generating random numbers, `mult` for computing $\tilde{A} = GA$, `svd` for computing $\tilde{\Sigma}$ and $\tilde{V}$ from $\tilde{A}$, and `iter` for LSQR. We see that, the running times of `randn`, `mult`, and `svd` increase linearly as $\gamma$ increases, while `iter` time decreases. Therefore there exists an optimal choice of $\gamma$. For this particular problem, we should choose $\gamma$ between 1.8 and 2.2. We experimented with various LS problems. The best choice of $\gamma$ ranges from 1.6 to 2.5, depending on the type and the size of the problem. We also note that, when $\gamma$ is given, the running time of the iteration stage is fully predictable. Thus we can initialize `LSRN` by measuring randn/sec and flops/sec for matrix-vector multiplication, matrix-matrix multiplication, and SVD, and then determine the best value of $\gamma$ by minimizing the total running time (3.13). For simplicity, we set $\gamma = 2.0$ in all later experiments; although this is not the optimal setting for all cases, it is always a reasonable choice.

### 3.5.4 Solution accuracy

Under the default settings $\gamma = 2.0$ and $\epsilon = 10^{-14}$, we test `LSRN`'s solution accuracy on three types of LS problems: full-rank, rank-deficient, and approximately rank-deficient. `LSRN` uses the common approach to determine the effective rank of $\tilde{A}$, whose singular values smaller than RCOND $\times \|\tilde{A}\|_2$ are treated as zeros, where RCOND is a user input. $A$ is generated by constructing its SVD. For full-rank problems, we use the following Matlab script:

```
U = orth(randn(m, n)); S = diag(linspace(1, 1/c, n)); V = orth(randn(n, n));
```

| | $\frac{\|\hat{x}\|_2-\|x^*\|_2}{c\|x^*\|_2}$ | $\frac{\|A\hat{x}-b\|_2-\|Ax^*-b\|_2}{c\|Ax^*-b\|_2}$ | $\frac{\|A^T(Ax^*-b)\|_2}{c}$ | $\frac{\|A^T(A\hat{x}-b)\|_2}{c}$ |
|---|---|---|---|---|
| full-rank | -8.5e-14 | 0.0 | 3.6e-18 | 2.5e-17 |
| rank-def | -5.3e-14 | 0.0 | 8.1e-18 | 1.5e-17 |
| approx. rank-def | 3.1e-12 | -8.6e-19 | 2.0e-17 | 2.9e-17 |

Table 3.2: Comparing LSRN's solution accuracy to DGELSD. DGELSD's solution is denoted by $x^*$, and LSRN's denoted by $\hat{x}$. The metrics are computed using quad precision. We show the average values of those metrics from 50 independent runs. LSRN should be accurate enough for most applications.

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

For rank-deficient problems, we use:

```
U = orth(randn(m, r)); S = diag(linspace(1, 1/c, r)); V = orth(randn(n, r));
```

The script for approximately rank-deficient problems is the same as the full-rank one except that

```
S = diag([linspace(1, 1/c, r), 1e-9 * ones(1, n-r)]);
```

We choose $m = 10^5$, $n = 100$, $r = 80$, and $c = 10^6$. DGELSD is used as a reference solver with RCOND set as $10^{-8}$. The metrics are relative differences in $\|x\|_2$ and $\|Ax-b\|_2$, and $\|A^T(Ax-b)\|_2$, all scaled by $1/c$ and computed using quad precision. Table 3.2 lists the average values of those metrics from 50 independent runs. We see that LSRN is accurate enough to meet the accuracy requirement of most applications.

### 3.5.5 Dense least squares

Though LSRN is not designed for dense problems, it is competitive with DGELSD/DGELSY and Blendenpik on large-scale strongly over- or under-determined LS problems. Figure 3.3 compares the running times of LSRN and competing solvers on randomly generated full-rank LS problems. We use the script from section 3.5.4 to generate test problems. The results show that Blendenpik is the overall winner. The follow-ups are LSRN and DGELSD. We find that the SVD-based DGELSD actually runs much faster than the QR-based DGELSY on strongly over- or under-determined systems on the shared-memory machine. It may be because of better use of multi-threaded BLAS, but we don't have a definitive explanation. The performance of LAPACK's solvers decreases significantly for under-determined problems. We monitored CPU usage and found that they couldn't fully use all the CPU cores, i.e., they couldn't effectively call multi-threaded BLAS. The performance of Blendenpik also decreases, while that of LSRN does not change much, making LSRN's performance very close to Blendenpik's.

**Remark** The performance of DGELSD/DGELSY varies greatly, depending on the LAPACK implementation. When we use the LAPACK library shipped with MATLAB R2010b, The DGELSD from it takes near 150 seconds to solve a $10^6 \times 10^3$ LS problem, which is slower than LSRN. However, after we switch to MATLAB R2011b, it runs slightly faster than LSRN does on the same problem.

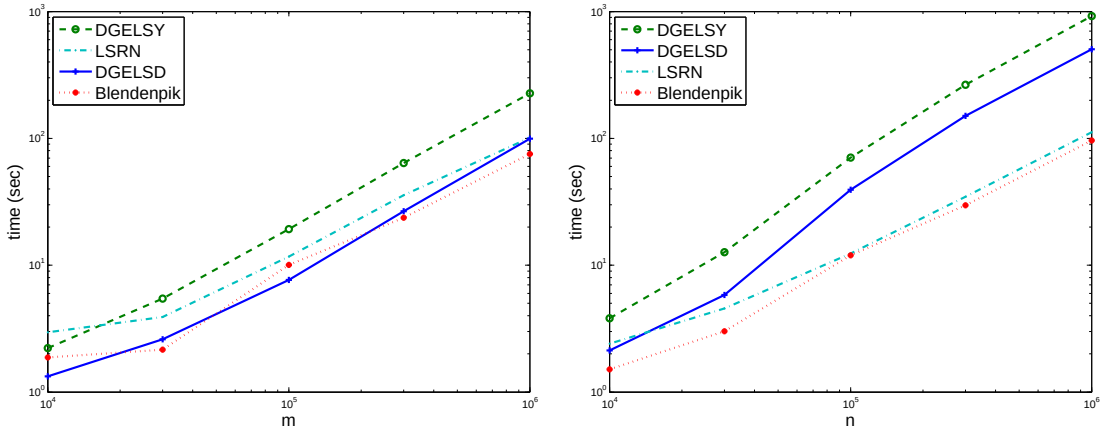


Figure 3.3: Running times on $m \times 1000$ dense over-determined problems with full rank (left) and on $1000 \times n$ dense under-determined problems with full rank (right). On the problem of size $10^6 \times 10^3$, we have Blendenpik > DGELSD > `LSRN` > DGELSY in terms of speed. On under-determined problems, LAPACK's performance decreases significantly compared with the over-determined cases. Blendenpik's performance decreases as well, while `LSRN` doesn't change much.

`LSRN` is also capable of solving rank-deficient problems, and in fact it takes advantage of any rank-deficiency (in that it finds a solution in fewer iterations). Figure 3.4 shows the results on over- and under-determined rank-deficient problems generated the same way as in previous experiments, except that we set $r = 800$. Blendenpik is not included because it is not designed to handle rank deficiency. DGELSY/DGELSD remains the same speed on over-determined problems as in full-rank cases, and runs slightly faster on under-determined problems. On the problem of size $10^6 \times 10^3$, DGELSD spends 99.5 seconds, almost the same as in the full-rank case, while `LSRN`'s running times reduce to 95.0 seconds, from 109 seconds on its full-rank counterpart.

We see that, for strongly over- or under-determined problems, DGELSD is the fastest and most reliable routine among the LS solvers provided by LAPACK. However, it (or any other LAPACK solver) runs much slower on under-determined problems than on over-determined problems, while `LSRN` works symmetrically on both cases. Blendenpik is the fastest dense least squares solver in our tests. Though it is not designed for solving rank-deficient problems, Blendenpik should be modifiable to handle such problems following Theorem 6. We also note that Blendenpik's performance depends on the distribution of the row norms of $U$. We generate test problems randomly so that the row norms of $U$ are homogeneous, which is ideal for Blendenpik. When the row norms of $U$ are heterogeneous, Blendenpik's performance may drop. See Avron, Maymounkov, and Toledo [7] for a more detailed analysis.

### 3.5.6 Sparse least squares

In `LSRN`, $A$ is only involved in the computation of matrix-vector and matrix-matrix multiplications. Therefore `LSRN` accelerates automatically when $A$ is sparse, without exploring $A$'s sparsity pattern.

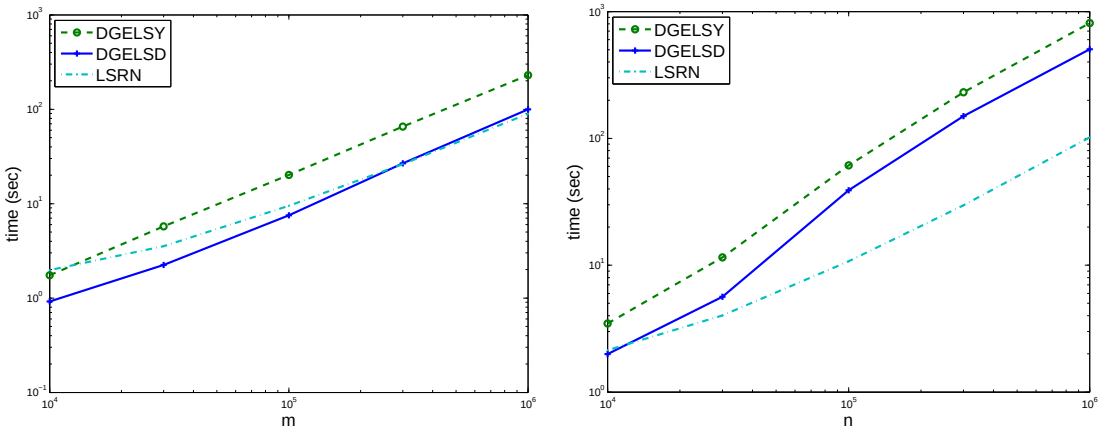


Figure 3.4: Running times on $m \times 1000$ dense over-determined problems with rank 800 (left) and on $1000 \times n$ dense under-determined problems with rank 800 (right). `LSRN` takes advantage of rank deficiency. We have `LSRN` > DGSLS/DGELSD > DGELSY in terms of speed.

SPQR requires explicit knowledge of $A$'s sparsity pattern to obtain a sparse QR factorization. LAPACK does not have any direct sparse LS solver, and Blendenpik utilizes fast transforms, which assume that the input matrix is dense.

We generated sparse LS problems using MATLAB's "sprandn" function with density 0.01 and condition number $10^6$. All problems have full rank. Figure 3.5 shows the results. DGELSD/DGELSY and Blendenpik basically perform the same as in the dense case. For over-determined problems, we see that SPQR handles sparse problems very well when $m < 10^5$. As $m$ goes larger, it becomes harder to analyze $A$'s sparsity pattern in order to create a sparse QR factorization. SPQR runs even longer than DGELSD when $m \geq 3 \times 10^5$. `LSRN` becomes the fastest solver among the five when $m \geq 10^5$. It takes only 27.4 seconds on the over-determined problem of size $10^6 \times 10^3$. On large under-determined problems, `LSRN` still leads by a huge margin.

