PCMI Summer School Lectures, July 11 - 16, 2016:

Randomized algorithms for matrix computations and analysis of high dimensional data

Lecturer: Per-Gunnar Martinsson, Dept. of Applied Mathematics, Univ. of Colorado Boulder
TA: Nathan Heavner, Dept. of Applied Mathematics, Univ. of Colorado Boulder

(1). **Introduction.** These lectures will describe a set of highly computationally efficient techniques for computing low rank approximations to matrices. The techniques are based on randomized projections and achieve high computational efficiency when applied to large data sets and when executed on "modern" computing platforms such as multicore CPUs, GPUs, distributed memory parallel machines, and mobile processors (phones, etc). For concreteness, let us state in detail one of the computational problems we will address:

Model problem: Let $\mathbf{A}$ be an $m \times n$ matrix of approximate rank k, meaning that for some specified tolerance ε, there exists a matrix $\mathbf{A}_k$ of exact rank k such that $\|\mathbf{A} - \mathbf{A}_k\| \leq \varepsilon$. Then compute an approximate rank-k Singular Value Decomposition (SVD) of $\mathbf{A}$ in the form

$$\begin{array}{ccccc} \mathbf{A} & \approx & \mathbf{U} & \mathbf{D} & \mathbf{V}^*, \\ m \times n & & m \times k & k \times k & k \times n \end{array}$$

where $\mathbf{U}$ and $\mathbf{V}$ are matrices with orthonormal columns, and where $\mathbf{D}$ is diagonal.

While the lectures will focus on techniques for computing an approximate partial Singular Value Decomposition (SVD), the methods can with very minor modifications be used to compute a partial eigenvalue decomposition (EVD), determining spanning columns or rows of the matrix, etc. The low-rank approximation problems addressed arise frequently in computational statistics and in data mining, in areas such as Principal Component Analysis (PCA), Latent Semantic Indexing (LSI), PageRank (an algorithm that formed an important part of the original Google search engine), and many more.

The problem we address is a classical one, and many techniques for solving it have been developed over the years. These techniques include Rank-Revealing QR factorizations, Krylov methods, subspace iteration, and many more. Most of these techniques were originally developed for a computational environment that was quite different than the one we face now. One difference is that many of the data sets we are interested in these days are far too large to store in main memory (RAM) of a computer; instead they might be stored on a hard drive, on flash memory, distributed in "the cloud," etc. Another difference is that when the classical algorithms were developed, the key limitation on computing power was how many *floating point operations* one could afford to expend. These days, it is often *communication* that is the rate limiting factor. Here, the word "communication" should be interpreted in a broad sense: communication between a hard drive and RAM, communication between processors in a parallel machine, communication incurred when querying a data set stored "in the cloud," communication between levels of cache near the CPU, etc.

(2). **Resources.** The manuscript [10] (available at `http://arxiv.org/abs/1607.01649`) provides details and background on the material covered. Supplementary material can be found at:

`http://amath.colorado.edu/faculty/martinss/2016_PCMI/`

For the more theoretical material covered, the primary reference is the survey paper [3]. The surveys [6] and [13] provide complementary reading.

(3). **The two-stage approach to low-rank approximation.** The model problem we introduced in Section (1) can advantageously be split into two parts:

Stage A — find an approximate range: Construct an $m \times k$ matrix $\mathbf{Q}$ with orthonormal columns such that $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$. This step will be executed via a randomized process described in Section (4).

Stage B — form a specific factorization: Given the matrix $\mathbf{Q}$ computed in Stage A, form factors $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$, via classical deterministic techniques. For instance, this stage can be executed via the following steps:
(1) Form the $k \times n$ matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$.
(2) Decompose the matrix $\mathbf{B}$ in a singular value decomposition $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^*$.
(3) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.

The algorithms we describe use *randomized* techniques for solving Stage A. Then classical deterministic methods are used to execute Stage B.

In Section (12) we describe generalizations of the computational template introduced to problems beyond low rank approximation. Specifically, we will described other problems where highly efficient algorithms results by taking an approach where we first (in "Stage A") use randomized algorithms to develop a *sketch* of the data provided, with the purpose of telling us "where to look." Then once we have a sketch, we use deterministic techniques to compute a highly accurate solution to whatever problem was originally posed.

Reading suggestions: Sections 1.1 – 1.6 of [3] *and Sections 2.2 & 2.3 of* [10].

