Solving Sparse Linear Systems Faster than Matrix Multiplication

By Richard Peng and Santosh S. Vempala

Abstract
Can linear systems be solved faster than matrix multiplication? While there has been remarkable progress for the special cases of graph-structured linear systems, in the general setting, the bit complexity of solving an \( n \times n \) linear system \( Ax = b \) is \( \tilde{O}(n^\omega) \), where \( \omega \) is the matrix multiplication exponent. Improving on this has been an open problem even for sparse linear systems with \( \text{poly}(n) \) condition number.

In this paper, we present an algorithm that solves linear systems with sparse coefficient matrices asymptotically faster than matrix multiplication for any \( \omega > 2 \). This speedup holds for any input matrix \( A \) with \( o(n^{\omega-1}/\log(\kappa(A))) \) nonzeros, where \( \kappa(A) \) is the condition number of \( A \).

Our algorithm can be viewed as an efficient, randomized implementation of the block Krylov method via recursive low displacement rank factorization. It is inspired by an algorithm of Eberly et al. for inverting matrices over finite fields. In our analysis of numerical stability, we develop matrix anti-concentration techniques to bound the smallest eigenvalue and the smallest gap in the eigenvalues of semi-random matrices.

1. INTRODUCTION
Solving a linear system \( Ax = b \) is a basic algorithmic problem with direct applications to scientific computing, engineering, and physics, and is at the core of algorithms for many other problems, including optimization, data science, and computational geometry. It has enjoyed an array of elegant approaches, from Cramer’s rule and Gaussian elimination to numerically stable iterative methods to more modern randomized variants based on random sampling and sketching. Despite much recent progress on faster solvers for graph-structured linear systems\(^{5,9,19} \) progress on the general case has been elusive.

Most of the work in obtaining better running time bounds for linear systems solvers has focused on efficiently computing the inverse of \( A \), or some factorization of it. Such operations are in turn closely related to the cost of matrix multiplication. Matrix inversion can be reduced to matrix multiplication via divide-and-conquer, and this reduction was shown to be stable when the word size for representing numbers is increased by a factor of \( O(\log n) \). The current best runtime of \( \omega < 2.38 \) follows a long line of work on faster matrix multiplication algorithms and is also the current best running time for solving \( Ax = b \): when the input matrix/vector are integers, matrix multiplication based algorithms can obtain the exact rational solution using \( O(n^\omega) \) word operations.

Methods for matrix inversion and factorization are often referred to as direct methods in the linear systems literature. This is in contrast to iterative methods, which gradually converge to the solution. Iterative methods have low space overhead, and therefore are widely used for solving large, sparse, linear systems that arise in scientific computing. Another reason for their popularity is that iterative methods are naturally suited to producing approximate solutions of desired accuracy in floating point arithmetic, the de facto method for representing real numbers. Perhaps the most famous iterative method is the Conjugate Gradient (CG) / Lanczos algorithm. It was introduced as an \( O(n \cdot \text{nnz}) \) time algorithm under exact arithmetic, where \( \text{nnz} \) is the number of non-zeros in the input matrix. However, this bound only holds under the Real RAM model where words have unbounded precision. When taking bit sizes into account, it incurs an additional factor of \( n \). Despite much progress in iterative techniques in the intervening decades, obtaining gains over matrix multiplication in the presence of roundoff errors has remained an open question.

The convergence and stability of iterative methods typically depend on some condition number of the input. When all intermediate steps are carried out using precision close to the condition number of \( A \), the running time bounds of the CG algorithm, as well as other currently known iterative methods, depend polynomially on the condition number of the input matrix \( A \). Formally, the condition number of a symmetric matrix \( A, \kappa(A) \), is the ratio between the maximum and

---

\( b \) We will be measuring bit-complexity under fixed-point arithmetic. Here the machine word size is on the order of the maximum number of digits of precision in \( A \), and the total cost is measured by the number of word operations. The need to account for bit-complexity of the numbers naturally led to the notion of condition number. The logarithm of the condition number measures the additional number of words needed to store \( A^{-1} \) (and thus \( A^{-1} b \)) compared to \( A \). In particular, matrices with \( \text{poly}(n) \) condition number can be stored with a constant factor overhead in precision, and are numerically stable under standard floating point number representations.

The original version of this paper was published in the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA21).
minimum eigenvalues of $A$. Here the best known rate of convergence when all intermediate operations are restricted to bit-complexity $O(\log (k(A)))$ is $O(\sqrt{k(A)} \log (1/\epsilon))$ iterations to achieve error $\epsilon$. This is known to be tight if one restricts to matrix-vector multiplications in the intermediate steps.\(^{1,17}\)

This means for moderately conditioned (e.g., with $\kappa = \text{poly}(n)$), sparse, systems, the best runtime bounds are still via direct methods, which are stable when $O(\log (1/\kappa))$ words of precision are maintained in intermediate steps.\(^{4}\)

