

Implementing Randomized Matrix Algorithms in Parallel and Distributed Environments

In this paper, the authors review recent work on developing and implementing randomized matrix algorithms in large-scale parallel and distributed environments.

By JIYAN YANG, *Student Member IEEE*, XIANGRUI MENG, AND MICHAEL W. MAHONEY

ABSTRACT | In this era of large-scale data, distributed systems built on top of clusters of commodity hardware provide cheap and reliable storage and scalable processing of massive data. With cheap storage, instead of storing only currently relevant data, it is common to store as much data as possible, hoping that its value can be extracted later. In this way, exabytes (10^{18} bytes) of data are being created on a daily basis. Extracting value from these data, however, requires scalable implementations of advanced analytical algorithms beyond simple data processing, e.g., statistical regression methods, linear algebra, and optimization algorithms. Most such traditional methods are designed to minimize floating-point operations, which is the dominant cost of in-memory computation on a single machine. In parallel and distributed environments, however, load balancing and communication, including disk and network input/output (I/O), can easily dominate computation. These factors greatly increase the complexity of algorithm design and challenge traditional ways of thinking about the design of parallel and distributed algorithms. Here, we review recent work on developing and implementing randomized matrix algorithms in large-scale parallel and distributed environments. Randomized algorithms for matrix problems have received a great deal of attention in recent years, thus far typically either in theory or in machine learning applications or with implementations on a single machine.

Our main focus is on the underlying theory and practical implementation of random projection and random sampling algorithms for very large very overdetermined (i.e., over-constrained) ℓ_1 - and ℓ_2 -regression problems. Randomization can be used in one of two related ways: either to construct subsampled problems that can be solved, exactly or approximately, with traditional numerical methods; or to construct preconditioned versions of the original full problem that are easier to solve with traditional iterative algorithms. Theoretical results demonstrate that in near input-sparsity time and with only a few passes through the data one can obtain very strong relative-error approximate solutions, with high probability. Empirical results highlight the importance of various tradeoffs (e.g., between the time to construct an embedding and the conditioning quality of the embedding, between the relative importance of computation versus communication, etc.) and demonstrate that ℓ_1 - and ℓ_2 -regression problems can be solved to low, medium, or high precision in existing distributed systems on up to terabyte-sized data.

KEYWORDS | Big data; distributed matrix algorithms; least absolute deviation; least squares; preconditioning; randomized linear algebra; subspace embedding

I. INTRODUCTION

Matrix algorithms lie at the heart of many applications, both historically in areas such as signal processing and scientific computing as well as more recently in areas such as machine learning and data analysis. Essentially, the reason is that matrices provide a convenient mathematical structure with which to model data arising in a broad range of applications: an $m \times n$ real-valued matrix A provides a natural structure for encoding information

Manuscript received February 9, 2015; revised July 22, 2015; accepted August 11, 2015. Date of current version December 18, 2015. This work was supported in part by the U.S. Army Research Office, the Defense Advanced Research Projects Agency, and the U.S. Department of Energy.

J. Yang is with the Institute for Computational and Mathematical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: jiyang@stanford.edu).

X. Meng is with Databricks, San Francisco, CA 94105 USA (e-mail: meng@databricks.com).

M. W. Mahoney is with the International Computer Science Institute and Department of Statistics, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: mmahoney@stat.berkeley.edu).

Digital Object Identifier: 10.1109/JPROC.2015.2494219

about m objects, each of which is described by n features. Alternatively, an $n \times n$ real-valued matrix A can be used to describe the correlations between all pairs of n data points, or the weighted edge–edge adjacency matrix structure of an n -node graph. In astronomy, for example, very small angular regions of the sky imaged at a range of electromagnetic frequency bands can be represented as a matrix—in that case, an object is a region and the features are the elements of the frequency bands. Similarly, in genetics, DNA single nucleotide polymorphism (SNP) or DNA microarray expression data can be represented in such a framework, with A_{ij} representing the expression level of the i th gene or SNP in the j th experimental condition or individual. In another example, term-document matrices can be constructed in many Internet applications, with A_{ij} indicating the frequency of the j th term in the i th document.

Traditional algorithms for matrix problems are usually designed to run on a single machine, often focusing on minimizing the number of floating-point operations per second (FLOPS). On the other hand, motivated by the ability to generate very large quantities of data in relatively automated ways, analyzing data sets of billions or more of records has now become a regular task in many companies and institutions. In a distributed computational environment, which is typical in these applications, communication costs, e.g., between different machines, are often much more important than computational costs. What is more, if the data cannot fit into memory on a single machine, then one must scan the records from secondary storage, e.g., hard disk, which makes each pass through the data associated with enormous input/output (I/O) costs. Given that, in many of these large-scale applications, regression, low-rank approximation, and related matrix problems are ubiquitous, the fast computation of their solutions on large-scale data platforms is of interest.

In this paper, we will provide an overview of recent work in randomized numerical linear algebra (RandNLA) on implementing randomized matrix algorithms in large-scale parallel and distributed computational environments. RandNLA is a large area that applies randomization as an algorithmic resource to develop improved algorithms for regression, low-rank matrix approximation, and related problems [1]. To limit the presentation, here we will be most interested in very large, very rectangular linear regression problems on up to terabyte-sized data: in particular, in the ℓ_2 -regression [also known as least squares (LSs)] problem and its robust alternative, the ℓ_1 -regression [also known as least absolute deviations (LADs) or least absolute errors (LAEs)] problem, with strongly rectangular “tall” data. Although our main focus is on ℓ_2 - and ℓ_1 -regression, much of the underlying theory holds for ℓ_p -regression, either for $p \in [1, 2]$ or for all $p \in [1, \infty)$, and thus for simplicity we formulate many of our results in ℓ_p .

Several important conclusions will emerge from our presentation.

- First, many of the basic ideas from RandNLA in RAM extend to RandNLA in parallel/distributed environments in a relatively straightforward manner, assuming that one is more concerned about communication than computation. This is important from an algorithm design perspective, as it highlights which aspects of these RandNLA algorithms are peculiar to the use of randomization and which aspects are peculiar to parallel/distributed environments.
- Second, with appropriate engineering of random sampling and random projection algorithms, it is possible to compute good approximate solutions—to low precision (e.g., one or two digits of precision), medium precision (e.g., three or four digits of precision), or high precision (e.g., up to machine precision)—to several common matrix problems in only a few passes over the original matrix on up to terabyte-sized data. While low precision is certainly appropriate for many data analysis and machine learning applications involving noisy input data, the appropriate level of precision is a choice for user of an algorithm to make; and there are obvious advantages to having the developer of an algorithm provide control to the user on the quality of the answer returned by the algorithm.
- Third, the design principles for developing high-quality RandNLA matrix algorithms depend strongly on whether one is interested in low, medium, or high precision. (An example of this is whether to solve the randomized subproblem with a traditional method or to use the randomized subproblem to create a preconditioned version of the original problem.) Understanding these principles, the connections between them, and how they relate to traditional principles of NLA algorithm design is important for providing high-quality implementations of recent theoretical developments in the RandNLA literature.

Although many of the ideas we will discuss can be extended to related matrix problems such as low-rank matrix approximation, there are two main reasons for restricting attention to strongly rectangular data. The first, most obvious, reason is that strongly rectangular data arise in many fields to which machine learning and data analysis methods are routinely applied. Consider, e.g., Table 1, which lists a few examples.

- In genetics, SNPs are important in the study of human health. There are roughly ten million SNPs in the human genome. However, there are typically 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.

Table 1 Examples of Strongly Rectangular Data Sets

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

- In Internet applications, strongly rectangular data sets are common, for example, the image data set called TinyImages [2] which contains 80 million images of size 32×32 collected from the Internet.
- In spatial discretization of high-dimensional partial differential equations (PDEs), the number of degrees of freedom grows exponentially as dimension increases. For 3-D 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 1-D. Though depending on spatial discretization (e.g., the Courant–Friedrichs–Lewy condition for hyperbolic PDEs), the number of time steps is usually much lower than the number of degrees of freedoms in spatial discretization.
- In geophysical applications, especially in seismology, the number of sensors is much lower than the number of data points each sensor collects. For example, Werner-Allen *et al.* [3] deployed three wireless sensors to monitor volcanic eruptions. In 54 h, each sensor sent back approximately 20 million packets.
- In natural language processing (NLP), the number of documents is much lower than the number of n -grams, which grows geometrically as n increases. For example, the webspam¹ data set contains 350 000 documents and 254 unigrams, but 680 715 trigrams.
- In high-frequency trading, the number of relevant stocks is much lower than the number of ticks, changes to the best bid and ask. For example, in 2012 ISE Historical Options Tick Data² has daily files with average size greater than 100 GB uncompressed.

A second, less obvious, reason for restricting attention to strongly rectangular data is that many of the algorithmic methods that are developed for them (both the RandNLA methods we will review as well as deterministic NLA methods that have been used traditionally) have extensions to low-rank matrix approximation and to related problems on more general “fat” matrices. For example, many of the

methods for SVD-based low-rank approximation and related rank-revealing QR decompositions of general matrices have strong connections to QR decomposition methods for rectangular matrices; and, similarly, many of the methods for more general linear and convex programming arise in special (e.g., ℓ_1 -regression) linear programming problems. Thus, they are a good problem class to consider the development of matrix algorithms (either in general or for RandNLA algorithms) in parallel and distributed environments.

It is worth emphasizing that the phrase “parallel and distributed” can mean quite different things to different research communities, in particular to what might be termed high-performance computing (HPC) or scientific computing researchers versus data analytics or database or distributed data systems researchers. There are important technical and cultural differences here, but there are also some important similarities. For example, to achieve parallelism, one can use multithreading on a shared-memory machine, or one can use message passing on a multinode cluster. Alternatively, to process massive data on large commodity clusters, Google’s MapReduce [4] describes a computational framework for distributed computation with fault tolerance. For computation not requiring any internode communication, one can achieve even better parallelism. We do not want to dwell on many of these important details here: this is a complicated and evolving space; and no doubt the details of the implementation of many widely used algorithms will evolve as the space evolves. To give the interested reader a quick sense of some of these issues, though, here we provide a very high-level representative description of parallel environments and how they scale. See Table 2. As one goes down this list, one tends to get larger and larger.

In addition, it is also worth emphasizing that there is a great deal of related work in parallel and distributed computing, both in numerical linear algebra as well as more generally in scientific computing. For example, Valiant has provided a widely used model for parallel computation [5]; Aggarwal *et al.* have analyzed the communication complexity of parallel random-access machines (PRAMs) [6]; Lint and Agerwala have highlighted communication issues that arise in the design of parallel algorithms [7]; Heller has surveyed parallel algorithms in numerical linear algebra [8]; Toledo has provided a survey of out-of-core algorithms in numerical linear algebra [9]; Ballard *et al.* have focused

¹<http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html>

²<http://www.ise.com/hotdata>

Table 2 High-Level Representative Description of Parallel Environments

name	cores	memory	notes
Shared memory	[10, 10 ³] ³	[100GB, 100TB]	
Message passing	[200, 10 ⁵] ⁴	[1TB, 1000TB]	CUDA cores: [5 × 10 ⁴ , 3 × 10 ⁶] ⁵
MapReduce	[40, 10 ⁵] ⁶	[240GB, 100TB]	GPU memory: [500GB, 20TB]
Distributed computing	[−, 3 × 10 ⁵] ⁸		storage: [100TB, 100PB] ⁷

³<http://www.sgi.com/pdfs/4358.pdf>
⁴<http://www.top500.org/list/2011/11/100>
⁵<http://i.top500.org/site/50310>
⁶<http://www.cloudera.com/blog/2010/04/pushing-the-limits-of-distributed-processing/>
⁷<http://hortonworks.com/blog/an-introduction-to-hdfs-federation/>
⁸<http://folding.stanford.edu/>

on developing algorithms for minimizing communication in numerical linear algebra [10]; and Bertsekas and Tsitsiklis have surveyed parallel and distributed iterative algorithms [11]. We expect that some of the most interesting developments in upcoming years will involve coupling the ideas for implementing RandNLA algorithms in parallel and distributed environments that we describe in this review with these more traditional ideas for performing parallel and distributed computation.

In Section II, we will review the basic ideas underlying RandNLA methods, as they have been developed in the special case of ℓ_2 -regression in the RAM model. Then, in Section III, we will provide notation, some background, and preliminaries on ℓ_2 , and more general ℓ_p -regression problems, as well as traditional methods for their solution. Then, in Section IV, we will describe rounding and embedding methods that are used in a critical manner by RandNLA algorithms; and in Section V, we will review recent empirical results on implementing these ideas to solve up to terabyte-sized ℓ_2 - and ℓ_1 -regression problems. Finally, in Section VI, we will provide a brief discussion and conclusion. An overview of the general RandNLA area has been provided [1], and we refer the interested reader to this overview. In addition, two other reviews are available to the interested reader: an overview of how RandNLA methods can be coupled with traditional NLA algorithms for low-rank matrix approximation [12]; and an overview of how data-oblivious subspace embedding methods are used in RandNLA [13].

II. RandNLA IN RAM

In this section, we will highlight several core ideas that have been central to prior work in RandNLA in (theory and/or practice in) RAM that we will see are also important as design principles for extending RandNLA methods to larger scale parallel and distributed environments. We start in Section II-A by describing a prototypical example of a RandNLA algorithm for the very overdetermined LS problem; then, we describe in Section II-B two problem-specific complexity measures that are important

for low-precision and high-precision solutions to matrix problems, respectively, as well as two complementary ways in which randomization can be used by RandNLA algorithms; and we conclude in Section II-C with a brief discussion of running time considerations.

A. A Meta-Algorithm for RandNLA

A prototypical example of the RandNLA approach is given by the following meta-algorithm for very overdetermined LS problems [1], [14]–[16]. In particular, the problem of interest is to solve

$$\min_x \|Ax - b\|_2. \quad (1)$$

The following meta-algorithm takes as input an $m \times n$ matrix A , where $m \gg n$, a vector b , and a probability distribution $\{\pi_i\}_{i=1}^m$, and it returns as output an approximate solution \hat{x} , which is an estimate of the exact answer x^* of (1).

- Randomly sampling. Randomly sample $r > n$ constraints, i.e., rows of A and the corresponding elements of b , using $\{\pi_i\}_{i=1}^m$ as an importance sampling distribution.
- Subproblem construction. Rescale each sampled row/element by $1/(r\pi_i)$ to form a weighted LS subproblem.
- Solving the subproblem. Solve the weighted LS subproblem, formally given in (2), and then return the solution \hat{x} .

It is convenient to describe this meta-algorithm in terms of a random “sampling matrix” S , in the following manner. If we draw r samples (rows or constraints or data points) with replacement, then define an $r \times m$ sampling matrix S , where each of the r rows of S has one nonzero element indicating which row of A (and element of b) is chosen in a given random trial. In this case, the (i, k) th element of S equals $1/\sqrt{r\pi_k}$ if the k th data point is chosen in the i th random trial (meaning, in particular, that every nonzero element of S equals $\sqrt{n/r}$ for sampling uniformly at

random). With this notation, this meta-algorithm constructs and solves the weighted LS estimator

$$\hat{x} = \arg \min_x \|SAx - Sb\|_2. \quad (2)$$

Since this meta-algorithm samples constraints and not variables, the dimensionality of the vector \hat{x} that solves the (still overconstrained, but smaller) weighted LS subproblem is the same as that of the vector x^* that solves the original LS problem. The former may thus be taken as an approximation of the latter, where, of course, the quality of the approximation depends critically on the choice of $\{\pi_i\}_{i=1}^n$. Although uniform subsampling (with or without replacement) is very simple to implement, it is easy to construct examples where it will perform very poorly [1], [14], [16]. On the other hand, it has been shown that, for a parameter $\gamma \in (0, 1]$ that can be tuned, if

$$\pi_i \geq \gamma \frac{h_{ii}}{n} \quad \text{and} \quad r = \mathcal{O}\left(\frac{n \log(n)}{\gamma \epsilon^2}\right) \quad (3)$$

where the so-called statistical leverage scores h_{ii} are defined in (6), i.e., if one draws the sample according to an importance sampling distribution that is proportional to the leverage scores of A , then with constant probability (that can be easily boosted to probability $1 - \delta$, for any $\delta > 0$) the following relative-error bounds hold:

$$\|b - A\hat{x}\|_2 \leq (1 + \epsilon) \|b - Ax^*\|_2 \quad (4)$$

$$\|x^* - \hat{x}\|_2 \leq \sqrt{\epsilon} \left(\kappa(A) \sqrt{\xi^{-2} - 1} \right) \|x^*\|_2 \quad (5)$$

where $\kappa(A)$ is the condition number of A and where $\xi = \|UU^T b\|_2 / \|b\|_2$ is a parameter defining the amount of the mass of b inside the column space of A [1], [14], [15].

Due to the crucial role of the statistical leverage scores in (3), this canonical RandNLA procedure has been referred to as the algorithmic leveraging approach to approximating LS approximation [16]. In addition, although this meta-algorithm has been described here only for very overdetermined LS problems, it generalizes to other linear regression problems and low-rank matrix approximation problems on less rectangular matrices⁹ [17]–[21].

B. Leveraging, Conditioning, and Using Randomization

Leveraging and conditioning refer to two types of problem-specific complexity measures, i.e., quantities that

⁹Let A be a matrix with dimension m by n where $m > n$. A less rectangular matrix is a matrix that has smaller m/n .

can be computed for any problem instance that characterize how difficult that problem instance is for a particular class of algorithms. Understanding these, as well as different uses of randomization in algorithm design, is important for designing RandNLA algorithms, both in theory and/or practice in RAM as well as in larger parallel and distributed environments. For now, we describe these in the context of very overdetermined LS problems.

- Statistical leverage. (Related to eigenvectors; important for obtaining low-precision solutions.) If we let $H = A(A^T A)^{-1} A^T$, where the inverse can be replaced with the Moore–Penrose pseudoinverse if A is rank deficient, be the projection matrix onto the column span of A , then the i th diagonal element of H

$$h_{ii} = A_{(i)} (A^T A)^{-1} A_{(i)}^T \quad (6)$$

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

$$h_{ii} = \sum_{j=1}^n U_{ij}^2 = \|U_{(i)}\|^2 \quad (7)$$

where $U_{(i)}$ is the i th row of U . Leverage scores provide a notion of “coherence” or “outlierness,” in that they measure how well correlated the singular vectors are with the canonical basis [15], [18], [22] as well as which rows/constraints have largest “influence” on the LS fit [23]–[26]. Computing the leverage scores $\{h_{ii}\}_{i=1}^m$ exactly is generally as hard as solving the original LS problem (but $1 \pm \epsilon$ approximations to them can be computed more quickly, for arbitrary input matrices [15]). Leverage scores are important from an algorithm design perspective since they define the key nonuniformity structure needed to control the complexity of high-quality random sampling algorithms. In particular, naïve uniform random sampling algorithms perform poorly when the leverage scores are very nonuniform, while randomly sampling in a manner that depends on the leverage scores leads to high-quality solutions. Thus, in designing RandNLA algorithms, whether in RAM or in parallel-distributed environments, one must either quickly compute approximations

to the leverage scores or quickly preprocess the input matrix so they are nearly uniformized—in which case uniform random sampling on the preprocessed matrix performs well.¹⁰

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

- **Condition number.** (Related to eigenvalues; important for obtaining high-precision solutions.) If we let $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ denote the largest and smallest nonzero singular values of A , respectively, then $\kappa(A) = \sigma_{\max}(A)/\sigma_{\min}^+(A)$ is the ℓ_2 -norm condition number of A which is formally defined in Definition 3. (Here, $\sigma_{\min}^+(A)$ is the smallest nonzero singular value of A .) Computing $\kappa(A)$ exactly is generally as hard as solving the original LS problem. The condition number $\kappa(A)$ is important from an algorithm design perspective since $\kappa(A)$ defines the key nonuniformity structure needed to control the complexity of high-precision iterative algorithms, i.e., it bounds the number of iterations needed for iterative methods to converge. In particular, for ill-conditioned problems, e.g., if $\kappa(A) \approx 10^6 \gg 1$, then the convergence speed of iterative methods is very slow, while if $\kappa \gtrsim 1$ then iterative algorithms converge very quickly. Informally, $\kappa(A)$ defines the aspect ratio of the quadratic well implicitly defined by A in the high-dimensional Euclidean space. If one is interested in obtaining high-precision solutions, e.g., $\epsilon = 10^{-10}$, that can be obtained by iterating a low-precision solution to high precision with an iterative algorithm that converges as $\log(1/\epsilon)$, then the key quantity that must be dealt with is the condition number of the input data.
- **Monte Carlo versus Las Vegas** uses of randomization. Note that the guarantee provided by the meta-algorithm, as stated above, is of the following form: the algorithm runs in no more than a specified time T , and with probability at least $1 - \delta$ it returns a

solution that is an ϵ -good approximation to the exact solution. Randomized algorithms that provide guarantees of this form, i.e., with running time that is deterministic, but whose output may be incorrect with a certain small probability, are known as Monte Carlo algorithms [27]. A related class of randomized algorithms, known as Las Vegas algorithms, provide a different type of guarantee: they always produce the correct answer, but the amount of time they take varies randomly [27]. In many applications of RandNLA algorithms, guarantees of this latter form are preferable.

