

Randomized Algorithms for Low-Rank Matrix Decomposition

Benjamin J. Sapp

Written Preliminary Examination II

Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19103

May 6, 2011

Abstract

Low-rank matrix factorization is one of the most useful tools in scientific computing and data analysis. The goal of low-rank factorization is to decompose a matrix into a product of two smaller matrices of lower rank that approximates the original matrix well. Such a decomposition exposes the low-rank structure of the data, requires less storage, and subsequent matrix operations require less computation. Classical methods for low-rank factorization of an $m \times n$ matrix into two matrices of rank k are $O(mnk)$ and require at least k passes through the input matrix and assume random access. In modern applications, we deal more and more with matrices so large that accessing the data becomes the computational bottleneck, and the classical methods become unsuitable.

In this literature review, we discuss several randomized methods for computing approximations to the best low-rank factorization, in only 1 or 2 passes over the input matrix. The remaining computation is polynomial in the desired rank k and at most sublinear in the original dimensions $m \times n$. These methods provide a clear tradeoff between accuracy and computation time via the degree of sampling involved. We discuss the running time and bounds on the expected approximation error for each method. Finally, we present a unified framework for all methods, and compare them in terms of the computation-accuracy tradeoff, data access requirements and parallelizability.

Primary sources for this WPE II

- N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, September 2009.
(Halko et al., 2009)
- A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292—303, 2006.
(Deshpande and Vempala, 2006)
- A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025—1041, 2004.
(Frieze et al., 2004)

Contents

1	Introduction	5
1.1	The Classical Solution and Underlying Issues	5
1.2	Random Techniques	7
1.3	Methods for Comparison	8
2	Related work	10
3	Definitions and Preliminaries	11
3.1	Notation	11
3.2	The Singular Value Decomposition	12
3.3	Orthogonal Projections	12
3.4	Norms	13
3.5	The Optimal Low-Rank Decomposition	13
3.5.1	Exact methods for Optimal Decomposition	14
4	Randomly subsampling <i>a priori</i>:	
	SAMPLECOLS, SAMPLEROWSCOLS	17
4.1	A First Attempt: subsampling columns only	17
4.1.1	Analysis	19
4.2	“Constant-time SVD”: subsampling rows and columns	24
4.2.1	Analysis	24
5	Adaptive subsampling: ADASAMPLE	27
5.1	Analysis	28
6	Randomly projecting to lower dimensions: RANDPROJ	33
6.1	Analysis	33
6.2	Refinements	35
7	Comparison of methods	37
7.1	Running time versus expected error	37
7.2	Passes through the data	38
7.3	Parallelizability	39
8	Conclusion	40

1 Introduction

In real-world settings, often we deal with data stored in an $m \times n$ matrix X which we would like to decompose into a product of two matrices of lower rank $k < \min\{m, n\}$:

$$A \underset{m \times n}{\approx} \underset{m \times k}{B} \underset{k \times n}{C} \quad (1.1)$$

Such a decomposition lends itself to a multitude of practical applications which cover almost every field of engineering. To name a few in computer science, statistics and machine learning: least squares regression, principal component analysis, spectral clustering, and linear discriminative analysis all rely on a low-rank decomposition.

Oftentimes, the decomposition exposes the underlying structure of the data which we are trying to discover. A universal benefit of such a low-rank decomposition is that fewer elements are required to represent the matrix ($km + kn$ versus mn), requiring less storage and less operations to perform matrix-matrix or matrix-vector multiplication. Furthermore, determining A and B brings us only a few, computationally-efficient steps away from obtaining standard decompositions such as the QR decomposition, the Singular Value Decomposition (SVD) and eigendecomposition of $X^T X$ and XX^T .

In this paper, we concern ourselves with the *fixed-rank approximation problem*: find a matrix D of rank at most k (assumed known or given) that represents X well. The general optimization problem is thus posed as

$$\min_{\text{rank}(A') \leq k} \|A - A'\|^2 \quad (1.2)$$

where $\|\cdot\|$ is either the operator norm $\|\cdot\|_2$ or Frobenius norm $\|\cdot\|_F$, see §3 for definitions. All optimization methods for solving Equation 1.2 explicitly find matrices B and C such that $A' = \underset{m \times n}{B} \underset{m \times k}{C}$; thus A' never needs to be explicitly formed.

1.1 The Classical Solution and Underlying Issues

It is well known that Equation 1.2 has a global optimum, and we can construct a minimizer $A_k = \sum_{i=1}^k u_i u_i^T A$, where u_i are the top k left singular vectors of A . Thus the standard way to solve Equation 1.2 is to simply compute a partial Singular Value Decomposition of A , at computational cost

$O(mnk)$. (A proof of this elementary fact and details of these decomposition methods are provided in §3.5).

However, in modern engineering applications, it is not enough to measure the worth of an algorithm based on its complexity bound alone. A practitioner must take into consideration the entire computational climate: the intersection of dataset size, storage capabilities and computer architecture. We characterize the common case for today’s datasets and systems, and its implications to the classical decomposition methods performance as follows:

- **Dataset size.** The increase in size of real-world datasets has outpaced the memory capacity of modern computers, to the extent that it is not possible to hold a matrix containing the data into memory on a single machine. Some popular datasets and their sizes: The Netflix competition dataset is $> 2\text{GB}$; the FERET face database, 5.4GB ; the English version of Wikipedia, 14GB ; the knowledge base of IBM’s Jeopardy playing Watson, 1TB . Classical decomposition algorithms which assume access to an entire matrix at once are not feasible for such big problems.
- **Storage.** Since maintaining larger datasets in memory is not an option, data typically needs to be read off of disk storage or even fragmented across many remote machines. In such cases, the bottleneck of running time is accessing the data, not the number of floating-point operations. Another area of concern is data that comes as a potentially infinite *stream*, where each datum can be observed once in turn and then must be discarded. The standard decomposition methods require re-accessing the data $O(k)$ times, which makes them again unsuitable in this setting.
- **Architecture** For a variety of reasons both physical and economical, computing architectures are becoming more and more parallelized. Multi-core processors and cloud computing are becoming predominant. Again, classical decomposition methods were not designed to take advantage of parallel architecture.

All three of these concerns motivate the desire for decentralized methods with as little data transfer as possible. Also, in any application, it is always a nice feature to be able to choose to trade off the cost of resources (compute power, time, and money) with the performance of the method. The random

Algorithm 1.1 Meta-algorithm for randomized low-rank decompositions

Input: $m \times n$ matrix A , target rank k , number of samples $s = k + p$.

Output: $\tilde{A} \in \mathbb{R}^{m \times n}$ which *approximately* minimizes $\min_{\text{rank}(A') \leq k} \|A - A'\|_F^2$.

1. Form lower-dimensional matrix Y by sampling s rows and/or columns (SAMPLEROWSCOLS, ADASAMPLE) or by applying s random projections (RANDPROJ) to A . Y is either $m \times s$, $s \times n$ or $s \times s$.
 2. Compute orthonormal basis Q for the range of Y using exact methods.
 3. Return low-rank approximation \tilde{A} , by projecting A onto the space spanned by basis Q : $\tilde{A} = QQ^T A$.
-

decomposition approaches presented in this review attempt to address these issues.

1.2 Random Techniques

We present three different randomized algorithms for finding a low-rank approximation to a matrix A , specifically, approximately minimizing Equation 1.2. Each has its own error bounds which depend on the amount of samples required, thus allowing a trade-off between computational efficiency and accuracy. Two of the techniques employ intelligent subsampling of the input matrix, and the other uses random projections. A meta-algorithm that encompasses all three is outlined in Algorithm 1.1. We briefly describe how each of the three methods differ in step (2) of Algorithm 1.1 to stochastically obtain a lower dimensional matrix. In each, we use s to denote the number of samples required, which is always at least k plus some non-negative amount of oversampling $p \geq 0$: $s = k + p$.

Method SAMPLEROWSCOLS: Randomly sample columns of A proportional to the squared L_2 -norm of the column (with replacement), and scale to obtain Y . Then, randomly sample and scale rows of X in the same way to obtain W .

$m \times s$
 $s \times s$

Method ADASAMPLE: Let $\mathcal{P}_Y(A)$ denote the projection of A onto the subspace spanned by Y . Start with $Y = \emptyset$. Iterate: Let $E = A - \mathcal{P}_Y(A)$.

Subsample a collection of a few rows S from A proportional to the squared L_2 -norm of the residual E . Update $Y = Y \cup S$. Repeat.

Method RANDPROJ: Project A with random matrix Ω to form Y .

$$\begin{matrix} & m \times n & & n \times s & & m \times s \\ & & & & & \end{matrix}$$

1.3 Methods for Comparison

We analyze and compare the randomized and standard approaches along several different axes. We bound the running times of the algorithms, which are typically straightforward and depend on the number of matrix multiplications and the cost of computing exact decompositions of smaller matrices. We also note how many passes through the data are required and whether the algorithm is “embarrassingly parallel,” in that the algorithm can be trivially sped up by a factor of ℓ when run on ℓ processors.

Finally, we analyze accuracy for all methods in terms of *average Frobenius norm error*, which will always be presented in the form

$$\mathbb{E} \left[\|A - \tilde{A}\|_F^2 \right] \leq f_1(k, s) \|A - A_k\|_F^2 + f_2(k, s) \|A\|_F^2. \quad (1.3)$$

In the above, \tilde{A} is the rank- k matrix that is returned by each randomized algorithm. A_k is the *optimal* rank- k minimizer of Equation 1.2, which can be obtained by the classical decomposition methods. Thus the right-hand side can never be smaller than $\|A - A_k\|_F^2$. Multiplicative factors f_1 and f_2 depend on the desired rank constraint k and the number of samples used s , and are always positive and polynomial in k and $1/s$. Also, f_1 is always greater than or equal to 1, since $\|A - A_k\|_F^2$ is a lower bound on the error.