`LSRN` makes no distinction between dense and sparse problems. The speedup on sparse problems is due to faster matrix-vector and matrix-matrix multiplications. Hence, although no test was performed, we expect a similar speedup on fast linear operators as well. Also note that, in the multi-threaded implementation of `LSRN`, we use a naïve multi-threaded routine for sparse matrix-vector and matrix-matrix multiplications, which is far from optimized and thus leaves room for improvement.

### 3.5.7 Real-world problems

In this section, we report results on some real-world large data problems. The problems are summarized in Table 3.3, along with running times. DGELSY is not included in this test due to its inferiority to DGELSD.

`landmark` and `rail4284` are from the University of Florida Sparse Matrix Collection [28]. `landmark`

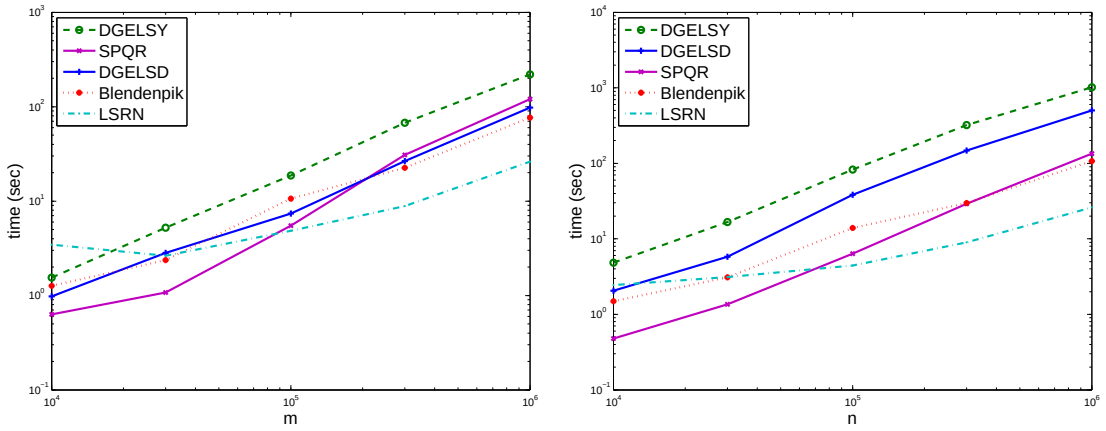


Figure 3.5: Running times on $m \times 1000$ sparse over-determined problems with full rank (left) and on $1000 \times n$ sparse under-determined problems with full rank (right). DGELSD/DGELSY and Blendenpik perform almost the same as in the dense case. SPQR performs very well for small and medium-scaled problems, but it runs slower than the dense solver Blendenpik on the problem of size $10^6 \times 10^3$. `LSRN` starts to lead as $m$ goes above $10^5$, and it leads by a huge margin on the largest one. The under-determined case is very similar to its over-determined counterpart.

Table 3.3: Real-world problems and corresponding running times in seconds. DGELSD doesn't take advantage of sparsity, with its running time determined by the problem size. Though SPQR may not output min-length solutions to rank-deficient problems, we still report its running times (marked with " *"). Blendenpik either doesn't apply to rank-deficient problems or runs out of memory (OOM). `LSRN`'s running time is mainly determined by the problem size and the sparsity.

| matrix | $m$ | $n$ | nnz | rank | cond | DGELSD | SPQR | Blendenpik | `LSRN` |
|---|---|---|---|---|---|---|---|---|---|
| `landmark` | 71952 | 2704 | 1.15e6 | 2671 | 1.0e8 | 18.64 | 4.920* | - | 17.89 |
| `rail4284` | 4284 | 1.1e6 | 1.1e7 | full | 400.0 | $> 3600$ | 505.9 | OOM | 146.1 |
| `tnimg_1` | 951 | 1e6 | 2.1e7 | 925 | - | 510.3 | 72.14* | - | 41.08 |
| `tnimg_2` | 1000 | 2e6 | 4.2e7 | 981 | - | 1022 | 168.6* | - | 82.63 |
| `tnimg_3` | 1018 | 3e6 | 6.3e7 | 1016 | - | 1628 | 271.0* | - | 124.5 |
| `tnimg_4` | 1019 | 4e6 | 8.4e7 | 1018 | - | 2311 | 371.3* | - | 163.9 |
| `tnimg_5` | 1023 | 5e6 | 1.1e8 | full | - | 3105 | 493.2 | OOM | 197.6 |

originated from a rank-deficient LS problem. `rail4284` has full rank and originated from a linear programming problem on Italian railways. Both matrices are very sparse and have structured patterns. Though SPQR runs extremely fast on `landmark`, it doesn't guarantee to return the min-length solution. Blendenpik is not designed to handle the rank-deficient `landmark`, and it unfortunately runs out of memory (OOM) on `rail4284`. `LSRN` takes 17.55 seconds on `landmark` and 136.0 seconds on `rail4284`. DGELSD is slightly slower than `LSRN` on `landmark` and much slower on `rail4284`.

`tnimg` is generated from the TinyImages collection [66], which provides 80 million color images of size $32 \times 32$. For each image, we first convert it to grayscale, compute its two-dimensional DCT (Discrete Cosine Transform), and then only keep the top 2% largest coefficients in magnitude. This gives a sparse matrix of size $1024 \times$ 8e7 where each column has 20 or 21 nonzero elements. Note that `tnimg` doesn't have apparent structured pattern. Since the whole matrix is too big, we work on submatrices of different sizes. `tnimg_`$i$ is the submatrix consisting of the first $10^6 \times i$ columns of the whole matrix for $i = 1, \ldots, 80$, where empty rows are removed. The running times of `LSRN` are approximately linear in $n$. Both DGELSD and SPQR are slower than `LSRN` on the `tnimg` problems. More importantly, their running times show that DGELSD and SPQR do not have linear scalability. Blendenpik either doesn't apply to the rank-deficient cases or runs OOM.

We see that, though both methods taking advantage of sparsity, SPQR relies heavily on the sparsity pattern, and its performance is unpredictable until the sparsity pattern is analyzed, while `LSRN` doesn't rely on the sparsity pattern and always delivers predictable performance and, moreover, the min-length solution.

### 3.5.8 Scalability and choice of iterative solvers on clusters

In this section, we move to the Amazon EC2 cluster. The goals are to demonstrate that (1) `LSRN` scales well on clusters, and (2) the CS method is preferred to LSQR on clusters with high communication cost. The test problems are submatrices of the `tnimg` matrix in the previous section: `tnimg_`4, `tnimg_`10, `tnimg_`20, and `tnimg_`40, solved with 4, 10, 20, and 40 cores respectively. Each process stores a submatrix of size $1024 \times$ 1e6. Table 3.4 shows the results, averaged over 5 runs. Ideally, from the complexity analysis (3.13), when we double $n$ and double the number of cores, the increase in running time should be a constant if the cluster is homogeneous and has perfect load balancing (which we have observed is not true on Amazon EC2). For `LSRN` with CS, from `tnimg_`10 to `tnimg_`20 the running time increases 27.6 seconds, and from `tnimg_`20 to `tnimg_`40 the running time increases 34.7 seconds. We believe the difference between the time increases is caused by the heterogeneity of the cluster, because Amazon EC2 doesn't guarantee the connection speed among nodes. From `tnimg_`4 to `tnimg_`40, the problem scale is enlarged by a factor of 10 while the running time only increases by a factor of 50%. The result still demonstrates `LSRN`'s good scalability. We also compare the performance of LSQR and CS as the iterative solvers in `LSRN`. For all problems LSQR converges in 84 iterations and CS converges in 106 iterations. However, LSQR is slower than CS. The communication cost saved by CS is significant on those tests. As a result, we recommend CS as the default `LSRN` iterative solver for cluster environments. Note that to reduce the communication cost on a cluster, we could also consider increasing $\gamma$ to reduce the number of iterations.

Table 3.4: Test problems on the Amazon EC2 cluster and corresponding running times in seconds. When we enlarge the problem scale by a factor of 10 and increase the number of cores accordingly, the running time only increases by a factor of 50%. It shows LSRN's good scalability. Though the CS method takes more iterations, it is faster than LSQR by saving communication cost.

| solver | $N_{\text{cores}}$ | matrix | $m$ | $n$ | nnz | $N_{\text{iter}}$ | $T_{\text{iter}}$ | $T_{\text{total}}$ |
|---|---|---|---|---|---|---|---|---|
| LSRN w/ CS | 4 | tnimg_4 | 1024 | 4e6 | 8.4e7 | 106 | 34.03 | 170.4 |
| LSRN w/ LSQR | | | | | | 84 | 41.14 | 178.6 |
| LSRN w/ CS | 10 | tnimg_10 | 1024 | 1e7 | 2.1e8 | 106 | 50.37 | 193.3 |
| LSRN w/ LSQR | | | | | | 84 | 68.72 | 211.6 |
| LSRN w/ CS | 20 | tnimg_20 | 1024 | 2e7 | 4.2e8 | 106 | 73.73 | 220.9 |
| LSRN w/ LSQR | | | | | | 84 | 102.3 | 249.0 |
| LSRN w/ CS | 40 | tnimg_40 | 1024 | 4e7 | 8.4e8 | 106 | 102.5 | 255.6 |
| LSRN w/ LSQR | | | | | | 84 | 137.2 | 290.2 |

| | $n = 1000$ | $n = 2000$ | $n = 3000$ | $n = 4000$ |
|---|---|---|---|---|
| CRT11 | 98.0 | 327.7 | 672.3 | 1147.9 |
| LSRN | 101.1 | 293.1 | 594.0 | 952.2 |

Table 3.5: Running times (in seconds) on full-rank dense over-determined problems of size $10^6 \times n$, where $n$ ranges from 1000 to 4000. LSRN is slightly slower than CRT11 when $n = 1000$ and becomes faster when $n = 2000$, 3000, and 4000, which is consistent with our theoretical analysis.

### 3.5.9 Comparison with Coakley et al.

Coakley et al. [22] introduced a least squares solver, referred to as CRT11, based on preconditioned normal equation, where the preconditioning matrix is computed via a random normal projection $G$, with $G \in \mathbb{R}^{(n+4)\times m}$. We implemented a multi-threaded version of CRT11 that shares the code base used by LSRN and uses $\mathcal{O}(m + n^2)$ RAM by computing in blocks. In this section, we report some comparison results between CRT11 and LSRN.

It is easy (and we omit details) to derive the time complexity of CRT11, which requires applying $A$ or $A^T$ $3n + 6$ times, while from section 3.3.4 we know that LSRN needs roughly $2n + 200$ matrix-vector multiplications under the default setting. So LSRN is asymptotically faster than CRT11 in theory. We compare the running times of LSRN and CRT11 on dense strongly over-determined least square problems, where $m$ is fixed at $10^6$ while $n$ ranges from 1000 to 3000, and $A$ has full rank. The test problems are generated the same way as in section 3.5.5. We list the running times in Table 3.5, where we see that LSRN is slightly slower than CRT11 when $n = 1000$ and becomes faster when $n = 2000$, 3000, and 4000.

