Finding Structure with Randomness

Probabilistic Algorithms for Approximate Matrix Decompositions
Research Project Report – "Sparsity and Compressed Sensing" course

Lilian Besson*
Department of Mathematics
École Normale Supérieure de Cachan (France)
lilian.besson@ens-cachan.fr

Abstract

Matrix factorization is a powerful tool to achieve tasks efficiently in numerical linear algebra. A problem arises when we compute low-rank approximations for massive matrices: we have to reduce the algorithms' complexity in time to hope to be efficient. A way to adapt these techniques for such computational environments is randomization. This report presents a framework for these new techniques, mainly based on the recent work by N. Halko, P.-G. Martinsson and J. Tropp presented in "*Finding Structure with Randomness: Probabilistic Algorithms for Constructing approximate Matrix Decompositions*" [HMT11, Tro14]. The main intuition behind the various algorithms presented herein is using random sampling to apprehend the action of the matrix in a "compressed" subspace. We can apply then the classical methods on the resulting matrix – the one acting on the "compressed" subspace – to obtain a low-rank approximation. Another application that rises from this explanation is the fact that this method is thus more robust addressing incomplete data sets that one can get in information sciences, while other. The benefits of such methods will depend on the matrix.

We will decline the results for two main classes: *dense matrices* where the new complexity is about $O(mn\log(k))$, compared to $O(mnk)$ for classical methods – k being the numerical rank; *sparse matrices*, and in general, structured matrices. [HMT11] also studied *massive matrices* that do not fit in fast memory (*RAM*) for which the access time – which surpasses computation time – can be reduced to one pass, comparing to $O(k)$ passes, but due to space constraint we will not present it here.

A clean Octave implementation was developed, and we will present 2 experiments, done on both dense and sparse matrices, in order to confirm the theoretical analysis.

*If needed, see on-line at http://lbo.k.vu/pcs2016 for an e-version of this report, as well as additional resources (slides, code, figures, complete bibliography etc), open-sourced under the MIT License.

Contents

Project Advisor: Gabriel Peyré (CEREMADE, Université Paris-Dauphine)
Course: *"Sparsity and Compressed Sensing"*, by G. Peyré, in 2015–2016
Master 2 **program:** Mathématiques, Vision, Apprentissage (MVA) at ENS de Cachan.
Grade: I got 19/20 for my project.
Ranked 1st amongst 36 students who passed the course (average 14.19/20), 57 were registered.

Outline: In this report, we start by introducing the problem, and the main approach we studied, in section 1. Then different algorithms are presented in section 2, and each comes with a short discussion about its complexity; followed in section 3 by some necessary preliminary results to a theoretical analysis for some algorithms (mainly proofs of deterministic or expected error bounds). We will also detail in section 4 our implementation, and the numerical experiments developed to confirm the theoretical results and apply the presented framework on dense or sparse matrices. At the end, we conclude with a short sum-up in section 5, along with a list of references, and links to additional on-line resources in appendix A.

1 Introduction

1.1 Presentation of the problem

Matrix factorization is frequently listed amongst the "10 most influential algorithms" of the 20^{st} century [Cip00, DS00, Giv01]. For instance, B. Cipra lists in 2000 3 algorithms related to matrix factorization amongst his top 10 algorithms: Krylov subspace methods (3^{rd} position), decompositional approach to matrix computations (4^{th} position), and the QR algorithm (6^{th} position) [Cip00].

In fact, linear algebra computational techniques should be separated from its applications. Ideally, one should focus only on one specialization. Matrix manipulators should develop frameworks with algorithms to solve numerical linear algebra problems. The classical algorithms, dating back to the middle of the previous century[0], are no longer adequate for some applications, which are the result of the developments in computer hardware, information sciences and big data.

About the new applications, we can cite modern data mining applications, inaccurate or missing data in information science, and new architectures involving, for instance, graphics processing units (GPU). In fact, new data mining applications involve very large matrices that classical methods cannot even hope to solve efficiently (*e.g.* Google page rank). Otherwise, in information sciences, it goes without mentioning the fact that data is usually missing or inaccurate. In such case, it seems misguided to spend too much of the highly expensive computational capacity on inaccurate information. In addition, some data matrices are so big that they cannot be stored in easy access memory[1]. Data access time surpasses thus the computational requirements while classical algorithms are multi-pass.