Many of the algorithms used in research highlights in scientific computing for solving linear systems involving large, sparse matrices are based on combining direct and iterative methods: we will briefly discuss these perspectives in Section 1.3. In terms of asymptotic complexity, the practical successes of many such methods naturally lead to the question of whether one can provably do better than the $O(\min \{ n^\omega, \text{nnz} \cdot \sqrt{k(A)} \})$ time corresponding to the faster of direct or iterative methods. Somewhat surprisingly, despite the central role of this question in scientific computing and numerical analysis, as well as extensive studies of linear systems solvers, progress on it has been elusive. The continued lack of progress on this question has led to its use as a hardness assumption for showing conditional lower bounds for numerical primitives such as linear elasticity problems\(^{25}\) and positive linear programs,\(^{10}\) One formalization of such hardness is the Sparse Linear Equation Approximation Hypothesis (SLTH) from.\(^{19}\) SLTH\(^{\omega}\) denotes the assumption that a sparse linear system with $\text{nnz}(A) = \omega$ cannot be solved in time faster than $\text{nnz}(A)^{\omega}$ to within relative error $\epsilon = n^{-10}$. Here, improving over the smaller running time of both direct and iterative methods can be succinctly encapsulated as refuting SLTH\(^{\omega}\).\(^{19}\)

We provide a faster algorithm for solving sparse linear systems. Our formal result is the following (we use the form defined in Kyng et al\(^{10}\) Linear Equation Approximation Problem, LEA).

**Theorem 1.** Given a matrix $A$ with maximum dimension $n$, $\text{nnz}(A)$ non-zeros (whose values fit into a single word), along with a parameter $k(A)$ such that $k(A) \geq \sigma_{\text{max}}(A)/\sigma_{\text{min}}(A)$, a vector $b$ and error requirement $\epsilon$, we can compute, under fixed point arithmetic, in time

$$O(\max \{ \text{nnz}(A)^{\omega} n, n^{\omega+1} \} \log (\kappa/\epsilon))$$

a vector $x$ such that

$$\|Ax - \Pi_A b\|_2 \leq \epsilon \left\| \Pi_A b \right\|_2,$$

where $\epsilon$ is a fixed constant and $\Pi_A$ is the projection operator onto the column space of $A$.

Note that $\| \Pi_A b \|_2 = \| A^T b \|_{\text{vec}(A)}$ and when $A$ is square and full rank, it is just $\| b \|_2$. The cross-over point for the two bounds is at $\text{nnz}(A) = n^{\omega+1}$. In particular, for the sparse case with $\text{nnz}(A) = \text{nnz}(n)$, and the bound of $\omega \leq 2.38$, we get an exponent of

$$\max \left\{ 2 + \frac{\omega - 2}{\omega - 1}, \frac{5\omega - 4}{\omega + 1} \right\} < \max \left\{ 2.28, 2.34 \right\} = 2.34.$$

As $n \leq \text{nnz}$, this also translates to a running time of $O(n^{\omega+1})$, which as $\frac{\text{nnz}}{n^{\omega+1}} = \omega - \frac{\omega - 2}{\omega + 1}$, refutes SLTH\(^{\omega}\) for constant values of $\omega$ and any value of $\omega > 2$.

We can parameterize the asymptotic gain over matrix multiplication for moderately sparse instances. Here we use the $O(\cdot)$ notation to hide lower-order terms, specifically $O(f(n))$ denotes $O(f(n) \cdot \log^a(f(n)))$ for some absolute constant $c$.

**Corollary 2.** For any matrix $A$ with dimension at most $n$, $O(n^{\omega+1})$ non-zeros, and condition number $n^{\omega+1}$, a linear system in $A$ can be solved to accuracy $n^{-\omega+1}$ in time $O(\max \{ n^{\omega+1}, n^{\omega+1} \})$.

Here the cross-over point happens at $\omega = \theta - \frac{1}{\epsilon}$. Also, because $\frac{\text{nnz}}{n^{\omega+1}} = \omega - \frac{\omega - 2}{\omega + 1}$, we can also infer that for any $0 < \theta \leq \omega - 2$ and any $\omega > 2$, the runtime is $O(n^{\theta})$, or asymptotically faster than matrix multiplication.

**1.1 Idea**

At a high level, our algorithm follows the block Krylov space method (see, e.g., Chapter 6.12 of Saad\(^{16}\)). This method is a multi-vector extension of the CG/Lanczos method, which in the single-vector setting is known to be problematic under round-off errors both in theory\(^{12}\) and in practice.\(^{16}\) Our algorithm starts with a set of $s$ initial vectors, $B \in \mathbb{R}^{n \times s}$, and forms a column space by multiplying these vectors by $A$ repeatedly, $m$ times. Formally, the block Krylov space matrix is

$$K = [B | AB | A^2B | \ldots | A^{m-1}B].$$

The core idea of Krylov space methods is to efficiently orthogonalize this column space. For this space to be spanning, block Krylov space methods typically choose $s$ and $m$ so that $sm = n$.

The conjugate gradient algorithm can be viewed as an efficient implementation of the case $s = 1$, $m = n$, with $B$ set to $b$, the RHS of the input linear system. The block case with larger values of $s$ was studied by Eberly, Giesbrecht, Giorgi, Storjohann, and Villard\(^{1}\) over finite fields, and they gave an $O(n^{2.38})$ time\(^{d}\) algorithm for computing the inverse of an $O(n)$-sparse matrix over a finite field.