The notions of condition number and leverage scores have been described here only for very overdetermined ℓ_2 -regression problems. However, as discussed in Section III (as well as previously [17], [19]), these notions generalize to very overdetermined ℓ_p , for $p \neq 2$, regression problems [19] as well as to $p = 2$ for less rectangular matrices, as long as one specifies a rank parameter k [17]. Understanding these generalizations, as well as the associated tradeoffs, will be important for developing RandNLA algorithms in parallel and distributed environments.

C. Running Time Considerations in RAM

As presented, the meta-algorithm of Section II-B has a running time that depends on both the time to construct the probability distribution, $\{\pi_i\}_{i=1}^n$, and the time to solve the subsampled problem. For uniform sampling, the former is trivial and the latter depends on the size of the subproblem. For estimators that depend on the exact or approximate [recall the flexibility in (3) provided by γ] leverage scores, the running time is dominated by the exact or approximate computation of those scores. A naïve algorithm involves using a QR decomposition or the thin SVD of A to obtain the exact leverage scores. This naïve implementation of the meta-algorithm takes roughly $\mathcal{O}(mn^2/\epsilon)$ time and is thus no faster (in the RAM model) than solving the original LS problem exactly [14], [17]. There are two other potential problems with practical implementations of the meta-algorithm: the running time dependence of roughly $\mathcal{O}(mn^2/\epsilon)$ time scales polynomially with $1/\epsilon$, which is prohibitive if one is interested in moderately small (e.g., 10^{-4}) to very small (e.g., 10^{-10}) values of ϵ ; and, since this is a randomized Monte Carlo algorithm, with some probability δ the algorithm might completely fail.

Importantly, all three of these potential problems can be solved to yield improved variants of the meta-algorithm.

- **Making the algorithm fast:** improving the dependence on m and n . We can make this meta-algorithm “fast” in worst case theory in RAM [14], [15], [20], [28], [29]. In particular, this meta-algorithm runs in $\mathcal{O}(mn \log n/\epsilon)$ time in RAM if one does either of the following: if one performs a Hadamard-based random projection and then performs uniform sampling in the randomly

¹⁰As stated, this is just an observation about how to approach RandNLA algorithm design. As a forward reference, however, we note that any random projection algorithm (whether Gaussian based or Hadamard based or input sparsity based or via some other construction) works (essentially) since it does the latter option, and any random sampling algorithm (that leads to high-quality, e.g., relative-error and not additive-error, bounds) works (essentially) since it does the former option.

rotated basis [28], [29] (which, recall, is basically what random projection algorithms do when applied to vectors in a Euclidean space [1]); or if one quickly computes approximations to the statistical leverage scores (using the algorithm of [15], the running time bottleneck of which is applying a random projection to the input data) and then uses those approximate scores as an importance sampling distribution [14], [15]. In addition, by using carefully constructed extremely sparse random projections, both of these two approaches can be made to run in so-called “input sparsity time,” i.e., in time proportional to the number of nonzeros in the input data, plus lower order terms that depend on the lower dimension of the input matrix [20].

- Making the algorithm high precision: improving the dependence on ϵ . We can make this meta-algorithm “fast” in practice, e.g., in “high-precision” numerical implementation in RAM [30]–[33]. In particular, this meta-algorithm runs in $\mathcal{O}(mn \log n \log(1/\epsilon))$ time in RAM if one uses the subsampled problem constructed by the random projection/sampling process to construct a preconditioner, using it as a preconditioner for a traditional iterative algorithm on the original full problem [30]–[32]. This is important since, although the worst case theory holds for any fixed ϵ , it is quite coarse in the sense that the sampling complexity depends on ϵ as $1/\epsilon$ and not $\log(1/\epsilon)$. In particular, this means that obtaining high-precision with (say) $\epsilon = 10^{-10}$ is not practically possible. In this iterative use case, there are several tradeoffs: e.g., one could construct a very high-quality preconditioner (e.g., using a number of samples that would yield a $1 + \epsilon$ error approximation if one solved the LS problem on the subproblem) and perform fewer iterations, or one could construct a lower quality preconditioner by drawing many fewer samples and perform a few extra iterations. Here too, the input sparsity time algorithm of [20] could be used to improve the running time still further.
- Dealing with the δ failure probability. Although fixing a failure probability δ is convenient for theoretical analysis, in certain applications having even a very small probability that the algorithm might return a completely meaningless answer is undesirable. In this case, one is interested in converting a Monte Carlo algorithm into a Las Vegas algorithm. Fortunately, those application areas, e.g., scientific computing, are often more interested in moderate- to high-precision solutions than in low-precision solutions. In these cases, using the subsampled problem to create a preconditioner for iterative algorithms on the original problem has the side effect that one changes a “fixed running time but might fail” algorithm to an “expected running time but will never fail” algorithm.

From above, we can make the following conclusions. The “fast in worst case theory” variants of the meta-algorithm [14], [15], [20], [28], [29] represent qualitative improvements to the $\mathcal{O}(mn^2)$ worst case asymptotic running time of traditional algorithms for the LS problem going back to Gaussian elimination. The “fast in numerical implementation” variants of the meta-algorithm [30]–[32] have been shown to beat LAPACK’s direct dense LS solver by a large margin on essentially any dense tall matrix, illustrating that the worst case asymptotic theory holds for matrices as small as several thousand by several hundred [31].

While these results are a remarkable success for RandNLA in RAM, they leave open the question of how these RandNLA methods perform in larger scale parallel/distributed environments, and they raise the question of whether the same RandNLA principles can be extended to other common regression problems. In the remainder of this paper, we will review recent work showing that if one wants to solve ℓ_2 -regression problems in parallel/distributed environments, and if one wants to solve ℓ_1 -regression problems in theory or in RAM or in parallel/distributed environments, then one can use the same RandNLA meta-algorithm and design principles. Importantly, though, depending on the exact situation, one must instantiate the same algorithmic principles in different ways, e.g., one must worry much more about communication rather than FLOPS.

III. PRELIMINARIES ON ℓ_p -REGRESSION PROBLEMS

In this section, we will start in Section III-A with a brief review of notation that we will use in the remainder of the paper. Then, in Section III-B–D, we will review ℓ_p -regression problems and the notions of condition number and preconditioning for these problems.

A. 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 ℓ_p -norm of a vector, $\|\cdot\|_2$ the spectral-norm of a matrix, $\|\cdot\|_F$ the Frobenius-norm of a matrix, and $|\cdot|_p$ the element-wise ℓ_p -norm of a matrix.
- We use 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 “ \sim ” accent is used for sketches of matrices, e.g., \tilde{A} , the “*” superscript is used for indicating optimal solutions, e.g., x^* , and the “ \wedge ” accent is used for estimates of solutions, e.g., \hat{x} .

B. ℓ_p -Regression Problems

In this work, a parameterized family of linear regression problems that is of particular interest is the ℓ_p -regression problem.

Definition 1 (ℓ_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 ℓ_p -regression problem specified by A , b , and p is the following optimization problem:

$$\text{minimize}_{x \in \mathbb{R}^n} \|Ax - b\|_p \quad (8)$$

where the ℓ_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 overdetermined if $m \gg n$, and strongly underdetermined if $m \ll n$.

Important special cases include the ℓ_2 -regression problem, also known as linear LS, and the ℓ_1 -regression problem, also known as LADs or LAEs. The former is ubiquitous; and the latter is of particular interest as a robust regression technique, in that it is less sensitive to the presence of outliers than the former.

For general $p \in [1, \infty]$, denote \mathcal{X}^* the set of optimal solutions to (8). Let $x^* \in \mathcal{X}^*$ be an arbitrary optimal solution, and let $f^* = \|Ax^* - b\|_p$ be the optimal objective value. We will be particularly interested in finding a relative-error approximation, in terms of the objective value, to the general ℓ_p -regression problem (8).

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 ℓ_p -regression problem (8) if and only if

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

In order to make our theory and our algorithms for general ℓ_p -regression simpler and more concise, we can use an equivalent formulation of (8) in our discussion

$$\begin{aligned} & \text{minimize}_{x \in \mathbb{R}^n} \|Ax\|_p \\ & \text{subject to} \quad c^T x = 1. \end{aligned} \quad (9)$$

Above, the “new” A is A concatenated with $-b$, i.e., $(A \ -b)$ and c is a vector with a 1 at the last coordinate and zeros elsewhere, i.e., $c \in \mathbb{R}^{n+1}$ and $c = (0 \dots 01)$, to force the last element of any feasible solution to be 1. We note that the same formulation is also used by [34] for solving unconstrained convex problems in relative scale. This formulation of ℓ_p -regression, which consists of a homogeneous objective and an affine constraint, can be shown to be equivalent to the formulation of (8).

Consider, next, the special case $p = 2$. If, in the LS problem

$$\text{minimize}_{x \in \mathbb{R}^n} \|Ax - b\|_2 \quad (10)$$

we let $r = \text{rank}(A) \leq \min(m, n)$, then recall that if $r < n$ (the LS problem is underdetermined or rank deficient), then (10) 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 (10) and hence it must have the minimum length. In either case, we denote this unique min-length solution to (10) by x^* , and we are interested in computing x^* in this work. This was defined in (1). In this case, we will also be interested in bounding $\|x^* - \hat{x}\|_2$, for arbitrary or worst case input, where \hat{x} was defined in (2) and is an approximation to x^* .

C. ℓ_p -Norm Condition Number

An important concept in ℓ_2 and more general ℓ_p -regression problems, and in developing efficient algorithms for their solution, is the concept of condition number. For linear systems and LS problems, the ℓ_2 -norm condition number is already a well-established term.

Definition 3 (ℓ_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 ℓ_2 -norm condition number of A is defined as $\kappa_2(A) = \sigma_2^{\max}(A)/\sigma_2^{\min}(A)$. For simplicity, we use κ_2 , σ_2^{\min} , and σ_2^{\max} when the underlying matrix is clear from context.

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

Definition 4 (ℓ_p -Norm Condition Number [19]): Given a matrix $A \in \mathbb{R}^{m \times n}$ and $p \in [1, \infty]$, let

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

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

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

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

Definition 5 ((α, β, p)-Conditioning [35]): 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 (α, β, p) -conditioned if: 1) $|A|_p \leq \alpha$; and 2) for all $z \in \mathbb{R}^n$, $\|z\|_q \leq \beta \|Az\|_p$. Define $\bar{\kappa}_p(A)$, the (α, β, p) -condition number of A , as the minimum value of $\alpha\beta$ such that A is (α, β, p) -conditioned. We use $\bar{\kappa}_p$ for simplicity if the underlying matrix is clear.

Lemma 1 (Equivalence of κ_p and $\bar{\kappa}_p$ [19]): Given a matrix $A \in \mathbb{R}^{m \times n}$ and $p \in [1, \infty]$, we always have

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

That is, by Lemma 1, if $m \gg n$, then the notions of condition number provided by Definitions 4 and 5 are equivalent, up to low-dimensional factors. These low-dimensional factors typically do not matter in theoretical formulations of the problem, but they can matter in practical implementations.

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

D. Preconditioning ℓ_p -Regression Problems

Preconditioning refers to the application of a transformation, called the preconditioner, to a given problem instance such that the transformed instance is more easily solved by a given class of algorithms. Most commonly, the preconditioned problem is solved with an iterative algorithm, the complexity of which depends on the condition number of the preconditioned problem.

To start, consider $p = 2$, and recall that for a square linear system $Ax = b$ of full rank, this preconditioning usually takes one of the following forms:

$$\begin{aligned} &\text{left preconditioning } M^T Ax = M^T b \\ &\text{right preconditioning } ANy = b, \quad x = Ny \\ &\text{left and right preconditioning } M^T ANy = M^T b, \quad x = Ny. \end{aligned}$$

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), more care should be taken so that the preconditioned system has the same min-length solution as the original one. In particular, if we apply left preconditioning to the LS problem \min_x

$\|Ax - b\|_2$, then 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 2 (Left and Right Preconditioning [32]): 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 $\text{range}(NN^T A^T) = \text{range}(A^T)$;
- 2) $A^\dagger = (M^T A)^\dagger M^T$ if and only if $\text{range}(MM^T A) = \text{range}(A)$.

Just as with $p = 2$, for more general ℓ_p -regression problems with matrix $A \in \mathbb{R}^{m \times n}$ with full column rank, although 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. (This is not the R from a QR decomposition of A , unless $p = 2$, but some other matrix R .) In this case, the ℓ_p -regression problem (9) is equivalent to the following well-conditioned problem:

$$\begin{aligned} &\text{minimize}_{y \in \mathbb{R}^n} \quad \|AR^{-1}y\|_p \\ &\text{subject to} \quad c^T R^{-1}y = 1. \end{aligned} \quad (11)$$

Clearly, if y^* is an optimal solution to (11), then $x^* = R^{-1}y^*$ is an optimal solution to (9), and *vice versa*; however, (11) may be easier to solve than (9) because of better conditioning.

Since we want to reduce the condition number of a problem instance via preconditioning, it is natural to ask what the best possible outcome would be in theory. For $p = 2$, an orthogonal matrix, e.g., the matrix Q computed from a QR decomposition, has $\kappa_2(Q) = 1$. More generally, for the ℓ_p -norm condition number κ_p , we have the following existence result.

Lemma 3: Given a matrix $A \in \mathbb{R}^{m \times n}$ with full column rank and $p \in [1, \infty]$, there exists a matrix $R \in \mathbb{R}^{n \times n}$ such that $\kappa_p(AR^{-1}) \leq n^{1/2}$.

This is a direct consequence of John's theorem [36] on ellipsoidal rounding of centrally symmetric convex sets.

For the (α, β, p) -condition number $\bar{\kappa}_p$, we have the following lemma.

Lemma 4: Given a matrix $A \in \mathbb{R}^{m \times n}$ with full column rank and $p \in [1, \infty]$, there exists a matrix $R \in \mathbb{R}^{n \times n}$ such that $\bar{\kappa}_p(AR^{-1}) \leq n$.

Note that Lemmas 3 and 4 are both existential results. Unfortunately, except the case when $p = 2$, no polynomial-time algorithm is known that can provide such preconditioning for general matrices. In Section IV, we will discuss two practical approaches for ℓ_p -norm preconditioning: via ellipsoidal rounding and via subspace embedding, as well as subspace-preserving sampling algorithms built on top of them.

IV. ROUNDING, EMBEDDING, AND SAMPLING ℓ_p -REGRESSION PROBLEMS

Preconditioning, ellipsoidal rounding, and low-distortion subspace embedding are three core technical tools underlying RandNLA regression algorithms. In this section, we will describe in detail how these methods are used for ℓ_p -regression problems, with an emphasis on tradeoffs that arise when applying these methods in parallel and distributed environments. Recall that, for any matrix $A \in \mathbb{R}^{m \times n}$ with full column rank, Lemmas 3 and 4 show that there always exists a preconditioner matrix $R \in \mathbb{R}^{n \times n}$ such that AR^{-1} is well conditioned, for ℓ_p -regression, for general $p \in [1, \infty]$. For $p = 2$, such a matrix R can be computed in $O(mn^2)$ time as the “R” matrix from a QR decomposition, although it is of interest to compute other such preconditioner matrices R that are nearly as good more quickly; and for $p = 1$ and other values of p , it is of interest to compute a preconditioner matrix R in time that is linear in m and low-degree polynomial in n . In this section, we will discuss these and related issues.

In particular, in Section IV-A and B, we discuss practical algorithms to find such R matrices, and we describe the tradeoffs between speed (e.g., FLOPS, number of passes, additional space/time, other communication costs, etc.) and conditioning quality. The algorithms fall into two general families: ellipsoidal rounding (Section IV-A) and subspace embedding (Section IV-B). We present them roughly in the order of speed (in the RAM model), from slower ones to faster ones. We will discuss practical tradeoffs in Section V. For simplicity, here we assume $m \gg \text{poly}(n)$, and hence $mn^2 \gg mn + \text{poly}(n)$; and if A is sparse, we assume that $mn \gg \text{nnz}(A)$. Hereby, the degree of $\text{poly}(n)$ depends on the underlying algorithm, which may range from $\mathcal{O}(n)$ to $\mathcal{O}(n^7)$.

Before diving into the details, it is worth mentioning a few high-level considerations about subspace embedding methods. (Similar considerations apply to ellipsoidal rounding methods.) Subspace embedding algorithms involve mapping data points, e.g., the columns of an $m \times n$ matrix, where $m \gg n$ to a lower dimensional space

such that some property of the data, e.g., geometric properties of the point set, is approximately preserved; see Definition 7 for definition for low-distortion subspace embedding matrix. As such, they are critical building blocks for developing improved random sampling and random projection algorithms for common linear algebra problems more generally, and they are one of the main technical tools for RandNLA algorithms. There are several properties of subspace embedding algorithms that are important in order to optimize their performance in theory and/or in practice. For example, given a subspace embedding algorithm, we may want to know:

- whether it is data oblivious (i.e., independent of the input subspace) or data aware (i.e., dependent on some property of the input matrix or input space);
- the time and storage it needs to construct an embedding;
- the time and storage to apply the embedding to an input matrix;
- the failure rate, if the construction of the embedding is randomized;
- the dimension of the embedding, i.e., the number of dimensions being sampled by sampling algorithms or being projected onto by projection algorithms,
- the distortion of the embedding;
- how to balance the tradeoffs among those properties.

Some of these considerations may not be important for typical theoretical analysis but still affect the practical performance of implementations of these algorithms.

After the discussion of rounding and embedding methods, we will then show in Section IV-C that ellipsoidal rounding and subspace embedding methods (that show that the ℓ_p -norms of the entire subspace of vectors can be well preserved) can be used in one of two complementary ways: one can solve an ℓ_p -regression problem on the rounded/embedded subproblem; or one can use the rounding/embedding to construct a preconditioner for the original problem. [We loosely refer to these two complementary types of approaches as low-precision methods and high-precision methods, respectively. The reason is that the running time complexity with respect to the error parameter ϵ for the former is $\text{poly}(1/\epsilon)$, while the running time complexity with respect to ϵ for the latter is $\log(1/\epsilon)$.] We also discuss various ways to combine these two types of approaches to improve their performance in practice.