Low-rank approximation

We will present herein a special case of matrix decomposition techniques: the approximation by **low-rank matrices**. As we have seen before hand, matrix decomposition techniques are very popular, and usually cited amongst the 10 most influential algorithms [Cip00, DS00, Giv01]. This fact highlights the importance of low rank approximations, which includes the QR factorization and the singular eigenvalue decomposition (SVD) [GR13]. These methods both expose the **range**

[0] [GR13] presents these algorithms with a historical perspective. For instance, the QR algorithm was developed independently by J.G.F. Francis in the UK and V.N. Kublanovskaya in USSR in the years 1959–1963.

[1] An example is the Laplacian for a social network graph, as studied in Michal Valko's MVA course in fall 2015

of the matrix. In general, we aim at decomposing a matrix[2] $\mathbf{A} \in \mathbb{M}_{m,n}(\mathbb{K})$ as a product:

$$\underbrace{\mathbf{A}}_{m\times n} \approx \underbrace{\mathbf{B}}_{m\times k} \times \underbrace{\mathbf{C}}_{k\times n} \tag{1}$$

Where k is the numerical **rank** of the matrix (*i.e.* dimension of its image space). So when the matrix $\mathbf{A}$ has a "low" rank, $k \ll \min(m,n)$, we can store it and compute linear operations on it quite efficiently thanks to equation (1). Indeed, storing $\mathbf{A}$ takes about $O(mn)$, while storing $\mathbf{B}, \mathbf{C}$ takes only[3] $O((m+n)k)$, and linear algebra computations depend on the matrices' size.

Low rank approximation has many applications. The principal component analysis (PCA), in statistics, for one, is nothing but a low rank approximation. They are also of use in parameter estimation with least squares, as the design matrix may not be inverted. We use then the QR decomposition to calculate the Moore-Penrose inverse [Alb72, Rak04, Cou08, GR13].

1.2 Randomized matrix approximation

Randomized algorithms provide simple and effective tools to perform matrix approximate factorizations [CLRS09, chapter 5]. The randomized methods are faster and more robust and, with a trade-off accuracy for speed, it yields results up to any precision ($\varepsilon > 0$ hereafter).
In general, matrix approximation through randomization is done following the paradigm [Tro14]:

(i) **Pre-processing**: by computing the sampling probabilities,

(ii) **Sampling**: applying a function on the matrix, usually linear (as seen in the sparsity course),

(iii) **Post-processing**: by applying *classical* techniques of linear algebra on the samples.

We quickly list the 3 most common techniques:

- **Sparsification:** replace the matrix by a substitute that contains much less non-zero entries (*i.e.* a sparser matrix). We can also quantize entries such that we obtain an approximate matrix, *e.g.* the Spielman-Srivastava algorithm [SS08]. For instance, the resulting matrix multiplication by a vector is less time consuming than multiplying the original matrix by a vector.

- **Dimension reduction:** start from the fact that the matrix rows are dependent: they can be embedded in a low dimension subspace without altering the geometric properties, based on the Johnson-Lindenstrauss lemma [JL84].

- **Column selection methods:** select a small set of columns that describes most of the range of the matrix. This method is based on the fact that [DMM08, part 3.1], [Zha15, slide 29]:

$$\|\mathbf{A} - \mathbf{C}\mathbf{C}^{\dagger}\mathbf{A}\| \leq \sqrt{1 + k(n-k)}\, \|\mathbf{A} - \mathbf{A}_{(k)}\| \tag{2}$$

  The dagger $\mathbf{C}^{\dagger}$ denotes the pseudo-inverse, $\mathbf{A}_{(k)}$ represents the *best* k-rank approximation of $\mathbf{A}$ and $\mathbf{C}$ is a k-column sub-matrix of $\mathbf{A}$. Although it is NP-hard (in the input size $\min(m,n)$, [GJ79, problem MP4], proof by J. Murty in 1972) to choose the best k columns, there are efficient (randomized or not) techniques, mainly based on the QR method [AM07, WLRT08]. We will focus on this last approach.