(4). **A randomized range-finder.** Let us describe a simple randomized sampling algorithm for solving the problem we introduced as "Stage A" in Section (3) — namely, how to find an orthonormal basis $\{\mathbf{q}_j\}_{j=1}^k$ that approximately spans the column space of a given $m \times n$ matrix $\mathbf{A}$. (The vectors $\{\mathbf{q}_j\}$ are the columns of the matrix $\mathbf{Q}$.) The algorithm proceeds as follows:

(1) Form a set of k random Gaussian vectors $\{\mathbf{g}_j\}_{j=1}^k$.
(2) Form a set $\{\mathbf{y}_j\}_{j=1}^k$ of samples from the column space of $\mathbf{A}$ where $\mathbf{y}_j = \mathbf{A}\mathbf{g}_j$.
(3) Perform Gram-Schmidt on the set $\{\mathbf{y}_j\}_{j=1}^k$ to form the ON-basis $\{\mathbf{q}_j\}_{j=1}^k$ for the space spanned by the vectors $\{\mathbf{y}_j\}_{j=1}^k$.

If the matrix $\mathbf{A}$ has *exact* rank k, one can prove that with probability one, the vectors $\{\mathbf{q}_j\}_{j=1}^k$ form an ON basis for the column space of $\mathbf{A}$. In this case, $\mathbf{A} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$, where $\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2\ \cdots\ \mathbf{q}_k]$ and so $\mathbf{Q}\mathbf{Q}^*$ is the orthogonal projection onto the span of the vectors $\{\mathbf{y}\}_{j=1}^k$.

Now observe that the k matrix-vector products are independent and can advantageously be executed in parallel. We the find that the algorithm can be written in matrix form as (with Matlab commands in red on the right):

(1) Draw an $n \times k$ Gaussian random matrix $\mathbf{G}$. `G = randn(n,k)`
(2) Form the $m \times k$ sampling matrix $\mathbf{Y} = \mathbf{A}\mathbf{G}$. `Y = A*G`
(3) Orthonormalize the columns of $\mathbf{Y}$ to obtain the $m \times k$ ON matrix $\mathbf{Q}$. `[Q,~] = qr(Y,0)`

What is perhaps surprising is that even in cases where $\mathbf{A}$ does not have exact rank k, the very simple procedure outlined often works very well. One complication arises in that in the general case, one needs to do a slight amount of *over-sampling*. In other words, we need to draw a few extra samples in building a basis for the column space. Here, "a few" means 5 or 10 extra, or some such. (It is a remarkable fact that these are universal numbers, and do not depend on either m or n.)

Putting all the observations we have made together, we obtain a full algorithm for computing an approximate SVD, which we summarize in Figure 1. This simple algorithm often works well, but we will in the lectures discuss

```
ALGORITHM: BASIC RANDOMIZED SVD

Inputs: An $m \times n$ matrix $\mathbf{A}$, a target rank $k$, and an over-sampling parameter $p$ (say $p = 10$).

Outputs: Matrices $\mathbf{U}$, $\mathbf{D}$, and $\mathbf{V}$ in an approximate rank-$(k+p)$ SVD of $\mathbf{A}$. (I.e. $\mathbf{U}$ and $\mathbf{V}$ are orthonormal
and $\mathbf{D}$ is diagonal.)

Stage A:
    (1) Form an $n \times (k+p)$ Gaussian random matrix $\mathbf{G}$.                G = randn(n,k+p)
    (2) Form the sample matrix $\mathbf{Y} = \mathbf{A}\,\mathbf{G}$.                Y = A*G
    (3) Orthonormalize the columns of $\mathbf{Y}$ to form the ON matrix $\mathbf{Q}$.   [Q,~] = qr(Y,0)
Stage B:
    (4) Form the $(k+p) \times n$ matrix $\mathbf{B} = \mathbf{Q}^{*}\mathbf{A}$.      B = Q'*A
    (5) Form the SVD of the small matrix $\mathbf{B}$ so that $\mathbf{B} = \hat{\mathbf{U}}\mathbf{D}\mathbf{V}^{*}$.   [Uhat,V,D] = svd(B,'econ')
    (6) Form $\mathbf{U} = \mathbf{Q}\hat{\mathbf{U}}$.                                U = Q*Uhat
```

FIGURE 1. A basic randomized algorithm. If a factorization of rank precisely k is desired, the factorization in Step 5 can be truncated to the k leading terms (corresponding to the Matlab command `Uhat = Uhat(:,1:k); D = D(1:k,1:k); V = V(:,1:k)`).

various modifications that improves the accuracy for matrices that are very "noisy" (in the sense that their singular values decay slowly), that allow us to execute the algorithm in "streaming environments" (where we cannot store the matrix at all), that allow us to determine the rank as part of the computation (as opposed to knowing it in advance), etc.