Since we will introduce several important and distinct but closely related concepts in this long section, in Fig. 1 we provide an overview of these relations as well as of the structure of this section.

A. Ellipsoidal Rounding and Fast Ellipsoid Rounding

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

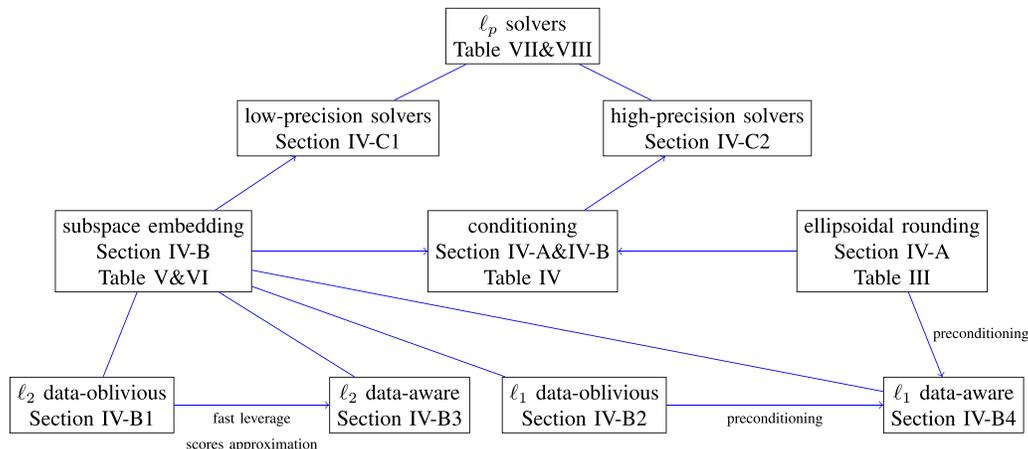


Fig. 1. Overview of relationships between several core technical components in RandNLA algorithms for solving ℓ_p -regression. Relevant subsection and tables in this section are also shown. A directed edge implies the tail component contributes to the head component.

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 κ -rounding of \mathcal{C} if it satisfies $\mathcal{E}/\kappa \subseteq \mathcal{C} \subseteq \mathcal{E}$, for some $\kappa \geq 1$, where \mathcal{E}/κ means shrinking \mathcal{E} by a factor of $1/\kappa$.

Finding an ellipsoidal rounding with a small κ factor for a given convex set has many applications such as in computational geometry [37], convex optimization [38], and computer graphics [39]. In addition, the ℓ_p -norm condition number κ_p naturally connects to ellipsoidal rounding. To see this, let $\mathcal{C} = \{x \in \mathbb{R}^n \mid \|Ax\|_p \leq 1\}$ and assume that we have a κ -rounding of \mathcal{C} : $\mathcal{E} = \{x \mid \|Rx\|_2 \leq 1\}$. This implies

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

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

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

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

Recall the well-known result due to John [36] that for a centrally symmetric convex set \mathcal{C} there exists a

$n^{1/2}$ -rounding. It is known that this result is sharp and that such rounding is given by the Löwner–John (LJ) ellipsoid of \mathcal{C} , i.e., the minimal-volume ellipsoid containing \mathcal{C} . This leads to Lemma 3. 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 [38].

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 an $(n(n+1))^{1/2}$ -rounding in polynomial time, in particular, in $\mathcal{O}(n^4 \log L)$ calls to the oracle; see [38, Th. 2.4.1]. (Polynomial time algorithms with better κ have been proposed for special convex sets, e.g., the convex hull of a finite point set [40] and the convex set specified by the matrix ℓ_∞ -norm [41].) This algorithmic result was used by Clarkson [42] and then by Dasgupta et al. [35] for ℓ_p -regression. Note that, in these works, only $\mathcal{O}(n)$ -rounding is actually needed, instead of $(n(n+1))^{1/2}$ -rounding.

Recent work has focused on constructing ellipsoid rounding (ER) methods that are much faster than these more classical techniques but that lead to only slight degradation in preconditioning quality. See Table 3 for a summary of these results. In particular, Clarkson et al. [19] follow the same construction as in the proof of Lovász [38] but 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.

Table 3 Summary of Several Ellipsoidal Rounding for ℓ_p Conditioning. The * Superscript Denotes That the Oracles Are Described and Called Through a Smaller Matrix With Size m/n by n

	κ	time	# passes	# calls to oracle
ER [35], [42]	$(n(n+1))^{1/2}$	$\mathcal{O}(mn^5 \log m)$	$n^3 \log m$	$\mathcal{O}(n^4 \log m)$
Fast ER [19]	$2n$	$\mathcal{O}(mn^3 \log m)$	$n \log m$	$\mathcal{O}(n^2 \log m)$
Single-pass ER [43]	$2n^{ 2/p-1 +1}$	$\mathcal{O}(mn^2 \log m)$	1	$\mathcal{O}(n^2 \log m)^*$

Lemma 5 (Fast Ellipsoidal Rounding [19]): 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 $\mathcal{O}(n^4 \log L)$ time to find a $2n$ -rounding of \mathcal{C} .

By applying Lemma 5 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 , one immediately improves the running time of the algorithm used by Clarkson [42] and by Dasgupta et al. [35] 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 $\kappa_p(AR^{-1}) \leq 2n$.

Unfortunately, even this improvement for computing a $2n$ -conditioning is not immediately applicable to very large matrices. The reason is that such matrices are usually distributively stored on secondary storage and each call to the oracle requires a pass through the data. We could group n calls together within a single pass, but this would still need $\mathcal{O}(n \log m)$ passes. Instead, Meng and Mahoney [43] present a deterministic single-pass conditioning algorithm that balances the cost-performance tradeoff to provide a $2n^{2/p-1+1}$ -conditioning of A [43]. This algorithm essentially invoke the fast ellipsoidal rounding (Lemma 5) method on a smaller problem which is constructed via a single pass on the original data set. Their main algorithm is stated in Algorithm 1, and the main result for Algorithm 1 is the following.

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 nonsingular matrix $E \in \mathbb{R}^{n \times n}$ such that

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

1: Partition A along its rows into submatrices of size $n^2 \times n$, denoted by A_1, \dots, 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 $i = 1, \dots, M$

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

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 $n^{\min\{1/p-1/2, 0\}} E$.

Lemma 6 (One-Pass Conditioning [43]): Algorithm 1 is a $2n^{2/p-1+1}$ -conditioning algorithm, and it runs in $\mathcal{O}((mn^2 + n^4) \log m)$ time. It needs to compute a $2n$ -rounding on a problem with size m/n by n which needs $\mathcal{O}(n^2 \log m)$ calls to the separation oracle on the smaller problem.

Remark 1: Solving the rounding problem of size $m/n \times n$ in Algorithm 1 requires $\mathcal{O}(m)$ RAM, which might be too much for very large-scale problems. In such cases, one can increase the block size from n^2 to, e.g., n^3 . A modification to the proof of Lemma 6 shows that this gives us a $2n^{\lfloor 3/p-3/2 \rfloor + 1}$ -conditioning algorithm that only needs $\mathcal{O}(m/n)$ RAM and $\mathcal{O}((mn + n^4) \log m)$ FLOPS for the rounding problem.

Remark 2: One can replace SVD computation in Algorithm 1 by a fast randomized ℓ_2 subspace embedding (i.e., a fast low-rank approximation algorithm as described in [1] and [12] and that we describe below). This reduces the overall running time to $\mathcal{O}((mn + n^4) \log(mn))$, and this is an improvement in terms of FLOPS; but this would lead to a nondeterministic result with additional variability due to the randomization (that in our experience substantially degrades the embedding/conditioning quality in practice). How to balance those tradeoffs in real applications and implementations depends on the underlying application and problem details.

B. Low-Distortion Subspace Embedding and Subspace-Preserving Embedding

In this section, we will describe in detail subspace embedding methods. Subspace embedding methods were first used in RandNLA by Drineas et al. in their relative-error approximation algorithm for ℓ_2 -regression (basically, the meta-algorithm described in Section II-A) [14]; they were first used in a data-oblivious manner in RandNLA by Sarlós [28]; and an overview of data-oblivious subspace embedding methods as used in RandNLA has been provided by Woodruff [13]. Based on the properties of the subspace embedding methods, we will present them in the following four categories. In Section IV-B1 and B2, we will discuss the data-oblivious subspace embedding methods for ℓ_2 - and ℓ_1 -norms, respectively; and then in Sections IV-B3 and B4, we will discuss the data-aware subspace embedding methods for ℓ_2 - and ℓ_1 -norms, respectively. Before getting into the details of these methods, we first provide some background and definitions.

Let us denote by $\mathcal{A} \subset \mathbb{R}^m$ the subspace spanned by the columns of A . A subspace embedding of \mathcal{A} into \mathbb{R}^s with $s > 0$ is a structure-preserving mapping $\phi: \mathcal{A} \hookrightarrow \mathbb{R}^s$, where the meaning of “structure-preserving” varies depending on the application. Here, we are interested in low-distortion linear embeddings of the normed vector space $\mathcal{A}_p = (\mathcal{A}, \|\cdot\|_p)$, the subspace \mathcal{A} paired with the ℓ_p -norm $\|\cdot\|_p$. (Again, although we are most interested in ℓ_1 and ℓ_2 , some of the

results hold more generally than for just $p = 2$ and $p = 1$, and so we formulate some of these results for general p .) We start with the following definition.

Definition 7 (Low-Distortion ℓ_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 an embedding of \mathcal{A}_p if $s = \mathcal{O}(\text{poly}(n))$, independent of m , and there exist $\sigma_\Phi > 0$ and $\kappa_\Phi > 0$ such that

$$\sigma_\Phi \cdot \|y\|_p \leq \|\Phi y\|_p \leq \kappa_\Phi \sigma_\Phi \cdot \|y\|_p \quad \forall y \in \mathcal{A}_p.$$

We call Φ 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 .

We remind the reader that low-distortion subspace embeddings can be used in one of two related ways: for ℓ_p -norm preconditioning and/or for solving directly ℓ_p -regression subproblems. We will start by establishing some terminology for their use for preconditioning.

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

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

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

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

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 ℓ_p -norm. We call a conditioning method that is obtained via computing the QR factorization of a low-distortion embedding a QR-type method; and we call a conditioning method that is obtained via an ER of a low-distortion embedding an ER-type method.

Furthermore, one can construct a well-conditioned basis by combining QR-like and ER-like methods. To see

this, let R be the matrix obtained by applying Corollary 1 to ΦA . We have

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

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

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

Hence

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

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

In Table 4, we summarize several different types of conditioning methods for ℓ_1 - and ℓ_2 -conditioning. Comparing the QR-type approach and the ER-type approach to obtaining the preconditioner matrix R , we see there are tradeoffs between running times and conditioning quality. Performing the QR decomposition takes $\mathcal{O}(sn^2)$ time [60], which is faster than fast ellipsoidal rounding that takes $\mathcal{O}(sn^3 \log s)$ time. However, the latter approach might provide a better conditioning quality when $2n < s^{1/p-1/2}$. We note that those tradeoffs are not important in most theoretical formulations, as long as both take $\mathcal{O}(\text{poly}(n))$ time and provide $\mathcal{O}(\text{poly}(n))$ conditioning, independent of m , but they certainly do affect the performance in practice.

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

Definition 8 (Subspace-Preserving Embedding): Given a matrix $A \in \mathbb{R}^{m \times n}$, $p \in [1, \infty]$ and $\epsilon \in (0, 1)$, $\Phi \in \mathbb{R}^{s \times m}$ is a subspace-preserving embedding of \mathcal{A}_p if $s = \mathcal{O}(\text{poly}(n))$, independent of m , and

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

1) *Data-Oblivious Low-Distortion ℓ_2 Subspace Embeddings:* An ℓ_2 subspace embedding is distinct from but closely related to the embedding provided by the Johnson–Lindenstrauss (J–L) lemma.

Lemma 7 (Johnson–Lindenstrauss Lemma [49]): Given $\epsilon \in (0, 1)$, a point set \mathcal{X} of N points in \mathbb{R}^m , there is a linear

Table 4 Summary of ℓ_1 - and ℓ_2 -Norm Conditioning Methods. QR and ER Refer, Respectively, to Methods Based on the QR Factorization and Methods Based on ER, as Discussed in the Text

name	κ	running time	# passes	type	norm
ER [35], [42]	$(n(n+1))^{1/2}$	$\mathcal{O}(mn^5 \log m)$	$\mathcal{O}(n^3 L)$	ER	ℓ_1
Fast ER [19]	$2n$	$\mathcal{O}(mn^3 \log m)$	$\mathcal{O}(nL)$	ER	ℓ_1
Single-pass ER [43]	$2n^2$	$\mathcal{O}(mn^2 \log m)$	1	ER	ℓ_1
CT [44]	$\mathcal{O}(n^{3/2} \log^{3/2} n)$	$\mathcal{O}(mn^2 \log n)$	1	QR	ℓ_1
FCT [19]	$\mathcal{O}(n^{9/2} \log^{9/2} n)$	$\mathcal{O}(mn \log n)$	1	QR	ℓ_1
SPCT [43]	$\mathcal{O}(n^{11/2} \log^{11/2} n)$	$\mathcal{O}(\text{nnz}(A))$	1	QR	ℓ_1
SPCT2 [43]	$6n$	$\mathcal{O}(\text{nnz}(A) \cdot \log n)$	2	QR+ER	ℓ_1
RET [45]	$\mathcal{O}(n^{5/2} \log^{5/2} n)$	$\mathcal{O}(\text{nnz}(A))$	1	QR	ℓ_1
Gaussian	$\mathcal{O}(1)$	$\mathcal{O}(mn^2)$	1	QR	ℓ_2
SRHT [15], [29], [46]	$\mathcal{O}(1)$	$\mathcal{O}(mn \log m)$	1	QR	ℓ_2
CW [20], [47], [48]	$\mathcal{O}(1)$	$\mathcal{O}(\text{nnz}(A))$	1	QR	ℓ_2

map $\phi : \mathbb{R}^m \mapsto \mathbb{R}^s$ 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. The projection can be represented by a random orthonormal matrix in $\mathbb{R}^{s \times m}$. Indyk and Motwani [50] 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. This simplifies the construction of a J-L transform, and it has improved algorithmic properties. Later, Achlioptas [51] showed that the random normal variables can be replaced by random signs, and moreover, we can zero out approximately 2/3 of the entries with proper scaling, while still maintaining the J-L property. The latter approach allows faster construction and projection with less storage, although still at the same order as the random normal projection.

The original J-L lemma applies to an arbitrary set of N vectors in \mathbb{R}^m . By using an ϵ -net argument and triangle inequality, Sarlós [28] 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 8 [28]: Let \mathcal{A}_2 be an arbitrary n -dimensional subspace of \mathbb{R}^m and $0 \leq \epsilon, \delta < 1$. If Φ 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

$$\Pr(\forall x \in \mathcal{A}_2 : \left| \|x\|_2 - \|\Phi x\|_2 \right| \leq \epsilon \|x\|_2) \geq 1 - \delta.$$

The result of Lemma 8 applies to any J-L transform, i.e., to any transform (including those with better or worse asymptotic FLOPS behavior) that satisfies the J-L distortion property.

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

$$\begin{aligned} \left| \|\Phi x\|_2^2 - \|x\|_2^2 \right| &= |(Ux)^T ((\Phi U)^T (\Phi U) - I)(Ux)| \\ &\leq \epsilon \|Ux\|_2^2 = \epsilon \|x\|_2^2 \end{aligned}$$

and hence

$$\left| \|\Phi x\|_2 - \|x\|_2 \right| \leq \frac{\epsilon \|x\|_2^2}{\|\Phi x\|_2 + \|x\|_2} \leq \epsilon \|x\|_2.$$

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 9 [52]: 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 $\sigma_1 \geq \dots \geq \sigma_n$. Then, for any $t > 0$

$$\max\{\Pr(\sigma_1 \geq \sqrt{s} + \sqrt{n} + t), \Pr(\sigma_n \leq \sqrt{s} - \sqrt{n} - t)\} < e^{-t^2/2}. \quad (12)$$

Using this concentration result, we can easily present a better analysis of random normal projection than in Lemma 8.

Corollary 2: Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$ 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 exists $s = \mathcal{O}((\sqrt{n} + \log(1/\delta))^2/\epsilon^2)$ such that, with probability at least $1 - \delta$, we have

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

Dense J–L transforms, e.g., a 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 Φ is a dense matrix of size $s \times m$ and $\text{nnz}(A)$ is the number of nonzero elements in A . There is also a line of research work on “fast” J–L transforms that started with [53] and [54]. These use fast Fourier transform (FFT)-like algorithms for the embedding, and thus they 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 Φ is a fast J–L transform. Before stating these results, we borrow the notion of fast Johnson–Lindenstrauss transform (FJLT) from [53] and [54] and use that to define a stronger and faster version of the simple J–L transform.

Definition 9 (FJLT): Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$, we say $\Phi \in \mathbb{R}^{s \times m}$ is an FJLT for \mathcal{A}_2 if Φ satisfies the following two properties:

- $\|(\Phi U)^T(\Phi U) - I_n\|_2 \leq \epsilon$, where U is an orthonormal basis of \mathcal{A}_2 ;
- given any $x \in \mathbb{R}^n$, Φx can be computed in at most $\mathcal{O}(m \log m)$ time.

Ailon and Chazelle construct the so-called FJLT [54], 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 an 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 [55], [56].

A subsequently refined FJLT was analyzed by Tropp [46], and it is named the subsampled randomized Hadamard transform (SRHT). As with other FJLT methods, the SRHT preserves the geometry of an entire ℓ_2 subspace of vectors by using a matrix Chernoff inequality to bound $\|(\Phi U)^T(\Phi U) - I\|_2$. We describe this particular FJLT in more detail.

Definition 10: 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 $m^{-1/2}$; and $R \in \mathbb{R}^{s \times m}$ restricts an n -dimensional vector to s coordinates, chosen uniformly at random.

Below we present the main results for SRHT from [15] since it has a better characterization of the subspace-preserving properties. We note that its proof is essentially a combination of the results in [29] and [46].

Lemma 10 (SRHT [15], [29], [46]): Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$ and $\epsilon, \delta \in (0, 1)$, let $\Phi \in \mathbb{R}^{s \times m}$ be a random SRHT with embedding dimension $s \geq (14n \ln(40mn)/\epsilon^2) \ln(30^2 n \ln(40mn)/\epsilon^2)$. Then, with probability at least 0.9, we have

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

Note that besides Walsh–Hadamard transform, other FFT-based transform, e.g., discrete Hartley transform (DHT) and discrete cosine transform (DCT) which have more practical advantages, can also be used; see [31] for details of other choices. Another important point to keep in mind (in particular, for parallel and distributed applications) is that, although called “fast,” a fast transform might be slower than a dense transform: when $\text{nnz}(A) = \mathcal{O}(m)$ (since machines are optimized for matrix–vector multiplies); when A ’s columns are distributively stored (since this slows down FFT-like algorithms, due to communication issues, e.g., if each machine has only certain rows from a tall matrix A , which is often the case, then it is not straightforward to perform an FFT on the columns); or for other machine-related issues.

More recently, Clarkson and Woodruff [20] developed an algorithm for the ℓ_2 subspace embedding that runs in so-called input-sparsity time, i.e., in $\mathcal{O}(\text{nnz}(A))$ time, plus lower order terms that depend polynomially on the low dimension of the input. Their construction is exactly the CountSketch matrix in the data stream literature [57], 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 ± 1 . 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} , i.e., based on the statistical leverage scores of A , they

proved that with an embedding dimension $\mathcal{O}(n^2/\epsilon^2)$, the above construction gives an ℓ_2 subspace embedding matrix. Improved bounds and simpler proofs (that have much more linear algebraic flavor) were subsequently provided by Meng and Mahoney [47] and Nelson and Nguyen [48]. In rest of this paper, we refer to this method as CW. Below, we present the main results from [20], [47], and [48].