Remark. In this paper, we do *not* discuss *tail/concentration/deviation bounds* for the algorithms. These types of bounds give us a guarantee that with very high probability (e.g., $> 95\%$), the actual error deviates from the expected error in Equation 1.3 by only a negligible amount (e.g., $< 5\%$) in any execution of the algorithm. Such bounds can be derived if we can get a handle on the variance of the Frobenius norm error as well, and then make use of Markov’s inequality (also often called Chebyshev’s inequality), a standard technique in probability theory and computational learning theory (Kearns and Vazirani, 1994):

$$P \left((X - \mathbb{E}[X])^2 \geq a^2 \right) \leq \frac{\mathbf{Var}[X]}{a^2}.$$

The proofs can be quite technical and involved, and it suffices to say that the errors for all of the algorithms presented here are adequately concentrated about their means. Thus with high probability, the algorithms obtain a result that is very close to the average Frobenius norm error, in the form of Equation 1.3. \square

We present an overview of the results we obtain in this review in Table 1. In §2 we go over related work, and mention a few of the major applications of low-rank factorization. We go over basic definitions and fundamental linear algebra machinery in §3. The randomized algorithms which are the focus of this survey are posed and analysed in §4, §5, and §6. We compare the strengths and weaknesses of the methods in §7, and conclude in §8.

Method, Section	Running Time	# Passes	Multicore?	Error $\mathbb{E} \ A - \tilde{A}\ _F^2$	w.h.p.
SAMPLECOLS, §4.1	$O(mn + ms^2)$	≤ 2	yes	$\leq \mathcal{O} + 2\sqrt{\frac{k}{s}} \ A\ _F^2$	
SAMPLEROWSCOLS, §4	$O(mn + s^3)$	≤ 2	yes	$\leq \mathcal{O} + O\left(\sqrt[4]{\frac{k^5}{s}}\right) \ A\ _F^2$	
ADASAMPLE, §5	$O(mnsT + (m+n)(sT)^2)$	$2T$	per iter.	$\leq \frac{1}{1-k/s} \mathcal{O} + \left(\frac{k}{s}\right)^T \ A\ _F^2$	
RANDPROJ, §6	$O(mns + ms^2)$	≤ 2	yes	$\leq \frac{1-1/s}{1-k/s-1/s} \mathcal{O}$	
Partial SVD or QR (exact)	$O(mnk)$	k	no	$\mathcal{O} = \ A - A_k\ _F^2$	

Table 1: Summary of different methods, explained in §7. Matrix A is $m \times n$, k is the desired rank, s is number of samples, T is number of adaptive iterations, and $\mathcal{O} \equiv \|A - A_k\|_F^2$ stands for the optimal solution. This is a stripped-down version of the full results in Table 2.

2 Related work

The earliest work surveyed here is the column subsampling approach of Frieze et al., first introduced in 1998. This was subsequently refined over the next 8 years including Frieze et al. (2004), culminating in Drineas et al. (2007). The adaptive column sampling method was developed by a set of authors with large overlap with the non-adaptive variant, in (Deshpande and Vempala, 2006). Very recently, and not considered in this review, (Deshpande and Rademacher, 2010) was introduced, which considers sampling based on volume rather than sample magnitude as a refinement to adaptive sampling, obtaining better bounds.

The idea of using random projections to compress data goes back at least to the Johnson-Lindenstrauss Lemma provided by Johnson and Lindenstrauss (1984). It was arguably first used for random matrix projections similar to works in this survey for Latent Semantic Indexing by Papadimitriou et al. (1998). The line of work containing the paper (Halko et al., 2009) reviewed here began with Martinsson et al. (2006b), which first derived bounds introduced the idea of oversampling beyond the desired rank to improve the bounds considerably. Then in (Martinsson et al., 2006a) the authors introduced the idea of combining the random projection method with power iterations for improved performance.

There has been a cottage industry in developing structured random matrices for random projections in recent years. The structure allows for either faster multiplications as in (Nguyen et al., 2009; Woolfe et al., 2008), or better random projections, e.g. Sarlos (2006) uses random Hadamard matrices.

3 Definitions and Preliminaries

In this section we go over various fundamental linear algebra ideas and intuitions which play a role in classical and randomized matrix decompositions.

3.1 Notation

Matrices are denoted with capital letters such as A, X, Q . Vectors and scalars are represented with lowercase letters, e.g. v, x, s . When not clear from context, the scalar-versus-vector question is answered explicitly. For simplicity, we choose to work with only real matrices in this paper, e.g., $A \in \mathbb{R}^{m \times n}$, but most of the proofs and reasoning works as well for $\mathbb{C}^{m \times n}$.

Other definitions, using $A \in \mathbb{R}^{m \times n}$ as a working example:

- The transpose of A as A^T .
- The trace of A is the sum of its diagonal elements, denoted $\text{tr}(A)$.
- For indexing elements and slices of a matrix, we use MATLAB's conventions: $A(i, :)$ is the i^{th} row of A , $A(:, j)$ is the j^{th} column, and $A(i, j)$ is the matrix entry at (i, j) .
- We stack vectors into a matrix using the notation $[a_1 \ a_2 \ \dots \ a_n]$, where a_i are vectors, assumed to always be vertical, in this case $m \times 1$.
- We use $\text{diag}(c_1, c_2, \dots)$ to denote a diagonal matrix with the scalars c_1, c_2, \dots on the diagonal.
- The linear subspace spanned by a set of vectors is denoted $\text{span}(\{a_1, \dots, a_n\})$.
- The range of A is the set of all vectors to which A maps:

$$\text{range}(A) = \{y \in \mathbb{R}^m \mid \exists x \in \mathbb{R}^n \text{ s.t. } y = Ax\}.$$

- The null space of A is the set of all vectors that A maps to zero:

$$\text{null}(A) = \{x \in \mathbb{R}^n \text{ s.t. } Ax = 0\}.$$

- The volume of the parallelepiped spanned by the columns of A is denoted $\text{vol}(A)$, which is equivalent to the absolute value of the determinant of A .

3.2 The Singular Value Decomposition

Every real matrix of arbitrary rectangular dimensions can be factored into the form

$$A = \underset{m \times n}{U} \underset{m \times m}{\Sigma} \underset{m \times n}{V^T},$$

where U, V both have orthonormal columns, and whose columns contain the *left and right singular vectors*, respectively. The matrix Σ is diagonal and contains *singular values* which correspond to the singular vectors. By convention, we denote the i^{th} singular value as $\sigma_i(A)$, and order them and their corresponding singular vectors into U and V such that $\Sigma = \text{diag}(\sigma_1(A), \sigma_2(A), \dots)$ and $\sigma_i(A) \geq \sigma_{i+1}(A)$ for all i . Equivalently we can write

$$A = \sum_{i=1}^{\min\{m,n\}} \sigma_i u_i v_i^T,$$

which is a decomposition of A into a sum of rank-1 matrices. We also note that $A^T A = V \Sigma^2 V^T$ and $A A^T = U \Sigma^2 U^T$, which tells us that the eigenvectors of $A^T A$ are $\{v_i\}$, and the eigenvectors of $A A^T$ are $\{u_i\}$, each with eigenvalues $\lambda_i = \sigma_i^2$.

Geometrically, U and V can be thought of as rotation matrices $\in SO(m)$ and $SO(n)$, and thus we can view the matrix-vector multiplication of A acting on vector x as a sequence of steps: (1) Take the vector $x \in \mathbb{R}^n$ and rotate it by V^T . (2) Scale each dimension of the result independently via multiplication with Σ . When $m < n$, Σ also discards the last $m - n$ dimensions to map to \mathbb{R}^m ; when $m > n$, it maps it to \mathbb{R}^m by zero padding the new $m - n$ dimensions. (3) Rotate a second time, by U .

Using this geometric intuition, one can infer that when A has rank k , the first k left singular values u_1, \dots, u_k form an orthonormal basis for $\text{range}(A)$, and the remaining left singular vectors u_{k+1}, \dots, u_m are a basis for the null space. Similar reasoning goes for the range and null space of A^T , using the right singular vectors.

3.3 Orthogonal Projections

Let us say we have an orthonormal basis for a linear subspace, stacked into a matrix: $Q = [q_1 \ q_2 \ \dots \ q_\ell]$. Then $Q Q^T$ is a projection matrix which operates

on any matrix A to project it orthogonally onto the subspace spanned by Q , which we denote $\mathcal{P}_Q(A)$:

$$\mathcal{P}_Q(A) \equiv QQ^T A.$$

3.4 Norms

There are two matrix norms typically used in the literature reviewed for this survey. We will try to deal exclusively with the *Frobenius norm* in this paper for simplicity and ease of comparison. The squared Frobenius norm and its equivalent formulations:

$$\|X\|_F^2 = \sum_{i,j} X(i,j)^2 = \mathbf{tr}(X^T X) = \sum_{i=1}^{\min\{m,n\}} \sigma_i(X)^2.$$

(in MATLAB notation, this would be `sum(X(:).^2)`).

Most papers under discussion here provide equivalent bounds for the L_2 -operator norm, also called the *spectral norm*:

$$\|X\|_2 = \max_{\|v\|_2=1} \|Xv\|_2 = \sigma_1(X).$$

where, for vectors v , $\|v\|_2$ is the standard L_2 (Euclidean) norm $\|v\|_2^2 = \sum_i v_i^2$.