Hardware limitations prevented testing larger problems. We believe that the difference should be much clearer if $A$ is an expensive operator, for example, if applying $A$ or $A^T$ requires solving a partial differential equation. Based on the evaluation result, we would recommend CRT11 over LSRN if $n \leq 1000$, and LSRN over CRT11 otherwise.

In [22], the authors showed that, unlike the original normal equation approach, CRT11 is very reliable on a broad range of matrices because the condition number of the preconditioned system

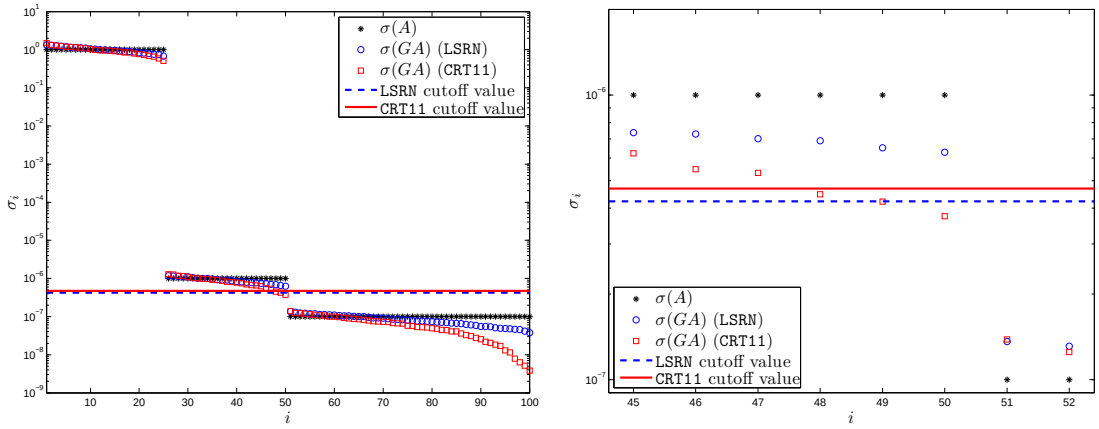


Figure 3.6: Left: Comparison of the spectrum of $A$ and $GA$ for both CRT11 and LSRN (rescaled by $1/\sqrt{s}$ for better alignment, where $s = n + 4$ for CRT11 and $s = 2n$ for LSRN) and the cutoff values in determining the effective rank of $A$. Right: Zoomed in to show that the effective rank estimated by CRT11 is 47, while LSRN outputs the correct effect rank, which is 50.

is not very large ($\approx 1000$). This is true for full-rank matrices. However, the authors did not show how CRT11 works on approximately rank-deficient problems. Based on our analysis in section 3.3.3, we need $\sigma_k$ and $\sigma_{k+1}$ well-separated with respect to the distortion introduced by $G$ in order to determine the effective rank correctly. In LSRN we choose $G \in \mathbb{R}^{2n\times m}$, which leads to a small constant distortion (with high probability), while in CRT11 we have $G \in \mathbb{R}^{(n+4)\times m}$, which leads to a relatively large distortion. It suggests CRT11 might be less reliable than LSRN in estimating the rank of an approximately rank-deficient problem. To verify this, we use the following Matlab script to generate a test problem:

```
sigma = [ones(1, n/4), 1/kappa*ones(1, n/4), e*ones(1, n/2)];
U = orth(randn(m, n)); A = U*diag(sparse(sigma));
```

where we choose $m = 10000$, $n = 100$, $\kappa = 10^6$, and $e = 10^{-7}$. Thus we have $A$'s effective rank $k = 50$, $\sigma_1(A) = 1$, $\sigma_k = 10^{-6}$, and $\sigma_{k+1} = 10^{-7}$. To estimate the effective rank, we set $c = \sqrt{\sigma_k\sigma_{k+1}}/\sigma_1 = 10^{-6.5}$, and singular values of $\tilde{A} = GA$ that are smaller than $c\sigma_1(\tilde{A})$ are treated as zeros. Figure 3.6 compares the singular values of $A$ and $GA$ for both CRT11 and LSRN (rescaled by $1/\sqrt{s}$ for better alignment, where $s = n + 4$ for CRT11 and $s = 2n$ for LSRN). We see that CRT11 introduces more distortion than LSRN to the spectrum of $A$. In this example, the rank determined by CRT11 is 47, while LSRN outputs the correct effective rank. We note that LSRN is not risk-free for approximately rank-deficient problems, which still should have sufficient separation between $\sigma_k$ and $\sigma_{k+1}$. However, it is more reliable than CRT11 on approximately rank-deficient problems because of less distortion introduced by $G$.

# Chapter 4

# $\ell_p$ Regression

From the analysis in Section 2.7, we see that a low-distortion $\ell_p$ subspace embedding is a fundamental building block (and very likely a bottleneck) for $(1 \pm \epsilon)$-distortion $\ell_p$ subspace embeddings, as well as for a $(1+\epsilon)$-approximation to an $\ell_p$ regression problem. In this chapter we show that, given a matrix $A \in \mathbb{R}^{m \times n}$ with $m \gg n$ and a $p \in [1, 2)$, with a constant probability we can construct a low-distortion embedding matrix $\Phi \in \mathbb{R}^{\text{poly}(n) \times m}$ that embeds $\mathcal{A}_p$, the $\ell_p$ subspace spanned by $A$'s columns, into $(\mathbb{R}^{\mathcal{O}(\text{poly}(n))}, \|\cdot\|_p)$. The distortion of our embeddings is only $\mathcal{O}(\text{poly}(n))$, and we can compute $\Phi A$ in $\mathcal{O}(\text{nnz}(A))$ time, i.e., input-sparsity time. Our result generalizes the input-sparsity time $\ell_2$ subspace embedding proposed by Clarkson and Woodruff [21] (see Theorem 3 for a simpler and improved analysis of their construction). These input-sparsity time $\ell_p$ embeddings are optimal, up to constants, in terms of their running time; and the improved running time propagates to applications such as $(1 \pm \epsilon)$-distortion $\ell_p$ subspace embedding and relative-error $\ell_p$ regression, our main interest in this work. Via the subspace-preserving sampling procedure, we show that a $(1 \pm \epsilon)$-distortion embedding of $\mathcal{A}_p$ into $\mathbb{R}^{\mathcal{O}(\text{poly}(n))}$ can be computed in $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time, and we also show that a $(1+\epsilon)$-approximate solution to the $\ell_p$ regression problem $\min_{x \in \mathbb{R}^n} \|Ax - b\|_p$ can be computed in $\mathcal{O}(\text{nnz}(A) \cdot \log m + \text{poly}(n) \log(1/\epsilon)/\epsilon^2)$ time. Moreover, we can improve the embedding dimension or equivalently the sample size to $\mathcal{O}(n^{3+p/2} \log(1/\epsilon)/\epsilon^2)$ without increasing the complexity.

## 4.1 Preliminaries

To state our results, we assume that we are capable of computing a $(1+\epsilon)$-approximate solution to an $\ell_p$ regression problem of size $m' \times n$ for some $\epsilon > 0$, as long as $m'$ is independent of $m$. Let us denote the running time needed to solve this smaller problem by $\mathcal{T}_p(\epsilon; m', n)$. In theory, we have $\mathcal{T}_p(\epsilon; m', n) = \mathcal{O}((m'n^2 + \text{poly}(n)) \log(n'/\epsilon))$ for general $p$ (see, e.g., Mitchell [52]).

Our analysis relies heavily on the property of stable distributionss and some related tail inequalities.

**Stable distributions.** The properties of $p$-stable distributions are essential for constructing input-sparsity time low-distortion $\ell_p$ subspace embeddings.

**Definition 10** ($p$-stable Distribution)**.** *A distribution $\mathcal{D}$ over $\mathbb{R}$ is called $p$-stable if for any $l$ real numbers $a_1, \ldots, a_l$ we have*

$$\sum_{i=1}^{l} a_i X_i \simeq \left( \sum_{i=1}^{l} |a_i|^p \right)^{1/p} X,$$

*where $X_i \overset{iid}{\sim} \mathcal{D}$ and $X \sim \mathcal{D}$. By "$X \simeq Y$", we mean $X$ and $Y$ have the same distribution.*

By a result due to Lévy [44], it is known that $p$-stable distributions exist for $p \in (0, 2]$; and from Chambers et al. [16], it is known that $p$-stable random variables can be generated efficiently, thus allowing their practical use. Let us use $\mathcal{D}_p$ to denote the "standard" $p$-stable distribution, for $p \in [1, 2]$, specified by its characteristic function $\psi(t) = e^{-|t|^p}$. It is known that $\mathcal{D}_1$ is the standard Cauchy distribution, and that $\mathcal{D}_2$ is the Gaussian distribution with mean 0 and variance 2.

**Tail inequalities.** We note two inequalities from Clarkson et al. [20] regarding the tails of the Cauchy distribution.

**Lemma 20** (Cauchy Upper Tail Inequality)**.** *For $i = 1, \ldots, l$, let $C_i$ be $l$ (not necessarily independent) standard Cauchy variables, and $\gamma_i > 0$ with $\gamma = \sum_i \gamma_i$. Let $X = \sum_i \gamma_i |C_i|$. For any $t > 1$,*

$$\Pr[X > t\gamma] \leq \frac{1}{\pi t} \left( \frac{\log(1 + (2lt)^2)}{1 - 1/(\pi t)} + 1 \right).$$

*For simplicity, we assume that $l \geq 3$ and $t \geq 1$, and then we have $\Pr[X > t\gamma] \leq 2\log(lt)/t$.*

**Lemma 21** (Cauchy Lower Tail Inequality)**.** *For $i = 1, \ldots, l$, let $C_i$ be independent standard Cauchy random variables, and $\gamma_i \geq 0$ with $\gamma = \sum_i \gamma_i$. Let $X = \sum_i \gamma_i |C_i|$. Then, for any $t > 0$,*

$$\log \Pr[X \leq (1 - t)\gamma] \leq \frac{-\gamma t^2}{3 \max_i \gamma_i}.$$

We also note the following result about Gaussian variables. This is a direct consequence of Maurer's inequality ([49]), and we use it to derive lower tail inequalities for $p$-stable distributions.

**Lemma 22** (Gaussian Lower Tail Inequality)**.** *For $i = 1, \ldots, l$, let $G_i$ be independent standard Gaussian random variables, and $\gamma_i \geq 0$ with $\gamma = \sum_i \gamma_i$. Let $X = \sum_i \gamma_i |G_i|^2$. Then, for any $t > 0$,*

$$\log \Pr[X \leq (1 - t)\gamma] \leq \frac{-\gamma t^2}{6 \max_i \gamma_i}.$$

## 4.2 Low-distortion $\ell_1$ embedding in input-sparsity time

Here is our main result for input-sparsity time low-distortion subspace embeddings for $\ell_1$.