Lemma 11 (Input-Sparsity Time Embedding for ℓ_2 [20], [47], [48]): Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$ and any $\delta \in (0, 1)$, let $s = (n^2 + n)/(\epsilon^2 \delta)$. Then, with probability at least $1 - \delta$

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

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

Remark 3: It is easy to see that computing ΦA , i.e., computing the subspace embedding, 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 ℓ_p -norms.

To summarize, in Table 5, we provide a summary of the basic properties of several data-oblivious ℓ_2 subspace embeddings discussed here (as well as of several data-aware ℓ_2 subspace-preserving embeddings that will be discussed in Section IV-B3).

Remark 4: With these low-distortion ℓ_2 subspace embeddings, one can use the QR-type method to compute an ℓ_2 preconditioner. That is, one can compute the QR factorization of the low-distortion subspace embeddings in Table 5 and use R^{-1} as the preconditioner; see Table 4 for

more details. We note that the tradeoffs in running time are implicit although they have the same conditioning quality. This is because the running time for computing the QR factorization depends on the embedding dimension which is varied from method to method. However, normally this is absorbed by the time for forming ΦA (theoretically, and it is in practice not the dominant effect).

2) *Data-Oblivious Low-Distortion ℓ_1 Subspace Embeddings:* General ℓ_p subspace embedding and even ℓ_1 subspace embedding is quite different from ℓ_2 subspace embedding. Here, we briefly introduce some existing results on ℓ_1 subspace embedding; for more general ℓ_p subspace embedding, see [47] and [20].

For ℓ_1 , the first question to ask is whether there exists an J-L transform equivalent. This question was answered in the negative by Charikar and Sahai [59].

Lemma 12 [59]: There exists a set of $\mathcal{O}(m)$ points in ℓ_1^m such that any linear embedding into ℓ_1^s has distortion at least $\sqrt{m/s}$. The tradeoff 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 ℓ_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 ℓ_1 , compared to the particularly “good” results provided by the J-L lemma for ℓ_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. In this case, Sohler and Woodruff [44] give the first linear oblivious embedding of an n -dimensional subspace of ℓ_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 . In particular, they prove the following quality bounds.

Lemma 13 (Cauchy Transform (CT) [44]): Let \mathcal{A}_1 be an arbitrary n -dimensional linear subspace of \mathbb{R}^m . Then, there is an $s_0 = s_0(n) = \mathcal{O}(n \log n)$ and a sufficiently large

Table 5 Summary of Data-Oblivious and Data-Aware ℓ_2 Embeddings. s Denotes the Embedding Dimension. By Running Time, We Mean the Time Needed to Compute ΦA . For Each Method, We Set the Failure Rate to Be a Constant. Moreover, “Exact lev. Scores Sampling” Means Sampling Algorithm Based on Using the Exact Leverage Scores (as Importance Sampling Probabilities); and “Appr. lev. Scores Sampling (SRHT)” and “Appr. lev. Scores Sampling (CW)” Are Sampling Algorithms Based on Approximate Leverage Scores Estimated by Using SRHT and CW (Using the Algorithm of [15]) as the Underlying Random Projections, Respectively. Note That Within the Algorithm (of [15]) for Approximating the Leverage Scores, the Target Approximation Accuracy Is Set to be a Constant

name	running time	s	κ_Φ
Gaussian (REF)	$\mathcal{O}(mns)$	$\mathcal{O}(n/\epsilon^2)$	$1 + \epsilon$
SRHT [15], [29], [46]	$\mathcal{O}(mn \log m)$	$\mathcal{O}(n \log(mn) \log(n/\epsilon^2)/\epsilon^2)$	$1 + \epsilon$
CW [20], [47], [48]	$\mathcal{O}(\text{nnz}(A))$	$(n^2 + n)/\epsilon^2$	$1 + \epsilon$
Exact lev. scores sampling [14]	$\mathcal{O}(mn^2)$	$\mathcal{O}(n \log n/\epsilon^2)$	$1 + \epsilon$
Appr. lev. scores sampling (SRHT) [15]	$\mathcal{O}(mn \log m)$	$\mathcal{O}(n \log n/\epsilon^2)$	$1 + \epsilon$
Appr. lev. scores sampling (CW) [15], [20]	$\mathcal{O}(\text{nnz}(A)) \log m$	$\mathcal{O}(n \log n/\epsilon^2)$	$1 + \epsilon$

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 chosen 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 [44]. The construction here is quite similar to the construction of the dense Gaussian embedding for ℓ_2 in Lemma 2, with several important differences. The most important differences are as follows:

- Cauchy random variables replace standard normal random variables;
- a larger embedding dimension does not always lead to better distortion quality;
- the failure rate becomes harder to control.

As CT is the ℓ_1 counterpart of the dense Gaussian transform, the fast Cauchy transform (FCT) proposed by Clarkson et al. [19] is the ℓ_1 counterpart of FJLT. There are several related constructions. For example, this FCT construction first preprocesses by a deterministic low-coherence matrix, then rescales by Cauchy random variables, and finally samples linear combinations of the rows. Then, they construct Φ 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 α sufficiently large, the parameter is set as $s = \alpha n \log(n/\delta)$, where $\delta \in (0, 1)$ controls the probability that the algorithm fails; $C \in \mathbb{R}^{2m \times 2m}$ is a diagonal matrix with diagonal entries chosen independently from a Cauchy distribution; and $H \in \mathbb{R}^{2m \times m}$ is a block-diagonal matrix composed of m/t blocks along the diagonal. Each block is the $2t \times t$ matrix $G_s = \begin{pmatrix} H_t \\ I_t \end{pmatrix}$, where I_t is the $t \times t$ identity matrix, and H_t is the normalized Hadamard matrix. (For simplicity, assume t is a power of two and m/t is an integer.)

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

Informally, 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 ℓ_1 -norm will not be too small relative to $\|y\|_1$. They prove the following quality bounds.

Lemma 14 (Fast Cauchy Transform (FCT) [19]): There is a distribution (given by the above construction) over matrices $\Phi \in \mathbb{R}^{s \times m}$, 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 $y \in \mathbb{R}^m$, the product Φy can be computed in $\mathcal{O}(m \log s)$ time.

To make the algorithm work with high probability, one has to set t to be at the order of s^6 and $s = \mathcal{O}(n \log n)$. It follows that $\kappa = \mathcal{O}(n^4 \log^4 n)$ in the above theorem. That is, while faster in terms of FLOPS than the CT, the FCT leads to worse embedding/preconditioning quality. Importantly, this result is different from how FJLT compares to dense Gaussian transform: FJLT is faster than the dense Gaussian transform, while both provide the same order of distortion; but FCT becomes faster than the dense Cauchy transform (CT), at the cost of somewhat worse distortion quality.

Similar to [20], [47], and [48] for computing an ℓ_2 subspace embedding, Meng and Mahoney [47] developed an algorithm for computing an ℓ_1 subspace embedding matrix in input-sparsity time, i.e., in $\mathcal{O}(\text{nnz}(A))$ time. They used a CountSketch-like matrix which can be written as the product of two matrices $\Phi = SC \in \mathbb{R}^{s \times m}$, 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 $C \in \mathbb{R}^{m \times m}$ is a diagonal matrix with diagonal entries chosen independently from the standard Cauchy distribution. We summarize the main theoretical results in the following lemma.

Lemma 15 (Sparse Cauchy Transform (SPCT) [47]): Given an n -dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^m$ and $\epsilon \in (0, 1)$, there is $s = \mathcal{O}(n^5 \log^5 n)$ such that with a constant probability

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

where Φ is the SPCT described above.

More recently, Woodruff and Zhang [45] proposed another algorithm that computes an ℓ_1 subspace embedding matrix in input-sparsity time. Its construction is similar to that of SPCT. That is, $\Phi = SD$ where D is a diagonal matrix with diagonal entries $1/u_1, 1/u_2, \dots, 1/u_n$ where u_i are exponential variables. Comparing to SPCT, the embedding dimension and embedding quality have been improved. We summarize the main results in the following lemma.

Lemma 16 [45]: Given an n -dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^m$ and $\epsilon \in (0, 1)$, there is $s = \mathcal{O}(n \log n)$ such that with a constant probability

$$\frac{1}{\mathcal{O}(n \log n) \|x\|_1} \leq \|\Phi x\|_1 \leq \mathcal{O}(n \log n) \|x\|_1 \quad \forall x \in \mathcal{A}_1$$

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

To summarize, in Table 6, we provide a summary of the basic properties of several data-oblivious ℓ_1 subspace embeddings discussed here (as well as of several data-aware ℓ_1 subspace-preserving embeddings that will be discussed in Section IV-B4).

3) *Data-Aware Low-Distortion ℓ_2 Subspace Embeddings:* All of the linear subspace embedding algorithms mentioned in previous sections are oblivious, i.e., independent of the input subspace. That has obvious algorithmic advantages, e.g., one can construct the embedding matrix without even looking at the data. Since using an oblivious embedding is not a hard requirement for the downstream task of solving ℓ_p -regression problems (and since one can use random projection embeddings to construct importance sampling probabilities [15] in essentially “random projection time,” up to small constant factors), a natural question is whether nonoblivious or data-aware embeddings could give better conditioning performance. In general, the answer is yes.

As mentioned in Section II-B, Drineas et al. [14] developed a sampling algorithm for solving ℓ_2 -regression by constructing a $(1 \pm \epsilon)$ -distortion ℓ_2 subspace-preserving sampling matrix. The underlying sampling distribution is

defined based on the statistical leverage scores of the design matrix which can be viewed as the “influence” of that row on the LS fit. That is, the sampling distribution is a distribution $\{p_i\}_{i=1}^m$ satisfying

$$p_i \geq \beta \cdot \frac{\ell_i}{\sum_j \ell_j}, \quad i = 1, \dots, m. \quad (13)$$

Above $\{\ell_i\}_{i=1}^m$ are the leverage scores of A and $\beta \in (0, 1]$. When $\beta = 1$ and $\beta < 1$, (13) implies we define $\{p_i\}_{i=1}^m$ according to the exact and estimated leverage scores, respectively.

More importantly, theoretical results indicate that, given a target desired accuracy, the required sampling complexity is independent of the higher dimension of the matrix. Similar construction of the sampling matrix appeared in several subsequent works, e.g., [14], [15], and [29], with improved analysis of the sampling complexity. For completeness, we include the main theoretical result regarding the subspace-preserving quality below, stated here for ℓ_2 .

Theorem 1 (ℓ_2 Subspace-Preserving Sampling [14], [15], [29]): Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$ represented by a matrix $A \in \mathbb{R}^{m \times n}$ and $\epsilon \in (0, 1)$, choose $s = \mathcal{O}(n \log n \log(1/\delta)/\beta\epsilon^2)$, and construct a sampling matrix $S \in \mathbb{R}^{m \times m}$ with diagonals

$$s_{ii} = \begin{cases} \frac{1}{\sqrt{q_i}}, & \text{with probability } q_i \\ 0, & \text{otherwise,} \end{cases} \quad i = 1, \dots, m$$

where

$$q_i \geq \min\{1, s \cdot p_i\}, \quad i = 1, \dots, m$$

and $\{p_i\}_{i=1}^m$ satisfies (13). Then, with probability at least 0.7

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

Table 6 Summary of Data-Oblivious and Data-Aware ℓ_1 Embeddings. Above, s Denotes the Embedding Dimension. By Running Time, We Mean the Time Needed to Compute ΦA . For Each Method, We Set the Failure Rate to Be a Constant. Moreover, “Sampling (FCT)” and “Sampling (SPCT)” Denote the ℓ_1 Sampling Algorithms Obtained by Using FCT and SPCT as the Underlying Preconditioning Methods, Respectively

name	running time	s	κ_Φ
CT [44]	$\mathcal{O}(mn^2 \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$
FCT [19]	$\mathcal{O}(mn \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^4 \log^4 n)$
SPCT [47]	$\text{nnz}(A)$	$\mathcal{O}(n^5 \log^5 n)$	$\mathcal{O}(n^3 \log^3 n)$
Reciprocal Exponential [45]	$\text{nnz}(A)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n^2 \log^2 n)$
Sampling (FCT) [19], [58]	$\mathcal{O}(mn \log n)$	$\mathcal{O}(n^{13/2} \log^{9/2} n \log(1/\epsilon)/\epsilon^2)$	$1 + \epsilon$
Sampling (SPCT) [19], [47], [58]	$\mathcal{O}(\text{nnz}(A) \cdot \log n)$	$\mathcal{O}(n^{15/2} \log^{11/2} n \log(1/\epsilon)/\epsilon^2)$	$1 + \epsilon$

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

On the other hand, by using the algorithm of [15], computing the leverage scores approximately can be done in essentially the time it takes to perform a random projection: in particular, Drineas *et al.* [15] suggested that one can estimate the leverage scores by replacing A with a “similar” matrix in the computation of the pseudoinverse (which is the main computational bottleneck in the exact computation of the leverage scores). To be more specific, by noticing that the leverage scores can be expressed as the row norms of AA^\dagger , we can use ℓ_2 subspace embeddings to estimate them. The high-level idea is

$$\|e_i AA^\dagger\|_2 \approx \left\| e_i A (\Phi_1 A)^\dagger \right\|_2 \approx \left\| e_i A (\Phi_1 A)^\dagger \Phi_2 \right\|_2$$

where e_i is a vector with zeros but 1 in the i th coordinate, $\Phi_1 \in \mathbb{R}^{n_1 \times m}$ is an FJLT, and $\Phi_2 \in \mathbb{R}^{n_2 \times r_2}$ is a JLT which preserves the ℓ_2 -norms of certain set of points. If the estimation of the leverage scores $\tilde{\ell}_i$ satisfies

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

then it is not hard to show that a sampling distribution $\{p_i\}_{i=1}^m$ defined according to $p_i = \tilde{\ell}_i / \sum_j \tilde{\ell}_j$ satisfies (13) with $\beta = (1 - \gamma)/(1 + \gamma)$. When γ is constant, say 0.5, from Theorem 1, the required sampling complexity will only need to be increased by a constant factor $1/\beta = 3$. This is less expensive, compared to the gain in the computation cost.

Suppose, now, we use SRHT (Lemma 10) or CW (Lemma 11) method as the underlying FJLT, i.e., Φ_1 , in the approximation of the leverage scores. Then, combining the theory suggested in [15] and Theorem 1, we have the following lemma.

Lemma 17 (Fast ℓ_2 Subspace-Preserving Sampling (SRHT) [15]): Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$ represented by a matrix $A \in \mathbb{R}^{m \times n}$ and $\epsilon \in (0, 1)$, it takes $\mathcal{O}(mn \log m)$ time to compute a sampling matrix $S \in \mathbb{R}^{s' \times m}$ (with only one nonzero element per row) with $s' = \mathcal{O}(n \log n / \epsilon^2)$ such that with constant probability

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

Lemma 18 (Fast ℓ_2 Subspace-Preserving Sampling (CW) [15], [20]): Given an n -dimensional subspace $\mathcal{A}_2 \subset \mathbb{R}^m$ represented by a matrix $A \in \mathbb{R}^{m \times n}$ and $\epsilon \in (0, 1)$, 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 $s' = \mathcal{O}(n \log n / \epsilon^2)$ such that with constant probability

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

Remark 5: Although using CW runs asymptotically faster than using SRHT, due to the poorer embedding quality of CW, in order to achieve the same embedding quality, and relatedly the same quality results in applications to ℓ_2 -regression, it may need a higher embedding dimension, i.e., r_1 . This results in a substantially longer QR factorization time for CW-based methods.

Finally, recall that a summary of both data-oblivious and data-aware subspace embedding for ℓ_2 -norm can be found in Table 5.

4) *Data-Aware Low-Distortion ℓ_1 Subspace Embeddings:* In the same way as we can use data-aware embeddings for ℓ_2 -regression, we can also use data-aware embeddings for ℓ_1 -regression. Indeed, the idea of using data-aware sampling to obtain $(1 \pm \epsilon)$ -distortion subspace embeddings for ℓ_1 -regression was first used in [42], where it was shown that an ℓ_1 subspace embedding can be done by weighted sampling after preprocessing the matrix, including preconditioning, using ellipsoidal rounding. Sampling probabilities depend on the ℓ_1 -norms of the rows of the preconditioned matrix. Moreover, the resulting sample has each coordinate weighted by the reciprocal of its sampling probability. Different from oblivious ℓ_1 subspace embeddings, the sampling approach can achieve a much better distortion.

Lemma 19 [42]: Given an n -dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^m$ (e.g., represented by a matrix $A \in \mathbb{R}^{m \times n}$) and $\epsilon, \delta \in (0, 1)$, it takes $\mathcal{O}(mn^5 \log m)$ time to compute a sampling matrix $S \in \mathbb{R}^{s' \times m}$ (with only one nonzero element per row) with $s' = \mathcal{O}(n^{3.5} \log(n/(\delta\epsilon)) / \epsilon^2)$ 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 ℓ_1 -norms of any vector from an 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.

Recall that the ℓ_2 leverage scores used in the ℓ_2 sampling algorithm described in Theorem 1 are the squared row norms of an orthonormal basis of \mathcal{A}_2 which can be viewed as a “nice” basis for the subspace of interest. Dasgupta *et al.* [35] generalized this method to the general ℓ_p case; in particular, they proposed to sample rows according to the ℓ_p row norms of AR^{-1} , where AR^{-1} is

a well-conditioned (in the ℓ_p sense of well conditioning) basis for \mathcal{A}_p . Different from ℓ_1 sampling algorithm [42] described above, computing such matrix R is usually sufficient, meaning it is not needed to preprocess A and form the basis AR^{-1} explicitly.

Theorem 2 (ℓ_p Subspace-Preserving Sampling [35]): Given an n -dimensional subspace $\mathcal{A}_p \subset \mathbb{R}^m$ represented by a matrix $A \in \mathbb{R}^{m \times n}$ and a matrix $R \in \mathbb{R}^{n \times n}$ such that AR^{-1} is well conditioned, $p \in [1, \infty)$, $\epsilon \in (0, 1/7)$, and $\delta \in (0, 1)$, choose

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

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

$$s_{ii} = \begin{cases} \frac{1}{p_i}, & \text{with probability } p_i \\ 0, & \text{otherwise,} \end{cases} \quad i = 1, \dots, m$$

where

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

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

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

In fact, Theorem 2 holds for any choice of R . When $R = I$, it implies sampling according to the ℓ_p row norms of A and the sampling complexity relies on $\bar{\kappa}_p^p(A)$. However, it is worth mentioning that a large condition number for A will lead to a large sampling size, which in turn affects the running time of the subsequent operations. Therefore, preconditioning is typically necessary. That is, one must find 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 the previous sections.

Given R such that AR^{-1} is well conditioned, computing the row norms of AR^{-1} takes $\mathcal{O}(\text{nnz}(A) \cdot n)$ time. Clarkson et al. [19] improve this running time by estimating the row norms of AR^{-1} instead of computing them exactly. The central idea is to postmultiply a random projection matrix $\Phi_2 \in \mathbb{R}^{n \times r}$ with $r = \mathcal{O}(\log m)$ which takes only $\mathcal{O}(\text{nnz}(A) \cdot \log m)$ time.

If one uses FCT or SPCT in Table 4 to compute a matrix R such that AR^{-1} is well conditioned and then uses the above idea to estimate quickly the ℓ_1 row norms of AR^{-1} to

define the sampling distribution, then by combining with Theorem 2, we have the following two results.