3.5 The Optimal Low-Rank Decomposition

Here we go over a proof of a classical result from Eckart and Young (1936), which states that the best rank- k approximation to $A = U\Sigma V^T$ is by truncating the singular values (setting $\sigma_j(A) = 0$ for $j > k$). We adorn the solution in boxes to emphasize the importance—all error bounds to follow will be composed of this optimal solution.

Theorem 3.1 (Eckart-Young Theorem). *Let $A \in \mathbb{R}^{m \times n}$, $A = U\Sigma V^T$. Then*

$$\min_{\text{rank}(A') \leq k} \|A - A'\|_F^2$$

is achieved by minimizer

$$A_k \equiv U\Sigma_k V^T,$$

where $\Sigma_k = \text{diag}(\sigma_1, \dots, \sigma_k, 0, \dots, 0)$. The optimum value is

$$\mathcal{O} \equiv \|A - A_k\|_F^2 = \sum_{j>k} \sigma_j^2(A).$$

Proof. First, we observe that the Frobenius norm is *unitarily invariant*: For any pair of unitary matrices U_1, U_2 , and any matrix X with agreeable dimensions,

$$\begin{aligned} \|U_1 X U_2^T\|_F^2 &= \text{tr}((U_1 X U_2^T)^T U_1 X U_2^T) \\ &= \text{tr}(U_2 X^T U_1^T U_1 X U_2^T) = \text{tr}(U_2 X^T X U_2^T) \\ &= \text{tr}(U_2^T U_2 X^T X) = \text{tr}(X^T X) \\ &= \|X\|_F^2 \end{aligned}$$

Now we can begin, remembering that $A = U\Sigma V^T$, and exploiting the unitarily invariant property right away by multiplying the left and right by U^T and V :

$$\|A - A'\|_F^2 = \|U^T(A - A')V\|_F^2 = \|\Sigma - U^T A' V\|_F^2$$

Since Σ is diagonal, we see that A' should be diagonal as well to minimize the Frobenius norm above. This can be achieved by letting A' have the form $A' = U \text{diag}(\sigma_1(B), \dots, \sigma_{\min\{m,n\}}(B)) V^T$, where the $\sigma_i(B)$ remain to be picked. Thus,

$$\|A - A'\|_F^2 = \|\Sigma - U^T A' V\|_F^2 = \sum_{j=1}^{\min\{m,n\}} (\sigma_j(A) - \sigma_j(A'))^2$$

Since A' is rank k , only k singular values of A' can be non-zero. From the last line above, and because $\sigma_1(A) \geq \sigma_2(A) \geq \dots$, it follows that the minimizer sets $\sigma_j(A') = \sigma_j(A)$ for $j = 1, \dots, k$, and $\sigma_j(A') = 0$ for $j > k$. Then we have fully specified the minimizer $A_k = U\Sigma_k V^T$. It follows that the optimal value is $\sum_{j>k} \sigma_j^2(A)$. \square

3.5.1 Exact methods for Optimal Decomposition

The easiest way to compute the minimizer A_k , is by computing a full SVD of $A = U\Sigma V^T$ and truncating the singular values to form $A_k = U\Sigma_k V^T$.

This requires $O(mn \min\{m, n\})$ work, which is overkill when we only care about the top k singular vectors and values. Below we discuss two standard approaches to compute both the partial SVD and partial QR decompositions in time $O(mnk)$. The books by Golub and van Loan (1984); Strang (2009) are excellent resources for these methods.

Power iteration methods. Given a random vector x , iterating $x \leftarrow Ax$ results in $A^p x = U \Sigma^p V^T x$ after p iterations. The top singular value of A is thus σ_1^p , which quickly dominates the other singular values, and $A^p x$ converges to $\sigma_1 u_1$. The rate of convergence depends on the ratio σ_1/σ_2 . If the top k singular values and singular vectors are desired, various methods (Orthogonal Iteration, Lanczos, block Lanczos, Arnoldi) involve maintaining a set of random vectors and/or previous iterates $\{A^1 v, A^2 v, \dots, A^{p-1} v\}$ for which an orthogonal basis is maintained throughout the iterative process. This orthogonal basis composes the top k left singular vectors. We can always obtain right singular vectors from left via $A^T u_i = \sigma_i v_i$. The orthogonalization process and the iterative nature of these methods implies extensive random access to the entire input at all times. Furthermore, practical implementations must be careful about numerical precision and redundant singular vectors, requiring restarts.

Rank-revealing QR decomposition. The QR decomposition decomposes a matrix A into two matrices:

$$A = \begin{matrix} & Q & R \\ m \times n & m \times m & m \times n \end{matrix}$$

such that Q has orthonormal columns, and R is upper triangular. The basis formed by the columns of Q are an *orthogonalization* of the columns of A . The full QR decomposition is computed by a sequence of special projections, the Householder projections of the form $H_q = I - qq^T$, which are reflections about a line determined by q . Vector q_i can always be chosen to transform $H_{q_{i-1}} \dots H_{q_1} A$ into a similar matrix which zeros out the elements of the i^{th} column below the diagonal. After n iterations, we obtain $H_{q_n} \dots H_{q_1} A = R$, so $Q = H_{q_1} \dots H_{q_n}$. This method is a more elegant and stable way of forming an orthogonal basis from a set of vectors than the standard Gram-Schmidt method, and is used in all of our randomized algorithms (in a low dimensional space).

The *rank-revealing QR*, or *RRQR* method, can be used for partial decomposition (Gu and Eisenstat, 1996). RRQR works much the same as

standard QR with Householder reflections, but also determines a pivoting of the columns to terminate when the Frobenius norm of the unprocessed columns is determined to be negligible. Note this only works if the input matrix is truly rank-deficient. The Q returned by this algorithm, consisting of k columns, can be used as a basis for the range of A , in the same way as the top k left singular vectors (though they are not necessarily equal, even modulo permutation). Once again, the iterative nature of this method forces it to make k passes over the input.

4 Randomly subsampling *a priori*: SAMPLECOLS, SAMPLEROWSCOLS

In this section we discuss the first of our three random decomposition techniques, and in many ways the most intuitive: randomly subsampling the rows and columns of the original matrix A to obtain a smaller, more manageable matrix Y . We hope that Y is not too different from A ; the analysis in this section will provide a bound on the expected difference. Since Y is small, we can perform exact decomposition methods on it cheaply. The algorithms and proofs presented here are due to Frieze et al. (2004) and Kannan and Vempala (2008).

A good set of columns does exist. Before we begin, we would like to point out a classical existential result, that bounds how well you can do by choosing the best possible subset of k columns:

Theorem 4.1 (Auerbach’s Theorem (Ruston, 1962), as parsed by Halko et al. (2009)). *Every $m \times n$ matrix A contains a k -column submatrix C for which*

$$\|A - \mathcal{P}_C(A)\|_2 \leq \sqrt{1 + k(n - k)}\|A - A_k\|_2 = O(\sqrt{n})\|A - A_k\|_2.$$

Thus a “good” set of columns always exists, in the sense that they are within a multiplicative factor of the optimal solution. Thus it is a worthwhile endeavor to try and find them. Performing exact optimal column selection to maximize the largest or smallest singular values or volume is known to be NP-hard (Eivril and Magdon-Ismail, 2009). The rank revealing QR method discussed in §3.5.1 can be viewed as a deterministic approximation algorithm to achieve maximal volume.

4.1 A First Attempt: subsampling columns only

We first present and analyze the technique of subsampling columns of the matrix A . The algorithm is presented in Algorithm 4.1. We offer an explanation of the steps as follows. At this point, the explanations are by necessity imprecise, but hopefully make the formal analysis easier to understand.

1. We sample columns proportional to how far they are from the origin, when viewed as points in \mathbb{R}^m . This is in accordance with the geomet-

Algorithm 4.1 SUBSAMPLECOLUMNS: obtain a low-rank factorization by subsampling columns \propto their magnitude.

Input: $m \times n$ matrix A , target rank k , number of samples $s = k + p$.

Output: $\tilde{A} \in \mathbb{R}^{m \times n}$ which *approximately* minimizes $\min_{\text{rank}(A') \leq k} \|A - A'\|_F^2$ in time $O(mn + ms^2)$.

1. Sample s columns $A(:, i_1), \dots, A(:, i_s)$ from A , proportional to their L_2 -norm squared, where the probability of choosing column i is $p_i = \|A(:, i)\|_2^2 / \|A\|_F^2$.

2. Form

$$Y_{m \times s} = \frac{\|A\|_F}{\sqrt{s}} \left[\frac{A(:, i_1)}{\|A(:, i_1)\|_2}, \dots, \frac{A(:, i_s)}{\|A(:, i_s)\|_2} \right]$$

3. Compute $Q = [q_1 \dots q_k]$, the top k left singular vectors of Y (via SVD).

4. Return low-rank approximation \tilde{A} , by projecting A onto basis Q : $\tilde{A} = QQ^T A$.

ric intuition that points close to the origin have less influence on the principal directions of the data.

2. The rescaling $\frac{1}{\sqrt{s}} \frac{\|A\|_F}{\|A(:, i)\|_2}$ of the sampled vectors is done so that the Frobenius norm of Y is comparable to the Frobenius norm of A .
3. In the decomposition $A = U\Sigma V^T$, if $\text{rank}(A) = k$, then $U_k = [u_1 \dots u_k]$ is an orthonormal basis for $\text{range}(A)$. Since we hope that Y is “close” to A , we expect that in the partial SVD $Y \approx QSW^T$, that Q is “close” to U_k , in that it is a basis that covers $\text{range}(A)$ well. The operation $QQ^T A$ projects A into the rank- k subspace spanned by Q , as desired.