**Theorem 11** (Low-distortion embedding for $\ell_1$)**.** *Given $A \in \mathbb{R}^{m\times n}$ with full column rank, let $\Phi = SC \in \mathbb{R}^{s\times m}$, where $S \in \mathbb{R}^{s\times m}$ has each column chosen independently and uniformly from the $s$ standard basis vectors of $\mathbb{R}^s$, and where $C \in \mathbb{R}^{m\times m}$ is a diagonal matrix with diagonals chosen independently from the standard Cauchy distribution. Set $s = \omega n^5 \log^5 n$ with $\omega$ sufficiently large. Then with a constant probability, we have*

$$1/\,\mathcal{O}(n^2 \log^2 n) \cdot \|Ax\|_1 \le \|\Phi Ax\|_1 \le \mathcal{O}(n \log n) \cdot \|Ax\|_1, \quad \forall x \in \mathbb{R}^n\,.$$

*In addition, $\Phi A$ can be computed in $\mathcal{O}(\mathrm{nnz}(A))$ time.*

The construction of the $\ell_1$ subspace embedding matrix is different from its $\ell_2$ norm counterpart (Theorem 3) only by the diagonal elements of $D$ (or $C$): whereas we use $\pm 1$ for the $\ell_2$ norm, we use Cauchy variables for the $\ell_1$ norm. Although our simpler direct proof leads to a better result for $\ell_2$ subspace embedding, the technique used in the proof of Clarkson and Woodruff [21], which splits coordinates into "heavy" and "light" sets based on the leverage scores, highlights an important structural property of $\ell_2$ subspace: that only a small subset of coordinates can have large $\ell_2$ leverage scores. (We note that the technique of splitting coordinates is also used by Ailon and Liberty [4] to get an unrestricted fast Johnson-Lindenstrauss transform; and that the difficulty in finding and approximating the large-leverage directions was—until recently [47, 29]—responsible for difficulties in obtaining fast relative-error random sampling algorithms for $\ell_2$ regression and low-rank matrix approximation.) An analogous structural fact holds for $\ell_1$ and other $\ell_p$ spaces. Using this property, we can construct novel input-sparsity time $\ell_p$ subspace embeddings for general $p \in [1, 2)$.

### 4.2.1 Proof of Theorem 11

The proof of Theorem 11 uses the technique of splitting coordinates, the fact that the Cauchy distribution is 1-stable, and the upper and lower tail inequalities regarding the Cauchy distribution from Lemmas 20 and 21.

We start with the following result, which establishes the existence of the so-called Auerbach's basis of a $d$-dimensional normed vector space. For our proof, we only need its existence and not an algorithm to construct it.

Auerbach's lemma (Lemma 4) implies that a $(n, 1, 1)$-conditioned basis matrix of $\mathcal{A}_1$ exists, which is denoted by $U$ throughout the proof. By definition, $U$'s columns are unit vectors in the $\ell_1$ norm (thus $|U|_1 = n$, where recall that $|\cdot|_1$ denotes the element-wise $\ell_1$ norm of a matrix) and $\|x\|_\infty \le \|Ux\|_1,\ \forall x \in \mathbb{R}^n$. Denote by $u_j$ the $j$-th row of $U$, $j = 1, \ldots, m$. Define $v_j = \|u_j\|_1$ the $\ell_1$ leverage scores of $A$. We have $\sum_j v_j = |U|_1 = n$. Let $\tau > 0$ to be determined later, and define two index sets $H = \{j \,|\, v_j \ge \tau\}$ and $L = \{j \,|\, v_j < \tau\}$. It is easy to see that $|H| \le \frac{n}{\tau}$ where $|\cdot|$ is used to denote the size of a finite set, and $\|v^L\|_\infty \le \tau$ where

$$v_j^L = \begin{cases} v_j, & \text{if } j \in L \\ 0, & \text{otherwise} \end{cases}, \quad j = 1, \ldots, m.$$

Similarly, when an index set appears as a superscript, we mean zeroing out elements or rows that do not belong to this index set, e.g., $v^L$ and $U^L$. Define

$$Y = \{y \in \mathbb{R}^m \mid y = Ux,\ \|x\|_\infty = 1,\ x \in \mathbb{R}^n\}.$$

For any $y = Ux \in Y$, we have $\|y\|_1 = \|Ux\|_1 \geq \|x\|_\infty = 1$,

$$|y_j| = |u_j^T x| \leq \|u_j\|_1 \|x\|_\infty = v_j, \quad j = 1, \ldots, m,$$

and thus $\|y\|_1 \leq \|v\|_1 = n$. Define $Y^L = \{y \in Y \mid \|y^L\|_1 \geq \frac{1}{2}\|y\|_1\}$ and $Y^H = Y \backslash Y^L$. Given $S$, define a mapping $\phi : \{1, \ldots, m\} \to \{1, \ldots, s\}$ such that $s_{\phi(j),j} = 1$, $j = 1, \ldots, m$, and split $L$ into two subsets: $\hat{L} = \{j \in L \mid \phi(j) \in \phi(H)\}$ and $\bar{L} = L \backslash \hat{L}$. Consider these events:

- $\mathcal{E}_U$: $|\Phi U|_1 \leq \omega_1 n \log n$ for some $\omega_1 > 0$.
- $\mathcal{E}_L$: $\|S v^L\|_\infty \leq \omega_2/(n \log n)$ for some $\omega_2 > 0$.
- $\mathcal{E}_H$: $\phi(j_1) \neq \phi(j_2),\ \forall\, j_1 \neq j_2,\ j_1, j_2 \in H$.
- $\mathcal{E}_C$: $\min_{j \in |H|} |c_j| \geq \omega_3/(n^2 \log^2 n)$ for some $\omega_3 > 0$.
- $\mathcal{E}_{\hat{L}}$: $|\Phi U^{\hat{L}}|_1 \leq \omega_4/(n^2 \log^2 n)$ for some $\omega_4 > 0$.

Recall that we set $s = \omega n^5 \log^5 n$ in Theorem 11. We show that, with $\omega$ sufficiently large and proper choices of $\omega_1$, $\omega_2$, $\omega_3$, and $\omega_4$, the event $\mathcal{E}_U$ leads to an upper bound of $\|\Phi y\|_1$ for all $y \in \text{range}(A)$, $\mathcal{E}_U$ and $\mathcal{E}_L$ lead to a lower bound of $\|\Phi y\|_1$ for all $y \in Y^L$ with probability at least 0.9, and $\mathcal{E}_H$, $\mathcal{E}_{\hat{L}}$, and $\mathcal{E}_C$ together imply an lower bound of $\|\Phi y\|_1$ for all $y \in Y^H$.

**Lemma 23.** *Provided $\mathcal{E}_U$, we have*

$$\|\Phi y\|_1 \leq \omega_1 n \log n \cdot \|y\|_1, \quad \forall y \in \text{range}(A).$$

*Proof.* For any $y \in \text{range}(A)$, we can find an $x$ such that $y = Ux$. Then,

$$\|\Phi y\|_1 = \|\Phi U x\|_1 \leq |\Phi U|_1 \|x\|_\infty \leq |\Phi U|_1 \|Ux\|_1 \leq \omega_1 n \log n \cdot \|y\|_1.$$

□

**Lemma 24.** *Provided $\mathcal{E}_L$, for any fixed $y \in Y^L$, we have*

$$\log \Pr\left[\|\Phi y\|_1 \leq \frac{1}{4}\|y\|_1\right] \leq -\frac{n \log n}{24 \omega_2}.$$

*Proof.* Let $z = \Phi y$. We have,

$$|z_i| = \left|\sum_j s_{ij} c_j y_j\right| \simeq \left(\sum_j s_{ij} |y_j|\right) |\tilde{c}_i| \succeq \left(\sum_j s_{ij} |y_j^L|\right) |\tilde{c}_i| := \tilde{\gamma}_i |\tilde{c}_i|,$$

where $\{\tilde{c}_i\}$ are independent Cauchy variables. Let $\tilde{\gamma} = \sum_i \tilde{\gamma}_i = \|y^L\|_1$. Since $|y| \leq v$, we have $\tilde{\gamma}_i \leq \|Sv^L\|_\infty$. By Lemma 21,

$$\log \Pr\left[X \leq \frac{\|y^L\|_1}{2}\right] \leq \frac{-\|y^L\|_1}{12\|Sv^L\|_\infty}.$$

By assumption $\mathcal{E}_L$ and $\|y^L\|_1 \geq \frac{1}{2}\|y\|_1 \geq \frac{1}{2}$, we obtain the result. □

**Lemma 25.** *Assume both $\mathcal{E}_U$ and $\mathcal{E}_L$. If $\omega_1$ and $\omega_2$ satisfy*

$$n \log (6n(1 + 4\omega_1 n \log n)) - \frac{n \log n}{24\omega_2} \leq \log \delta$$

*for some $\delta \in (0, 1)$ regardless of $n$, then, with probability at least $1 - \delta$, we have*

$$\|\Phi y\|_1 \geq \frac{1}{8}\|y\|_1, \quad \forall y \in Y^L.$$

*Proof.* Set $\epsilon = 1/(2 + 8\omega_1 n \log n)$ and create an $\epsilon$-net $Y^L_\epsilon \subseteq Y^L$ such that for any $y \in Y^L$, we can find a $y_\epsilon \in Y^L_\epsilon$ such that $\|y - y_\epsilon\|_1 \leq \epsilon$. Since $\|y\|_1 \leq n$ for all $y \in Y^L$, there exists such an $\epsilon$-net with at most $(3n/\epsilon)^n$ elements (Bourgain et al. [10]). By Lemma 24, we can apply a union bound for all the elements in $Y^L_\epsilon$:

$$\Pr[\|\Phi y_\epsilon\|_1 \geq \frac{1}{4}\|y_\epsilon\|_1, \ \forall y_\epsilon \in Y^L_\epsilon] \geq 1 - \left(\frac{3n}{\epsilon}\right)^n e^{-\frac{n \log n}{24\omega_2}} = 1 - e^{n \log \frac{3n}{\epsilon} - \frac{n \log n}{24\omega_2}} \geq 1 - \delta.$$

For any $y \in Y^L$, we have, noting that $y - y_\epsilon \in \text{range}(A)$,

$$\begin{aligned}\|\Phi y\|_1 &\geq \|\Phi y_\epsilon\|_1 - \|\Phi(y - y_\epsilon)\|_1 \geq \frac{1}{4}\|y_\epsilon\|_1 - \omega_1 n \log n \cdot \|y - y_\epsilon\|_1 \\ &\geq \frac{1}{4}\|y\|_1 - \left(\frac{1}{4} + \omega_1 n \log n\right)\epsilon \geq \frac{1}{8}\|y\|_1.\end{aligned}$$

So we establish a lower bound for all $y \in Y^L$. □

**Lemma 26.** *Provided $\mathcal{E}_H$ and $\mathcal{E}_{\hat{L}}$, if $\omega_3 > 4\omega_4$, we have*

$$\|\Phi y\|_1 \geq \frac{\omega_4}{n^2 \log^2 n}\|y\|_1, \quad \forall y \in Y^H.$$

*Proof.* For any $y = Ux \in Y^H$, we have,

$$\begin{aligned}\|\Phi y\|_1 &\geq \|\Phi(y^H + y^{\hat{L}})\|_1 \geq \|\Phi y^H\|_1 - \|\Phi U^{\hat{L}} x\|_1, \\ &\geq \sum_{j \in H} |c_j||y_j| - |\Phi U^{\hat{L}}|_1 \|x\|_\infty \geq \min_{j \in H} |c_j| \|y^H\|_1 - |\Phi U^{\hat{L}}|_1 \\ &\geq \left(\frac{\omega_3}{2n^2 \log^2 n} - \frac{\omega_4}{n^2 \log^2 n}\right)\|y\|_1 \geq \frac{\omega_4}{n^2 \log^2 n} \cdot \|y\|_1,\end{aligned}$$

which creates a lower bound for all $y \in Y^H$. □

We continue to show that, with $\omega$ sufficiently large, by setting $\tau = \omega^{1/4}/(n \log^2 n)$ and choosing $\omega_1$, $\omega_2$, $\omega_3$, and $\omega_4$ properly, we have each event with probability at least $1 - 0.08 = 0.92$ and thus

$$\Pr[\mathcal{E}_U \cap \mathcal{E}_L \cap \mathcal{E}_H \cap \mathcal{E}_{\hat{L}} \cap \mathcal{E}_C] \geq 0.6.$$

Moreover, the condition in Lemma 25 holds with $\delta = 0.1$, and the condition in Lemma 26 holds. Therefore, $\Phi = SC$ has the desired property with probability at least 0.5, which would conclude the proof of Theorem 11.