Lemma 20 (Fast ℓ_1 Subspace-Preserving Sampling (FCT) [19], [58]): Given an n -dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^m$ represented by a matrix $A \in \mathbb{R}^{m \times n}$ and $\epsilon \in (0, 1)$, it takes $\mathcal{O}(mn \log m)$ time to compute a sampling matrix $S \in \mathbb{R}^{s' \times m}$ (with only one nonzero element per row) with $s' = \mathcal{O}(n^{13/2} \log^{9/2} n \log(1/\epsilon)/\epsilon^2)$ such that with a constant probability

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

Lemma 21 (Fast ℓ_1 Subspace-Preserving Sampling (SPCT) [19], [47], [58]): Given an n -dimensional subspace $\mathcal{A}_1 \subset \mathbb{R}^m$ (e.g., represented by a matrix $A \in \mathbb{R}^{m \times n}$) and $\epsilon \in (0, 1)$, 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 $s' = \mathcal{O}(n^{15/2} \log^{11/2} n \log(1/\epsilon)/\epsilon^2)$ such that with a constant probability

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

Remark 6: Fast sampling algorithm also exists for ℓ_p -regression. That is, after computing a matrix R such that AR^{-1} is well conditioned, one can use a similar idea to approximate the ℓ_2 row norms of AR^{-1} , e.g., postmultiplying a random matrix with independent Gaussian variables (JLT), which lead to estimation of the ℓ_p row norms of AR^{-1} up to small factors; see [19] for more details.

Remark 7: We note that the speedup comes at the cost of increased sampling complexity, which does not substantially affect most theoretical formulations, since the sampling complexity is still $\mathcal{O}(\text{poly}(n) \log(1/\epsilon)/\epsilon^2)$. In practice, however, 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 tradeoff when implementing a subspace-preserving sampling algorithm.

Finally, recall that a summary of both data-oblivious and data-aware subspace embeddings for ℓ_1 -norm can be found in Table 6.

C. Application of Rounding/Embedding Methods to ℓ_1 - and ℓ_2 -Regression

In this section, we will describe how the ellipsoidal rounding and subspace embedding methods described in Section IV-A and B can be applied to solve ℓ_2 - and ℓ_1 -regression problems. In particular, by combining the tools we have introduced in Section IV-A and B, e.g., solving subproblems and constructing preconditioners with ER and subspace-embedding methods, we are able to describe

Table 7 Summary of RandNLA-Based ℓ_2 -Regression Solvers; PC Stands for Preconditioning

type	precision	example	reference
embedding + solving subproblem	low	CW + (FJLT+SVD)	[20]
		appr. lev. samp. (SRHT) + SVD	[15]
direct solver	high	SVD or QR	[60]
PC + direct solver	high	PC (Gaussian) + normal equation	[33]
PC + iterative alg.	high	PC (FJLT) + LSQR	[30], [31]

several approaches to compute very fine $(1 + \epsilon)$ relative-error solutions to ℓ_p -regression problems.

Depending on the downstream task of interest, e.g., how the solution to the regression problem will be used, one might be interested in obtaining low-precision solutions, e.g., $\epsilon = 10^{-1}$, medium-precision solutions, e.g., $\epsilon = 10^{-4}$, or high-precision solutions, e.g., $\epsilon = 10^{-10}$. As described in Section II, the design principles for these cases are somewhat different. In particular, the use of ℓ_2 and ℓ_1 well-conditioned bases is somewhat different, depending on whether one is interested in low precision. Here, we elaborate on how we can use the methods described previously construct low-precision solvers and high-precision solvers for solving ℓ_p -regression problems. As a reference, see Tables 7 and 8 for a summary of several representative RandNLA algorithms for solving ℓ_2 - and ℓ_1 -regression problems, respectively. (Most of these have been previously introduced for smaller scale computations in RAM; and in Section V we will describe several variants that extend to larger scale parallel and distributed environments.)

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

- 1) Construct a subspace-preserving embedding matrix Φ with distortion $1 \pm \epsilon/4$.
- 2) Using a black-box solver, solve the reduced-sized problem exactly, i.e., exactly solve

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

We refer to this approach as low precision since the running time complexity with respect to the error

parameter ϵ is $\text{poly}(1/\epsilon)$. Thus, while this approach can be analyzed for a fixed ϵ , this dependence means that as a practical matter this approach cannot achieve high-precision solutions.

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

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

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

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

Here, A is any matrix, but in particular it could be the constraint matrix A above augmented by the right-hand side vector b . For completeness, we include the following lemma stating this result more precisely.

Lemma 22: Given an ℓ_p -regression problem specified by $A \in \mathbb{R}^{m \times n}$ and $p \in [1, \infty)$ using the constrained formulation (9), let Φ be a $(1 \pm \epsilon/4)$ -distortion embedding of \mathcal{A}_p , and \hat{x} be an optimal solution to the reduced-sized problem $\min_{x \in \mathbb{R}^n} \|\Phi Ax\|_p$. Then, \hat{x} is a $(1 + \epsilon)$ -approximate solution to the original problem.

A great deal of work has followed this general approach. In particular, the meta-algorithm for ℓ_2 -regression from Section II is of this general form. Many other authors have proposed related algorithms that require solving the subproblem by first computing a subspace-preserving

Table 8 Summary of RandNLA-Based ℓ_1 -Regression Solvers; PC Stands for Preconditioning

type	precision	example	reference
(PC + sampling) + solving subproblem	low	(ER/fast ER + sampling) + IPM	[35], [42]
		(SCT/FCT + sampling) + IPM	[19], [44]
second-order	high	IPM	[61]
PC + first-order	high	ER + accelerated gradient descent	[41]

sampling matrix. See, e.g., [1] and references therein. Here, we simply cite several of the most immediately relevant for our subsequent discussion.

- Sampling for ℓ_2 -regression. One could use the original algorithm of [14] and [17], which performs a data-aware random sampling and solves the subproblem in $\mathcal{O}(mn^2)$ time to obtain an approximate solution. Using the algorithm of [15], the running time of this method was improved to roughly $\mathcal{O}(mn \log(n))$ time, and by combining the algorithm of [15] with the algorithm of [20], the running time was still further improved to input-sparsity time.
- Projections for ℓ_2 -regression. Alternatively, one could use the algorithm of [28] and [29], which performs a data-oblivious Hadamard-based random projection and solves the subproblem in roughly $\mathcal{O}(mn \log(n))$ time, or one could use the algorithm of [20], which runs in input-sparsity time.
- Sampling and projections for ℓ_1 - and ℓ_p -regression. See [19], [42], and [44] and see [20], [35], and [47] and references therein for both data-oblivious and data-aware methods, respectively.

To summarize these and other results, depending on whether the idealization that $m \gg n$ holds, either the Hadamard-based projections for ℓ_2 -regression (e.g., the projection algorithm of [29] or the sampling algorithm of [14] combined with the algorithm of [15]) and ℓ_1 -regression (e.g., the algorithm of [19]) or the input-sparsity time algorithms for ℓ_2 - and ℓ_1 -regression (e.g., the algorithms of [20] and [47]) lead to the best worst-case asymptotic performance. There are, however, practical tradeoffs, both in RAM and in parallel-distributed environments, and the most appropriate method to use in any particular situation is still a matter of ongoing research.

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

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

We refer to this approach as high precision since the running time complexity with respect to the error parameter ϵ is $\log(1/\epsilon)$. Among other things, this means

that, given a moderately good solution—e.g., the one obtained from the embedding that could be used in a low-precision solver—one can very easily obtain a very high precision solution.

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

- For ℓ_2 -regression. Recall that theoretical (and empirical) results suggest that the required number of iterations in many iterative solvers such as LSQR [62] depends strongly on the condition number of the system. Thus, a natural idea is first to compute a randomized preconditioner and then to apply one of these iterative solvers on the preconditioned system. For example, if we use SRHT (Lemma 10) 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. Rokhlin and Tygert [30] and Avron *et al.* [31] developed algorithms that use FJLT for preconditioning and LSQR as an iterative solver. In [32], Meng *et al.* developed a randomized solver for ℓ_2 -regression using Gaussian transform and LSQR or the Chebyshev semi-iterative method; see Section V-A for more details.

As with the low-precision solvers, note that if we use the input-sparsity time algorithm of [20] for embedding and then use an (SRHT+LSQR) approach above to solve the reduced-sized problem, then under the assumption that $m \geq \text{poly}(n)$ and ϵ is fixed, this particular combination would become the best approach proposed. However, there are various tradeoffs among those approaches. For instance, there are tradeoffs between running time and conditioning quality in preconditioning for computing the subspace-preserving sampling matrix, and there are tradeoffs between embedding dimension/sample size and failure rate in embedding/sampling. Some of the practical tradeoffs on different problem types and computing platforms will be discussed in Section V-C.

- For ℓ_1 -regression. While most of the work in RandNLA for high-precision solvers has been for ℓ_2 -regression, we should point out related work for ℓ_1 -regression. In particular, Nesterov [41] proposed an algorithm that employs a combination of ER and accelerated gradient descent; and second-order methods from [61] use interior point techniques more generally. See also the related solvers of Portnoy *et al.* [63], [64]. For

ℓ_1 -regression, Meng and Mahoney [43] coupled these ideas with RandNLA ideas to develop an iterative medium-precision algorithm for ℓ_1 -regression; see Section V-B for more details.

V. IMPLEMENTATIONS AND EMPIRICAL RESULTS

In this section, we describe several implementations in large-scale computational environments of the theory described in Section IV. In particular, in Section V-A, we will describe LSRN, an ℓ_2 -regression solver appropriate for parallel environments using multithreads and MPI; and then in Section V-B, we will describe the results of both a low-precision algorithm as well as a related medium-precision iterative algorithm for the ℓ_1 -regression problem. Both of these subsections summarize recent previous work, and they both illustrate how implementing RandNLA algorithms in parallel and distributed environments requires paying careful attention to computation-communication tradeoffs. These prior results do not, however, provide a comprehensive evaluation of any particular RandNLA method. Thus, for completeness, we also describe in Section V-C several new results: a comprehensive empirical evaluation of low-precision, medium-precision, and high-precision random sampling and random projection algorithms for the very overdetermined ℓ_2 -regression problem. Hereby, by “medium precision,” typically we mean calling a high-precision solver but executing fewer iterations in the underlying iterative solver. These implementations were done in Apache Spark¹¹; they have been applied to matrices of up to terabyte size; and they illustrate several points that will be important to understand as other RandNLA algorithms are implemented in very large-scale computational environments.¹²

A. Solving ℓ_2 -Regression in Parallel Environments

In this section, we describe implementation details for a high-precision ℓ_2 -regression solver designed for large-scale parallel environments. LSRN [32] is designed to solve the minimum-length LS problem (1) to high precision; and it works for linear systems that are either strongly overdetermined, i.e., $m \gg n$ or strongly underdetermined, i.e., $m \ll n$, and possibly rank deficient. LSRN uses random normal projections to compute a preconditioner matrix such that the preconditioned system is provably extremely well conditioned. In particular, either LSQR [62] (a conjugate-gradient-based method) or the Chebyshev semi-iterative (CS) method [66] can be used at the iterative step to compute the min-length solution within just a few iterations. As we will describe, the latter

method is preferred on clusters with high communication cost. Here, we only present the formal description of the LSRN algorithm for strongly overdetermined systems in Algorithm 2.

Algorithm 2: LSRN for strongly overdetermined systems

- 1: Choose an oversampling factor $\gamma > 1$, e.g., $\gamma = 2$. Set $s = \lceil \gamma n \rceil$.
- 2: Generate $G = \text{randn}(s, m)$, a Gaussian matrix.
- 3: Compute $\tilde{A} = GA$.
- 4: Compute \tilde{A} 's economy-sized SVD: $\tilde{U}\tilde{\Sigma}\tilde{V}^T$.
- 5: Let $N = \tilde{V}\tilde{\Sigma}^{-1}$. (Note that this is basically R^{-1} from QR on the embedding, but it is written here into the SVD.)
- 6: Iteratively compute the min-length solution \hat{y} to

$$\text{minimize}_{y \in \mathbb{R}^r} \|ANy - b\|_2.$$

- 7: Return $\hat{x} = N\hat{y}$.
-

Two important aspects of LSRN are the use of the Gaussian transform and the CS method, and they are coupled in a nontrivial way. In the remainder of this section, we discuss these issues.

To start, note that among the available choices for the random projection matrix, the Gaussian transform has particularly good conditioning properties. In particular, the distribution of the spectrum of the preconditioned system depends only on that of a certain Gaussian matrix, not the original linear system. In addition, one can show that

$$\mathcal{P}\left(\kappa(AN) \leq \frac{1 + \alpha + \sqrt{\frac{\tilde{r}}{s}}}{1 - \alpha - \sqrt{\frac{\tilde{r}}{s}}}\right) \geq 1 - 2e^{-\alpha^2 s/2}$$

where $\kappa(AN)$ is the condition number of the preconditioned system, r is the rank of A , and α is a parameter [32]. For example, if we choose the oversampling factor γ in Algorithm 2 to be 2, then the condition number of the new linear system is less than 6 with high probability. In addition, a result on bounds on the singular values provided in [32] enables CS to work more efficiently.

Moreover, while slower in terms of FLOPS than FFT-based fast transforms, the Gaussian transform comes with several other advantages for large-scale environments. First, it automatically speeds up with sparse input matrices and fast linear operators (in which case FFT-based fast transforms are no longer “fast”). Second, the preconditioning process is embarrassingly parallel and thus scales well in parallel environments. Relatedly, it is easy to implement using multithreads or MPI. Third, it still works (with an extra “allreduce” operation) when A is partitioned

¹¹<http://spark.apache.org>

¹²An important point that we will not describe in detail is that generating the random bits to implement the randomized algorithm and then randomly sampling parts of the data can be challenging when we do not have random access to the data; see, e.g., [65] and references therein.

along its bigger dimension. Last, when implemented properly, Gaussian random variables can be generated very fast [67] (which is nontrivial, given that the dominant cost in naïvely implemented Gaussian-based projections can be generating the random variables). For example, it takes less than 2 s to generate 10^9 random Gaussian numbers using 12 central processing unit (CPU) cores [32].

To understand why CS is preferable as a choice of iterative solver compared to other methods such as the conjugate-gradient-based LSRN, one has to take the convergence rate and computation/communication costs into account. In general, if (a bound for) the condition number of the linear system is large or not known precisely, then the CS method will fail ungracefully (while LSQR will just converge very slowly). However, with the very strong preconditioning guarantee of the Gaussian transform, we have very strong control on the condition number of the embedding, and thus the CS method can be expected to converge within a very few iterations. In addition, since CS does not have vector inner products that require synchronization between nodes (while the conjugate-gradient-based LSQR does), CS has one less synchronization point per iteration, i.e., it has improved communication properties. See Fig. 2 for the Python code snippets of LSQR and CS, respectively. On each iteration, both methods have to do two matrix–vector multiplications, while CS only needs one cluster-wide synchronization compared to two in LSQR. Thus, the more communication-efficient CS method is enabled by the very strong control on conditioning that is provided by the more expensive Gaussian projection. It is this advantage that makes CS favorable in the distributed environments, where communication costs are considered more expensive.

B. Solving ℓ_1 -Regression in Distributed Environments

In this section, we describe implementation details for both low-precision and high-precision solvers for the ℓ_1 -regression problem in large-scale distributed environments. These algorithms were implemented using

Listing 1: One iteration in LSQR

```
u = A.matvec(v) - alpha*u
beta = sqrt(comm.allreduce(np.dot(u,u)))
...
v = comm.allreduce(A.rmatvec(u)) - beta*v
```

Listing 2: One iteration in CS

```
v = comm.allreduce(A.rmatvec(r)) - beta*v
x += alpha*v
r -= alpha*A.matvec(v)
```

Fig. 2. Python code snippets for LSQR-based and CS-based iterations, respectively, illustrating that the latter has one synchronization point per iteration, while the former has two.

MapReduce framework [4] which (at least until the relatively recent development of the Apache Spark framework) was the *de facto* standard parallel environment for analyzing massive data sets.

1) *Low-Precision Solver*: Recall that one can use the sampling algorithm described in Section IV-C to obtain a low-precision approximate solution for ℓ_1 -regression. This can be summarized in the following three steps.

- 1) Compute an ℓ_1 well-conditioned basis $U = AR^{-1}$ for A .
- 2) Construct an importance sampling distribution $\{p_i\}_{i=1}^m$ based on the ℓ_1 row norms of U . Randomly sample a small number of constraints according to $\{p_i\}_{i=1}^m$ to construct a subproblem.
- 3) Solve the ℓ_1 -regression problem on the subproblem.

Next, we will discuss some of the implementation details of the above three steps in the MapReduce framework. The key thing to note is that, for the problems we are considering, the dominant cost is the cost of input/output, i.e., communicating the data,¹³ and hence we want to extract as much information as possible for each pass over the data.

The first step, as described in Section IV-C, is to construct an ℓ_1 well-conditioned basis for A ; and for this one can use one of the following three methods: ER; a QR factorization of ΦA , where ΦA is a low-distortion subspace embedding matrix in terms of ℓ_1 -norm (QR); or a combination of these two (QR+ER method). See Table 4 for summary of these approaches to conditioning. Note that many conditioning methods are embarrassingly parallel, in which case it is straightforward to implement them in MapReduce. For example, the CT with embedding dimension r can be implemented in the following manner.

Mapper:

- 1: For each row a_i of A , generate a vector $c_i \in \mathbb{R}^{r \times 1}$ consisting r standard Cauchy random variables.
- 2: For $j = 1, \dots, r$, emit $(j, c_{i,j}a_i)$ where $c_{i,j}$ denotes the j th element of c_i .

Reducer:

- 1: Reduce vectors associated with key k to v_k with addition operation.
 - 2: Return v_k .
-

After collecting all the vectors v_k , for $k = 1, \dots, r$, one only has to assemble these vectors and perform QR decomposition on the resulting matrix, which completes the preconditioning process.

¹³There are other communication costs, but the dominant cost in the MapReduce framework is the number of iterations, each of which involves an enormous amount of communication. This particular cost is partly mitigated by the state maintained in the Apache Spark framework, and as more algorithms are developed in Apache Spark, more attention will need to be paid to the other communication costs.

With the matrix R^{-1} such that AR^{-1} is well conditioned, a second pass over the data set is sufficient to construct a subproblem and obtain several approximate solutions to the original problem, i.e., the second and three steps of the sampling algorithm above. Note that since computation is a less precious resource than communication here, one can exploit this to compute multiple subsampled solutions in this single pass. (For example, performing, say, 100 matrix–vector products is only marginally more expensive than performing 1, and thus we can solve multiple subsampled solutions in a single “pass” with almost no extra effort. To provide an example, on a ten-node Hadoop cluster, with a matrix of size ca. $10^8 \times 50$, a single query took 282 s, while 100 queries took only 383 s, meaning that the extra 99 queries come almost “for free.”) We summarize the basic steps as follows. Assume that $A \in \mathbb{R}^{m \times n}$ has condition number κ_1 , s is the sampling size, and n_x is the number of approximate solutions desired.¹⁴ Then, the following algorithm returns n_x approximate solutions to the original problem.

Mapper:

- 1: For each row a_i of A , let $p_i = \min\{s\|a_i\|_1/(\kappa_1 n^{1/2}), 1\}$.
- 2: For $k = 1, \dots, n_x$, emit $(k, a_i/p_i)$ with probability p_i .

Reducer:

- 1: Collect row vectors associated with key k and assemble A_k .
 - 2: Compute $\hat{x}_k = \arg \min_{c^T x=1} \|A_k x\|_1$ using interior-point methods.
 - 3: Return \hat{x}_k .
-

Note here, in the second step of the reducer above, since the size of the subsampled matrix A_k typically only depends on the low dimension n , the subproblem can be fit into the memory of a single machine and can be solved locally.