Running time

It takes $O(mn)$ time to compute the distribution $\{p_i\}$ and construct $Y_{m \times s}$. Obtaining the left singular values of Y takes $O(ms^2)$. We omit the time to actually form \tilde{A} (which is $O(mnk)$) because typically we never need it explicitly. Thus the total running time is $O(mn + ms^2)$.

4.1.1 Analysis

To determine what the approximation error is in Algorithm 4.1, we answer two questions:

- Q1. How different is the subsampled version Y from the original A ? This can be quantified as the residual $\mathbb{E} [\|AA^T - YY^T\|_F^2]$.
- Q2. Given any matrices A and Y , suppose we project A onto the basis Q comprised of the top k left singular vectors of Y . How different is that from the minimal distortion projection of A down to rank- k ? This can be measured by the quantity $\|A - QQ^T A\|_F^2 - \|A - A_k\|_F^2$.

We can combine the answer to these questions (in Lemma 4.1 and Lemma 4.2 respectively) to bound the expected approximation error incurred in Algorithm 4.1.

Lemma 4.1. *Given any matrices $A \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{m \times s}$, form Y as in*

$$\mathbb{E} [YY^T] = AA^T \text{ and } \mathbb{E} [\|YY^T - AA^T\|_F^2] \leq \frac{1}{s} \|A\|_F^4.$$

Proof. We begin with an analysis of randomly subsampled matrix-vector multiplication. Let's say we want to compute the matrix-vector multiplication of A time v :

$$Av = \sum_{i=1}^n A(:, i)v_i.$$

If n is very large, we might want to get away with only computing the above sum for a subset of the columns. Define a sampling distribution where p_i is the probability of picking column i . Then intuitively we could approximate Av by computing:

$$Av \approx \frac{1}{s} \sum_{j=1}^s \frac{A(:, i_j)v_{i_j}}{p_{i_j}}.$$

where i_j is the j^{th} random column index. Now consider the random variable

$$X = \frac{A(:, i)v_i}{p_i}.$$

We can examine the expectation and variance of X to get a handle on the expected value and variance of our randomized matrix-vector multiply technique:

$$\mathbb{E}[X] = \sum_{i=1}^n p_i \frac{A(:, i)v_i}{p_i} = \sum_{i=1}^n A(:, i)v_i = Av.$$

so X is an unbiased estimator, and

$$\mathbf{Var}[X] = \mathbb{E}[\|X\|^2] - \|\mathbb{E}[X]\|^2 = \sum_{i=1}^n \frac{\|A(:, i)\|^2 v_i^2}{p_i} - \|Av\|^2$$

To cleanly bound this variance, we can choose p_i as we did in [Algorithm 4.1](#): set p_i proportional to the squared L_2 -norm of each column. Thus when

$$p_i = \|A(:, i)\|^2 / \|A\|_F^2,$$

we can summarize

$$\mathbf{Var}[X] \leq \|A\|_F^2 \|v\|^2.$$

We can then quickly extend this bound on variance to multiple samples, and to matrix-matrix multiplication to finish up the lemma. If we use s samples, our variance estimate gets s times better:

$$\mathbf{Var}\left[\frac{1}{s} \sum_{j=1}^s X_j\right] \leq \frac{1}{s} \|A\|_F^2 \|v\|^2.$$

Now, we can apply the matrix-vector idea ℓ times to perform randomized matrix-matrix multiplication between A and B :

$$\text{Let } Z = \frac{1}{s} \sum_{j=1}^s \frac{A(:, i_j)B(i_j, :)}{p_{i_j}}.$$

then

$$\mathbb{E}[Z] = AB \text{ and } \mathbf{Var}[Z] \leq \frac{1}{s} \|A\|_F^2 \|B\|_F^2 \quad (4.1)$$

using similar arguments and again setting p_i proportional to squared L_2 -norm. Finally, when $B = A^T$, we can write $Z = YY^T$, with

$$Y_{m \times s} = \frac{\|A\|_F}{\sqrt{s}} \left[\frac{A(:, i_1)}{\|A(:, i_1)\|_2}, \dots, \frac{A(:, i_s)}{\|A(:, i_s)\|_2} \right]$$

and obtain

$$\mathbf{Var} [YY^T] = \mathbb{E} [\|YY^T - \mathbb{E} [YY^T]\|_F^2] = \mathbb{E} [\|YY^T - AA^T\|_F^2] \leq \frac{1}{s} \|A\|_F^4$$

by plugging into Equation 4.1 above, which completes the proof. Also note that, in this case,

$$\mathbf{Var} [Z] = \frac{1}{s} \sum_{i=1}^n \frac{\|A(:, i)\|_2^4}{p_i} - \|AA^T\|_F^2.$$

Choosing the p_i 's proportional to the column's squared magnitude *exactly* minimizes this variance. \square

Lemma 4.2. *Given any matrices $A \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{m \times s}$, let $Q \in \mathbb{R}^{m \times k}$ contain the top k left singular vectors of Y . Then*

$$\|A - QQ^T A\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k} \|AA^T - YY^T\|_F.$$

Proof. By massaging quantities under the trace definition of the Frobenius norm, it is straightforward to show that

$$\|A - QQ^T A\|_F^2 = \|A\|_F^2 - \|Q^T A\|_F^2$$

and

$$\|A - A_k\|_F^2 = \|A\|_F^2 - \|A_k\|_F^2$$

Now we can proceed:

$$\begin{aligned}
& \|A - QQ^T A\|_F^2 - \|A - A_k\|_F^2 \\
&= \|A\|_F^2 - \|Q^T A\|_F^2 - (\|A\|_F^2 - \|A_k\|_F^2) && \text{via massaging} \\
&= \|A_k\|_F^2 - \|Q^T A\|_F^2 \\
&= \|A_k\|_F^2 - \|Q^T A\|_F^2 - \|Q^T Y\|_F^2 + \|Q^T Y\|_F^2 && \text{add/subtract a scalar} \\
&= \sum_{i=1}^k \sigma_i^2(A) - \sigma_i^2(Y) + \sum_{i=1}^k \sigma_i^2(Y) - \|q_i^T A\|_F^2 && \text{def. of Frobenius norm} \\
&\leq \sqrt{k \sum_{i=1}^k (\sigma_i^2(A) - \sigma_i^2(Y))^2} + \sqrt{k \sum_{i=1}^k (\sigma_i^2(Y) - \|q_i^T A\|_F^2)^2} && \text{Cauchy-Schwarz both terms} \\
&= \sqrt{k \sum_{i=1}^k (\sigma_i(AA^T) - \sigma_i(YY^T))^2} + \sqrt{k \sum_{i=1}^k (q_i^T(YY^T - AA^T)q_i)^2} && \text{because } \sigma_i^2(X) = \sigma_i^2(XX^T) \\
&\leq 2\sqrt{k}\|AA^T - YY^T\|_F && \text{Hoffman-Wielandt inequality}
\end{aligned}$$

The last line used the useful Hoffman-Wielandt inequality, that relates differences in singular values to Frobenius norm difference in symmetric matrices:

$$\sum_i (\sigma_i(A) - \sigma_i(B))^2 \leq \|A - B\|_F^2.$$

This is straightforward to prove using von Neumann's Lemma, see Mirsky (1975). \square

We can combine Lemma 4.1 and Lemma 4.2 to prove the main bound on the performance for Algorithm 4.1.

Theorem 4.2. *Algorithm 4.1 finds a rank- k matrix \tilde{A} such that*

$$\mathbb{E} \left[\|A - \tilde{A}\|_F^2 \right] \leq \|A - A_k\|_F^2 + 2\sqrt{\frac{k}{s}} \|A\|_F^2.$$

Proof. Using Q and Y found in the algorithm, we take expectations with respect to Y :

$$\mathbb{E} \left[\|A - QQ^T A\|_F^2 \right] \leq \|A - A_k\|_F^2 + 2\sqrt{k}\mathbb{E} \left[\|AA^T - YY^T\|_F \right] \quad (4.2)$$

$$(4.3)$$

Algorithm 4.2 SUBSAMPLE: obtain a low-rank factorization by subsampling columns and rows.

Input: $m \times n$ matrix A , target rank k , number of samples $s = k + p$.

Output: $\tilde{A} \in \mathbb{R}^{m \times n}$ which *approximately* minimizes $\min_{\text{rank}(A') \leq k} \|A - A'\|_F^2$, in $O(mn + ms^2)$ time.

1. Sample s columns $A(:, i_1), \dots, A(:, i_s)$ from A , proportional to their L_2 -norm squared, where the probability of choosing column i is

$$p_i^{\text{col}} = \|A(:, i)\|_2^2 / \|A\|_F^2.$$

2. Form

$$Y_{m \times s} = \frac{\|A\|_F}{\sqrt{s}} \left[\frac{A(:, i_1)}{\|A(:, i_1)\|_2}, \dots, \frac{A(:, i_s)}{\|A(:, i_s)\|_2} \right]$$

3. Sample s rows $Y(j_1, :), \dots, Y(j_s, :)$ from Y , proportional to their L_2 -norm squared, where the probability of choosing row j is

$$p_j^{\text{row}} = \|Y(j, :)\|_2^2 / \|Y\|_F^2.$$

4. Form

$$W_{s \times s} = \frac{\|Y\|_F}{\sqrt{s}} \left[\frac{Y(j_1, :)}{\|Y(:, j_1)\|_2}, \dots, \frac{Y(j_s, :)}{\|Y(:, j_s)\|_2} \right]$$

5. Compute $V = [v_1 \dots v_s]$, the top right singular vectors of W via SVD.

6. Compute $q_i = \frac{Y v_i}{\|Y v_i\|_2}$ for $i \in [1, s]$.

7. Discard q_i for i such that $\|Y v_i\|_2 < \text{const} \cdot \left(\frac{k}{s}\right)^{\frac{1}{4}} \|Y\|_F^2$. From the remaining l vectors, form $Q = [q_1 \dots q_l]$.