Our algorithm also leverages the top-level insight of the Eberly et al. results: the Gram matrix of the Krylov space matrix, is a block Hankel matrix. Solving linear systems in this Gram matrix, $(AK)^T(AK)$, lead to solvers for linear systems in $A$ because

$$(AK)^T(AK)^{-1} = (K^T A^T AK)^{-1} = K^T A^{-1} A^T K^T,$$

so as long as $A$ and $K$ are both invertible, composing this on the left by $K$ and on the right by $A$ gives

$$A^{-1} = K((AK)^T(AK))^{-1} A^T K^T.$$
Eberly et al. viewed the Gram matrix as an $m$-by-$m$ matrix containing $s$-by-$s$ sized blocks, and critically leveraged the fact that the blocks along each anti-diagonal are identical:

$$\begin{align*}
(AK)^T(AK) &= \\
&= \begin{bmatrix}
B^n A^2 B & B^2 A^2 B & B^2 A^4 B & \cdots & B^2 A^{m+1} B \\
B^n A^2 B & B^2 A^2 B & B^2 A^4 B & \cdots & B^2 A^{m+1} B \\
B^n A^2 B & B^2 A^2 B & B^2 A^4 B & \cdots & B^2 A^{m+1} B \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
B^n A^2 B & B^2 A^2 B & B^2 A^4 B & \cdots & B^2 A^{m+1} B \\
\end{bmatrix}
\end{align*}$$

Formally, the $s$-by-$s$ inner product matrix formed from $A^T B$ and $A^T B$ is $B^T A^T B$, and depends only on $i + j$. So instead of $m^2$ blocks each of size $s \times s$, we are able to represent an $n$-by-$n$ matrix with only about $m$ blocks.

Operations involving these $m$ blocks of the Hankel matrix can be handled using $O(m)$ block operations. This is perhaps easiest seen for computing matrix-vector products using $K$. If we use $\{i\}$ to denote the $i^{th}$ block of the Hankel matrix, and define

$$H_{\{ij\}} = M(i + j)$$

for a sequence of matrices $M$, we get that the $i^{th}$ block of the product $Hx$ can be written in block-form as

$$(Hx)_{\{i\}} = \sum H_{\{ij\}} x_{\{j\}} = \sum M(i + j) x_{\{j\}}.$$

Note this is precisely the convolution of (a sub-interval) of $M$ and $x$, with shifts indicated by $i$. Therefore, in matrix-vector multiplication (the “forward” direction), a speedup by a factor of about $m$ is possible with fast convolution algorithms. The performance gains of the Eberly et al. algorithms$^5$ can be viewed as being of a similar nature, albeit in the more difficult direction of solving linear systems. Specifically, they utilize algorithms for the Padé problem of computing a polynomial from the result of its convolution.$^4$ Over finite fields, or under exact arithmetic, such algorithms for matrix Padé problems take $O(m \log m)$ block operations,$^4$ for a total of $O(K(s^m))$ operations.

The overall time complexity is based on two opposing goals:

1. Quickly generate the Krylov space: repeated multiplication by $A$ allows us to generate $A^T B$ using $O(ms \cdot \text{nnz}) = O(n \cdot \text{nnz})$ arithmetic operations. Choosing a sparse $B$ then allows us to compute $B^T A^T B$ in $O(n \cdot s)$ arithmetic operations, for a total overhead of $O(n^2)$.2

2. Quickly invert the Hankel matrix. Each operation on an $s$-by-$s$ block takes $O(s^3)$ time. Under the optimistic assumption of $O(m)$ block operations, the total is $O(m \cdot s^3)$. Under these assumptions, and the requirement of $n = ms$, the total cost becomes $O(n \cdot \text{nnz} + m \cdot s^3)$, which is at most $O(n \cdot \text{nnz})$ as long as $m > n^2$. However, this runtime complexity is over finite fields, where numerical stability is not an issue. Over the reals, under round-off errors, one must contend with numerical errors without blowing up the bit complexity. This is a formidable challenge; indeed, as mentioned earlier, with exact arithmetic, the CG method takes time $O(n \cdot \text{nnz})$, but this is misleading since the computation is effective only when the word sizes are increased by a factor of $n$ (to about $n \log \kappa$ words), which leads to an overall complexity of $O(n^2 \cdot \text{nnz} \cdot \log \kappa)$.

### 1.2 Our Contributions

Our algorithm can be viewed as the numerical generalization of the algorithms from.$^5$ We work with real numbers of bounded precision, instead of entries over a finite field. The core of our approach can be summarized as follows.

The block Krylov space method together with fast Hankel/Toeplitz matrices. Many of the prior algorithms rely on algebraic identities that do not generalize to the block setting, and are often (experimentally) numerically unstable.$^7$

2. Develop matrix anti-concentration bounds for analyzing the word lengths of inverses of random Krylov spaces. This is to upper bound the probability of random matrices being in some set of small measure, which in our case is the set of nearly singular matrices. Previously, such bounds were known assuming the matrix entries are independent$^{18,21}$ but Krylov matrices have correlated columns.