[2] In all this report, the only field we consider is $\mathbb{K} \overset{\text{def}}{=} \mathbb{C}$, and if not specified every scalar is complex.
[3] And obviously $(m+n)k \ll mn$ as soon as $k \ll \min(m,n)$, hence the name "low rank".

1.3 The two stages approach framework

The main idea in this framework is to capt the most of the action of the matrix: *i.e.* we want to apply the matrix on a dimension-reduced space so as to obtain an approximation of the result obtained in the full space. It is a dual to the dimension reduction technique. We then apply the classical methods on the "reduced" matrix. In this framework, we are using then two stage algorithms.

Formally, suppose that $k \in \mathbb{N}^\star$ is the numerical rank of the $m \times n$ matrix $\mathbf{A}$ (the same notation will be used hereafter). Let $\mathbf{Q}$ be the matrix representation of k orthonormal vectors (columns). The "reduced" matrix is then $\mathbf{B}$, simply written as: $\mathbf{B} \overset{\text{def}}{=} \mathbf{Q}^*\mathbf{A}$. If the subspace generated by $\mathbf{Q}$ captures the most of $\mathbf{A}$'s action, we should then get: $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.

We can then describe briefly the two stages[4] setting as follows:

Stage 1: First, (randomly) produce a $m \times k$ matrix $\mathbf{Q}$ such that:

$$\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}. \tag{3}$$

Stage 2: Then, apply *classical* techniques – such as QR and SVD – on the "reduced" matrix:

$$\mathbf{B} = \mathbf{Q}^*\mathbf{A}. \tag{4}$$

It is in stage 1 that the randomization intervenes. In fact, as stated previously, we will use the column selection approach: we draw k random vectors to sweep over the range of $\mathbf{A}$. Hopefully, the resulting vectors will form an approximation of the range of the matrix we are $\mathbf{A}$. We then just apply an ortho-normalization (OR) technique – such as Gramm-Schmidt – to deduct $\mathbf{Q}$. This idea is not especially new [GR13, GE96].

The new idea presented here is from [HMT11], and is based on the fact that we could try to sample "too much" vectors instead of exactly k vectors, *i.e.* l vectors with $l > k$. Let us try to explain why. Suppose we drowned k vectors $\{\omega^i\}_{1\leq i\leq k}$. We hope that these vectors being drawn randomly are independent, and the resulting subspace does not intersect (non-trivially) with the matrix's kernel $\ker(\mathbf{A})$. We are the confident that the vectors y^i will describe the range of $\mathbf{A}$, where:

$$y^i \overset{\text{def}}{=} \mathbf{A}\omega^{(i)}, \;\; i = 1 \dots k. \tag{5}$$

The problem is that when we approximate $\mathbf{A}$ the error we make shifts these vectors out from the range of $\mathbf{A}$. We thus have to **oversample**, *i.e.* to draw more ω^i: $\{\omega^i\}_{1\leq i\leq l}$. The algorithms presented later all use the notation $\mathbf{\Omega} \overset{\text{def}}{=} [\omega^i]_{1\leq i\leq l}$ and $\mathbf{Y} \overset{\text{def}}{=} [y^i]_{1\leq i\leq l}$.

But another issue arises. All this time we considered that $k \overset{\text{def}}{=} \text{rank}(\mathbf{A}) = \dim(\text{range}(\mathbf{A}))$ is already set. The problem is that we do not know k in general, and in fact, we have to find k. We reformulate then the "stage 1" equation (3):

$$\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \varepsilon. \tag{6}$$

The numerical rank depends then on the aimed precision[5]: $k = k(\varepsilon)$.

To solve the problem, we will use the SVD. Let σ_j the j-th largest singular value of $\mathbf{A}$ (for $1 \leq j \leq \min(m, n)$. A famous theorem, due to L. Mirsky (from 1969, see [WLRT08, lemma 3.1]), can then be used to state:

$$\min_{\text{range}(\mathbf{X})\leq k} \|\mathbf{A} - \mathbf{X}\| = \sigma_{k+1}. \tag{7}$$

[4] [HMT11] uses stages "A" and "B", but I preferred to use "1" and "2", as $\mathbf{A}$ and $\mathbf{B}$ are matrices here.