**Lemma 27.** *With probability at least* 0.92, $\mathcal{E}_U$ *holds with* $\omega_1 = 500(1 + \log \omega)$.

*Proof.* With $S$ fixed, we have,

$$|\Phi U|_1 = |SCU|_1 = \sum_{k=1}^{n} \sum_{i=1}^{s} |\sum_{j=1}^{m} s_{ij} c_j u_{jk}| \simeq \sum_{k=1}^{n} \sum_{i=1}^{s} \sum_{j=1}^{m} (|s_{ij} u_{jk}|) \, |\tilde{c}_{ik}|,$$

where $\{\tilde{c}_{ik}\}$ are *dependent* Cauchy random variables. We have

$$\sum_{k=1}^{n} \sum_{i=1}^{s} \sum_{j=1}^{m} |s_{ij} u_{jk}| = \sum_{k=1}^{n} \sum_{j=1}^{m} |u_{jk}| = |U|_1 = n.$$

Apply Lemma 20,

$$\Pr[|\Phi U|_1 \geq tn \,|\, S] \leq \frac{2 \log(snt)}{t}.$$

Setting $\omega_1 = 500(1 + \log \omega)$ and $t = \omega_1 \log m$, we have

$$\frac{2 \log(snt)}{t} = \frac{2 \log(\omega \omega_1 n^6 \log^5 n)}{\omega_1 \log m} \leq 0.08.$$

We assume that $\log m \geq 1$ and $\log \omega \geq 1$. □

**Lemma 28.** *For any* $\delta \in (0, 0.1)$, *if* $s \geq n/\tau$, *we have,*

$$\Pr\left[\|Sv^L\|_\infty \geq \left(1 + 2 \log \frac{n}{\delta \tau}\right) \cdot \tau\right] \leq \delta.$$

*Proof.* Let $X_{ij} = s_{ij} v_j^L$. We have $\mathbf{E}[X_{ij}] = v_j^L/s$, $\mathbf{E}[X_{ij}^2] = (v_j^L)^2/s$, and $0 \leq X_{ij} \leq v_j^L \leq \tau$. Fixed $i$, $X_{ij}$ are independent, $j = 1, \ldots, m$. By Bernstein's inequality,

$$\log \Pr\left[\sum_j X_{ij} \geq \frac{\|v^L\|_1}{s} + t\right] \leq \frac{-t^2/2}{\|v^L\|_2^2/s + \tau t/3} \leq \frac{-t^2/2}{\tau(\|v^L\|_1/s + t/3)} \leq \frac{-t^2/(2\tau)}{n/s + t/3}.$$

where we use Holder's inequality: $\|v^L\|_2^2 \leq \|v^L\|_1 \|v^L\|_\infty \leq n\tau$. To obtain a union bound for all $i$

with probability $1-\delta$, we need

$$\frac{-t^2/(2\tau)}{n/s+t/3}+\log s\le\log\delta.$$

Given $\delta<0.1$, it suffices to choose $s=n/\tau$ and $t=2\log(n/(\delta\tau))\tau$. Note that $\|v^L\|_1/s\le\|v\|_1/s=\tau$. We have

$$\Pr\left[\|Sv^L\|_\infty\ge\left(1+2\log\frac{n}{\delta\tau}\right)\cdot\tau\right]\le\delta.$$

Increasing $s$ decreases the failure rate, so it holds for all $s\ge n/\tau$. □

**Lemma 29.** *With probability at least* 0.92, $\mathcal{E}_L$ *holds with* $\omega_2=(15+\log\omega)/\omega^{1/4}$.

*Proof.* By Lemma 28, with probability at least 0.92, $\mathcal{E}_L$ holds with

$$\omega_2=\frac{1+2\log\frac{\omega^{1/4}n^2\log^2 n}{0.08}}{\omega^{1/4}\log n}\le\frac{15+\log\omega}{\omega^{1/4}}.$$

□

**Lemma 30.** *With the above choices of* $\omega_1$ *and* $\omega_2$*, the condition in Lemma 24 holds with* $\delta=0.1$ *for sufficiently large* $\omega$.

*Proof.* With $\omega_1=500(1+\log\omega)$, and $\omega_2=(15+\log\omega)/\omega^{1/4}$, the first term in

$$n\log\left(6n(1+4\omega_1 n\log n)\right)-\frac{n\log n}{24\omega_2}$$

increases much slower than the second term as $\omega$ increases, while both are of order $n\log n$. Therefore, if $\omega$ is sufficiently large, the condition holds with $\delta=0.1$. □

**Lemma 31.** *If* $\omega\ge160$*, event* $\mathcal{E}_H$ *holds with probability at least* 0.92.

*Proof.* Given $j_1,j_2\in H$ and $j_1\ne j_2$, let $X_{j_1j_2}=1$ if $\phi(j_1)=\phi(j_2)$ and $X_{j_1j_2}=0$ otherwise. It is easy to see that $\Pr[X_{j_1j_2}=1]=\frac{1}{s}$. Therefore,

$$\Pr[\mathcal{E}_H]\ge1-\sum_{j_1<j_2}\Pr[X_{j_1j_2}=1]\ge1-\frac{|H|^2}{s}\ge1-\frac{n^2}{s\tau^2}\ge1-\frac{1}{\omega^{1/2}}.$$

It suffices to have $\omega\ge160$. □

**Lemma 32.** *With probability at least* 0.92*, event* $\mathcal{E}_C$ *holds with* $\omega_3=1/(8\omega^{1/4})$.

*Proof.* Let $c$ be a Cauchy variable. We have

$$\Pr[|c|\le t]=\frac{2}{\pi}\tan^{-1}t\le\frac{2t}{\pi}.$$

$|H|$ is at most $n/\tau = \omega^{1/4} n^2 \log^2 n$. Then

$$\begin{aligned}\Pr[\mathcal{E}_C] &\geq 1 - |H| \cdot \Pr\left[|c| < \frac{\omega_3}{n^2 \log^2 n}\right] \\ &\geq 1 - \omega^{1/4} n^2 \log^2 n \cdot \frac{2\omega_3}{\pi n^2 \log^2 n}.\end{aligned}$$

Therefore, $\omega_3 = 1/(8\omega^{1/4})$ would suffice. □

**Lemma 33.** *With probability at least* 0.92*, event* $\mathcal{E}_{\hat{L}}$ *holds with* $\omega_4 = 25000(1 + \log \omega)/\omega^{3/4}$*. Thus with* $\omega$ *sufficiently large and the above choice of* $\omega_3$*, the condition in Lemma 26* $\omega_3 > 4\omega_4$ *holds.*

*Proof.* We have,

$$\mathbf{E}[|U^{\hat{L}}|_1] = \frac{|H|}{s}|U^L|_1 \leq \frac{\omega^{1/4} n^2 \log^2 n}{\omega d^5 \log^5 n} \cdot n = \frac{1}{\omega^{3/4} n^2 \log^3 n}.$$

By Markov's inequality,

$$\Pr\left[|U^{\hat{L}}|_1 \geq \frac{25}{\omega^{3/4} n^2 \log^3 n}\right] \leq 0.04.$$

Assume that $|U^{\hat{L}}|_1 \leq \frac{25}{\omega^{3/4} n^2 \log^3 n}$. Similar to the proof of Lemma 27, we have

$$|\Phi U^{\hat{L}}|_1 = \sum_{k=1}^{n} \sum_{i \in \phi(H)} |\sum_j s_{ij} c_j u^{\hat{L}}_{jk}| \simeq \sum_{k=1}^{n} \sum_{i \in \phi(H)} \left(\sum_j s_{ij} |u^{\hat{L}}_{jk}|\right) |\tilde{c}_{ik}|,$$

where $\{\tilde{c}_{ik}\}$ are *dependent* Cauchy variables. Apply Lemma 20,

$$\Pr[|\Phi U^{\hat{L}}| \geq |U^{\hat{L}}| t] \leq \frac{2\log(|H| nt)}{t}.$$

It suffices to choose $t = 1000(1 + \log \omega) \log n$ to make the RHS less than 0.04. So with probability at least 0.92, we have $\mathcal{E}_{\hat{L}}$ holds with $\omega_4 = 25000(1 + \log \omega)/\omega^{3/4}$. □

**Remark.** As mentioned above, the $\mathcal{O}(\text{nnz}(A))$ running time is optimal. Whether the distortion $\mathcal{O}(n^3 \log^3 n)$ is optimal is still an open question. However, for the same construction of $\Phi$, we can provide a "bad" case that provides a lower bound. Choose $A = \begin{pmatrix} I_n & \mathbf{0} \end{pmatrix}^T$. Suppose that $s$ is sufficiently large such that with an overwhelming probability, the top $d$ rows of $A$ are perfectly hashed, i.e., $\|\Phi Ax\|_1 = \sum_{k=1}^{n} |c_k||x_k|$, $\forall x \in \mathbb{R}^n$, where $c_k$ is the $k$-th diagonal of $C$. Then, the distortion of $\Phi$ is $\max_{k \leq d} |c_k| / \min_{k \leq d} |c_k| \approx \mathcal{O}(n^2)$. Therefore, at most an $\mathcal{O}(n \log^3 n)$ factor of the distortion is due to artifacts in our analysis.

## 4.3 Application to $\ell_1$ regression

Our input-sparsity time $\ell_1$ subspace embedding of Theorem 11 improves the $\mathcal{O}(\text{nnz}(A) \cdot n \log n)$-time embedding by Sohler and Woodruff [64] and the $\mathcal{O}(mn \log m)$-time embedding of Clarkson et al. [20]. In addition, by combining Theorem 11 and Lemma 17, we can compute a $(1 \pm \epsilon)$-distortion embedding in $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time, i.e., in *nearly* input-sparsity time.

**Theorem 12** (Fast $\ell_1$ subspace-preserving embedding)**.** *Given $A \in \mathbb{R}^{m \times n}$, it takes $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time to compute a sampling matrix $S \in \mathbb{R}^{s \times m}$, where $s = \mathcal{O}(\text{poly}(n) \cdot \log(1/\epsilon)/\epsilon^2)$, such that with a constant probability, $S$ embeds $\mathcal{A}_1$ into $(\mathbb{R}^s, \|\cdot\|_1)$ with distortion $1 \pm \epsilon$.*

Our improvements in Theorems 11 and 12 also propagate to related $\ell_1$-based applications, including the $\ell_1$ regression and the $\ell_1$ subspace approximation problem considered in [64, 20]. As before, only the regression improvement is stated here explicitly. For completeness, we present in Algorithm 5 our algorithm for solving $\ell_1$ regression problems in nearly input-sparsity time.