Before we describe the difficulties and new tools needed, we first provide some intuition on why a factor $m$ increase in word lengths may be the right answer by upper-bounding the magnitudes of entries in an $m$-step Krylov space. By rescaling, we may assume that the minimum singular value of $A$ is at least $1/\kappa$, and the maximum entry in $A$ is at most $\kappa$. The maximum magnitude of (entries of) $A^m b$ is bounded by the maximum magnitude of $A$ to the power of $m$, times a factor corresponding to the number of summands in the matrix product:

$$\left| A^m b \right| \leq n^m |A| |b| \leq (nk)^m \cdot |b| \leq \kappa^m |b|.$$

Here the last inequality is via the assumption of $\kappa \geq n$. So by forming $K$ via horizontally concatenating $A^T g$ for sparse Gaussian vectors $g$, we have with high probability that the maximum magnitude of an entry of $K$, and in turn $AK$, is at most $\kappa^m$. In other words, $O(m \log \kappa)$ words in front of the decimal point is sufficient with high probability.

Should such a bound of $O(m \log \kappa)$ hold for all numbers that arise in the algorithm, including the matrix inversion steps, and the matrix $B$ is sparse with $O(n)$ entries, the cost of computing the block Krylov matrices becomes $O(m \log \kappa \cdot s^2 \cdot \text{nnz})$, while the cost of the matrix inversion portion encounters an overhead of $O(m \log \kappa)$, for a total of $O(m^2 s^3 \log \kappa)$. In the sparse case of $\text{nnz} = O(n)$, and $n = ms$, this becomes:

$$O(n^2 m \log \kappa + m^2 s^3 \log \kappa) = O(n^2 m \log \kappa + n^w \cdot \log \kappa). \quad (1)$$

Due to the gap between $n^2$ and $n^w$, setting $m$ appropriately gives improvement over $n^w$ when $\log \kappa = n^{w/2}$.
However, the magnitude of an entry in the inverse depends on the smallest magnitude, or in the matrix case, its minimum singular value. Bounding and propagating the min singular value, which intuitively corresponds to how close a matrix is to being degenerate, represents our main challenge. In exact/finite fields settings, non-degeneracies are certified via the Schwartz-Zippel lemma about polynomial roots. The numerical analog of this is more difficult—the Krylov space matrix \( K \) is asymmetric, even for a symmetric matrix \( A \). It is much easier for an asymmetric matrix with correlated entries to be close to singular.

Consider for example a two-banded, two-block matrix with all diagonal entries set to the same random variable \( \alpha \) (see Figure 1):

\[
A_{ij} = \begin{cases} 
1 & \text{if } i = j \text{ and } j \leq n/2, \\
\alpha & \text{if } i = j + 1 \text{ and } j < n/2, \\
\alpha & \text{if } i = j + 1 \text{ and } n/2 < j, \\
2 & \text{if } i = j + 1 \text{ and } n/2 < j, \\
0 & \text{otherwise.}
\end{cases}
\]

In the exact case, this matrix is full rank unless \( \alpha = 0 \), even over finite fields. On the other hand, its minimum singular value is close to 0 for all values of \( \alpha \). To see this, it’s useful to first make the following observation about the min singular value of one of the blocks.

**Observation 3.** The minimum singular value of a matrix with 1s on the diagonal, \( \alpha \) on the entries immediately below the diagonal, and 0 everywhere else is at most \( |\alpha|^{-n-1} \), due to the test vector \([1; -\alpha; \alpha^2; \ldots; (-\alpha)^{n+1}]\).

Then, in the top-left block, as long as \( |\alpha| > 3/2 \), the top left block has minimum singular value at most \( (2/3)^{n+1} \). On the other hand, rescaling the bottom-right block by \( 1/\alpha \) to get 1s on the diagonal gives \( 2/\alpha \) on the off-diagonal. So as long as \( |\alpha| < 3/2 \), this value is at least \( 4/3 \), which in turn implies a minimum singular value of at most \( (3/4)^{n+1} \) in the bottom right block. This means no matter what value \( \alpha \) is set to, this matrix will always have a singular value that’s exponentially close to 0. Furthermore, the Gram matrix of this matrix also gives such a counter example to symmetric matrices with (non-linearly) correlated entries. Previous works on analyzing condition numbers of asymmetric matrices also encounter similar difficulties; a more detailed discussion of it can be found in Section 7 of Sankar et al.

To bound the bit complexity of all intermediate steps of the block Krylov algorithm by \( O(m \log k) \), we devise a more numerically stable algorithm for solving block Hankel matrices, as well as provide a new perturbation scheme to quickly generate a well-conditioned block Krylov space. Central to both of our key components is the close connection between condition number and bit complexity bounds.

First, we give a more numerically stable solver for block Hankel/Toeplitz matrices. Fast solvers for Hankel (and closely related Toeplitz) matrices have been extensively studied in numerical analysis, with several recent developments on more stable algorithms.24 However, the notion of numerical stability studied in these algorithms is the variant where the number of bits of precision is fixed. Our attempts at converting these into asymptotic bounds yielded dependencies quadratic in the number of digits in the condition number, which in our setting translates to a prohibitive cost of \( \tilde{O}(m^2) \) (that is, the overall cost would be higher than \( n^\omega \)).