[5] And not the other way around, it is absurd to consider a target precision depending on the unknown rank!

One way to find the minimizer is to we choose $\mathbf{X} = \mathbf{Q}\mathbf{Q}^*\mathbf{A}$, where $\mathbf{Q}$ are the k dominant (left) singular vectors of $\mathbf{A}$, and k is numerically determined by the fact that $\sigma_{k+1} \leq \varepsilon$ (this is a classical result, see for example [GR13, section 10.6.2]).

Therefore, the *fixed rank problem* can be solved by a first prototype algorithm, which uses a matrix $\mathbf{\Omega}$. More discussion on the possible choices of $\mathbf{\Omega}$ will come in the next section.

1. Draw an $n \times (k+p)$ random matrix $\mathbf{\Omega}$, (from a certain distribution, for a certain p)
2. Form the matrix $\mathbf{Y} \overset{\text{def}}{=} \mathbf{A}\mathbf{\Omega}$,
3. Construct a matrix $\mathbf{Q}$ whose columns form an orthonormal basis for the range of $\mathbf{Y}$.

Figure 1: PROTO-ALGORITHM (Idea for stage 1)

2 Algorithms for the two stages matrix approximation framework

In this section, we present different algorithms to solve the "stage 1" part, and then the "stage 2" part of the matrix approximation two stage framework introduced in the previous section 1.

2.1 Stage 1 algorithms

Below are presented 5 different algorithms for the first stage 1, respectively in figures 2, 3, 4, 5, 6.

2.1.1 Randomized Range Finder

The simplest implementation of the proto-algorithm from Figure 1 is the *randomized range finder algorithm* [Tro14, slide 52]. Given a matrix $\mathbf{A}$, and a integer l (with $l > k$) (the "oversampled" numerical rank), it computes an orthonormal basis $\mathbf{Q}$. The $\mathbf{\Omega}$ matrix is drawn using a Gaussian distribution with mean 0 and variance 1 (*i.e.* the normal distribution, $\mathcal{N}(0,1)$).

1. Draw an $n \times l$ standard Gaussian random matrix $\mathbf{\Omega}$,
2. Form the $m \times l$ matrix $\mathbf{Y} \overset{\text{def}}{=} \mathbf{A}\mathbf{\Omega}$,
3. Construct the $m \times l$ matrix $\mathbf{Q}$, from $\mathbf{Y}$'s QR factorization: $\mathbf{Y} = \mathbf{Q}\mathbf{R}$.

Figure 2: RANDOMIZED RANGE FINDER (1st algorithm for Stage 1)

The oversampling parameter $p \overset{\text{def}}{=} l - k > 0$ depends on the matrix dimensions, the decrease of the ordered singular spectrum and the random test matrices. For Gaussian matrices $\mathbf{A}$, p of the order of 5 or 10, yields good results[6] and there is no need for more oversampling than k [Tro12a].

The QR factorization is done either with Gramm-Schmidt algorithm, Householder reflections or Givens rotations [GR13, section 10.4], and it is usually in $O(mn\min(m,n))$, but we will not give anymore detail on this part.

The number of operations ("*flops*") necessary for this algorithm is *about*:

$$T_{\text{RandomizedRangeFinder}}(m,n,l) \approx n \times l \times T_{\text{rand}} + l \times T_{\text{mult}} + m \times l^2 = O(mnl). \tag{8}$$

[6] In practice, my first experiment was to check this numerically.