As an aside, note that such an algorithm can be used to compute approximate solutions for other problems such as the quantile regression problem by only increasing the sampling size by a constant factor. In [58], Yang et al. evaluate the empirical performance of this algorithm by using several different underlying preconditioners, e.g., CT, FCT, etc., on a terabyte-size data set in Hadoop to solve ℓ_1 -regression and other quantile regression problems.

2) *High-Precision Solver*: To obtain a high-precision solution for the ℓ_1 -regression problem, we have to resort to iterative algorithms. See Table 9, where we summarize several iterative algorithms [34], [42], [68], [69] in terms of their convergence rates and complexity per iteration. Note that, among these methods, although the interior point cutting plane method (IPCPM) needs additional

¹⁴The condition number parameter κ_1 can be estimated with a traditional condition number estimator, or it can be computed to within relative error ϵ in an additional pass over the data [15].

Table 9 Iterative Algorithms for Solving ℓ_1 -Regression

	passes	extra work per pass
subgradient [42]	$\mathcal{O}(n^4/\epsilon^2)$	
gradient [34]	$\mathcal{O}(m^{1/2}/\epsilon)$	
ellipsoid [68]	$\mathcal{O}(n^2 \log(\kappa_1/\epsilon))$	
IPCPM [69]	$\mathcal{O}(n \log(\kappa_1/\epsilon))$	$\mathcal{O}(n^{7/2} \log n)$

work at each iteration, the needed number of passes is linear in the low dimension n and it only has a dependence on $\log(1/\epsilon)$. Again, since communication is a much more precious resource than computation in the distributed application where this was implemented, this can be an acceptable tradeoff when, e.g., a medium-precision solution is needed.

Meng and Mahoney [43] proposed a randomized IPCPM algorithm to solve the ℓ_1 -regression problem to medium precision in large-scale distributed environments. It includes several features specially designed for MapReduce and distributed computation. (To describe the method, recall that IPCPM is similar to a bisection method, except that it works in a high-dimensional space. It starts with a search region $\mathcal{S}_0 = \{x | Sx \leq t\}$, which contains a ball of desired solutions described by a separation oracle. At step k , we first compute the maximum-volume ellipsoid \mathcal{E}_k inscribing \mathcal{S}_k . Let y_k be the center of \mathcal{E}_k , and send y_k to the oracle. If y_k is not a desired solution, the oracle returns a linear cut that refines the search region $\mathcal{S}_k \rightarrow \mathcal{S}_{k+1}$.) The algorithm of [43] is different from the standard IPCPM, mainly for the following two reasons.

- Initialization using all the solutions returned by sampling algorithms. To construct a search region \mathcal{S}_0 , one can use the multiple solutions returned by calling the sampling algorithm, e.g., low-precision solutions, to obtain a much better initial condition. If we denote by $\hat{x}_1, \dots, \hat{x}_N$ the N approximation solution, then given each \hat{x} , let $\hat{f} = \|A\hat{x}\|_1$ and $\hat{g} = A^T \text{sign}(A\hat{x})$. Note that given $\hat{x}_1, \dots, \hat{x}_N$, computing \hat{f}_i, \hat{g}_i for $i = 1, \dots, N$ can be done in a single pass. Then, we have

$$\begin{aligned} \|x^* - \hat{x}\|_2 &\leq \|A(x^* - \hat{x})\|_1 \leq \|Ax^*\|_1 + \|A\hat{x}\|_1 \\ &\leq 2\hat{f}. \end{aligned}$$

Hence, for each subsampled solution \hat{x}_i , we have a hemisphere that contains the optimal solution. We use all these hemispheres to construct a better initial search region \mathcal{S}_0 , which may potentially reduce the number of iterations needed for convergence.

- Performing multiple queries per iteration. Instead of sending one query point at each iteration, one can exploit the fact that it is inexpensive to compute multiple query points per iteration, and one can send multiple query points at a time. Let

us still use \hat{x}_i to denote the multiple query points. Note that by convexity

$$\|Ax^*\|_1 \geq \|A\hat{x}\|_1 + \hat{g}^T(x^* - \hat{x}).$$

This implies $g^T x^* \leq g^T \hat{x}$. That is, given any query point \hat{x} , the subgradient serves as a separation oracle which returns a half-space that contains the desired ball. This means that, for each query point \hat{x}_i , a half-space containing the ball of desired solutions will be returned.

Note that both of these differences take advantage of performing extra computation while minimizing the number of iterations (which is strongly correlated with communication for MapReduce computations).

C. Detailed Empirical Evaluations of ℓ_2 -Regression Solvers in Parallel/Distributed Environments

In this section, we provide a detailed empirical evaluation of the performance of RandNLA algorithms for solving very overdetermined very large-scale ℓ_2 -regression problems. Recall that the subspace embedding that is a crucial part of RandNLA algorithms can be data aware (i.e., a sampling algorithm) or data oblivious (i.e., a projection algorithm). Recall also that, after obtaining a subspace embedding matrix, one can obtain a low-precision solution by solving the resulting subproblem, or one can obtain a high-precision solution by invoking an iterative solver, e.g., LSQR [62], for ℓ_2 -regression, with a preconditioner constructed from by the embedding. Thus, in this empirical evaluation, we consider both random sampling and random projection algorithms, and we consider solving the problem to low precision, medium precision, and high precision on a suite of data sets chosen to be challenging for different classes of algorithms. We consider a range of matrices designed to “stress test” all of the variants of the basic meta-algorithm of Section II that we have been describing, and we consider matrices of size ranging up to just over the terabyte size scale.¹⁵

1) *Experimental Setup*: In order to illustrate a range of uniformity and nonuniformity properties for both the leverage scores and the condition number, we considered the following four types of data sets:

- UG (matrices with uniform leverage scores and good condition number);
- UB (matrices with uniform leverage scores and bad condition number);
- NG (matrices with nonuniform leverage scores and good condition number);

¹⁵While our empirical evaluation is detailed, it is not exhaustive. In particular, we do not evaluate weak scaling versus strong scaling, two common metrics of interest in scientific computing and high-performance computing.

- NB (matrices with nonuniform leverage scores and bad condition number).

These matrices are generated in the following manner. For matrices with uniform leverage scores, we generated the matrices by using the commands that are listed in Table 10. For matrices with nonuniform leverage scores, we considered matrices with the following structure:

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

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

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

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

We call this stacking method `STACK2`. The two different stacking methods lead to different properties for the linear system being solved—we summarize these in Table 11—and, while they yielded results that were usually similar, as

Table 10 Commands (Presented in MATLAB format) Used to Generate Matrices With Uniform Leverage Scores, i.e., the UG and UB Matrices. Here, `kappa` Is a Parameter Used to Determine the Condition Number of the Generated Matrix

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

Table 11 Summary of Methods for Stacking Matrices, to Generate Matrices Too Large to Fit Into RAM; Here, REPNUM Denotes the Number of Replications and Coherence Is Defined as the Largest Leverage Score of the Matrix

NAME	CONDITION NUMBER	LEVERAGE SCORES	COHERENCE
STACK1	UNCHANGED	DIVIDED BY REPNUM	DIVIDED BY REPNUM
STACK2 (FOR NB AND NG ONLY)	INCREASED	UNKNOWN	ALWAYS 1

we mention below, the results were different in certain extreme cases. With either method of stacking matrices, the optimal solution remains the same, so that we can evaluate the approximate solutions of the new large LS problems. We considered these and other possibilities, but in the results reported below, unless otherwise specified, we choose the following: for large-scale UG and UB matrices, we use STACK1 to generate the data; and, for large-scale NG and NB matrices, we use STACK2 to generate the data.

Recall that Table 5 provides several methods for computing an ℓ_2 subspace embedding matrix. Since a certain type of random projection either can be used to obtain an embedding directly or can be used (with the algorithm of [15]) to approximate the leverage scores for use in sampling, we consider both data-aware and data-oblivious methods. Throughout our evaluation, we use the following notations to denote various ways of computing the subspace embedding:

- PROJ CW: random projection with the input-sparsity time CW method;
- PROJ GAUSSIAN: random projection with Gaussian transform;
- PROJ RADEMACHER: random projection with Rademacher transform;
- PROJ SRDHT: random projection with subsampled randomized discrete Hartley transform [70];
- SAMP APPR: random sampling based on approximate leverage scores;
- SAMP UNIF: random sampling with uniform distribution.

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

Throughout this section, by embedding dimension, we mean the projection size for projection-based methods and the sampling size for sampling-based methods. Also, it is worth mentioning that for sampling algorithm with approximate leverage scores, and we fix the underlying embedding method to be PROJ CW and the projection size c to be $d^2/4$. In our experiments, we found that—when they were approximated sufficiently well—the precise quality of the

approximate leverage scores does not have a strong influence on the quality of the solution obtained by the sampling algorithm. We will elaborate on this more in Section V-C3.

The computations for Table 12, Fig. 4, and Table 13 (i.e., for the smaller sized problems) were performed on a

Table 12 Quality of the Approximate Leverage Scores. The Test Was Performed on an NB Matrix With Size $1e6$ by 500. \hat{p} Denotes the Distribution by Normalizing the Approximate Leverage Scores and p^* Denotes the Exact Leverage Score Distribution. $D_{KL}(p||q)$ Is the KL Divergence [71] of q From p Defined as $\sum_i p_i \ln(p_i/q_i)$. Let $L = \{i|p_i^L = 1\}$ and $S = \{i|p_i^S < 1\}$. In This Case, \hat{p}^L Denotes the Corresponding Slice of \hat{p} , and the Quantities $\hat{p}^S, p^{*L}, p^{*S}$ Are Defined Similarly

c	CW	GAUSSIAN	RADEMACHER	SRDH
	$\ \hat{p} - p^*\ _2 / \ p^*\ _2$			
5E2	0.9205	0.7738	0.7510	0.5008
1E3	0.9082	0.0617	0.0447	0.0716
5E3	0.9825	0.0204	0.0072	0.0117
1E4	0.9883	0.0143	0.0031	0.0075
5E4	0.9962	0.0061	0.0006	0.0030
1E5	0.0016	0.0046	0.0003	0.0023
	$D_{KL}(p^* \hat{p})$			
5E2	18.5241	0.0710	0.6372	0.1852
1E3	19.7773	0.0020	0.0015	0.0029
5E3	20.3450	0.0002	0.0001	0.0001
1E4	20.0017	0.0001	0.0001	0.0001
5E4	19.2417	1.9E-5	1.0E-5	1.0E-5
1E5	0.0001	1.0E-5	5E-6	5E-6
	$\alpha_L = \max_i \{\hat{p}_i^L / p_i^{*L}\}$			
5E2	28.6930	7.0267	7.3124	4.0005
1E3	11.4425	1.1596	1.1468	1.2201
5E3	50.3311	1.0584	1.0189	1.0379
1E4	82.6574	1.0449	1.0099	1.0199
5E4	218.9658	1.0192	1.0018	1.0094
1E5	1.0016	1.0108	1.0009	1.0060
	$\alpha_S = \max_i \{\hat{p}_i^S / p_i^{*S}\}$			
5E2	0	24.4511	16.8698	4.5227
1E3	0	1.3923	1.3718	1.3006
5E3	0	1.1078	1.1040	1.1077
1E4	0	1.0743	1.0691	1.0698
5E4	0	1.0332	1.0317	1.0310
1E5	1.0236	1.0220	1.0218	1.0198
	$\beta_L = \min_i \{\hat{p}_i^L / p_i^{*L}\}$			
5E2	0	0.0216	0.0448	0.4094
1E3	0	0.8473	0.8827	0.8906
5E3	0	0.9456	0.9825	0.9702
1E4	0	0.9539	0.9916	0.9827
5E4	0	0.9851	0.9982	0.9922
1E5	0.9969	0.9878	0.9993	0.9934
	$\beta_S = \min_i \{\hat{p}_i^S / p_i^{*S}\}$			
5E2	0	0.0077	0.0141	0.1884
1E3	0	0.7503	0.7551	0.7172
5E3	0	0.9037	0.9065	0.9065
1E4	0	0.9328	0.9306	0.9356
5E4	0	0.9704	0.9691	0.9710
1E5	0.9800	0.9787	0.9789	0.9803

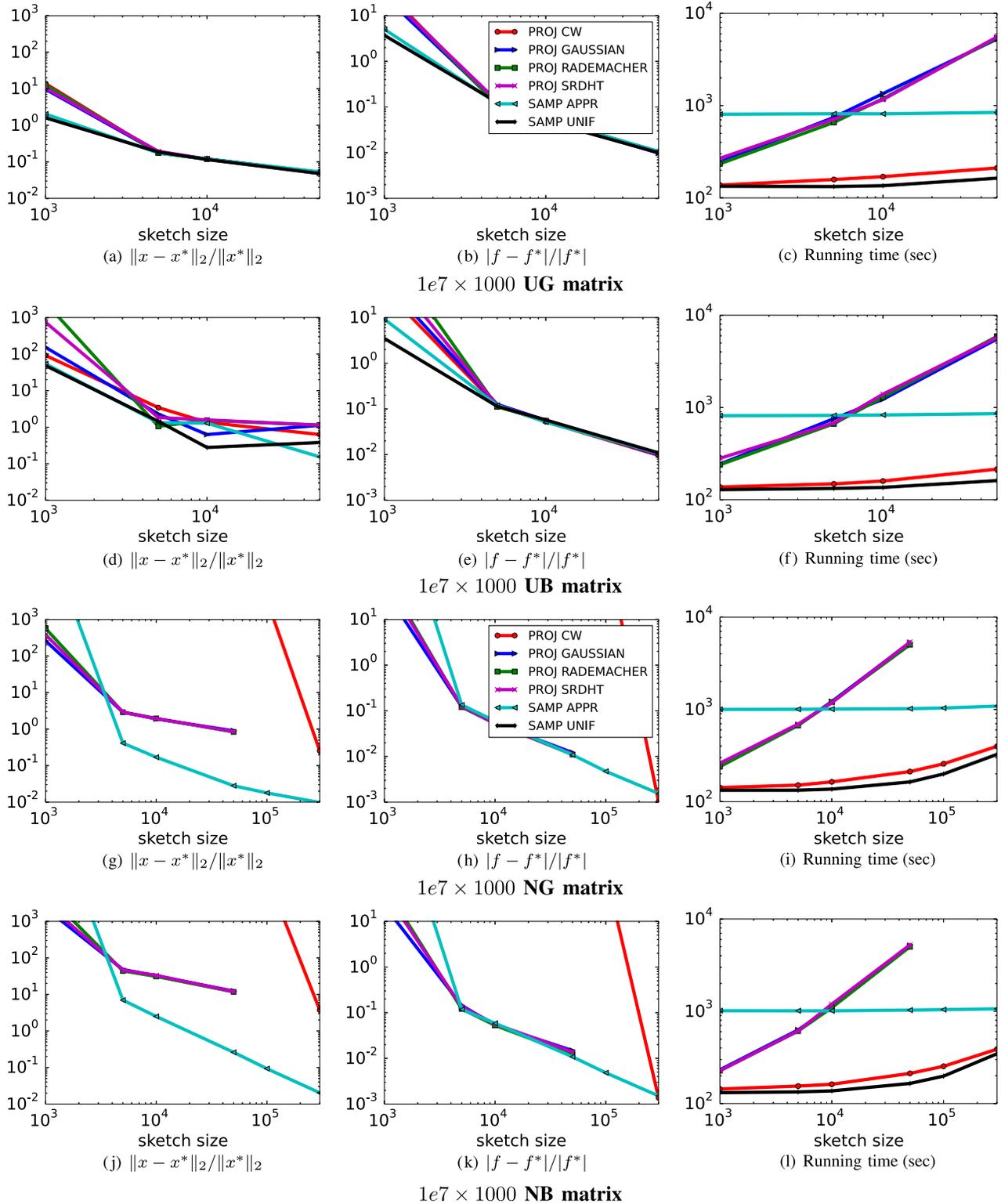


Fig. 3. Evaluation of all six of the algorithms on the four different types of matrices of size 1e7 by 1000. For each method, the following three quantities are computed: relative error of the objective $|f - f^*|/|f^*|$; relative error of the certificate $\|x - x^*\|_2/\|x^*\|_2$; and the running time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

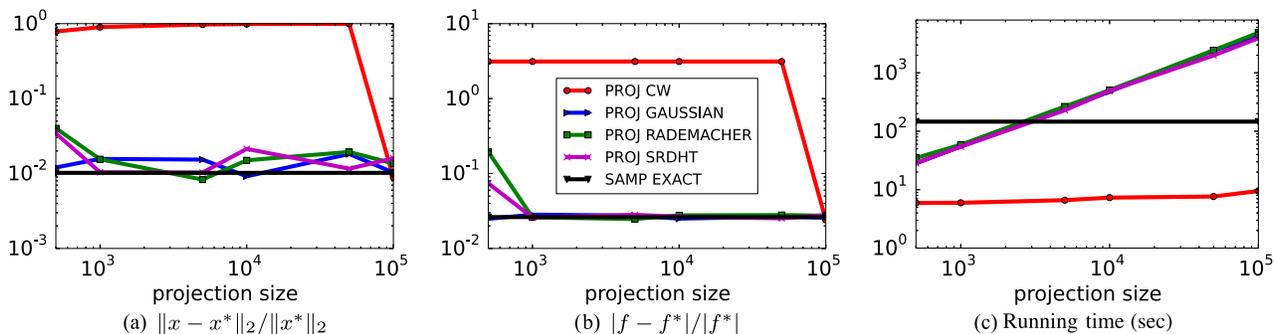


Fig. 4. Performance of sampling algorithms with approximate leverage scores, as computed by several different underlying projections. The test was performed on an NB matrix of size $1e6$ by 500 and the sampling size was $1e4$. Each subplot shows one of the following three quantities versus the projection size used in the underlying random projection phase: relative error of the objective $|f - f^*|/|f^*|$; relative error of the certificate $\|x - x^*\|_2/\|x^*\|_2$; and the running time. For each setting, five independent trials are performed and the median is reported.

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

2) *Overall Performance of Low-Precision Solvers:* Here, we evaluate the performance of the six kinds of embedding methods described above (with different embedding dimension) on the four different types of data set described above (with size $1e7$ by 1000). For dense transforms, e.g., PROJ GAUSSIAN, due to the memory capacity, the largest embedding dimension we can handle is $5e4$. For each data set and each kind of the embedding, we compute the following three quantities: relative error of the objective $|f - f^*|/|f^*|$; relative error of the solution certificate $\|x - x^*\|_2/\|x^*\|_2$; and the total running time to compute the approximate solution. The results are presented in Fig. 3.

As we can see, when the matrices have uniform leverage scores, all the methods including SAMP UNIF behave similarly. As expected, SAMP UNIF runs fastest, followed by PROJ CW. On the other hand, when the leverages scores are nonuniform, SAMP UNIF breaks down even with large

sampling size. Among the projection-based methods, the dense transforms, i.e., PROJ GAUSSIAN, PROJ RADEMACHER, and PROJ SRDHT, behave similarly. Although PROJ CW runs much faster, it yields very poor results until the embedding dimension is large enough, i.e., $c = 3e5$. Meanwhile, the sampling algorithm with approximate leverage scores, i.e., SAMP APPR, tends to give very reliable solutions. (This breaks down if the embedding dimension in the approximate leverage score algorithm is chosen to be too small.) In particular, the relative error is much lower throughout all choices of the embedding dimension. This can be understood in terms of the theory; see [14], [17], and [29] for details. In addition, its running time becomes more favorable when the embedding dimension is larger.