Instead, we combine developments in recursive block Gaussian elimination with the low displacement rank representation of Hankel/Toeplitz matrices. Such representations allow us to implicitly express both the Hankel matrix and its inverse by displaced versions of low-rank matrices. This means the intermediate size of instances arising from recursion is \( O(s) \) times the dimension, for a total size of \( O(n \log n) \), giving a total of \( \tilde{O}(ns^{\omega - 1}) \) arithmetic operations involving words of size \( \tilde{O}(m) \). We provide a rigorous analysis of the accumulation of round-off errors similar to the analysis of recursive matrix multiplication based matrix inversion from.4

Motivated by this close connection with the condition number of Hankel matrices, we then try to initialize with Krylov spaces of low condition number. Here we show that a sufficiently small perturbation suffices for producing a well conditioned overall matrix. In fact, the first step of our proof, showing that a small sparse random perturbation to \( A \) guarantees good separations between its eigenvalues, is a direct combination of bounds on eigenvalue separation of random Gaussians as well as min eigenvalue of random sparse matrices.11 This separation then ensures that the powers of \( A, A^2, \ldots, A^m \), are sufficiently distinguishable from each other. Such considerations also come up in the smoothed analysis of numerical algorithms.

The randomness of the Krylov matrix induced by the initial set of random vectors \( B \) is more difficult to analyze: each column of \( B \) affects \( m \) columns of the overall Krylov space matrix. In contrast, all existing analyses of lower bounds of singular values of possibly asymmetric ran-
dom matrices\textsuperscript{16,21} rely on the randomness in the columns of matrices being independent. The dependence between columns necessitates analyzing singular values of random linear combinations of matrices, which we handle by adapting \( \epsilon \)-net based proofs of anti-concentration bounds. Here we encounter an additional challenge in bounding the minimum singular value of the block Krylov matrix. We resolve this issue algorithmically: instead of picking a Krylov space that spans the entire \( \mathbb{R}^n \), we stop short by picking \( ms = n - \Omega(m) \). This set of extra columns significantly simplifies the proof of singular value lower bounds. This is similar in spirit to the analysis of the minimum singular value of a random matrix, which is easier for a non-square matrix.\textsuperscript{15} In the algorithm, the remaining columns are treated as a separate block that we handle via a Schur complement at the very end of the algorithm. Since this block is small, so is its overhead on the running time.

1.3 History and Related Work

Our algorithm has close connections with multiple lines of research on efficient solvers for sparse linear systems. The topic of efficiently solving linear systems has been extensively studied in computer science, applied mathematics and engineering. For example, in the Society of Industrial and Applied Mathematics News ‘top 10 algorithms of the 20\textsuperscript{th} century’, three of them (Krylov space methods, matrix decompositions, and QR factorizations) are directly related to linear systems solvers.\textsuperscript{3}

At a high level, our algorithm is a hybrid linear systems solver. It combines iterative methods, namely block Krylov space methods, with direct methods that factorize the resulting Gram matrix of the Krylov space. Hybrid methods have their origins in the incomplete Cholesky method for speeding up elimination/factorization based direct solvers. A main goal of these methods is to reduce the \( \Omega(n^2) \) space needed to represent matrix factorizations/inverses. This high space requirement is often even more problematic than time requirements when handling large sparse matrices. Such reductions can occur in two ways: either by directly dropping entries from the (intermediate) matrices, or by providing more succinct representations of these matrices using additional structure.

The main structure of our algorithm is based on the latter line of work on solvers for structured matrices. Such systems arise from physical processes where the interactions between objects have invariances (e.g., either by time or space differences). Examples of such structure include circulant matrices, Hankel/Toeplitz matrices and distances from \( n \)-body simulations.\textsuperscript{7} Many such algorithms require exact preservation of the structure in intermediate steps. As a result, many of these works develop algorithms over finite fields.

More recently, there has been work on developing numerically stable variants of these algorithms for structured matrices, or more generally, matrices that are numerically close to being structured.\textsuperscript{24} However, these results only explicitly discussed in the entry-wise Hankel/Toeplitz case (which corresponds to block size \( s = 1 \)). Furthermore, because they rely on domain-decomposition techniques similar to fast multiple methods, they produce one bit of precision for each outer iteration loop. As the Krylov space matrix has condition number \( \exp(\Omega(m)) \), such methods would lead to another factor of \( m \) in the solve cost if invoked directly.

Instead, our techniques for handling and bounding numerical errors are more closely related to recent developments in provably efficient sparse Cholesky factorizations.\textsuperscript{9} These methods generate efficient preconditioners using only the condition that intermediate steps of Gaussian elimination, known as Schur complements, have small representations. They avoid the explicit generation of the dense representations of Schur complements by treating them as operators, and apply randomized tools to directly sample/sketch the final succinct representations, which have much smaller size and algorithmic cost.

On the other hand, previous works on sparse Cholesky factorizations required the input matrix to be decomposable into a sum of simple elements, often through additional combinatorial structure of the matrices. In particular, this line of work on combinatorial preconditioning was initiated through a focus on graph Laplacians, which are built from 2-by-2 matrix blocks corresponding to edges of undirected graphs.\textsuperscript{19} Since then, there have been substantial generalizations to the structures amenable to such approaches, notably to finite element matrices, and directed graphs/irreversible Markov chains. However, recent works have also shown that many classes of structures involving more than two variables are complete for general linear systems.\textsuperscript{25} Nonetheless, the prevalence of approximation errors in such algorithms led to the development of new ways to bound numerical round-off errors in algorithms, which will be critical to our elimination routine for block-Hankel matrices.