**Algorithm 5** Fast $\ell_1$ regression approximation in $\mathcal{O}(\text{nnz}(A) \cdot \log m + \text{poly}(n) \log(1/\epsilon)/\epsilon^2)$ time

**Input:** $A \in \mathbb{R}^{m \times n}$ with full column rank, $b \in \mathbb{R}^m$, and $\epsilon \in (0, 1/2)$.
**Output:** A $(1+\epsilon)$-approximation solution $\hat{x}$ to $\min_{x \in \mathbb{R}^n} \|Ax - b\|_1$, with a constant probability.

1: Let $\bar{A} = \begin{pmatrix} A & b \end{pmatrix}$ and denote $\bar{\mathcal{A}}_1$ the $\ell_1$ subspace spanned by $A$'s columns and $b$.
2: Compute a low-distortion embedding $\Phi \in \mathbb{R}^{\mathcal{O}(\text{poly}(n)) \times m}$ of $\bar{\mathcal{A}}_1$ (Theorem 11).
3: Compute $\bar{R} \in \mathbb{R}^{(n+1) \times (n+1)}$ from $\Phi \bar{A}$ such that $\bar{A}\bar{R}^{-1}$ is well-conditioned (QR or Corollary 1).
4: Compute a $(1 \pm \epsilon/4)$-distortion embedding $S \in \mathbb{R}^{\mathcal{O}(\text{poly}(n) \log(1/\epsilon)/\epsilon^2) \times m}$ of $\bar{\mathcal{A}}_1$ (Lemma 17).
5: Compute a $(1+\epsilon/4)$-approximate solution $\hat{x}$ to $\min_{x \in \mathbb{R}^n} \|SAx - Sb\|_1$.

**Corollary 2** (Fast $\ell_1$ regression)**.** *With a constant probability, Algorithm 5 computes a $(1+\epsilon)$-approximate solution to an $\ell_1$ regression problem in $\mathcal{O}(\text{nnz}(A) \cdot \log m + \mathcal{T}_1(\epsilon; \text{poly}(n) \log(1/\epsilon)/\epsilon^2, n))$ time.*

*Proof.* By Theorem 11 and Lemma 17, we know that Steps 2 and 4 of Algorithm 5 succeed with a constant probability. Conditioning on this event, we have

$$\begin{aligned} \|A\hat{x} - b\|_1 &\le \frac{1}{1-\epsilon/4} \|SA\hat{x} - Sb\|_1 \le \frac{1+\epsilon/4}{1-\epsilon/4} \|SAx^* - Sb\|_1 \\ &\le \frac{(1+\epsilon/4)^2}{1-\epsilon/4} \|Ax^* - b\|_1 \le (1+\epsilon)\|Ax^* - b\|_1, \end{aligned}$$

where the last inequality is due to $\epsilon < 1/2$. By Theorem 11, Step 2 takes $\mathcal{O}(\text{nnz}(A))$ time, and Step 3 takes $\mathcal{O}(\text{poly}(n))$ time because $\Phi A$ has $\mathcal{O}(\text{poly}(n)$ rows. Then, by Lemma 17, Step 4 takes $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time, and Step 5 takes $\mathcal{T}_1(\epsilon/4; \mathcal{O}(\text{poly}(n) \cdot \log(1/\epsilon)/\epsilon^2), d)$ time. Thus, the total running time of Algorithm 5 is as stated. □

## 4.4 Low-distortion $\ell_p$ embedding in input-sparsity time

In this section, we use the properties of $p$-stable distributions to generalize the input-sparsity time $\ell_1$ subspace embedding to $\ell_p$ norms, for $p \in (1,2)$. Generally, $\mathcal{D}_p$ does not have explicit PDF/CDF, which increases the difficulty for theoretical analysis. Indeed, the main technical difficulty here is that we are not aware of $\ell_p$ analogues of Lemmas 20 and 21 that would provide upper and lower tail inequality for $p$-stable distributions. (Indeed, even Lemmas 20 and 21 were established only recently [20].)

Instead of analyzing $\mathcal{D}_p$ directly, for any $p \in (1,2)$, we establish an order among the Cauchy distribution, the $p$-stable distribution, and the Gaussian distribution, and then we derive upper and lower tail inequalities for the $p$-stable distribution similar to the ones we used to prove Theorem 11. We state these technical results here since they are of independent interest.

**Lemma 34.** *For any $p \in (1,2)$, there exist constants $\alpha_p > 0$ and $\beta_p > 0$ such that*

$$\alpha_p |C| \succeq |X_p|^p \succeq \beta_p |G|^2,$$

*where $C$ is a standard Cauchy variable, $X_p \sim \mathcal{D}_p$, $G$ is a standard Gaussian variable. By "$X \succeq Y$" we mean $\Pr[X \geq t] \geq \Pr[Y \geq t]$, $\forall t \in \mathbb{R}$, i.e., $F_X(t) \leq F_Y(t)$, $\forall t \in \mathbb{R}$, where $F(\cdot)$ is the corresponding CDF.*

*Proof.* First, we know that

$$\Pr[|X_p|^p \geq t] = \Pr[|X_p| \geq t^{1/p}] = 2 \cdot \Pr[X_p \geq t^{1/p}].$$

Next, we state the following lemma, which is due to Nolan [59].

**Lemma 35.** *(Nolan [59, Thm. 1.12]) Let $X \sim \mathcal{D}_p$ with $p \in [1,2)$. Then as $x \to \infty$,*

$$\Pr[X > x] \sim c_p x^{-p},$$

*where $c_p = \sin \frac{\pi p}{2} \cdot \Gamma(p)/\pi$.*

By Lemma 35, it follows that, as $t \to \infty$,

$$\Pr[|X_p|^p \geq t] \sim 2c_p t^{-1}.$$

For the Cauchy distribution, we have

$$\Pr[|C| \geq t] = 1 - \frac{2}{\pi}\tan^{-1} t = \frac{2}{\pi}\tan^{-1}\frac{1}{t} \sim \frac{2}{\pi} \cdot t^{-1}.$$

Hence, there exist $\alpha_p' > 0$ and $t_1 > 0$ such that for all $t > t_1$,

$$\Pr[\alpha_p' |C| \geq t] \geq \Pr[|X_p|^p \geq t].$$

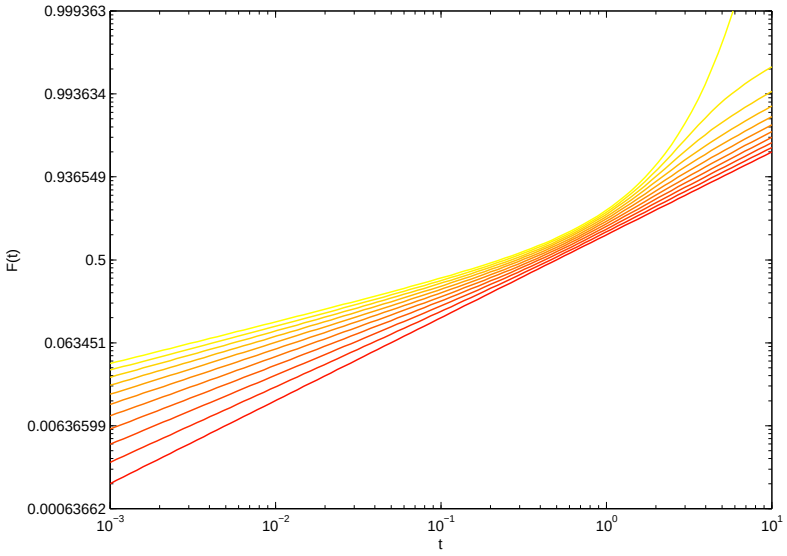


Figure 4.1: The CDFs ($F(t)$) of $|X_p/2|^p$ for $p = 1.0$ (bottom, i.e., red or dark gray), $1.1, \ldots, 2.0$ (top, i.e., yellow or light gray), where $X_p \sim \mathcal{D}_p$ and the scales of the axes are chosen to magnify the upper (as $t \to \infty$) and lower (as $t \to 0$) tails. These empirical results suggest $|X_{p_1}/2|^{p_1} \succeq |X_{p_2}/2|^{p_2}$ for all $1 \leq p_1 \leq p_2 \leq 2$.

Note that all the $p$-stable distributions with $p \in [1, 2]$ have finite and positive density at $x = 0$. Therefore, there exists $\alpha_p'' > 0$ such that for all $0 \leq t \leq t_1$,

$$\Pr[\alpha_p''|C| \geq t] \geq \Pr[|X_p|^p \geq t].$$

Let $\alpha_p = \max\{\alpha_p', \alpha_p''\}$. We get $\alpha_p|C| \succeq |X_p|^p$. For the Gaussian distribution, we have, as $t \to \infty$,

$$\Pr[|G|^2 \geq t] \sim 2e^{-t/2}t^{-1/2}.$$

which converges to zero much faster than $t^{-1}$, so we can apply similar arguments to obtain $\beta_p$. □

Our numerical results suggest that the constants $\alpha_p$ and $\beta_p$ are not too far away from 1. See Figure 4.1, which plots of the CDFs of $|X_p/2|^p$ for $p = 1, 0, 1.1, \ldots, 2.0$, based on which we conjecture $|X_{p_1}/2|^{p_1} \succeq |X_{p_2}/2|^{p_2}$, for all $1 \leq p_1 \leq p_2 \leq 2$. This implies that $2^{p-1}|C| \succeq |X_p|^p$ and $|X_p|^p \succeq 2^{p-2}|X_2|^2 \simeq 2^{p-1}|G|^2$, which therefore provides a value for the constants $\alpha_p$ and $\beta_p$.

Lemma 34 suggests that we can use Lemma 20 (regarding Cauchy random variables) to derive upper tail inequalities for general $p$-stable distributions and that we can use Lemma 22 (regarding Gaussian variables) to derive lower tail inequalities for general $p$-stable distributions. The following two lemmas establish these results.