(5). **Single-pass algorithms.** The basic randomized algorithm described in Figure 1 has as its main virtue that it interacts with the given matrix $\mathbf{A}$ only twice: First on line (2) when we multiply $\mathbf{A}$ with the random matrix $\mathbf{G}$ and then on line (5) when we multiply $\mathbf{A}$ by the computed orthonormal matrix $\mathbf{Q}$.

It turns out to be possible to rearrange the computation a little to avoid the need to revisit the matrix. This enables us to compute a factorization of a matrix $\mathbf{A}$ that is so large that we cannot store it at all! Such algorithms are called *streaming algorithms* since they can be executed on a "stream" of data without storing it.

The streaming algorithms described in this section of the lectures are computationally efficient, but there is a price to be paid for giving up the second visit to the matrix, and that is that we incur slightly higher approximation errors.

Reading suggestions: Sections 2.4 of [10].

(6). **Cost analysis and the SRFT.** In this part of the lectures, we analyze the cost of randomized algorithms for computing a low-rank approximation to a matrix. These algorithms are intended for the situation where the approximate rank k is much smaller than the matrix dimensions m and n so that

$$k \ll \min(m, n).$$

In this environment, the cost of executing the algorithm in Figure 1 is dominated by the matrix-matrix multiplications on lines (2) and (5). These operations have cost $O(mnk)$, while all remaining operations have cost $O((m+n)\,k^2)$. (These estimates apply to general dense matrices stored in RAM. For sparse or structured matrices, the cost estimates change, of course.) One very important point here is the following:

> ***Fact:*** *The matrix-matrix multiplication is a simple operation that executes very rapidly in many different environments. It is particularly fast on parallel computing platforms such as multicore CPUs, or massively multicore GPUs. Highly optimized implementations of the matrix-matrix multiplication are often available for distributed memory parallel computers.*

One of the key benefits of the randomized algorithms is that they cast the costly $O(mnk)$ steps in terms of specifically matrix-matrix multiplications. We discuss the various implications of this observation.

In this part of the lectures, we will also describe how the asymptotic complexity of the low-rank approximation algorithms can be meaningfully reduced by using better random projections. The idea is that one can construct a random projection that on the one hand has enough structure that the matrix-matrix multiplication can be executed faster than $O(mnk)$ operations, and on the other hand is "random enough" that the computed result is highly accurate. Specifically, we will show that by using "structured" random matrices, and exploiting ideas from the "single-pass" randomized algorithms, the asymptotic complexity of computing a rank-k SVD of an $m \times n$ matrix can be reduced from $O(mnk)$ to $O(mn \log k)$.

There are many different options for how "structured" random projections can be implemented. In the lecture, we will focus on the so called sub-sampled random Fourier transform (SRFT). Random projections like these are sometimes referred to as "Fast Johnson-Lindenstrauss transforms."

Reading suggestions: Section 2.5 of [10]. *For foundational work, see* [14] *and* [1].

(7). **The accuracy of randomized algorithms.** Having described the computational cost of the randomized algorithms, the lectures will next discuss the *accuracy* of the low-rank approximations computed.

We will through numerical examples illustrate two points:

(1) For matrices whose singular values decay *rapidly,* the simple scheme in Figure 1 produces a factorization of close to optimal accuracy.
(2) For matrices whose singular values decay *slowly,* the basic randomized scheme produces much higher errors than an optimal scheme.

We will briefly state some theorems that formalize the observations made from empirical data. These theorems concern the errors resulting from the basic scheme in Figure 1 when it is applied to a matrix of size $m \times n$ with singular values $\{\sigma_j\}_{j=1}^{\min(m,n)}$. Let $\mathbf{A}_{\text{approx}}$ be the rank-k approximation to $\mathbf{A}$ computed by the scheme in Figure 1. We then define the error

$$e_k = \|\mathbf{A} - \mathbf{A}_{\text{approx}}\|.$$

Observe first that e_k is a random variable since it depends on the draw of the Gaussian random matrix $\mathbf{G}$.