Key to recent developments in combinatorial preconditioning is matrix concentration.\textsuperscript{22} Such bounds provide guarantees for (relative) eigenvalues of random sums of matrices. For generating preconditioners, such randomness arises from whether each element is kept or not, and a small condition number (which in turn implies a small number of outer iterations using the preconditioner) corresponds to a small deviation between the original and sampled matrices. In contrast, we introduce randomness in order to obtain block Krylov spaces whose minimum eigenvalue is large. As a result, the matrix tool we need is anti-concentration, which somewhat surprisingly is far less studied. Previous works on it are related to similar problems of numerical precision\textsuperscript{16,21} and address situations where the entries in the resulting matrix are independent. Our bound on the min singular value of the random Krylov space also yields a crude bound for a sum of rectangular random matrices.

1.4 Subsequent Improvements and Extensions

Nie\textsuperscript{24} gave a more general and tighter version of matrix anti-concentration that also works for square matrices, answering an open question we posed. For an \( m \)-step Krylov space instantiated using \( \mathbb{R}^n \) vectors, Nie’s bound reduces the middle term in our analysis from \( \Omega(n^2 m) \) to \( \Omega(n^m) \), thus leading to a running time of \( nmz \cdot n \cdot m + n^a \cdot m^2 \), which match-
es the bound for finite fields. Moreover, it does so without the padding step at the end. We elect to keep for this article our epsilon-net based analyses, and the padded algorithm required for it, both for completeness, and due to it being a more elementary approach toward the problem with a simpler proof.

Faster matrix multiplication is an active area of work with recent progresses. Due to the dependence on fast multiplication in our algorithm, such improvements also lead to improvements in the running time of solving sparse systems as well.

Our main result for solving sparse linear systems has also been extended to solving sparse regression, with faster than matrix multiplication bounds for sufficiently sparse matrices. The complexity of sparse linear programming remains an interesting open problem.

2. ALGORITHM

We describe the algorithm, as well as the running times of its main components in this section. To simplify the discussion, we assume the input matrix $A$ is symmetric, and has polyn($n$) condition number. If it is asymmetric (but invertible), we implicitly apply the algorithm to $A^T A$, using the identity $A^{-1} = (A^T A)^{-1} A^T$ derived from $(A^T A)^{-1} = A^{-1} A^{-T}$. Also, recall from the discussion after Theorem 1 that we use $\hat{O}(\cdot)$ to hide logarithmic terms in order to simplify runtimes.

Before giving details of our algorithm, we first discuss what constitutes a linear system solver algorithm, specifically the equivalence between many such algorithms and linear operators.

For an algorithm $\text{ALG}$ that takes a matrix $B$ as input, we say that $\text{ALG}$ is linear if there is a matrix $Z_{\text{ALG}}$ such that for any input $B$, the output of running the algorithm on $B$ is the same as multiplying $B$ by $Z_{\text{ALG}}$:

$$\text{ALG}(B) = Z_{\text{ALG}} B.$$ 

In this section, in particular in the pseudocode in Algorithm 2, we use the name of the procedure, $\text{SOLVE}(b, \delta)$, interchangeably with the operator corresponding to a linear algorithm that solves a system in $A$, on vector $b$, to error $\delta > 0$. In the more formal analysis, we will denote such corresponding linear operators using the symbol $Z$, with subscripts corresponding to the routine if appropriate.

This operator/matrix-based analysis of algorithms was first introduced in the analysis of a recursive Chebyshev iteration by Spielman and Teng, with credits to the technique also attributed to V. Rokhlin. It has the advantage of simplifying the analysis of multiple iterations of such algorithms, as we can directly measure Frobenius norm differences between such operators and the exact ones that they approximate.

Under this correspondence, the goal of producing an algorithm that solves $Ax = b$ for any $b$ as input becomes equivalent to producing a linear operator $Z_A$ that approximates $A^{-1}$, and then running it on the input $b$. For convenience, we also let the solver take as input a matrix instead of a vector, in which case the output is the result of solves against each of the columns of the input matrix as the RHS.

The high-level description of our algorithm is in Figure 2. Some of the steps of the algorithm require care for efficiency as well as for tracking the number of words needed to represent the numbers. We assume a bound on bit complex-

---

**Algorithm 2**

**Figure 2. Pseudocode for block Krylov space algorithm: Solve($\cdot, \cdot$) are operators corresponding to linear system solving algorithms whose formalization we discuss at the start of this section.**

- **FORM KRYLOV SPACE**
  1. Set $s \leftarrow \lceil n/m \rceil \cdot O(m)$, $h \leftarrow (m^2 \log 1/\varepsilon_{\text{ms}})$.
  2. Let $G'$ be an $n \times s$ random matrix sampled from $\mathcal{N}(0, 1)$ with probability $1/2$ and 0 otherwise.
  3. Use MatVec($G'$, $x$) to compute $K = G' A G' T A^{-1} G'$. (a) Set $s \leftarrow \lceil n/m \rceil \cdot O(m)$. 
  