As a more minor point, theoretical results also indicate that the upper bound of the relative error of the solution vector depends on the condition number of the system as well as the amount of mass of b lies in the range space of A , denoted by γ [15]. Across the four data sets, γ is roughly the same. This is why we see the relative error of the certificate, i.e., the vector achieving the minimum solution tends to be larger when the condition number of the matrix becomes higher.

3) *Quality of the Approximate Leverage Scores:* Here, we evaluate the quality of the fast approximate leverage score algorithm of [15], and we investigate the quality of the approximate leverage scores with several underlying embeddings. (The algorithm of [15] considered only Hadamard-based projections, but other projection methods could be used, leading to similar approximation quality but different running times.) We consider only an NB matrix since leverage scores with nonuniform distributions are harder to approximate. In addition, the size of the matrix we considered is only rather small, $1e6$ by 500 , due to the need to compute the exact leverage scores for comparison. Our implementation follows closely the main algorithm of [15], except that we

Table 13 Quality of Preconditioning on an NB Matrix With Size $1e6$ by 500 Using Several Kinds of Embeddings. For Each Setting, Five Independent Trials Are Performed and the Median Is Reported

c	CW	GAUSSIAN	RADEMACHER	SRDHT	APPR
5E2	1.08E8	2.17E3	1.42E3	1.19E2	1.21E2
1E3	1.1E6	5.7366	5.6006	7.1958	75.0290
5E3	5.5E5	1.9059	1.9017	1.9857	25.8725
1E4	5.1E5	1.5733	1.5656	1.6167	17.0679
5E4	1.8E5	1.2214	1.2197	1.2293	6.9109
1E5	1.1376	1.1505	1.1502	1.1502	4.7573

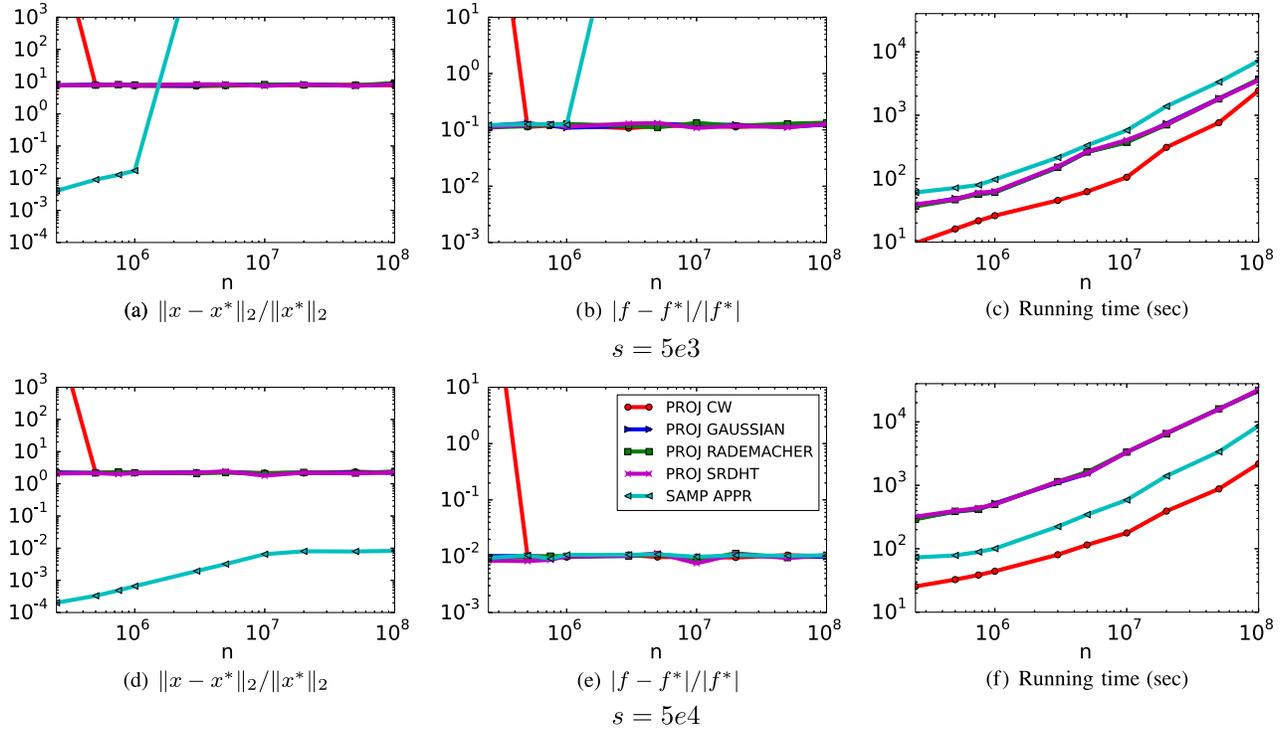


Fig. 5. Performance of all the algorithms on NB matrices with varying n from $2.5e5$ to $1e8$ and fixed $d = 1000$. The matrix is generated using STACK1. For each method, the embedding dimension is fixed to be $5e3$ or $5e4$. The following three quantities are computed: relative error of the objective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

consider other random projection matrices. In particular, we used the following four ways to compute the underlying embedding: namely, PROJ CW, PROJ GAUSSIAN, PROJ RADEMACHER, and PROJ SRDHT. For each kind of embedding and embedding dimension, we compute a series of quantities which characterize the statistical properties of the approximate leverage scores. The results are summarized in Table 12.

As we can see, when the projection size is large enough, all the projection-based methods to compute approximations to the leverage scores produce highly accurate leverage scores. Among these projection methods, PROJ CW is typically faster but also requires a much larger projection size in order to yield reliable approximate leverage scores. The other three random projections perform similarly. In general, the algorithms approximate the large leverage scores (those that equal or are close to 1) better than the small leverage scores, since α_L and β_L are closer to 1. This is crucial when calling SAMP APPR since the important rows should not be missed, and it is a sufficient condition for the theory underlying the algorithm of [15] to apply.

Next, we invoke the sampling algorithm for the ℓ_2 -regression problem, with sampling size $s = 1e4$ by using these approximate leverage scores. We evaluate the relative error on both the solution vector and objective

and the total running time. For completeness and in order to evaluate the quality of the approximate leverage score algorithm, we also include the results by using the exact leverage scores. The results are presented in Fig. 4.

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

4) *Performance of Low-Precision Solvers When n Changes:* Here, we explore the scalability of the low-precision solvers by evaluating the performance of all the embeddings on NB matrices with varying n . We fix $d = 1000$ and let n take values from $2.5e5$ to $1e8$. These matrices are generated by stacking an NB matrix with size $2.5e5$ by

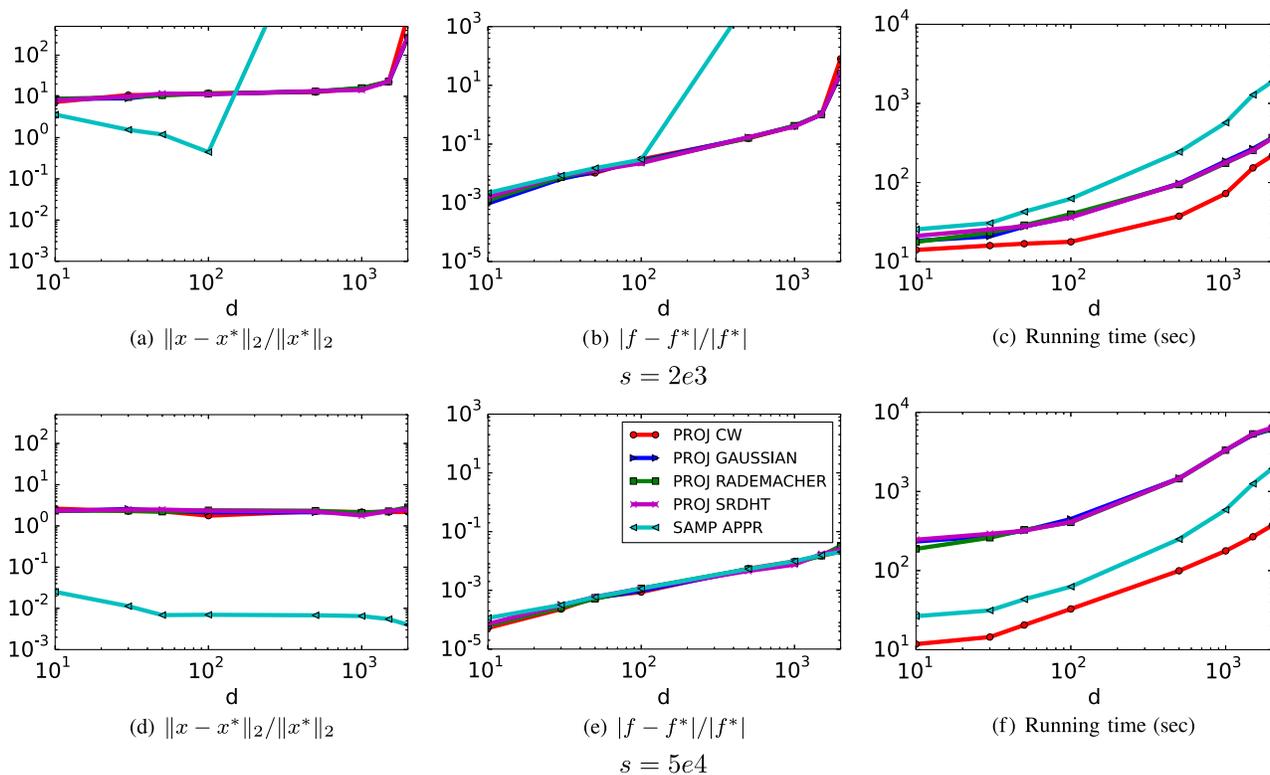


Fig. 6. Performance of all the algorithms on NB matrices with varying d from 10 to 2000 and fixed $n = 1e7$. The matrix is generated using `STACK1`. For each method, the embedding dimension is fixed to be $2e3$ or $5e4$. The following three quantities are computed: relative error of the objective $|f - f^*|/|f^*|$; relative error of the certificate $\|x - x^*\|_2/\|x^*\|_2$; and the running time to compute the approximate solution. For each setting, three independent trials are performed and the median is reported.

1000 REPNUM times, with REPNUM varying from 1 to 400 using `STACK1`. For conciseness, we fix the embedding dimension of each method to be either $5e3$ or $5e4$. The relative error on certificate and objective and running time are evaluated. The results are presented in Fig. 5.

Especially worthy mentioning is that when using `STACK1`, by increasing REPNUM, as we pointed out, the coherence of the matrix, i.e., the maximum leverage score, is decreasing, as the size is increased. We can clearly see that, when $n = 2.5e5$, i.e., the coherence is 1, PROJ CW fails. Once the coherence gets smaller, i.e., n gets larger, the projection-based methods behave similarly and the relative error remains roughly the same as we increased n . This is because `STACK1` does not alter the condition number and the amount of mass of the right-hand side vector that lies in the range space of the design matrix and the lower dimension d remains the same. However, SAMP APPR tends to yield larger error on approximating the certificate as we increase REPNUM, i.e., the coherence gets smaller. Moreover, it breaks down when the embedding dimension is very small.

5) *Performance of Low-Precision Solvers When d Changes:* Here, we evaluate the performance of the low-precision solvers by evaluating the performance of all the embed-

dings on NB matrices with changing d . We fix $n = 1e7$ and let d take values from 10 to 2000. For each d , the matrix is generated by stacking an NB matrix with size $2.5e5$ by d 40 times using `STACK1`, so that the coherence of the matrix is $1/40$. For conciseness, we fix the embedding of each method to be $2e3$ or $5e4$. The relative error on certificate and objective and running time are evaluated. The results are shown in Fig. 6.

As can be seen, overall, all the projection-based methods behave similarly. As expected, the relative error goes up as d gets larger. Meanwhile, SAMP APPR yields lower error as d increases. However, it seems to have a stronger dependence on the lower dimension of the matrix, as it breaks down when d is 100 for small sampling size, i.e., $s = 2e3$.

6) *Performance of High-Precision Solvers:* Here, we evaluate the use of these methods as preconditioners for high-precision iterative solvers. Since the embedding can be used to compute a preconditioner for the original linear system, one can invoke iterative algorithms such as LSQR [62] to solve the preconditioned LS problem. Here, we will use LSQR. We first evaluate the conditioning quality, i.e., $\kappa(\text{AR}^{-1})$, on an NB matrix with size $1e6$ by 500 using several different ways for computing the embedding. The

results are presented in Table 13. Then, we test the performance of LSQR with these preconditioners on an NB matrix with size $1e8$ by 1000 and an NG matrix with size $1e7$ by 1000 . For simplicity, for each method of computing the embedding, we try a small embedding dimension where some of the methods fail, and a large embedding dimension where most of the methods succeed. See Figs. 7 and 8 for details.

The convergence rate of the LSQR phase depends on the preconditioning quality, i.e., $\kappa(AR^{-1})$ where R is obtained by the QR decomposition of the embedding of A , ΦA . See Section IV-B for more details. Table 13 implies that all the projection-based methods tend to yield preconditioners with similar condition numbers once the embedding dimension is large enough. Among them, PROJ CW needs a much larger embedding dimension to be reliable (clearly consistent with its use in low-precision solvers). In addition, overall, the conditioning quality of the sampling-based embedding method, i.e., SAMP APPR tends to be worse than that of projection-based methods.

As for the downstream performance, from Fig. 7 we can clearly see that, when a small embedding dimension

is used, i.e., $s = 5e3$, PROJ GAUSSIAN yields the best preconditioner, as its better preconditioning quality translates immediately into fewer iterations for LSQR to converge. This is followed by SAMP APPR. This relative order is also suggested by Table 13. As the embedding dimension is increased, i.e., using large embedding dimension, all the methods yield significant improvements and produce much more accurate solutions compared to that of NOCO (LSQR without preconditioning), among which PROJ CW with embedding dimension $3e5$ converges to a nearly machine-precision solution within only five iterations. As for the running time, since each iteration of LSQR only involves with two matrix-vector multiplications (costs less than 2 min in our experiments), the overall running time is dominated by the time for computing the preconditioner. As expected, PROJ CW runs the fastest and the running time of PROJ GAUSSIAN scales linearly in the embedding dimension. In SAMP APPR, the sampling process needs to make one to two passes over the data set but the running time is relatively stable regardless of the sampling size, as reflected in Fig. 7(c) and (f). Finally, note that the reason that the error does not drop monotonically in the

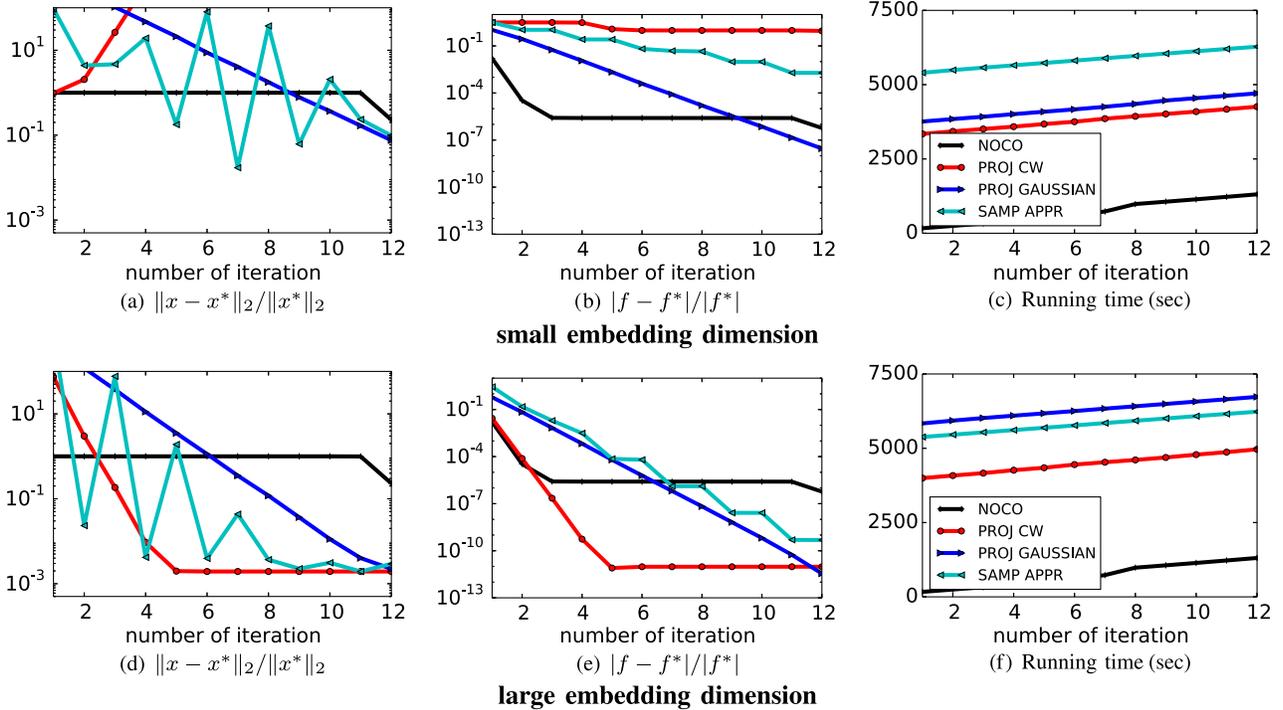


Fig. 7. Evaluation of LSQR with randomized preconditioner on an NB matrix with size $1e8$ by 1000 and condition number $1e6$. Here, several ways for computing the embedding are implemented. In SAMP APPR, the underlying random projection is PROJ CW with projection dimension $3e5$. For completeness, LSQR without preconditioner is evaluated, denoted by NOCO. By small embedding dimension, we mean $5e3$ for all the methods. By large embedding dimension, we mean $3e5$ for PROJ CW, $1e4$ for PROJ GAUSSIAN, and $5e4$ for SAMP APPR. For each method and embedding dimension, the following three quantities are computed: relative error of the objective $|f - f^*|/|f^*|$; relative error of the certificate $\|x - x^*\|_2/\|x^*\|_2$; and the running time to compute the approximate solution. Each subplot shows one of the above quantities versus the number of iterations, respectively. For each setting, only one trial is performed.