- When the errors are measured in the Frobenius norm, the *expectation* of the error satisfies the bound

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_{\text{Fro}}\big] \le \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}. \tag{1}$$

  The bound (1) is very gratifying since the factor high-lighted in red is the theoretically minimal error incurred by a rank-k approximation to a matrix (as specified by the Eckart-Young theorem).
- When the errors are measured in the spectral norm, the *expectation* of the error satisfies the bound

$$\mathbb{E}\big[\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|\big] \le \left(1 + \sqrt{\frac{k}{p-1}}\right)\sigma_{k+1} + \frac{e\sqrt{k+p}}{p}\left(\sum_{j=k+1}^{\min(m,n)} \sigma_j^2\right)^{1/2}. \tag{2}$$

  The bound (2) is much less gratifying than the corresponding bound for the Frobenius norm (1). The reason is that in the spectral norm, the blue factor is the theoretical minimum. For matrices whose singular values decay slowly, the red factor is much larger, and the right hand side is therefore much larger than the theoretically optimal value.

- We will also show theorems that demonstrate that the risk of getting a large deviation from the expected value is extremely small. The reason is in some sense a result of the fact that the error depends on a large number of random variables. For a very large error to happen, a lot of these random variables have to "conspire" to seemingly act in unison.

Reading suggestions: Section 2.6 of [10].

(8). **Randomized power method and subspace iteration.** In this section of the lectures, we show how the basic algorithm in Figure 1 can be modified to produce close to optimally small errors even for matrices whose singular values decay slowly. The key idea is this: Suppose that A is an $n \times n$ symmetric matrix with eigen value decomposition

$$\mathsf{A} = \mathsf{U}\mathsf{D}\mathsf{U}^*,$$

where $\mathsf{D} = \texttt{diag}(\lambda_1, \lambda_2, \dots,)$ is a diagonal matrix holding the eigenvalues of A, ordered by magnitude so that

$$|\lambda_1| \geq |\lambda_2| \geq |\lambda_3| \geq \cdots$$

Our objective is now to compute an ON basis that approximately spans the same space as that spanned by the dominant k eigenvectors of A. We have seen that the basic algorithm in Figure 1 does an excellent job of this when the magnitudes of the eigenvalues of A decay rapidly. But suppose that they do not decay rapidly. We then pick a small integer q, and observe that the q'th power of A has the eigenvalue decomposition

$$\mathsf{A}^q = \mathsf{U}\mathsf{D}^q\mathsf{U}^*.$$

The key observation is now that A^q has precisely the same eigenvectors as A, but its eigenvalues decay much faster in magnitude. If we just change line (2) in Figure 1 to

$$\mathsf{Y} = \mathsf{A}^q\mathsf{G},$$

we will see that the new randomized algorithm will produce *far* more accurate results. In fact, one can prove that in this case

$$\mathbb{E}\Big[\|\mathsf{A} - \mathsf{A}^{\text{computed}}\|\Big] \leq \left(1 + 4\sqrt{\frac{2\,n}{k-1}}\right)^{1/q} |\lambda_{k+1}|. \tag{3}$$

Even for small q (say $q = 2$ or $q = 3$), the bound (3) is often much tighter than (2).

Of course, applying the accuracy enhancement technique described here will increase the computational cost since more applications of A to tall thin matrices are required.

In the lectures, we will describe the "power method" outlined here in more detail, and will also discuss some numerical issues that arise due to ill-conditioning of the matrices involved.

Remark 1. In this summary, we discussed a symmetric matrix just to keep things simple and familiar. The methods work just as well for general matrices. To be precise, suppose that A is an $m \times n$ matrix with singular value decomposition

$$\mathsf{A} = \mathsf{U}\mathsf{D}\mathsf{V}^*.$$

We then pick a small integer q and form the new matrix

$$\mathsf{A}^{(q)} = \big(\mathsf{A}\mathsf{A}^*\big)^q\mathsf{A}.$$

It is easily verified that this new matrix $\mathsf{A}^{(q)}$ has the SVD

$$\mathsf{A} = \mathsf{U}\mathsf{D}^{2q+1}\mathsf{V}^*.$$

In other words, A and $\mathsf{A}^{(q)}$ have exactly the same left singular vectors, but the singular values of $\mathsf{A}^{(q)}$ decay much faster. The replacement for Line (2) in Figure 1 is now

$$\mathsf{Y} = \big(\mathsf{A}\mathsf{A}^*\big)^q\mathsf{A}\mathsf{G}.$$

Reading suggestions: Subspace iteration is a classical method that is covered in many standard text books on numerical linear algebra. The connection to randomized projections is described in [10, Sec. 2.7] *and* [9, Sec. 4.5] *For original work, see* [11].