- **SPARSE INVERSE**
  1. Use fast solvers for block Hankel matrices to obtain a solver for the matrix $M = (AK)^T (AK)$, to error \( \varepsilon_{\text{ms}} \). 
  2. (PAD AND SOLVE)
  3. (a) Generate an $n \times (n - m s)$ dense Gaussian matrix $G$, use it to complete the basis as: 
  4. (b) Compute the Schur complement of $(AQ)^T$ 
  5. (c) Use the inverse of this Schur complement, as well as $\text{SOLVE}_{\text{ms}}$ to obtain a solver for $Q^T Q$, $\text{SOLVE}_{\text{ms}}(\cdot, \varepsilon)$. 
  6. Return $Q \cdot \text{SOLVE}_{\text{ms}}(\cdot, \varepsilon)$ as an approximate solver for $A$. 

---

**Figure 3. Randomized $m$-step Krylov Space Matrix with $n$-by-$s$ sparse Gaussian $G$ as starter.**

- $n$ 
- $G^s$ 
- $A^G^s$ 
- $A^G^s$ 
- $\cdots$ 
- $A^G^s$
ity of $O(m)$ when $\kappa = \text{poly}(n)$ in the brief description of costs in the outline of the steps below.

We start by perturbing the input matrix, resulting in a symmetric positive definite matrix where all eigenvalues are separated by $\alpha$. Then we explicitly form a Krylov matrix from a sparse random Gaussian matrix, see Figure 3. For any vector $u$, we can compute $A^i u$ from $A^{i-1} u$ via a single matrix-vector multiplication in $A$. So computing each column of $K$ requires $O(nnz(A))$ operations, each involving a length $n$ vector with words of length $O(m)$. So we get the matrix $K$, as well as $AK$, in time

$$O(nnz(A) \cdot n \cdot m).$$

To obtain a solver for $AK$, we instead solve its Gram matrix $(AK)^T(AK)$. Each block of $K^i K$ has the form $(G^i)^T A^i G^i$ for some $2 \leq i \leq 2m$, and can be computed by multiplying $(G^i)^T$ and $A^i G^i$. As $A^i G^i$ is an $n$-by-$s$ matrix, each non-zero in $G^i$ leads to a cost of $O(s)$ operations involving words of length $O(m)$. Then because we chose $G^i$ to have $O(m^2)$ non-zeroes per column, the total number of non-zeroes in $G^i$ is about $O(s \cdot m^2) = O(n m^2)$. This leads to a total cost (across the $m$ values of $i$) of:

$$O(n^2 m^3).$$

The key step is then Step 2, a block version of the Conjugate Gradient method. It will be implemented using a recursive data structure based on the notion of displacement rank. To get a sense of why a faster algorithm may be possible, note that there are only $O(m)$ distinct blocks in the matrix $(AK)^T(AK)$. So a natural hope is to invert these blocks by themselves; the cost of (stable) matrix inversion, of course, it does not suffice to solve these $m$ $s$-by-$s$ blocks independently. Instead, the full algorithm, as well as the $\text{SOLVE}_\mu$ operator, is built from efficiently convolving such $s$-by-$s$ blocks with matrices using Fast Fourier Transforms. Such ideas can be traced back to the development of super-resolution tools\cite{10.1145/307295.307310,10.1145/2997719.2997720}, namely lower bounding the inner product of a random vector against any vector. Thus, we can lower bound the minimum singular value of $Q$, and in turn $AQ$, by exp $(-O(m))$.

This bound in turn translates to a lower bound on the minimum eigenvalue of the Gram matrix of $AQ$, $(AQ)^T(AQ)$. Partitioning its entries by those from $K$ and $G$ gives four blocks: one $(sm)$-by-$(sm)$ block corresponding to $(AK)^T(AK)$, one $\Theta(m)$-by-$\Theta(m)$ block corresponding to $(AG)^T(AG)$, and then the cross terms. To solve this matrix, we apply block-Gaussian elimination, or equivalently, form the Schur complement onto the $\Theta(m)$-by-$\Theta(m)$ corresponding to the columns in $AG$.

To compute this Schur complement, it suffices to solve the top-left block (corresponding to $(AK)^T(AK)$) against every column in the cross term. As there are at most $\Theta(m) < s$ columns, this solve cost comes out to less than $O(s^2 m)$ as well. We are then left with a $\Theta(m)$-by-$\Theta(m)$ matrix, whose solve cost is a lower order term.

So the final solver costs

$$O(nnz(A) \cdot nm + n^2 m^3 + n^s m^2 \cdot \omega)$$

which leads to the final running time by choosing $m$ to balance the terms. This bound falls short of the ideal case given in Equation 1 mainly due to the need for a denser $B$ to the well-conditionedness of the Krylov space matrix. Instead of $O(m)$ non-zeroes total, or about $O(m)$ per column, we need $\text{poly}(m)$ non-zero variables per column to ensure the an $O(m)$ condition number of the block Krylov space matrix $K$. This in turn leads to a total cost of $O(n \cdot nnz \cdot \text{poly}(m))$ for computing the blocks of the Hankel matrix, and a worse tradeoff when summed against the $\Theta(m)$ term.

**Acknowledgments**

Richard Peng was supported in part by NSF CAREER award.
research highlights

1846218/2330255, and Santosh Vempala by NSF awards AF-1909756 and AF-2007443. We thank Mark Giesbrecht for bringing to our attention the literature on block-Krylov space algorithms.