8. Return low-rank approximation \tilde{A} , by projecting A onto basis Q :
 $\tilde{A} = Q Q^T A$.

from Lemma 4.2 and using linearity of expectation. Applying Lemma 4.1 to the last term gives the bound stated in this theorem. \square

4.2 “Constant-time SVD”: subsampling rows and columns

In the previous section, we discussed an algorithm which subsamples columns of the input A , and performs exact decompositions of the resulting $m \times s$ matrix, resulting in a running time of $O(ms^2)$. We now consider subsampling both the rows *and* columns of A , which, for a fixed s , increases approximation error, but decreases running time down to $O(s^3)$. For a matrix of rank k and a fixed error tolerance, s is a fixed quantity that does not depend on the ambient dimensions of the matrix m and n . Hence the term “constant time SVD.” The algorithm is presented in [Algorithm 4.2](#).

The algorithm subsamples rows and columns to obtain W . It then extracts an orthonormal basis V for the row space of W in step 5. We can then derive a good basis Q for the column space of Y by forming YV as in Step 6. We handle numerical issues regarding division by zero or numbers close to zero in step 7 by throwing out vectors with small norm. Finally, once we have our basis for Y , the reasoning is the same as in [Algorithm 4.1](#) to conclude that $QQ^T A$ is a good low-rank approximation to A .

Running time

It takes $O(mn)$ time to compute $\{p_i^{col}\}$. Computing the singular vectors of W to obtain V takes $O(s^3)$ only. All other intermediate steps take $O(ms)$, so the entire algorithm is $O(mn + s^3)$, again, ignoring the explicit formation of \tilde{A} .

4.2.1 Analysis

The analysis of [Algorithm 4.2](#) uses pieces of analysis from [Algorithm 4.1](#). Namely [Lemma 4.1](#), regarding the error incurred from sampling, is used twice: once for the columns (bounding $\|A^T A - Y^T Y\|_F^2$), and once for the rows (bounding $\|Y^T Y - W^T W\|_F^2$). We need two additional pieces to help glue all the reasoning together.

First, we need to relate the error incurred by going from an approximately “good” right projection (e.g., one where $Y \approx YV V^T$) to obtain an approximately “good” left projection (e.g., $Y \approx QQ^T Y$), as is done in step 6 of [Algorithm 4.2](#).

Lemma 4.3 (Good left projections from good right projections, (Corollary 6.9 in (Kannan and Vempala, 2008))). *Let $q_i = Yv_i/\|Yv_i\|_2$ with sufficient*

magnitude as specified in Algorithm 4.2, Steps 5 and 6. Then

$$\mathbb{E} [\|Y - QQ^T Y\|_F^2] \leq \|Y - YVV^T\|_F^2 + O\left(\sqrt[4]{\frac{k^5}{s}}\right) \|A\|_F^2. \quad (4.4)$$

Second, we state a useful result which relates the difference in projection error of two matrices onto the same subspace.

Lemma 4.4 (Difference in projection error, (Lemma 6.5 in (Kannan and Vempala, 2008))). *For any k vectors $\{u_i\}_{i=1}^k$ with $\|u_i\|_2 = 1$, stacked as $U = [u_1 \dots u_k]$, and for any matrices B and C with an equal number of rows,*

$$\|B - UU^T B\|_F^2 - \|C - UU^T C\|_F^2 \leq \|BB^T - CC^T\|_F (k+1)^2. \quad (4.5)$$

Proofs of these intermediate results have been omitted for brevity, please refer to Kannan and Vempala (2008). We can now use these two facts to state and sketch out the main result of this section.

Theorem 4.3. *Algorithm 4.2 finds a rank- k matrix \tilde{A} such that*

$$\mathbb{E} [\|A - \tilde{A}\|_F^2] = \|A - A_k\|_F^2 + O\left(\sqrt[4]{\frac{k^5}{s}}\right) \|A\|_F^2.$$

Proof sketch. From Lemma 4.1, we have

$$\mathbb{E} [\|Y^T Y - W^T W\|_F] \leq \frac{1}{\sqrt{s}} \mathbb{E} [\|Y\|_F^2] = \frac{1}{\sqrt{s}} \|A\|_F^2. \quad (4.6)$$

We can use Lemma 4.2 equivalently with right projections (e.g., YVV^T) instead of left projections (e.g., $QQ^T A$) to say that

$$\mathbb{E} [\|Y - YVV^T\|_F^2] \leq \|Y - Y_k\|_F^2 + 2\sqrt{\frac{k}{s}} \mathbb{E} [\|Y\|_F^2] \quad (4.7)$$

$$= \|Y - Y_k\|_F^2 + 2\sqrt{\frac{k}{s}} \|A\|_F^2 \quad (4.8)$$

$$\leq \|A - A_k\|_F^2 + 2\sqrt{\frac{k}{s}} \|A\|_F^2. \quad (4.9)$$

We can apply Equation 4.5 to compare projecting both A and Y onto Q , and then subsequently chain together the inequalities Equation 4.4, Equation 4.9 and Equation 4.6 to get the following derivation:

$$\begin{aligned}
\mathbb{E} \left[\|A - \tilde{A}\|_F^2 \right] &= \mathbb{E} \left[\|A - QQ^T A\|_F^2 \right] \\
&\leq \mathbb{E} \left[\|Y - QQ^T Y\|_F^2 \right] + \mathbb{E} \left[\|AA^T - YY^T\|_F^2 \right] (k+1)^2 && \text{from Equation 4.5} \\
&\leq \mathbb{E} \left[\|Y - YVV^T\|_F^2 \right] + O \left(\sqrt[4]{\frac{k^5}{s}} \right) \|A\|_F^2 + \mathbb{E} \left[\|AA^T - YY^T\|_F^2 \right] (k+1)^2 && \text{from Equation 4.4} \\
&\leq \|A - A_k\|_F^2 + O \left(\sqrt{\frac{k}{s}} + \sqrt[4]{\frac{k^5}{s}} \right) \|A\|_F^2 + \|AA^T - YY^T\|_F^2 (k+1)^2 && \text{from Equation 4.9} \\
&\leq \|A - A_k\|_F^2 + O \left(\sqrt{\frac{k}{s}} + \sqrt[4]{\frac{k^5}{s}} + \frac{(k+1)^2}{s} \right) \|A\|_F^2 && \text{from Lemma 4.2} \\
&\leq \|A - A_k\|_F^2 + O \left(\sqrt[4]{\frac{k^5}{s}} \right) \|A\|_F^2 && \text{simplification}
\end{aligned}$$

□

5 Adaptive subsampling: ADASAMPLE

In the last section, we saw a algorithm which took s samples from A independently, with an error bound of $\|A - \tilde{A}\|_F^2 \leq \|A - A_k\|_F^2 + 2\sqrt{k/s}\|A\|_F^2$. This is an additive type error, which scales with the size of the Frobenius norm of the matrix. It is easy to construct matrices where such a sampling method would not perform well. Imagine a matrix with one column having an extremely large magnitude compared to the others. This column would get resampled many times in [Algorithm 4.1](#) providing no additional information about the range of the matrix. In another scenario, with all rows and columns of comparable magnitude, an overwhelming amount of columns or rows that are all linearly dependent will again lead to redundant sampling: think of finding the best rank-2 subspace of a dataset where all but one point lies on a line. The true rank-2 subspace would capture both the line and the point, but in [Algorithm 4.1](#), the single point may never be found by sampling (getting chosen with probability $\approx 1/n$).

These failure modes motivate a new algorithm: sample once as in [§4](#). Then sample again, but proportional to the portion of A not already cap-

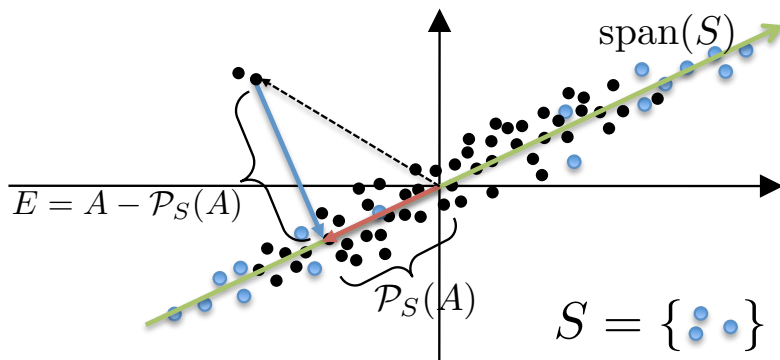


Figure 1: A simple example to motivate the need for adaptive sampling and geometric intuition of the quantities in [Algorithm 5.1](#): Sampling from the points shown, it is very unlikely to sample the outlying points in the upper left to capture the optimal rank-2 subspace. The ADASAMPLE method, however, in the next iteration, samples points proportional to their as-yet unexplained portion: $E = A - \mathcal{P}_S(A)$, which is largest for those outlier points, which ensures a high probability of them getting sampled.

tured by the span of sample. The algorithm is presented in [Algorithm 5.1](#). [Figure 1](#) shows our motivating example. The algorithm was first suggested in [Deshpande and Vempala \(2006\)](#), with improved bounds in [Kannan and Vempala \(2008\)](#). [Eivril and Magdon-Ismail \(2009\)](#) offer a similar, deterministic greedy algorithm to find a k -column submatrix of A with maximal volume.