**Lemma 36** (Upper tail inequality for $p$-stable distributions)**.** *Given $p \in (1, 2)$, for $i = 1, \ldots, l$, let $X_i$ be $l$ (not necessarily independent) random variables sampled from $\mathcal{D}_p$, and $\gamma_i > 0$ with $\gamma = \sum_i \gamma_i$.*

*Let $X = \sum_i \gamma_i |X_i|^p$. Assume that $l \geq 3$. Then for any $t \geq 1$,*

$$\Pr[X \geq t\alpha_p \gamma] \leq \frac{2\log(lt)}{t}.$$

*Proof.* Let $C_i = F_c^{-1}(F_p(X_i))$, $i = 1, \ldots, l$, where $F_c$ is the CDF of the standard Cauchy distribution and $F_p$ is the CDF of $\mathcal{D}_p$. $C_i$ follows the standard Cauchy distribution, and, by Lemma 34, we have $\alpha_p |C_i| \geq |X_i|^p$. Therefore, for any $t \geq 1$,

$$\Pr[X \geq t\alpha_p \gamma] \leq \Pr\left[\sum_i \gamma_i |C_i| \geq t\gamma\right] \leq \frac{2\log(lt)}{t}.$$

The last inequality is from Lemma 20. □

**Lemma 37** (Lower tail inequality for $p$-stable distributions)**.** *For $i = 1, \ldots, l$, let $X_i$ be independent random variables sampled from $\mathcal{D}_p$, and $\gamma_i \geq 0$ with $\gamma = \sum_i \gamma_i$. Let $X = \sum_i \gamma_i |c_i|$. Then,*

$$\log \Pr[X \leq (1-t)\beta_p \gamma] \leq \frac{-\gamma t^2}{6 \max_i \gamma_i}.$$

*Proof.* Let $G_i$ be independent random variables sampled from the standard Gaussian distribution, $i = 1, \ldots, l$. By Lemma 34, we have

$$\log \Pr[X \leq \beta_p (1-t)\gamma] \leq \log \Pr\left[\sum_i \gamma_i |G_i|^2 \leq (1-t)\gamma\right].$$

The lower tail inequality from Lemma 22 concludes the proof. □

Given these results, here is our main result for input-sparsity time low-distortion subspace embeddings for $\ell_p$. The proof of this theorem is similar to the proof of Theorem 11, except that we replace the $\ell_1$ norm $\|\cdot\|_1$ by $\|\cdot\|_p^p$ and use the tail inequalities from Lemmas 36 and 37 (rather than Lemmas 20 and 21).

**Theorem 13** (Low-distortion embedding for $\ell_p$)**.** *Given $A \in \mathbb{R}^{m \times n}$ with full column rank and $p \in (1, 2)$, let $\Phi = SD \in \mathbb{R}^{s \times m}$ where $S \in \mathbb{R}^{s \times m}$ has each column chosen independently and uniformly from the $s$ standard basis vectors of $\mathbb{R}^s$, and where $D \in \mathbb{R}^{m \times m}$ is a diagonal matrix with diagonals chosen independently from $\mathcal{D}_p$. Set $s = \omega n^5 \log^5 n$ with $\omega$ sufficiently large. Then with a constant probability, we have*

$$1/\,\mathcal{O}((n \log n)^{2/p}) \cdot \|Ax\|_p \leq \|\Phi Ax\|_p \leq \mathcal{O}((n \log n)^{1/p}) \cdot \|Ax\|_p, \quad \forall x \in \mathbb{R}^n.$$

*In addition, $\Phi A$ can be computed in $\mathcal{O}(\mathrm{nnz}(A))$ time.*

Similar to the $\ell_1$ case, our input-sparsity time $\ell_p$ subspace embedding of Theorem 13 improves the $\mathcal{O}(mn \log m)$-time embedding of Clarkson et al. [20]. Their construction (and hence the construction

of [21]) works for all $p \in [1, \infty)$, but it requires solving a rounding problem of size $\mathcal{O}(m/\operatorname{poly}(n)) \times n$ as an intermediate step, which may become intractable when $n$ is very large in a streaming environment, while our construction only needs $\mathcal{O}(\operatorname{poly}(n))$ storage. By combining Theorem 13 and Lemma 17, we can compute a $(1 \pm \epsilon)$-distortion embedding in $\mathcal{O}(\operatorname{nnz}(A) \cdot \log m)$ time.

**Theorem 14** ($(1 \pm \epsilon)$-distortion embedding for $\ell_p$)**.** *Given $A \in \mathbb{R}^{m \times n}$ and $p \in [1, 2)$, it takes $\mathcal{O}(\operatorname{nnz}(A) \cdot \log m)$ time to compute a sampling matrix $S \in \mathbb{R}^{s \times m}$ with $s = \mathcal{O}(\operatorname{poly}(n) \log(1/\epsilon)/\epsilon^2)$ such that with a constant probability, $S$ embeds $\mathcal{A}_p$ into $(\mathbb{R}^s, \|\cdot\|_p)$ with distortion $1 \pm \epsilon$.*

These improvements for $\ell_p$ subspace embedding also propagate to related $\ell_p$-based applications. In particular, we can establish an improved algorithm for solving the $\ell_p$ regression problem in nearly input-sparsity time.

**Corollary 3** (Fast $\ell_p$ regression)**.** *Given $p \in (1, 2)$, with a constant probability, a $(1+\epsilon)$-approximate solution to an $\ell_p$ regression problem can be computed in*

$$\mathcal{O}(\operatorname{nnz}(A) \cdot \log m + \mathcal{T}_p(\epsilon; \operatorname{poly}(n) \log(1/\epsilon)/\epsilon^2, n))$$

*time.*

For completeness, we also present a result for low-distortion dense embeddings for $\ell_p$ that the tail inequalities from Lemmas 36 and 37 enable us to construct.

**Theorem 15** (Low-distortion dense embedding for $\ell_p$)**.** *Given $A \in \mathbb{R}^{m \times n}$ with full column rank and $p \in (1, 2)$, let $\Phi \in \mathbb{R}^{s \times m}$ whose entries are i.i.d. samples from $\mathcal{D}_p$. If $s = \omega n \log n$ for $\omega$ sufficiently large, with a constant probability, we have*

$$1/\mathcal{O}(1) \cdot \|Ax\|_p \le \|\Phi Ax\|_p \le \mathcal{O}((n \log n)^{1/p}) \cdot \|Ax\|_p, \quad \forall x \in \mathbb{R}^n .$$

*In addition, $\Phi A$ can be computed in $\mathcal{O}(\operatorname{nnz}(A) \cdot n \log n)$ time.*

*Proof.* The proof is similar to the proof of Sohler and Woodruff [64, Theorem 5], except that the Cauchy tail inequalities are replaced by tail inequalities for the stable distributions. For simplicity, we omit the complete proof but show where to apply those tail inequalities. By Lemma 4, there exists a $(n^{1/p}, 1, p)$-conditioned basis matrix of $\mathcal{A}_p$, denoted by $U$. Thus, $|U|_p^p = n$, where recall that $|\cdot|_p$ denotes the element-wise $\ell_p$ norm of a matrix. We have,

$$|\Phi U|_p^p = \sum_{k=1}^{n} \|\Phi u_k\|_p^p = \sum_{k=1}^{n} \sum_{i=1}^{s} \left| \sum_{j=1}^{m} \Phi_{ij} u_{jk} \right|^p \simeq \sum_{k=1}^{n} \sum_{i=1}^{s} \|u_k\|_p^p |\tilde{X}_{ik}|^p,$$

where $\tilde{X}_{ik} \sim \mathcal{D}_p$. Applying Lemma 36, we get $\|\Phi U\|_p^p / s = \mathcal{O}(n \log n)$ with a constant probability. Define $Y = \{Ux \mid \|x\|_q = 1, x \in \mathbb{R}^n\}$. For any fixed $y \in Y$, we have

$$\|\Phi y\|_p^p = \sum_{i=1}^{s} \left| \sum_{j=1}^{m} \Phi_{ij} y_j \right|^p \simeq \sum_{i=1}^{s} \|y\|_p^p |\tilde{X}_i|^p,$$

where $\tilde{X}_i \overset{\text{iid}}{\sim} \mathcal{D}_p$. Applying Lemma 36, we get $\|\Phi y\|_p^p/s \leq 1/\mathcal{O}(1)$ with an exponentially small probability with respect to $s$. By choosing $s = \omega n \log n$ with $\omega$ sufficiently large and an $\epsilon$-net argument on $Y$, we can obtain a union lower bound of $\|\Phi y\|_p^p$ on all the elements of $Y$ with a constant probability. Then,

$$1/\mathcal{O}(1) \cdot \|y\|_p^p \leq \|\Phi y\|_p^p/s \leq |\Phi U|_p^p \|x\|_q^p \leq \mathcal{O}(n \log n) \cdot \|Ux\|_p^p = \mathcal{O}(n \log n)\|y\|_p^p, \quad y \in Y,$$

which gives us the desired result. □

**Remark.** The result in Theorem 15 is based on a dense $\ell_p$ subspace embeddings that is analogous to the dense Gaussian embedding for $\ell_2$ and the dense Cauchy embedding of [64] for $\ell_1$. Although the running time (if one is simply interested in FLOP counts in RAM) of Theorem 15 is somewhat worse than that of Theorem 13, the embedding dimension and condition number quality (the ratio of the upper bound on the distortion and the lower bound on the distortion) are much better. Our numerical implementations, both with the $\ell_1$ norm [20] and with the $\ell_2$ norm [50], strongly suggest that the latter quantities are more important to control when implementing randomized regression algorithms in large-scale parallel and distributed settings.

## 4.5 Improving the embedding dimension

In Theorems 11 and 13, the embedding dimension is $s = \mathcal{O}(\text{poly}(n) \log(1/\epsilon)/\epsilon^2)$, where the $\text{poly}(n)$ term is a somewhat large polynomial of $d$ that directly multiplies the $\log(1/\epsilon)/\epsilon^2$ term. (See the remark below for comments on the precise value of the $\text{poly}(n)$ term.) This is not ideal for the subspace embedding and the $\ell_p$ regression, because we want to have a small embedding dimension and a small subsampled problem, respectively. Here, we show that it is possible to decouple the large polynomial of $n$ and the $\log(1/\epsilon)/\epsilon^2$ term via another round of sampling and conditioning without increasing the complexity. See Algorithm 6 for details on this procedure. Theorem 16 provides our main quality-of-approximation result for Algorithm 6.

**Algorithm 6** Improving the embedding dimension

**Input:** $A \in \mathbb{R}^{m \times n}$ with full column rank, $p \in [1, 2)$, and $\epsilon \in (0, 1)$.

**Output:** A $(1 \pm \epsilon)$-distortion embedding $S \in \mathbb{R}^{\mathcal{O}(n^{3+p/2} \log(1/\epsilon)/\epsilon^2) \times m}$ of $\mathcal{A}_p$.

1: Compute a low-distortion embedding $\tilde{\Phi} \in \mathbb{R}^{\mathcal{O}(\text{poly}(n)) \times m}$ of $\mathcal{A}_p$ (Theorems 11 and 13).
2: Compute $\tilde{R} \in \mathbb{R}^{n \times n}$ from $\tilde{\Phi} A$ such that $A\tilde{R}^{-1}$ is well-conditioned (QR or Corollary 1).
3: Compute a $(1 \pm 1/2)$-distortion embedding $\tilde{S} \in \mathbb{R}^{\mathcal{O}(\text{poly}(n) \times m)}$ of $\mathcal{A}_p$ (Lemma 17).
4: Compute $R \in \mathbb{R}^{n \times n}$ such that $\kappa_p(\tilde{S}AR^{-1}) \leq 2n$ (Corollary 1).
5: Compute a $(1 \pm \epsilon)$-distortion embedding $S \in \mathbb{R}^{\mathcal{O}(n^{3+p/2} \log(1/\epsilon)/\epsilon^2) \times n}$ of $\mathcal{A}_p$ (Lemma 17).