References

1. Beckermann, B. and Labahn, G. A uniform approach for the fast computation of matrix-type Padé approximants. SIAM J. on Matrix Analysis and Applications 15, 3 (1994), 804–823.
2. Blum, L. Computing over the reals: Where Turing meets Newton. Notices of the AMS 52, 9 (2004), 1024–1034.
3. Cipra, B.A. The best of the 20 century: Editors name top 10 algorithms. SIAM News 33, 4 (2000), 1–2; https://archive.siam.org/pdf/news/837.pdf.
4. Demmel, J., Dumitriu, I., Holtz, O., and Kleinberg, R. Fast matrix multiplication is stable. Numerische Mathematik 106, 2 (2007), 189–224.
5. Eberly, W. et al. Faster inversion and other black box matrix computations using efficient block projections. In Symbolic and Algebraic Computation, Intern. Symp., ISSAC 2007, Waterloo, Ontario, Canada, July 28 - August 1, 2007, Proceedings, 2007, 143–150.
6. Ghadiri, M., Peng, R., and Vempala, S.S. The bit complexity of other black box matrix computations have simple spectrum. arXiv preprint arXiv:1802.03662, 2018.
7. Gross, C., Musco, C., and Sidford, A. Stability of the Lanczos method for matrix function approximation. In Proceedings of the 29th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, 1605–1624; https://arxiv.org/abs/1708.07788.
8. Kyng, R., Wang, D., and Zhang, P. Packing LPs are hard to solve accurately, assuming linear equations are hard. In Proceedings of the 2020 ACM-SIAM Symp. on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5–8, 2020. S. Chawla, Ed. SIAM, 2020. 279–296; https://dl.acm.org/doi/pdf/10.1137/19.m1208108. 3831106.
9. Kyng, R. Approximate Gaussian Elimination. Ph.D. thesis, Yale University, 2017; http://rasmuskyng.com/rykyng-dissertation.pdf.
10. Kyng, R., Wang, D., and Zhang, P. Packing LPs are hard to solve accurately, assuming linear equations are hard. In Proceedings of the 2020 ACM-SIAM Symp. on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5–8, 2020. S. Chawla, Ed. SIAM, 2020. 279–296; https://dl.acm.org/doi/pdf/10.1137/19.m1208108. 3831106.
11. Luh, K. and Vu, V. Sparse random matrices have simple spectrum. arXiv preprint arXiv:1802.03662, 2018.
12. Musco, C. and Musco, C. Stability of the Lanczos method for matrix function approximation. In Proceedings of the 29th Annual ACM-SIAM Symp. on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018, 1605–1624; https://arxiv.org/abs/1708.07788.
13. Nguyen, H., Tao, T., and Vu, V. Random matrices: Tail bounds for gaps between eigenvalues. Probability Theory and Related Fields 167, 3-4, (2017), 777–818.
14. Nie, Z. Matrix anti-concentration inequalities with applications. In STOC '22: 54th Annual ACM SIGACT Symp. on Theory of Computing, Rome, Italy, June 20–24, 2022. S. Leonardo and A. Gupta (eds). ACM, (2022), 568–581; https://arxiv.org/abs/2111.05553.
15. Rudelson, M. and Vershynin, R. Non-asymptotic theory of random matrices: Extreme singular values. In Proceedings of the Intern. Congress of Mathematicians 2010 (2 Vol.), (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures. World Scientific, 1578–1602.
16. Saad, Y. Iterative Methods for Sparse Linear Systems, 2nd ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003; http://www-users.cs.umn.edu/~saad/saadtoc.pdf.
17. Sachdeva, S. and Vishnoi, N.K. Faster algorithms via approximation theory. Theoretical Computer Science 9, 2 (2013), 125–210.
18. Sankar, A., Spielman, D.A., and Teng, S. Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. SIAM J. on Matrix Analysis and Applications 35, 3 (2014), 835–885; http://arxiv.org/abs/cs/0707.1093.
19. Stojanovic, A. The shifted number system for fast linear algebra on integer matrices. J. of Complexity 21, 4 (2005), 609–630; https://cs.uwaterloo.ca/~astojn/shifted.pdf.
20. Tao, T. and Vu, H.V. Smooth analysis of the condition number and the least singular value. Math. Comput. 79, 272 (2010), 2333–2352; https://arxiv.org/abs/0805.3317.
21. Tropp, J.A. An introduction to matrix concentration inequalities. Found. Trends Mach. Learn. 8, 1-2, (2015), 1–220.
22. Woodruff, D.P. Sketching as a tool for numerical linear algebra. Foundations and Trends in Theoretical Computer Science 10, 1-2, (2014), 1–157; https://arxiv.org/abs/1411.4357.
23. Xi, Y., Xia, J., Cauley, S., and Balakrishnan, V. Superfast and stable structured solvers for toeplitz least squares via randomized sampling. SIAM J. on Matrix Analysis and Applications 35, 1 (2014), 44–72.
24. Zhang, P. Hardness and Tractability For Structured Numerical Problems. Ph.D. thesis, Georgia Institute of Technology, 2018.

Richard Peng (peng@cs.cmu.edu), Carnegie Mellon University, Pittsburgh, PA, USA. Work in this article done while at Georgia Institute of Technology, Atlanta, GA, USA.

Santosh S. Vempala (vempala@gatech.edu), Georgia Institute of Technology, Atlanta, GA, USA.

© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.