Where T_{rand} is the time to sample 1 value from a standard Gaussian distribution ($O(1)$ is reasonable, see the Ziggurat method [MT+00]), and T_{mult} is the time to evaluate a matrix-vector product of size $(m,n)\times(n,1)$, so about $O(mn)$ (in line 2, this is the most consuming part!). So this first algorithm is in $O(mnk)$, still linear in k. A $\log(k)$ will be obtained by the Fast Randomized Range Finder algorithm (see below, in section 2.1.4)

2.1.2 Adaptive Randomized Range Finder

In the previous setting, the rank k is fixed and assumed to be know. Actually, we can do better by adapting the "randomized range finder" algorithm from Figure 2, (almost) for free. In fact, due to the following lemma 2.1 [Tro12a, WLRT08], we can evaluate the error by setting[7] $\alpha = 10$, and applying it to $\mathbf{B} \overset{\text{def}}{=} (\mathbb{I}_m - \mathbf{Q}\mathbf{Q}^*)\mathbf{A}$. We thus get, with probability $1-\alpha^{-r}$:

$$\|(\mathbb{I}_m - \mathbf{Q}\mathbf{Q}^*)\mathbf{A})\| \leq 10\sqrt{\frac{2}{\pi}} \max_{i=1\dots r} \|\left(\mathbb{I}_m - \mathbf{Q}\mathbf{Q}^*\right)\mathbf{A}\omega^{(i)}\|. \tag{9}$$

Lemma 2.1. *Let* $\mathbf{B}$ *be an* $m\times n$ *matrix. Fix* $r>0$ *an integer and a real number* $\alpha>1$*. Draw an* independent *family* $\{\omega^{(i)}, i=1\dots r\}$ *of standard Gaussian vectors. Then, with probability at least* $0<1-\alpha^{-r}<1$*:*

$$\|\mathbf{B}\| \leq \alpha\sqrt{\frac{2}{\pi}} \max_{i=1\dots r} \|\mathbf{B}\omega^{(i)}\|.$$

(See [WLRT08, observation 3.3] for a proof).

In consequence, we do not need to fix the over-sampling parameter p, as l will be determined numerically, depending on the aimed tolerance ε. To be more precise, we want to find the smallest l such as the $m\times l$ matrix $\mathbf{Q}^{(l)}$ verifies:

$$\|\left(\mathbb{I}_m - \mathbf{Q}^{(l)}(\mathbf{Q}^{(l)})^*\right)\mathbf{A}\| \leq \varepsilon. \tag{10}$$

We then adapt the algorithm from Figure 2 by adding columns to Q, so that we get the desired precision, that can now be measured before hand (see Figure 3).

2.1.3 Randomized Power iteration

The *Randomized Range Finder* algorithm from Figure 2 (both naive and adaptive versions) supposes that matrices have singular values that decay quickly, as highlighted by equation (7). In fact, it performs poorly on matrices that have singular values $\sigma_j(\mathbf{A})$ that decay slowly or are just too large [Tro14, slide 63]. Singular vector associated with the small singular values interfere in the calculation.

The idea behind the **power iteration** is to reduce the weight associated to those values. In order to do so, we compute the matrix $\mathbf{A}_q$ instead of $\mathbf{A} = \mathbf{A}_0$, where $q\in\mathbb{N}$ is an "iteration" parameter:

$$\mathbf{A}_q \overset{\text{def}}{=} (\mathbf{A}\mathbf{A}^*)^q\,\mathbf{A}. \tag{11}$$

Then, we get its singular values, simply by the usual properties on $\mathbf{A}^*$:

$$\forall q\in\mathbb{N},\ \ \sigma_j(\mathbf{B_q}) = \sigma_j(\mathbf{A})^{2q+1},\ \text{for } j=1,2,3\dots \tag{12}$$

And thus we can propose the Randomized Power Iteration stage 1 algorithm, where $\mathbf{A}$ and $q\geq 0$ are given, see Figure 4.

[7] This $\alpha = 10$ is an arbitrary choice, for sake of elegance.