**Theorem 16** (Improving the embedding dimension)**.** *Given $p \in [1, 2)$, with a constant probability, Algorithm 6 computes a $(1 \pm \epsilon)$-distortion embedding of $\mathcal{A}_p$ into $(\mathbb{R}^{\mathcal{O}(d^{3+p/2} \log(1/\epsilon)/\epsilon^2)}, \|\cdot\|_p)$ in $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time.*

*Proof.* Each of Steps 1, 3, and 5 of Algorithm 6 succeeds with a constant probability. We can control the success rate of each by adjusting the constant factor in the embedding dimension, such that all steps succeed with a constant probability. Conditioning on this event, we have $\kappa_p(AR^{-1}) = 6n$ because

$$\|AR^{-1}x\|_p \le 2\|\tilde{S}AR^{-1}x\|_p \le 4n\|x\|_2,$$
$$\|AR^{-1}x\|_p \ge \frac{2}{3}\|\tilde{S}AR^{-1}x\|_p \ge \frac{2}{3}\|x\|_2, \quad \forall x \in \mathbb{R}^n .$$

By Lemma 1, $\bar{\kappa}_p(AR^{-1}) \le 6n^{1/p+1}$, and then by Lemma 17, the embedding dimension of $S$ is $\mathcal{O}(\bar{\kappa}_p^p(AR^{-1})n^{|p/2-1|}n\log(1/\epsilon)/\epsilon^2) = \mathcal{O}(n^{3+p/2}\log(1/\epsilon)/\epsilon^2)$. □

Then, by applying Theorem 16 to the $\ell_p$ regression problem, we can improve the size of the subsampled problem and hence the overall running time.

**Corollary 4** (Improved fast $\ell_p$ regression)**.** *Given $p \in [1, 2)$, with a constant probability, a $(1+\epsilon)$-approximate solution to an $\ell_p$ regression problem can be computed in*

$$\mathcal{O}(\text{nnz}(A)\cdot\log m + \mathcal{T}_p(\epsilon; n^{3+p/2}\log(1/\epsilon)/\epsilon^2, n))$$

*time. The second term comes from solving a subsampled problem of size $\mathcal{O}(n^{3+p/2}\cdot\log(1/\epsilon)/\epsilon^2)\times n$.*

**Remark.** We have stated our results in the previous sections as $\text{poly}(n)$ without stating the value of the polynomial because there are numerous trade-offs between the conditioning quality and the running time. For example, let $p = 1$. We can use a rounding algorithm instead of QR to compute the $R$ matrix. If we use the input-sparsity time embedding with the $\mathcal{O}(n)$-rounding algorithm of [20], then the running time to compute the $(1\pm\epsilon)$-distortion embedding is $\mathcal{O}(\text{nnz}(A)\cdot\log m + n^8/\epsilon^2)$ and the embedding dimension is $\mathcal{O}(n^{6.5}/\epsilon^2)$ (ignoring log factors). If, on the other hand, we use QR to compute $R$, then the running time is $\mathcal{O}(\text{nnz}(A)\cdot\log m + n^7/\epsilon^2)$ and the embedding dimension is $\mathcal{O}(n^8/\epsilon^2)$. However, with the result from this section, the running time is simply $\mathcal{O}(\text{nnz}(A)\cdot\log m + \text{poly}(n) + \mathcal{T}_p(\epsilon; n^{3+p/2}/\epsilon^2, n))$ and the $\text{poly}(n)$ term can be absorbed by the $\text{nnz}(A)$ term.

# Bibliography


[1] D. Achlioptas. Database-friendly random projections. In *Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 274–281. ACM, 2001.

[2] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 557–563. ACM, 2006.

[3] N. Ailon and E. Liberty. Fast dimension reduction using Rademacher series on dual BCH codes. *Discrete & Computational Geometry*, 42(4):615–630, 2009.

[4] N. Ailon and E. Liberty. An almost optimal unrestricted fast Johnson-Lindenstrauss transform. In *Proceedings of the 22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 185–191, 2011.

[5] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, et al. *LAPACK Users' Guide*, volume 9. Society for Industrial Mathematics, 1987.

[6] H. Auerbach. *On the area of convex curves with conjugate diameters.* PhD thesis, University of Lwów, 1930.

[7] H. Avron, P. Maymounkov, and S. Toledo. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM J. Sci. Comput.*, 32(3):1217–1236, 2010.

[8] Gill Barequet and Sariel Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38(1):91–109, 2001.

[9] Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, 1996.

[10] J. Bourgain, J. Lindenstrauss, and V. Milman. Approximation of zonoids by zonotopes. *Acta Mathematica*, 162:73–141, 1989.

[11] Christian Bouville. Bounding ellipsoids for ray-fractal intersection. In *ACM SIGGRAPH Computer Graphics*, volume 19, pages 45–52. ACM, 1985.

[12] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[13] M. Bădoiu and K. L. Clarkson. Smaller core-sets for balls. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 801–802. SIAM, 2003.

[14] M. Bădoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 250–257. ACM, 2002.

[15] C. S. Burrus. Iterative reweighted least squares, 12 2012. http://cnx.org/content/m45285/1.12/.

[16] J. M. Chambers, C. L. Mallows, and B. W. Stuck. A method for simulating stable random variables. *Journal of the American Statistical Association*, 71(354):340–344, 1976.

[17] M. Charikar and A. Sahai. Dimension reduction in the $\ell_1$ norm. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 551–560, 2002.

[18] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*, pages 693–703. Springer, 2002.

[19] K. L. Clarkson. Subgradient and sampling algorithms for $\ell_1$ regression. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 257–266. SIAM, 2005.

[20] 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 *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013.

[21] K. L. Clarkson and D. P. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM symposium on Theory of Computing (STOC)*, 2013.

[22] E. S. Coakley, V. Rokhlin, and M. Tygert. A fast randomized algorithm for orthogonal projection. *SIAM J. Sci. Comput.*, 33(2):849–868, 2011.

[23] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1998.

[24] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney. Sampling algorithms and coresets for $\ell_p$ regression. *SIAM J. Comput.*, 38(5):2060–2078, 2009.

[25] K. R. Davidson and S. J. Szarek. Local operator theory, random matrices and Banach spaces. In *Handbook of the Geometry of Banach Spaces*, volume 1, pages 317–366. North Holland, 2001.

[26] T. A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, 2006.

[27] T. A. Davis. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Softw.*, 38(1), 2011.

[28] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Trans. Math. Softw.*, 38(1), 2011.

[29] P. Drineas, M. Magdon-Ismail, M. W. Mahoney, and D. P. Woodruff. Fast approximation of matrix coherence and statistical leverage. In *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[30] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Sampling algorithms for $\ell_2$ regression and applications. In *Proceedings of the 17th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1127–1136. ACM, 2006.

[31] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós. Faster least squares approximation. *Numer. Math.*, 117(2):219–249, 2011.

[32] D. C.-L. Fong and M. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.*, 33:2950, 2011.

[33] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins Univ Press, third edition, 1996.

[34] G. H. Golub and R. S. Varga. Chebyshev semi-iterative methods, successive over-relaxation methods, and second-order Richardson iterative methods, parts I and II. *Numer. Math.*, 3(1):147–168, 1961.

[35] M. Gu and S. C. Eisenstat. A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem. *SIAM J. Matrix Anal. Appl.*, 15(4):1266–1276, 1994.

[36] M. H. Gutknecht and S. Rollin. The Chebyshev iteration revisited. *Parallel Comput.*, 28(2):263–283, 2002.

[37] G. T. Herman, A. Lent, and H. Hurwitz. A storage-efficient algorithm for finding the regularized solution of a large, inconsistent system of equations. *IMA J. Appl. Math.*, 25(4):361–366, 1980.

[38] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Stand.*, 49(6):409–436, 1952.

[39] P. W. Holland and R. E. Welsch. Robust regression using iteratively reweighted least-squares. *Communications in Statistics - Theory and Methods*, 6(9):813–827, 1977.

[40] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–613. ACM, 1998.

[41] F. John. Extremum problems with inequalities as subsidiary conditions. In *Studies and Essays presented to R. Courant on his 60th Birthday*, pages 187–204, 1948.

[42] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.

[43] L. G. Khachiyan and M. J. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Math. Prog.*, 61(1):137–159, 1993.

[44] P. Lévy. *Calcul des Probabilités*. Gauthier-Villars, Paris, 1925.

[45] L. Lovász. *An Algorithmic Theory of Numbers, Graphs, and Convexity*. SIAM, 1986.

[46] D. G. Luenberger. *Introduction to Linear and Nonlinear Programming*. Addison-Wesley, 1973.

[47] M. W. Mahoney. *Randomized Algorithms for Matrices and Data*. Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011. Also available at arXiv:1104.5557v2.

[48] G. Marsaglia and W. W. Tsang. The ziggurat method for generating random variables. *J. Stat. Softw.*, 5(8):1–7, 2000.

[49] A. Maurer. A bound on the deviation probability for sums of non-negative random variables. *J. Inequalities in Pure and Applied Mathematics*, 4(1), 2003.

[50] 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):95–118, 2014.

[51] Xiangrui Meng and Michael W Mahoney. Robust regression on MapReduce. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, 2013.

[52] J. E. Mitchell. Polynomial interior point cutting plane methods. *Optimization Methods and Software*, 18(5):507–534, 2003.

[53] J. Nelson and H. Nguyen. OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 117–126. IEEE, 2013.

[54] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87. Springer, 2004.

[55] Y. Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 103(1):127–152, 2005.

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

[57] Y. Nesterov. Unconstrained convex minimization in relative scale. *Mathematics of Operations Research*, 34(1):180–193, 2009.

[58] Y. Nesterov and A. Nemirovsky. *Interior Point Polynomial Methods in Convex Programming*. SIAM, 1994.

[59] J. P. Nolan. *Stable Distributions - Models for Heavy Tailed Data.* Birkhauser, Boston, 2013. In progress, Chapter 1 online at academic2.american.edu/∼jpnolan.

[60] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM J. Numer. Anal.*, 12(4):617–629, 1975.

[61] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.*, 8(1):43–71, 1982.

[62] V. Rokhlin and M. Tygert. A fast randomized algorithm for overdetermined linear least-squares regression. *Proc. Natl. Acad. Sci. USA*, 105(36):13212–13217, 2008.

[63] T. Sarlós. Improved approximation algorithms for large matrices via random projections. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 143–152. IEEE, 2006.

[64] C. Sohler and D. P. Woodruff. Subspace embeddings for the $\ell_1$-norm with applications. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 755–764. ACM, 2011.

[65] M. J. Todd. On minimum volume ellipsoids containing part of a given ellipsoid. *Mathematics of Operations Research*, pages 253–261, 1982.

[66] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for non-parametric object and scene recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(11):1958–1970, 2008.

[67] J. A. Tropp. Improved analysis of the subsampled randomized Hadamard transform. *Adv. Adapt. Data Anal.*, 3(1-2):115–126, 2011.

[68] S. A. Vavasis and Y. Ye. Condition numbers for polyhedra with real number data. *Operations Research Letters*, 17(5):209–214, 1995.

[69] P. Wedin. Perturbation theory for pseudo-inverses. *BIT Numerical Mathematics*, 13(2):217–232, 1973.

[70] Geoffrey Werner-Allen, Jeff Johnson, Mario Ruiz, Jonathan Lees, and Matt Welsh. Monitoring volcanic eruptions with a wireless sensor network. In *Proceeedings of the 2nd European Workshop on Wireless Sensor Networks*, pages 108–120. IEEE, 2005.

[71] Yinyu Ye. *Interior Point Algorithms: Theory and Analysis*, volume 44. John Wiley & Sons, 2011.