Running time

The only computational step in the algorithm is to compute $E = A - \mathcal{P}_S(A)$. For a single iteration t , we need only update our current residual E with the orthogonal extension of the subspace coming from the S_t we just added. Hence we really update $E \leftarrow A - E - \mathcal{P}_{S_t}(A)$. Computing $\mathcal{P}_{S_t}(A)$ requires two steps.

- Obtain an orthonormal basis Y_t for S_t by orthogonalizing the s vectors in S_t against the currently stored orthonormal basis $\underset{ts \times n}{Y}$ which covers span $(\bigcup_{i=1}^{t-1} S_i)$. This takes $O(nts^2)$.
- Project A onto Y_t by computing $\mathcal{P}_{Y_t}(A) = AY_tY_t^T$. This takes $O(mns)$.

Thus, iteration t takes $O(mns + nts^2)$ time, so the iterative process takes $O(\sum_{t=1}^T mns + ns^2t) = O(mnsT + ns^2T^2)$. Finally, to compute the optimal k right singular vectors for $\mathcal{P}_S(A)$ in the final step takes $O(ms^2T^2)$, giving us a final running time of $O(mnsT + ns^2T^2 + ms^2T^2) = O(mnsT + (sT)^2(m+n))$.

5.1 Analysis

We begin with a generalization of [Theorem 3.1](#) which provides a bound for a single iteration of [Algorithm 5.1](#). Then by induction we can prove an error bound for [Algorithm 5.1](#). Recall from [§3](#) that the projection of A down to a linear subspace L is denoted $\mathcal{P}_L(A)$. Now, we define a new concept: Define $\mathcal{P}_{L,k}(A)$ to be the best rank- k approximation to A , limited to the subspace L . In other words,

$$\mathcal{P}_{L,k}(A) \equiv \arg \min_{\substack{\text{rank}(A') \leq k, \\ \text{span}(A') \subseteq L}} \|A - A'\|^2.$$

Equipped with this new concept and definition, we can bound the residual E of a single iteration, when a sample S has just been added to our current span of samples L :

Algorithm 5.1 ADASAMPLE obtain a low-rank factorization by adaptively sampling rows.

Input: $m \times n$ matrix A , target rank k , number of samples $s = k + p$, number of sampling iterations T .

Output: $\tilde{A} \in \mathbb{R}^{m \times n}$ which *approximately* minimizes $\min_{\text{rank}(A') \leq k} \|A - A'\|_F^2$, in $O(mnsT + (sT)^2(m + n))$ time.

1. Start with the empty linear subspace $S = \{\}$, such that $A - \mathcal{P}_S(A) = A$. Let $E = A - \mathcal{P}_S(A)$.

2. For $t = 1$ to T

(a) Pick a subset S_t of s rows of A , with row i chosen according to

$$p_i = \frac{\|E(i, :)\|_2^2}{\|E\|_F^2}.$$

(b) $S \leftarrow S \cup S_t$

(c) $E \leftarrow A - \mathcal{P}_S(A)$

3. Return $\tilde{A} = \mathcal{P}_{S,k}(A)$, the best rank- k approximation to the span of the samples in S .

Lemma 5.1. *Let $A \in \mathbb{R}^{m \times n}$ and $L \subseteq \mathbb{R}^n$ be a linear subspace. Let $E = A - \mathcal{P}_L(A)$. Let S be a collection of s rows of A sampled with distribution $p_i = \|E(i, :)\|_2^2 / \|E\|_F^2$. Then, for $k \geq 0$,*

$$\mathbb{E} [\|A - \mathcal{P}_{L+\text{span}(S),k}(A)\|_F^2] \leq \|A - A_k\|_F^2 + \frac{k}{s} \|E\|_F^2.$$

Remark. This can be seen as an extension to Theorem 4.2, in that when $L = \emptyset$ it makes sense that we get the same bounds as we do in SAMPLECOLS. In fact, the following analysis provides a way of proving a tighter bound on the performance of SAMPLECOLS than Theorem 4.2, where we get an additive error factor of k/s instead of $2\sqrt{k/s}$ \square

Proof. We construct vectors $w_1, \dots, w_k \in L + \text{span}(S)$, and show that $W = \text{span}\{w_1, \dots, w_k\}$ is a good approximation to the optimal right projection of

A , which uses $\text{span}\{v_1, \dots, v_k\}$ to achieve the optimal A_k . Formally, we'll show that

$$\mathbb{E} [\|A - \mathcal{P}_W(A)\|_F^2] \leq \|A - A_k\|_F^2 + \frac{k}{s} \|E\|_F^2. \quad (5.1)$$

First, we have that

$$\mathbb{E} [\|A - \mathcal{P}_{L+\text{span}(S),k}(A)\|_F^2] \leq \mathbb{E} [\|A - \mathcal{P}_W(A)\|_F^2],$$

since $W \subseteq L + \text{span}(S)$, that is, the optimal projection of A onto a potentially larger subspace (i.e., $\mathcal{P}_{L+\text{span}(S)}(A)$) has to be at least as good as an orthogonal projection onto a contained subspace (i.e., $\mathcal{P}_W(A)$). Thus if we can prove Equation 5.1, we are finished. Construct

$$w_j = \mathcal{P}_L(A)^T u_j + \frac{1}{s} \sum_{r=1}^s \frac{u_{i_r}}{p_{i_r}} E(i_r, :),$$

where i_r is the r^{th} random index.

We can derive the expected value of w_j (with respect to the random sample) like so:

$$\begin{aligned} \mathbb{E} [w_j] &= \mathcal{P}_L(A)^T u_j + \frac{1}{s} \sum_{r=1}^s \mathbb{E} \left[\frac{u_{i_r}}{p_{i_r}} E(i_r, :) \right] \\ &= \mathcal{P}_L(A)^T u_j + E^T u_j \\ &= A^T u_j = \sigma_j v_j. \end{aligned}$$

Through a few more tedious calculations that are similar to the ones in Lemma 4.1, we can also bound the variance:

$$\mathbf{Var} [w_j] = \mathbb{E} [\|w_j - \sigma_j v_j\|_F^2] \leq \frac{1}{s} \|E\|_F^2. \quad (5.2)$$

Now, consider the right projection $F = A \sum_{i=1}^k \frac{1}{\sigma_i} v_i w_i$. The row space of F is contained in W , i.e., $\text{span}(F^T) \subseteq W$. It follows that $\|A - \mathcal{P}_W(A)\|_F^2 \leq \|A - F\|_F^2$. So by bounding F , we can finish up the theorem:

$$\mathbb{E} [\|A - \mathcal{P}_{L+\text{span}(S)}(A)\|_F^2] \leq \mathbb{E} [\|A - \mathcal{P}_W(A)\|_F^2] \leq \|A - F\|_F^2 \quad (5.3)$$

$$= \sum_{i=1}^m \mathbb{E} [\|(A - F)^T u_i\|_F^2] \quad (5.4)$$

$$= \sum_{i=1}^k \mathbb{E} [\|\sigma_i v_i - w_i\|_F^2] + \sum_{i=k+1}^m \sigma_i^2 \quad (5.5)$$

$$\leq \|A - A_k\|_F^2 + \frac{k}{s} \|E\|_F^2. \quad (5.6)$$

□

Theorem 5.1. *After T iterations, Algorithm 5.1 obtains a sample S such that*

$$\mathbb{E} [\|A - \tilde{A}\|_F^2] \leq \frac{1}{1 - (k/s)} \|A - A_k\|_F^2 + \left(\frac{k}{s}\right)^T \|A\|_F^2,$$

where $\tilde{A} \equiv \mathcal{P}_{\text{span}(S)}(A)$.

Proof. Let's define the expected approximation error after round t as

$$\epsilon(t) \equiv \|A - \mathcal{P}_{\text{span}(S)}(A)\|_F^2,$$

and for shorthand, denote the optimal rank- k projection error as $\mathcal{O} \equiv \|A - A_k\|_F^2$, and $c \equiv \frac{k}{s}$. Then we have the following recurrence relation:

$$\text{base case: } \epsilon(1) \leq \mathcal{O} + \frac{k}{s} \|A\|_F^2$$

$$\text{inductive case: } \epsilon(t) \leq \mathcal{O} + \frac{k}{s} \|E\|_F^2 \leq \mathcal{O} + \frac{k}{s} \epsilon(t-1),$$

using the fact that the residual $\|E\|_F^2$ is less than the error from the previous

iteration. Expanding the recurrence, we get

$$\begin{aligned}
\epsilon(t) &\leq \mathcal{O} + c\epsilon(t-1) \\
&\leq \mathcal{O} + c(\mathcal{O} + c\epsilon(t-2)) \\
&\leq \mathcal{O} + c(\mathcal{O} + c(\mathcal{O} + c(\mathcal{O} + \dots + c(\mathcal{O} + c\|A\|_F^2)\dots)) \\
&= \sum_{k=0}^{t-1} c^k \mathcal{O} + c^t \|A\|_F^2 = \frac{1-c^t}{1-c} \mathcal{O} + c^t \|A\|_F^2 \\
&\leq \frac{1}{1-c} \mathcal{O} + c^t \|A\|_F^2 \\
&\equiv \frac{1}{1-(k/s)} \|A - A_k\|_F^2 + \left(\frac{k}{s}\right)^t \|A\|_F^2
\end{aligned}$$

□

6 Randomly projecting to lower dimensions: RANDPROJ

In the last two sections, we saw two methods that attempted to approximate the column or row space of a matrix A by constructing a smaller matrix of dimensions $m \times s$ or $s \times s$ comprised of rows and columns from the original matrix, in an effort to ensure similar row or column spaces. An alternative way to find a small matrix with the same range as A is by trying to capture the range of the matrix by applying it to a diverse set of random vectors $\Omega = [\omega_1 \dots \omega_s]$. We would expect that if Ω is diverse enough, then $A\Omega$ can generally capture $\text{range}(A)$.