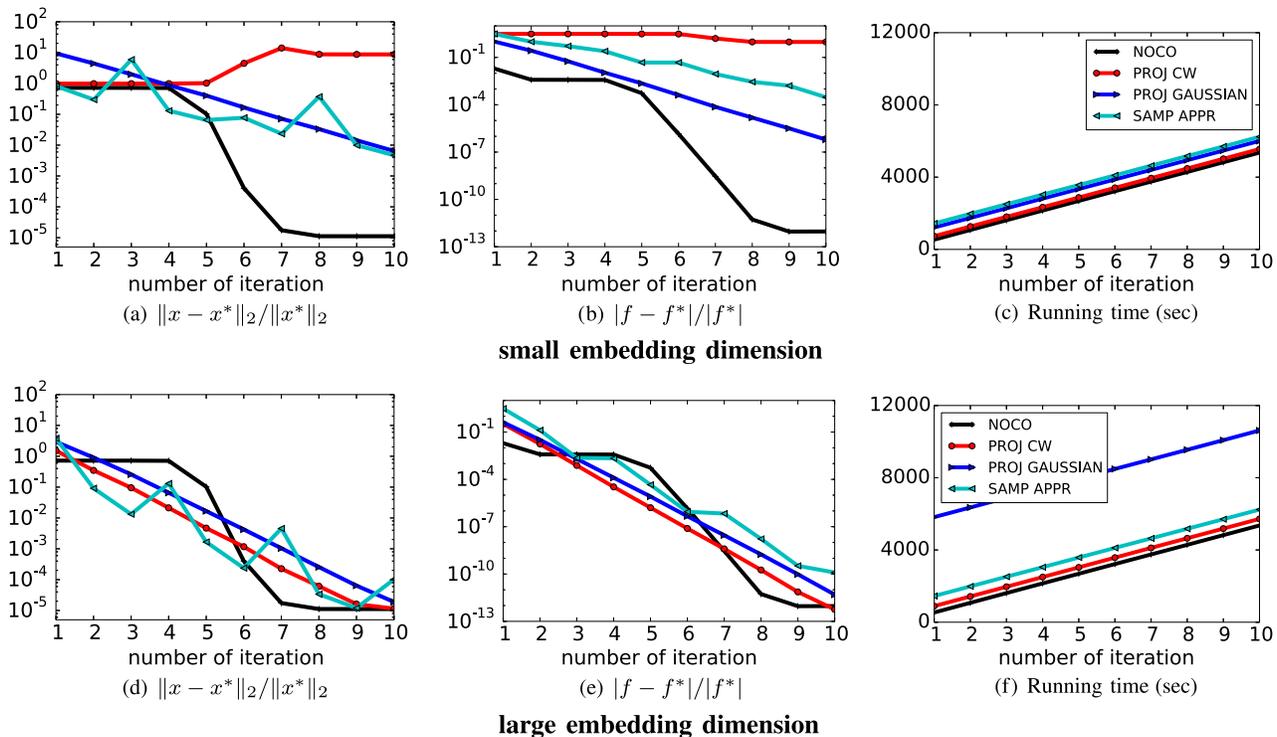


Fig. 8. Evaluation of LSQR with randomized preconditioner on an NB matrix with size $1e7$ by 1000 and condition number 5 . Here, several ways for computing the embedding are implemented. In SAMP APPR, the underlying random projection is PROJ CW with projection dimension $3e5$. For completeness, LSQR without preconditioner is evaluated, denoted by NOCO. By small embedding dimension, we mean $5e3$ for all the methods. By large embedding dimension, we mean $3e5$ for PROJ CW and $5e4$ for the rest. For each method and embedding dimension, the following three quantities are computed: relative error of the objective $\|x - x^*\|_2 / \|x^*\|_2$; relative error of the certificate $|f - f^*| / |f^*|$; and the running time to compute the approximate solution. Each subplot shows one of the above quantities versus the number of iterations, respectively. For each setting, three independent trials are performed and the median is reported.

solution vector is the following. With the preconditioners, we work on a transformed system, and the theory only guarantees monotonicity in the decreasing of the relative error of the certificate of the transformed system, not the original one.

Finally, a minor but potentially important point should be mentioned as a word of caution. As expected, when the condition number of the linear system is large, vanilla LSQR does not converge at all. On the other hand, when the condition number is very small, from Fig. 8, there is no need to precondition. If, in this latter case, a randomized preconditioning method is used, then the embedding dimension must be chosen to be sufficiently large: unless the embedding dimension is large enough such that the conditioning quality is sufficiently good, then preconditioned LSQR yields larger errors than even vanilla LSQR.

VI. DISCUSSION AND CONCLUSION

Large-scale data analysis and machine learning problems present considerable challenges and opportunities to signal processing, electrical engineering, scientific com-

puting, numerical linear algebra, and other research areas that have historically been developers of and/or consumers of high-quality matrix algorithms. RandNLA is an approach, originally from theoretical computer science, that uses randomization as a resource for the development of improved matrix algorithms, and it has had several remarkable successes in theory and in practice in small- to medium-scale matrix computations in RAM. The general design strategy of RandNLA algorithms (for problems such as ℓ_2 -regression and low-rank matrix approximation) in RAM is by now well known: construct a sketch (either by performing a random projection or by random sampling according to a judiciously chosen data-dependent importance sampling probability distribution), and then use that sketch to approximate the solution to the original problem (either by solving a subproblem on the sketch or using the sketch to construct a preconditioner for the original problem).

The work reviewed here highlights how, with appropriate modifications, similar design strategies can extend (for ℓ_2 -regression problems as well as other problems such as ℓ_1 -regression problems) to much larger scale parallel and distributed environments that are increasingly

common. Importantly, though, the improved scalability often comes due to restricted communications, rather than improvements in FLOPS. (For example, the use of the Chebyshev semi-iterative method versus LSQR in LSRN on MPI; and the use of the MIE with multiple queries on MapReduce.) In these parallel/distributed settings, we can take advantage of the communication-avoiding nature of RandNLA algorithms to move beyond FLOPS to design matrix algorithms that use more computation than the traditional algorithms but that have much better communication profiles, and we can do this by mapping the RandNLA algorithms to the underlying architecture in very nontrivial ways. (For example, using more computationally expensive Gaussian projections to ensure stronger

control on the condition number; and using the MIE with multiple initial queries to construct a very good initial search region.) These examples of performing extra computation to develop algorithms with improved communication suggest revisiting other methods from numerical linear algebra, optimization, and scientific computing, looking in other novel ways beyond FLOPS for better communication properties for many large-scale matrix algorithms. ■

Acknowledgment

The authors would like to thank M. Saunders for advice and helpful discussions.

REFERENCES

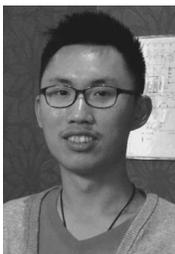
- [1] M. W. Mahoney, "Randomized algorithms for matrices and data," *Found. Trends Mach. Learn.*, vol. 3, no. 2, pp. 123–224, 2011.
- [2] A. Torralba, R. Fergus, and W. T. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 11, pp. 1958–1970, Nov. 2008.
- [3] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring volcanic eruptions with a wireless sensor network," in *Proc. IEEE 2nd Eur. Workshop Wireless Sensor Netw.*, 2005, pp. 108–120.
- [4] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Oper. Syst. Des. Implement.*, 2004, pp. 137–149.
- [5] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.
- [6] A. Aggarwal, A. K. Chandra, and M. Snir, "Communication complexity of PRAMs," *Theor. Comput. Sci.*, vol. 71, no. 1, pp. 3–28, 1990.
- [7] B. Lint and T. Agerwala, "Communication issues in the design and analysis of parallel algorithms," *IEEE Trans. Softw. Eng.*, vol. SE-7, no. 2, pp. 174–188, Mar. 1981.
- [8] D. Heller, "A survey of parallel algorithms in numerical linear algebra," *SIAM Rev.*, vol. 20, no. 4, pp. 740–777, 1976.
- [9] S. Toledo, "A survey of out-of-core algorithms in numerical linear algebra," *External Memory Algorithms*, J. M. Abello and J. S. Vitter, Eds. Providence, RI, USA: AMS, 1999, pp. 161–179.
- [10] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Minimizing communication in numerical linear algebra," *SIAM J. Matrix Anal. Appl.*, vol. 32, no. 3, pp. 866–901, 2011.
- [11] D. P. Bertsekas and J. N. Tsitsiklis, "Some aspects of parallel and distributed iterative algorithms—A survey," *Automatica*, vol. 27, no. 1, pp. 3–21, 1991.
- [12] N. Halko, P. G. Martinsson, and J. A. Tropp, "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions," *SIAM Rev.*, vol. 53, no. 2, pp. 217–288, 2011.
- [13] D. P. Woodruff, "Sketching as a tool for numerical linear algebra," *Found. Trends Theor. Comput. Sci.*, vol. 10, pp. 1–157, 2014.
- [14] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, "Sampling algorithms for ℓ_2 regression and applications," in *Proc. 17th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2006, pp. 1127–1136.
- [15] P. Drineas, M. Magdon-Ismael, M. W. Mahoney, and D. P. Woodruff, "Fast approximation of matrix coherence and statistical leverage," *J. Mach. Learn. Res.*, vol. 13, pp. 3475–3506, 2012.
- [16] P. Ma, M. W. Mahoney, and B. Yu, "A statistical perspective on algorithmic leveraging," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 861–911, 2015.
- [17] P. Drineas, M. W. Mahoney, and S. Muthukrishnan, "Relative-error CUR matrix decompositions," *SIAM J. Matrix Anal. Appl.*, vol. 30, pp. 844–881, 2008.
- [18] M. W. Mahoney and P. Drineas, "CUR matrix decompositions for improved data analysis," *Proc. Nat. Acad. Sci. USA*, vol. 106, pp. 697–702, 2009.
- [19] K. L. Clarkson et al., "The fast Cauchy transform and faster robust linear regression," in *Proc. 24th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2013, pp. 466–477.
- [20] K. L. Clarkson and D. P. Woodruff, "Low rank approximation and regression in input sparsity time," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, 2013, pp. 81–90.
- [21] A. Gittens and M. W. Mahoney, "Revisiting the Nyström method for improved large-scale machine learning," Tech. Rep., arXiv:1303.1849
- [22] E. J. Candes and B. Recht, "Exact matrix completion via convex optimization," *Commun. ACM*, vol. 55, no. 6, pp. 111–119, 2012.
- [23] D. C. Hoaglin and R. E. Welsch, "The hat matrix in regression and ANOVA," *Amer. Stat.*, vol. 32, no. 1, pp. 17–22, 1978.
- [24] S. Chatterjee and A. S. Hadi, "Influential observations, high leverage points, outliers in linear regression," *Stat. Sci.*, vol. 1, no. 3, pp. 379–393, 1986.
- [25] P. F. Velleman and R. E. Welsch, "Efficient computing of regression diagnostics," *Amer. Stat.*, vol. 35, no. 4, pp. 234–242, 1981.
- [26] S. Chatterjee and A. S. Hadi, *Sensitivity Analysis in Linear Regression*. New York, NY, USA: Wiley, 1988.
- [27] R. Motwani and P. Raghavan, *Randomized Algorithms*. New York, NY, USA: Cambridge Univ. Press, 1995.
- [28] T. Sarlós, "Improved approximation algorithms for large matrices via random projections," in *Proc. 47th Annu. IEEE Symp. Found. Comput. Sci.*, 2006, pp. 143–152.
- [29] P. Drineas, M. W. Mahoney, S. Muthukrishnan, and T. Sarlós, "Faster least squares approximation," *Numer. Math.*, vol. 117, no. 2, pp. 219–249, 2011.
- [30] V. Rokhlin and M. Tytgert, "A fast randomized algorithm for overdetermined linear least-squares regression," *Proc. Nat. Acad. Sci. USA*, vol. 105, no. 36, pp. 13 212–13 217, 2008.
- [31] H. Avron, P. Maysounkov, and S. Toledo, "Blendenpik: Supercharging LAPACK's least-squares solver," *SIAM J. Sci. Comput.*, vol. 32, no. 3, pp. 1217–1236, 2010.
- [32] 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.*, vol. 36, no. 2, pp. 95–118, 2014.
- [33] E. S. Coakley, V. Rokhlin, and M. Tytgert, "A fast randomized algorithm for orthogonal projection," *SIAM J. Sci. Comput.*, vol. 33, no. 2, pp. 849–868, 2011.
- [34] Y. Nesterov, "Unconstrained convex minimization in relative scale," *Math. Oper. Res.*, vol. 34, no. 1, pp. 180–193, 2009.
- [35] A. Dasgupta, P. Drineas, B. Harb, R. Kumar, and M. W. Mahoney, "Sampling algorithms and coresets for ℓ_p regression," *SIAM J. Comput.*, vol. 38, no. 5, pp. 2060–2078, 2009.
- [36] F. John, "Extremum problems with inequalities as subsidiary conditions," *Studies and Essays presented to R. Courant on his 60th Birthday, 1948*, pp. 187–204.
- [37] G. Barequet and S. Har-Peled, "Efficiently approximating the minimum-volume bounding box of a point set in three dimensions," *J. Algorithms*, vol. 38, no. 1, pp. 91–109, 2001.
- [38] L. Lovász, *An Algorithmic Theory of Numbers, Graphs, Convexity*. Philadelphia, PA, USA: SIAM, 1986.
- [39] C. Bouville, "Bounding ellipsoids for ray-fractal intersection," *ACM SIGGRAPH Comput. Graph.*, vol. 19, pp. 45–52, 1985.
- [40] L. G. Khachiyan and M. J. Todd, "On the complexity of approximating the maximal inscribed ellipsoid for a polytope," *Math. Program.*, vol. 61, no. 1, pp. 137–159, 1993.
- [41] Y. Nesterov, "Rounding of convex sets and efficient gradient methods for linear programming problems," *Optim. Methods Softw.*, vol. 23, no. 1, pp. 109–128, 2008.
- [42] K. L. Clarkson, "Subgradient and sampling algorithms for ℓ_1 regression," in *Proc. 16th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2005, pp. 257–266.
- [43] X. Meng and M. W. Mahoney, "Robust regression on MapReduce," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 888–896.

- [44] C. Sohler and D. P. Woodruff, "Subspace embeddings for the ℓ_1 -norm with applications," in *Proc. 43rd Annu. ACM Symp. Theory Comput.*, 2011, pp. 755–764.
- [45] D. P. Woodruff and Q. Zhang, "Subspace embeddings and ℓ_p -regression using exponential random variables," *COLT*, vol. 30, pp. 546–567, 2013.
- [46] J. A. Tropp, "Improved analysis of the subsampled randomized Hadamard transform," *Adv. Adapt. Data Anal.*, vol. 3, no. 1–2, pp. 115–126, 2011.
- [47] X. Meng and M. W. Mahoney, "Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression," in *Proc. 45th Annu. ACM Symp. Theory Comput.*, 2013, pp. 91–100.
- [48] J. Nelson and H. Nguyen, "OSNAP: Faster numerical linear algebra algorithms via sparser subspace embeddings," in *Proc. 54th Annu. IEEE Symp. Found. Comput. Sci.*, 2013, pp. 117–126.
- [49] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," *Contemporary Math.*, vol. 26, pp. 189–206, 1984.
- [50] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. 30th Annu. ACM Symp. Theory Comput.*, 1998, pp. 604–613.
- [51] D. Achlioptas, "Database-friendly random projections," in *Proc. 20th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.*, 2001, pp. 274–281.
- [52] K. R. Davidson and S. J. Szarek, "Local operator theory, random matrices and Banach spaces," *Handbook of the Geometry of Banach Spaces*, vol. 1. Amsterdam, The Netherlands: North Holland, 2001, pp. 317–366.
- [53] N. Ailon and B. Chazelle, "Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform," in *Proc. 38th Annu. ACM Symp. Theory Comput.*, 2006, pp. 557–563.
- [54] N. Ailon and B. Chazelle, "The fast Johnson-Lindenstrauss transform and approximate nearest neighbors," *SIAM J. Comput.*, vol. 39, no. 1, pp. 302–322, 2009.
- [55] N. Ailon and E. Liberty, "Fast dimension reduction using Rademacher series on dual BCH codes," *Discrete Comput. Geometry*, vol. 42, no. 4, pp. 615–630, 2009.
- [56] N. Ailon and E. Liberty, "An almost optimal unrestricted fast Johnson-Lindenstrauss transform," in *Proc. 22nd Annu. ACM-SIAM Symp. Discrete Algorithms*, 2011, pp. 185–191.
- [57] M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proc. Int. Colloq. Automata Lang. Program.*, 2002, pp. 693–703.
- [58] J. Yang, X. Meng, and M. W. Mahoney, "Quantile regression for large-scale applications," *SIAM J. Sci. Comput.*, vol. 36, no. 5, pp. S78–S110, 2014.
- [59] Y. Nesterov and A. Nemirovsky, "Dimension reduction in the ℓ_1 -norm," in *Proc. 43rd Annu. IEEE Symp. Found. Comput. Sci.*, 2002, pp. 551–560.
- [60] G. H. Golub and C. F. Van Loan, *Matrix Computation*, 3rd ed. Baltimore, MD, USA: Johns Hopkins Univ. Press, 1996.
- [61] Y. Nesterov and A. Nemirovsky, *Interior Point Polynomial Methods in Convex Programming*. Philadelphia, PA, USA: SIAM, 1994.
- [62] C. C. Paige and M. A. Saunders, "LSQR: An algorithm for sparse linear equations and sparse least squares," *ACM Trans. Math. Softw.*, vol. 8, no. 1, pp. 43–71, 1982.
- [63] S. Portnoy and R. Koenker, "The Gaussian hare and the Laplacian tortoise: Computability of squared-error versus absolute-error estimators," *Stat. Sci.*, vol. 12, no. 4, pp. 279–300, 1997.
- [64] S. Portnoy, "On computation of regression quantiles: Making the Laplacian tortoise faster," *Lecture Notes—Monograph Ser.*, vol. 31, L_1 -Statistical Procedures and Related Topics, pp. 187–200, 1997.
- [65] X. Meng, "Scalable simple random sampling and stratified sampling," in *Proc. 30th Int. Conf. Mach. Learn.*, 2013, pp. 531–539.
- [66] G. H. Golub and R. S. Varga, "Chebyshev semi-iterative methods, successive over-relaxation methods, second-order Richardson iterative methods parts I and II," *Numer. Math.*, vol. 3, no. 1, pp. 147–168, 1961.
- [67] G. Marsaglia and W. W. Tsang, "The ziggurat method for generating random variables," *J. Stat. Softw.*, vol. 5, no. 8, pp. 1–7, 2000.
- [68] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.
- [69] S. Tarasov, L. G. Khachiyan, and I. Erlikh, "The method of inscribed ellipsoids," *Soviet Mathematics Doklady*, vol. 37, pp. 226–230, 1988.
- [70] R. N. Bracewell, "Discrete hartley transform," *J. Opt. Soc. Amer.*, vol. 73, no. 12, pp. 1832–1835, 1983.
- [71] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Stat.*, vol. 22, no. 1, pp. 79–86, 1951.

ABOUT THE AUTHORS

Jiyan Yang (Student Member, IEEE) received the B.S. degree in mathematics from Nanjing University, Nanjing, China, in 2011 and the M.S. degree in computational mathematics from Stanford University, Stanford, CA, USA, in 2013, where he is currently working toward the Ph.D. degree in computational mathematics.

His research interests include numerical linear algebra and optimization, large-scale machine learning, and randomized algorithms.



Xiangrui Meng received the Ph.D. degree in computational and mathematical engineering from Stanford University, Stanford, CA, USA, in 2014.

He is currently a Software Engineer at Databricks, San Francisco, CA, USA and an Apache Spark project management committee member. His main interests center around developing and implementing scalable algorithms for scientific applications. At Databricks, he has been actively involved in the development and maintenance of MLlib, the machine learning component of Apache Spark. Before Databricks, he worked as an applied research engineer at LinkedIn, where he was the main developer of an offline machine learning framework in Hadoop MapReduce.



Michael W. Mahoney received the Ph.D. degree from Yale University, New Haven, CT, USA, in 2000, with a dissertation in computational statistical mechanics.

He is in the Department of Statistics and at the International Computer Science Institute (ICSI), University of California at Berkeley, Berkeley, CA, USA. He works on algorithmic and statistical aspects of modern large-scale data analysis. Much of his recent research has focused on large-scale machine learning, including randomized matrix algorithms and randomized numerical linear algebra; geometric network analysis tools for structure extraction in large informatics graphs; scalable implicit regularization methods; and applications in genetics, astronomy, medical imaging, social network analysis, and internet data analysis. He has worked and taught in the Mathematics Department, Yale University; at Yahoo Research, Sunnyvale, CA, USA; and in the Mathematics Department, Stanford University, Stanford, CA, USA. Among other things, he is on the National Advisory Committee of the Statistical and Applied Mathematical Sciences Institute (SAMSI), he was on the National Research Council's Committee on the Analysis of Massive Data, he coorganized the Simons Institute's Fall 2013 program on the Theoretical Foundations of Big Data Analysis, and he runs the biennial MADS Workshops on Algorithms for Modern Massive Data Sets.