(9). **The Interpolative Decomposition (ID) and the CUR.** Up to this point in the lectures, the discussion has focussed on algorithms for computing approximations to the dominant singular values and singular vectors (or eigenvalues and eigenvectors). But in many applications, we are of course interested in other matrix factorizations. For instance, recently popularized factorizations such as the "interpolative decomposition (ID)," or the closely related CUR factorization have many advantages both in terms of computational efficiency and in terms of opportunities for data interpretation. As an illustration, consider a matrix A of size $m \times n$ and rank k, where $k < \min(m, n)$. Any such matrix admits a so called "interpolative decomposition (ID)" which takes the form

$$\begin{array}{ccccc} \mathsf{A} & = & \mathsf{C} & \mathsf{Z}, \\ m \times n & & m \times k & k \times n \end{array} \tag{4}$$

where the matrix C is given by a subset of the columns of A and where Z is well-conditioned in a sense that we will make precise shortly. The ID has several advantages, as compared to, e.g., the QR or SVD factorizations:

- If A is sparse or non-negative, then C shares these properties.
- The ID require less memory to store than either the QR or the SVD.
- Finding the indices associated with the spanning columns is often helpful in *data interpretation.*
- In the context of numerical algorithms for discretizing PDEs and integral equations, the ID often preserves "the physics" of a problem in a way that the QR or SVD do not.

In this lecture, we briefly introduce the ID and the CUR decompositions, describe why they are of interest, and review both deterministic and randomized techniques for computing them.

Reading suggestions: [10, Ch. 4]*, and* [12]. *For original work, see* [7, 5, 2, 8].

(10). **Theoretical analysis of randomized algorithms.** In Lecture 1, we briefly discussed theoretical results that provide bounds on the expected error of the basic randomized scheme in Figure 1. In the last of the three lectures, we will look at how theorems of this type can be proven. In this discussion, we will in particular investigate spectral properties of Gaussian random matrices. As an example, consider an $m \times n$ Gaussian matrix G, where m and n are reasonably large, and where m is much larger than n (in other words, G is very tall and thin). It is well known that with high probability, the norm of each column is about $\sqrt{m}$, and, importantly, these columns form almost an orthogonal set. We will demonstrate that the success of randomized algorithms such as the one in Figure 1 is closely related to the fact that rectangular Gaussian matrices are well-conditioned. In particular, this property kicks in very rapidly as m grows larger than n. For instance, even a matrix of size $(n + 10) \times n$ can be proven to be reasonably well conditioned, with high likelihood.

Reading suggestions: Part III of [3].

(11). **Adaptive rank determination and blocked algorithms.** Time permitting, we will in this part of the lectures discuss how to design randomized algorithms for low rank factorization that perform well even in situations where there is a high degree of uncertainty about their numerical rank. (Observe that the basic algorithm in Figure 1 assumes that the rank is known in advance.) We will describe two classes of techniques:

- *Techniques that update the matrix:* When the matrix A to be factored can easily be updated (such as, e.g., a dense matrix stored in RAM), then randomized techniques can be implemented that are *guaranteed* to meet any requested tolerance.
- *Techniques that do not touch the matrix:* When A cannot be updated, we rely on randomized techniques to *estimate* the residual error at any step.

Reading suggestions: Sections 2.9 and 2.10 of [10].

(12). **Randomized pre-conditioning and randomized nearest neighbor search.** The final topic (if time permits) will be to discuss generalizations of the two-stage approach described in Section (3) to address other problems where the high dimensionality of a data set presents the main challenge.