Input: matrix $\mathbf{A}$, and $r = \min(m, n)$.
1. Draw r *iid* standard Gaussian vectors $\{\omega^{(i)}, i = 1 \dots r\}$, of size n,
2. **For** $i = 1 \dots r$, compute $y^{(i)} \overset{\text{def}}{=} \mathbf{A}\omega^{(i)}$,
3. $l \overset{\text{def}}{=} 0$
4. $\mathbf{Q}^{(0)} \overset{\text{def}}{=} [\,]$ (We use MATLAB notation for matrices)
5. **While** $\max_{i=1\dots r} \|y^{(i+l)}\| > \varepsilon / \left(10\sqrt{\frac{2}{\pi}}\right)$: (Using bound from equation (9))
6. $l \longleftarrow l + 1$
7. $y^{(i)} \overset{\text{def}}{=} (\mathbb{I}_m - \mathbf{Q}^{(l-1)}(\mathbf{Q}^{(l-1)})^*) y^{(l)}$ (New column, orthogonal to the previous ones)
8. $q^{(i)} \overset{\text{def}}{=} y^{(i)} / \|y^{(i)}\|$ (Normalized new column)
9. $\mathbf{Q}^{(l)} \longleftarrow \left[\mathbf{Q}^{(l-1)} \;\; q^{(i)}\right]$ (One more column in $\mathbf{Q}$)
10. Draw a Gaussian $\omega^{(l+r)}$, of length n
11. $y^{(l+r)} \overset{\text{def}}{=} \left(\mathbf{I} - \mathbf{Q}^{(l)}(\mathbf{Q}^{(l)})^*\right) \mathbf{A}\omega^{(l+r)}$
12. **For** $i = 1 \dots r - 1$: (Simple Gramm-Schmidt process)
13. $y^{(i+l)} \longleftarrow y^{(i+l)} - q^{(i+l)} \langle q^{(i+l)}, y^{(i+l)} \rangle$
14. **End For**.
15. **End While.**
16. **Return** $\mathbf{Q} \overset{\text{def}}{=} \mathbf{Q}^{(\text{last})} = \mathbf{Q}^{(l)}$.

Figure 3: ADAPTIVE RANDOMIZED RANGE FINDER (2nd algorithm for Stage 1)

1. Draw an $n \times l$ standard Gaussian random matrix $\mathbf{\Omega}$,
2. Form the $m \times l$ matrix $\mathbf{Y}_q \overset{\text{def}}{=} \mathbf{A}_q\mathbf{\Omega} = (\mathbf{A}\mathbf{A}^*)^q\mathbf{A}\mathbf{\Omega}$, via alternative application[8] of $\mathbf{A}$ and $\mathbf{A}^*$,
3. Construct the $m \times l$ matrix $\mathbf{Q}$, from $\mathbf{Y}_q$'s QR factorization: $\mathbf{Y}_q = \mathbf{Q}\mathbf{R}$.

Figure 4: RANDOMIZED POWER ITERATION (3rd algorithm for Stage 1)

Remark 2.2. ***What if*** $q = 0$***?*** *From equation (11),* $\mathbf{A}_0 = \mathbf{A}$*, so this third algorithm Figure 4 boils down to the first one ("Randomized Range Finder", Figure 2). In practice, we implemented the "Randomized Power Iteration" algorithm with a general parameter* $q \in \mathbb{N}$*, and using it with* $q = 0$ *allows to have the first one "for free" (see in subsection 4.3, or the file* `RandomizedRangeFinder.m`*).*

Robustness to rounding errors

Remark 2.3. *Rounding errors in this algorithm for floating-point operations result in killing singular modes associated with singular values small compared to* $\|\mathbf{A}\|$*. One way to overcome this issue is done by ortho-normalizing the columns after each application of* $\mathbf{A}$ *and* $\mathbf{A}^*$*.*

Remark 2.4. *(*Adaptive Power/Subspace Iteration Algorithm*) Like in the previous case, we can easily derive an adaptive version of the power iteration algorithm [HMT11, remark 4.3], using the same trick to evaluate the error on-line. Due to space constraint, this will not be developed here.*

2.1.4 Fast Randomized Range Finder

In fact, random Gaussian matrices are not adapted for dense matrices [Tro12b]. One way to adapt to dense matrices (without special structures) is to use the Fast Fourier Transform (FFT)

1. Draw an $n \times l$ Gaussian random matrix $\mathbf{\Omega}$,
2. Let $Y_0 \overset{\text{def}}{=} \mathbf{A\Omega}$, and compute its QR factorization: $Y_0 = Q_0 R_0$,
3. **For** $j = 1 \dots q$
4. Let $Y'_j \overset{\text{def}}{=} A^* Q_{j-1}$, and compute its QR factorization: $Y'_j = Q'_j R'_j$,
5. Let $Y_j \overset{\text{def}}{=} A Q'_j$, and compute its QR factorization: $Y_j = Q_j R_j$,
6. **End For**.
7. **Return:** the $m \times l$ matrix $\mathbf{Q} \overset{\text{def}}{=} Q_{\text{fin}} = Q_q$.