From another perspective, it is known that random projections of high dimensional data are norm-preserving with high probability. This is a consequence of the *Johnson-Lindenstrauss Lemma*, see (Dasgupta and Gupta, 2003). Thus, the algorithm can be thought of as randomly projecting the data down to a lower dimensional space, in which it is not so different in a $\|\cdot\|_F^2$ sense, but in which it is much cheaper to compute decompositions. The algorithm is detailed in Algorithm 6.1.

Running time

Computing the matrix-matrix product $Y = A\Omega$ takes $O(mns)$ (but, see §6.2 for an improvement). Computing Q then takes $O(ms^2)$, for a total running time of $(mns + ms^2)$.

6.1 Analysis

There are several possible choices for obtaining Ω , the simplest being that each element is drawn of a zero-mean, unit-variance normal distribution: $\Omega(i, j) \sim \mathcal{N}(0, 1)$. Using Gaussian random vectors also implies that the running time is $O(mns)$, where the bottleneck computation is computing $A\Omega$. For other choices of random vectors, mentioned in §6.2, this matrix-matrix multiply can actually be performed in $O(mn \log s)$.

To bound the error of the algorithm, we first state a theorem from (Halko et al., 2009) (Theorem 9.1, p. 51) omitting most of the technical details:

Theorem 6.1 (Deterministic random projection error bound). *Let A be an*

Algorithm 6.1 RANDPROJ: obtain a low-rank factorization via random projections.

Input: $m \times n$ matrix A , target rank k , number of samples $s = k + p$.

Output: $\tilde{A} \in \mathbb{R}^{m \times n}$ which *approximately* minimizes $\min_{\text{rank}(A') \leq k} \|A - A'\|_F^2$ in time $O(mns + ms^2)$.

1. Draw a random test matrix $\Omega_{n \times s}$.
 2. Form the product $Y_{m \times s} = A\Omega$.
 3. Compute an orthornormal basis $Q_{m \times k}$ for the range of Y via SVD.
 4. Return $\tilde{A} = QQ^T A$.
-

$m \times n$ matrix, whose singular value decomposition we partition as

$$A = U\Sigma V^T = U \begin{bmatrix} k & n-k \\ \Sigma_1 & \Sigma_2 \end{bmatrix} \begin{bmatrix} n \\ V_1^T \\ V_2^T \end{bmatrix} \begin{matrix} k \\ n-k \end{matrix} \quad (6.1)$$

Let $\Omega_1 = V_1^T \Omega$ and $\Omega_2 = V_2^T \Omega$. Let \tilde{A} be as in Algorithm 6.1. Then we have the following, deterministic bound on the random projection:

$$\|A - \tilde{A}\|_F^2 \leq \|\Sigma_2\|_F^2 + \|\Sigma_2 \Omega_2 \Omega_1^\dagger\|_F^2,$$

where Ω_1^\dagger denotes the pseudo-inverse of Ω_1 .

The proof can be found in (Halko et al., 2009). It is quite involved and relies on perturbation theory; beyond the scope of this review. The key idea is that $\Sigma_1 \Omega_1$ captures most of the range of the matrix; if the true rank of A is k , then $\Sigma_1 \Omega_1$ almost surely captures the entire range. The remainder of the range, $\Sigma_2 \Omega_2$, is treated as a noise we wish to ignore; in the analysis, it is treated as a perturbation of the ideal setting where $\Sigma_2 = 0$.

We also use a few properties of expectations of Gaussian random matrices, from (Halko et al., 2009), Propositions 10.1 and 10.2:

Theorem 6.2 (Gaussian random matrix properties). *Let G be a normal Gaussian matrix with dimensions $k \times (k + p)$, and B and C be any fixed*

matrices with compatible dimensions. Then

$$(1) \quad \mathbb{E} [\|BGC\|_F^2] = \|B\|_F^2 \|C\|_F^2 \quad (6.2)$$

$$(2) \quad \mathbb{E} [\|G^\dagger\|_F^2] \leq \frac{k}{p-1} \quad (6.3)$$

Theorem 6.3. Let \tilde{A} be obtained from Algorithm 6.1, with number of samples $s = k + p$. Then

$$\mathbb{E} [\|A - \tilde{A}\|_F^2] \leq \left(1 + \frac{k}{p-1}\right) \|A - A_k\|_F^2.$$

Proof. Using Theorem 6.1 and its partitioning of Σ and Ω , we have that

$$\mathbb{E}_\Omega [\|A - \tilde{A}\|_F^2] \leq \|\Sigma_2\|_F^2 + \mathbb{E}_{\Omega_1, \Omega_2} [\|\Sigma_2 \Omega_2 \Omega_1^\dagger\|_F^2]$$

with expectation with respect to Ω . We can first take expectation with respect to Ω_2 , and then w.r.t Ω_1 of the last term from the above equation.

$$\mathbb{E}_{\Omega_1, \Omega_2} [\|\Sigma_2 \Omega_2 \Omega_1^\dagger\|_F^2] = \mathbb{E}_{\Omega_1} [\|\Sigma_2\|_F^2 \|\Omega_1^\dagger\|_F^2] \leq \frac{k}{p-1} \|\Sigma_2\|_F^2$$

in which we used Equation 6.2 and Equation 6.3. \square

6.2 Refinements

The simplicity of the random projection technique lends itself to several improvements, depending on the properties of the matrix A we are dealing with.

Power iterations

When the singular values of A decay very slowly, the method fails to work as well. This is intuitive since the singular vectors associated with the tail singular values $\sigma_{k+1} \geq \sigma_{k+2} \geq \dots \geq \sigma_{\min\{m,n\}}$ capture a significant fraction of the range of A , and thus a significant fraction of the range of the random projection $Y = A\Omega$. To remedy this, it would be nice to downweigh the importance of the directions associated with these singular values without

corrupting the singular vectors themselves. Applying a few rounds of power iteration will achieve this:

$$B = (AA^T)^q A\Omega = U\Sigma^{2q+1}V^T\Omega$$

Applying q power iterations like so preserves the singular vectors, but “peakifies” the singular value decay to $\sigma_i(B) = \sigma_i^{2q+1}(A)$, exponentially increasing the relative importance of the larger singular vectors. It is not surprising that this provides improved error bounds of the form

$$\mathbb{E} \left[\|A - \tilde{A}\|_F^2 \right] \leq \left(1 + \frac{k}{p-1} \right)^{1/2q+1} \|A - A_k\|_F^2,$$

but it comes with the cost of $2q$ extra matrix multiplications requiring $2qmn$ s more flops than before.

Faster random multiplication with structured matrices

It turns out that for matrices $\Omega_{n \times \ell}$ with special structure, matrix multiplication can be performed in $O(mn \log \ell)$ time instead of the standard $O(mn\ell)$, which is the bottleneck of the random projection method. This is the case with *subsampled random Fourier transform* (SRFT) matrices. SRFT matrices allow the $O(mn \log \ell)$ matrix multiplication via Fast Fourier Transform (FFT) techniques, a description of which is beyond the scope of this paper—see (Woolfe et al., 2008). The authors in (Halko et al., 2009) note that it is difficult to analyze and bound accuracy using the SRFT, but it works comparably to using standard Gaussian random projections in practice.

Method, Section	Running Time	# Passes	Multicore?	Error $\mathbb{E} \left[\ A - \tilde{A}\ _F^2 \right]$ w.h.p.
SAMPLECOLS, §4.1	$O(mn + ms^2)$	≤ 2	yes	$\leq \mathcal{O} + 2\sqrt{\frac{k}{s}} \ A\ _F^2$
SAMPLEROWSCOLS, §4	$O(mn + s^3)$	≤ 2	yes	$\leq \mathcal{O} + O\left(\sqrt[4]{\frac{k^5}{s}}\right) \ A\ _F^2$
ADASAMPLE, §5	$O(mnsT + (m+n)(sT)^2)$	$2T$	per iter.	$\leq \frac{1}{1-k/s} \mathcal{O} + \left(\frac{k}{s}\right)^T \ A\ _F^2$
RANDPROJ, §6	$O(mns + ms^2)$	≤ 2	yes	$\leq \frac{1-1/s}{1-k/s-1/s} \mathcal{O}$
RANDPROJ + power iter, §6.2	$O((q+1)mns + ms^2)$	$2q$	yes	$\leq \left(\frac{1-1/s}{1-k/s-1/s}\right)^{\frac{1}{2q+1}} \mathcal{O}$
RANDPROJ + SRFT, §6.2	$O(mn \log s + ms^2)$	2	no	empirically comparable
Partial SVD or QR (exact)	$O(mnk)$	k	no	$\mathcal{O} = \ A - A_k\ _F^2$

Table 2: Summary of different methods, explained in §7. Matrix A is $m \times n$, k is the desired rank, s is number of samples, T is number of adaptive iterations, q is the number of power iterations, and $\mathcal{O} = \|A - A_k\|_F^2$ stands for the optimal solution.

7 Comparison of methods

In the previous three sections, we have discussed three different methods for approximating low rank matrix decomposition. Each comes with its own expected error bounds and running times. The pointed question now becomes: *Which one is the best?* We try to answer the question along 4 different important axes which are relevant in a modern computational environment: complexity, accuracy, passes through the data, and ease of parallelization. A summary is in Table 2.