As an illustration, consider the following version of "nearest neighbor search": Suppose you are given N points $\{\boldsymbol{x}_j\}_{j=1}^N$ in $\mathbb{R}^D$. How do you find the k nearest neighbors for every point when the ambient dimension D is large? This is a classical curse-of-dimensionality problem since the standard techniques (such as those based on tree-searches associated with a hierarchical partitioning of space) become prohibitively expensive in high dimensions. It was recently demonstrated by Jones, Osipov, and Rokhlin that the problem is highly tractable to the following randomized approach:

- *Randomized probing of the data:* Use a Johnson-Lindenstrauss random projection to map the N-particle problem in $\mathbb{R}^D$ (where D is large) to an N-particle problem in $\mathbb{R}^d$ where $d \sim \log N$. Run a deterministic nearest-neighbor search in $\mathbb{R}^d$ and store a list of the ℓ nearest neighbors for each particle (for simplicity, one can set $\ell = k$). Then repeat the process several times. If for a given particle a previously undetected neighbor is discovered, then simply add it to a list of potential neighbors.
- *Deterministic post-processing:* The randomized probing will result in a list of putative neighbors that typically contains more than k elements. But it is now easy to compute the pairwise distances in the original space $\mathbb{R}^D$ to judge which candidates in the list are the k nearest neighbors.

Observe that if only a single randomized projection were used, a very unreliable list of nearest neighbors would result. The key to high reliability is to substantially over-sample the point cloud by taking several randomized "snap-shots."

The idea is here is that while Johnson-Lindenstrauss type techniques result in quite large distortions, we can still get highly accurate and reliable answers out of the computation, as long as we only use the randomized methods to construct a *sketch* of the data set.

Reading suggestions: [4, 13]

APPROXIMATE TIME-LINE

Lecture 1:

- Introduction.
- A two-stage approach to low-rank approximation.
- A randomized algorithm for determining a basis for the range of a matrix.
- The accuracy of randomized algorithms.

Lecture 2:

- The accuracy of randomized algorithms. (Continued from Lecture 1.)
- Randomized power method and subspace iteration.
- The Interpolative Decomposition (ID) and the CUR decomposition.

Lecture 3:

- The Interpolative Decomposition (ID) and the CUR decomposition. (Continued from Lecture 2.)
- Theory of randomized methods for low-rank approximation. (Review of proofs of main theorems.)
- Adaptive rank determination and blocked algorithms. (If time permits.)
- Randomized pre-conditioning and randomized nearest neighbor search. (If time permits.)

References

[1] N. Ailon and B. Chazelle. Approximate nearest neighbors and the fast Johnson–Lindenstrauss transform. In *STOC '06: Proc. 38th Ann. ACM Symp. Theory of Computing*, pages 557–563, 2006.

[2] H. Cheng, Z. Gimbutas, P.G. Martinsson, and V. Rokhlin. On the compression of low rank matrices. *SIAM Journal of Scientific Computing*, 26(4):1389–1404, 2005.

[3] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[4] Peter W Jones, Andrei Osipov, and Vladimir Rokhlin. A randomized approximate nearest neighbors algorithm. *Applied and Computational Harmonic Analysis*, 34(3):415–444, 2013.

[5] Edo Liberty, Franco Woolfe, Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci. USA*, 104(51):20167–20172, 2007.

[6] Michael W Mahoney. Randomized algorithms for matrices and data. *Foundations and Trends® in Machine Learning*, 3(2):123–224, 2011.

[7] Michael W Mahoney and Petros Drineas. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702, 2009.

[8] Per-Gunnar Martinsson, Vladimir Rokhlin, and Mark Tygert. A randomized algorithm for the approximation of matrices. Technical Report Yale CS research report YALEU/DCS/RR-1361, Yale University, Computer Science Department, 2006.

[9] P.G. Martinsson. A fast randomized algorithm for computing a hierarchically semiseparable representation of a matrix. *SIAM Journal on Matrix Analysis and Applications*, 32(4):1251–1274, 2011.

[10] P.G. Martinsson. Randomized methods for matrix computations and analysis of high dimensional data, 2016. Arxiv.org manuscript #1607.01649.

[11] Vladimir Rokhlin, Arthur Szlam, and Mark Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2009.

[12] S. Voronin and P.G. Martinsson. A CUR factorization algorithm based on the interpolative decomposition. *arXiv.org*, 1412.8447, 2014.

[13] David P Woodruff. Sketching as a tool for numerical linear algebra. *arXiv preprint arXiv:1411.4357*, 2014.

[14] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335–366, 2008.