7.1 Running time versus expected error

An attempt to relate the methods to each other for the common case is in Figure 2. There, we fix k and s for all methods, and also assume that additive errors always outweigh the multiplicative errors, which holds for sufficiently large $\|A\|_F^2$ (i.e., for large enough matrices). There is a direct trade-off between accuracy and computation time across the four methods—

however, this trade-off is not necessarily linear, as the illustration suggests. The figure only preserves both the horizontal (error) and vertical (runtime) orderings of the methods. The shape of the trade-off depends on the relative size of m , n , k and s . As s increases, all methods approach the optimum asymptotically at a polynomial rate. Combining RANDPROJ with power iterations, we can approach the optimum exponentially fast; this is essentially a hybrid of random approaches and the traditional power method. Also, as we increase the number of iterations of adaptive sampling in ADASAMPLE, we approach a multiplicative error bound that is essentially equal to the error of RANDPROJ when not using power iterations.

Most of the trade-offs are intuitive in that more work yields tighter bounds: SAMPLEROWSCOLS is faster than SAMPLECOLS, but pays the price of a looser bound due to the error incurred by sampling rows as well. ADASAMPLE obtains a multiplicative bound by multiple rounds of work that SAMPLECOLS performs. For slightly less work than ADASAMPLE, RANDPROJ has a tighter bound. This may be due to the limitations of the sampling methods which can only approximate the range of A from a basis spanned by columns of the original matrix. One possible argument is that random combinations of the columns as in RANDPROJ make it easier to cover the entire range with the same number of samples.

Another issue in the comparison of these methods is that the bounds are not tight. In most proofs, upper bounds were obtained by dropping terms, applying Cauchy-Schwartz, and making other worst-case analysis arguments. The important properties to note are the coarse ones: multiplicative versus additive error, and that all bounds are polynomial in k and $1/s$. This makes a strong case for an empirical comparison on realistic matrices. To my knowledge, neither SAMPLEROWSCOLS nor ADASAMPLE have been tried on real data; Halko et al. (2009) demonstrated the benefits of RANDPROJ over RRQR on several real applications.

7.2 Passes through the data

The basic methods visit the input matrix twice: once to compute sampling distributions (SAMPLECOLS, SAMPLEROWSCOLS) or project (RANDPROJ), and an optional second pass to form \hat{A} . In ADASAMPLE, the input needs to be visited twice per iteration: once to compute $\mathcal{P}_S(A)$, and once to compute $E \leftarrow A - E - \mathcal{P}_{S_t}(A)$. Parameters T and q in these methods are essentially constant with respect to k , so all methods make many fewer passes over

the data than traditional methods, such as RRQR. The more data access becomes a bottleneck, the more desirable the randomized approaches over the traditional methods.

7.3 Parallelizability

Methods SAMPLECOLS and SAMPLEROWSCOLS require sampling distributions, where each column or row can be processed completely independently, and are thus trivially parallelizable to reduce the $O(mn)$ term down to $O(mn/\ell)$, where ℓ is the number of CPUs available. ADASAMPLE is iterative, and requires as much work as the classical methods in each iteration. Finally, the bottleneck of RANDPROJ is the multiplication $A\Omega$, which can be parallelized as well, but this is not possible when performing power iterations or performing the matrix multiplication via FFT, as in the case with SRFT matrices.

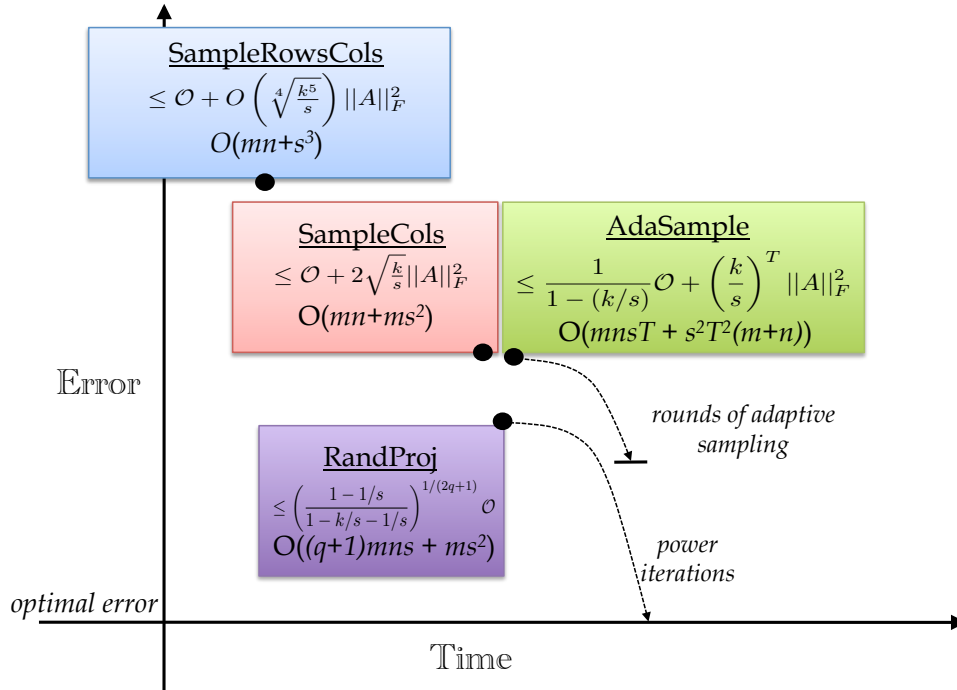


Figure 2: A cartoon illustration comparing running time versus expected error for all random methods. The relationships here are meant to coarsely reflect the statements in Table 2. For discussion, see §7.1. The trade-off between time and accuracy across the different methods is not necessarily linear as the illustration suggests.

8 Conclusion

In this review, we have presented several randomized methods which address the fixed-rank low-rank matrix factorization problem. We went over the standard methods for solving this problem (partial SVD and rank revealing QR decomposition) and discussed their shortcomings when applied to modern large datasets: they require random access to the data, many passes over the input and are iterative, thus not easily parallelized. We presented 3 basic methods of randomization— (i) independently sampling rows and columns, (ii) conditionally sampling rows and columns, and (iii) taking random combinations of columns—all attempting to capture the range of the matrix in

a much lower dimension.

It is clear that the randomized algorithms provide several attractive advantages over standard approaches, in terms of parallelizability, minimal data access, and allowing a trade-off between accuracy and computation by changing the amount of sampling. When fixing the sampling, each method has its own accuracy-computation trade-off point. The best accuracy-computation ratio on any specific dataset depends on the relative sizes of m , n , k and $\|A\|_F^2$.

Finally, an additional heretofore unspoken benefit of these algorithms is their simplicity—they are numerically stable, only have a few parameters to tune, and each takes only a few lines of code to implement, which call standard BLAS operations on low-dimensional data.

In light of the simplicity of the methods, and their different accuracy-computation operating points depending on the application, it makes sense for the practitioner to try a variety of these methods in practice. We leave it as future work to perform rigorous empirical tests to analyse the performance of these methods on real problems.

References

- S. Dasgupta and A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.
- A. Deshpande and L. Rademacher. Efficient volume sampling for row/column subset selection. In *Foundations of Computer Science*, pages 329–338. IEEE, 2010.
- A. Deshpande and S. Vempala. Adaptive sampling and fast low-rank matrix approximation. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 292–303, 2006.
- P. Drineas, R. Kannan, and M.W. Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2007.
- Carl Eckart and Gale Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1:211–218, 1936. 10.1007/BF02288367.
- A. Ēivriļ and M. Magdon-Ismail. On selecting a maximum volume submatrix of a matrix and related problems. *Theoretical Computer Science*, 410(47-49):4801–4811, 2009. ISSN 0304-3975.
- A. Frieze, R. Kannan, and S. Vempala. Fast Monte-Carlo algorithms for finding low-rank approximations. *Journal of the ACM*, 51(6):1025–1041, 2004.
- G. Golub and C. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1984.
- Ming Gu and Stanley C. Eisenstat. Efficient algorithms for computing a strong rank-revealing qr factorization. *SIAM Journal on Scientific Computing*, 17:848–869, July 1996.
- N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, September 2009.
- W.B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Math*, 26:189–206, 1984.

- Ravindran Kannan and Santosh Vempala. Spectral algorithms. *Foundations and Trends in Theoretical Computer Science*, 4:157–288, 2008.
- M.J. Kearns and U.V. Vazirani. *An introduction to computational learning theory*. The MIT Press, 1994.
- P.G. Martinsson, V. Rokhlin, and M. Tygert. A Randomized Algorithm for the Approximation of Matrices. 2006a.
- P.G. Martinsson, VA Rokhlin, and M. Tygert. A randomized algorithm for the approximation of matrices. Technical report, Yale University, Department of Computer Science, 2006b.
- L. Mirsky. A trace inequality of John von Neumann. *Monatshefte fur mathematik*, 79(4):303–306, 1975.
- N.H. Nguyen, T.T. Do, and T.D. Tran. A fast and efficient algorithm for low-rank approximation of a matrix. In *Proceedings of Symposium on Theory of Computing*, pages 215–224. ACM, 2009.
- C.H. Papadimitriou, H. Tamaki, P. Raghavan, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *Proceedings ACM Symposium on Principles of Database Systems*, pages 159–168. ACM, 1998. ISBN 0897919963.
- AF Ruston. Auerbach’s theorem and tensor products of Banach spaces. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 58, pages 476–480. Cambridge Univ Press, 1962.
- Tamas Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science*, pages 143–152, oct. 2006.
- G. Strang. *Introduction to Linear Algebra*. Wellesley - Cambridge Press, 2009.
- F. Woolfe, E. Liberty, V. Rokhlin, and M. Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25:335–336, 2008.