Figure 5: RANDOMIZED SUBSPACE ITERATION (more stable 3rd algorithm for Stage 1)

[CT65, DV90]. Therefore, we will use the sub-sampled random Fourier transform (SRFT) [HMT11, section 4.6]. The goal being to choose a *structured* random matrix $\mathbf{\Omega}$.One advantage of this last algorithm is that it runs in $O(mn\log(l))$ for dense matrices, where previous algorithms where all in $O(mnl)$.

We need to consider *sub-sampled RFT* because we sample only for l columns, and random because we randomly rotate on each direction. Without giving too much details due to space constraints (see [HMT11, Eq. (4.6)]), a **SRFT** is a matrix $\mathbf{\Omega}$ of size $n \times l$, defined by 3 components:

$$\mathbf{\Omega} \overset{\text{def}}{=} \sqrt{\frac{n}{l}} \times \mathbf{D} \times \mathcal{F} \times \mathbf{R}. \tag{13}$$

- Where $\mathbf{D}$ is **a** $n \times n$ diagonal matrix whose entries are iid random variables distributed on the complex unit circle[9] (*i.e.* complex Rademacher variables),
- $\mathcal{F}$ is **the** $n \times n$ unitary discrete Fourier transform (DFT),
- $\mathbf{R}$ is **an** $n \times l$ matrix whose l columns are uniformly drawn from the $n \times n$ identity matrix $\mathbb{I}_n$ (it is a random sample of columns, see above about the column selection methods).

Having introduced this notion of SRFT, we can then present this last stage 1 algorithm, more suited for dense matrices than the previous ones:

1. Draw an $n \times l$ SRFT test matrix $\mathbf{\Omega}$ as above (randomly),
2. Form the $m \times l$ matrix $\mathbf{Y} \overset{\text{def}}{=} \mathbf{A\Omega}$ using sub-sampled FFT,
3. Construct the $m \times l$ matrix $\mathbf{Q}$ using $\mathbf{Y}$'s QR factorization (as before).

Figure 6: FAST RANDOMIZED RANGE FINDER (4^{th} algorithm for Stage 1)

For step 2, the cost of forming $\mathbf{Y} = \mathbf{A\Omega}$ will be about $O(mn\log l)$, by using the FFT algorithm [CT65, DV90]. So, the number of operations (*flops*) necessary for this algorithm is about:

$$T_{\text{FastRandomizedRangeFinder}}(m,n,l) \approx m \times n \times \log(l) + m \times l^2 = O(mn\log(l)). \tag{14}$$

No error bound for this last stage 1 algorithm will be presented, cf. [BG13, theorem 7] if needed.

[9] Note that some authors impose $\mathbf{D}$ to be real, in which case its diagonal elements are just random signs, *i.e.* classical Rademacher random variables in $\{-1, 1\}$, *cf.* [Tro14, slide 63] and [HMT11, section 4.6].

2.2 Stage 2 algorithms

In this subsection, we first give a general discussion about factorizations transforms, and then one example of algorithm for stage 2 (using SVD).

2.2.1 From one factorization to another?

The main low rank approximations are the QR decomposition and the truncated SVD [Cip00]. In general, we can derive any matrix factorization from any other one, almost for free. Indeed, suppose we got the following low-rank approximation for $\mathbf{A}$, with precision at least $\varepsilon > 0$:

$$\| \underbrace{\mathbf{A}}_{m\times n} - \underbrace{\mathbf{B}}_{m\times k} \times \underbrace{\mathbf{C}}_{k\times n} \| \leq \varepsilon. \tag{15}$$

Obtaining A's QR factorization: We can easily get $\mathbf{A}$'s QR factorization through:
1. Computing $\mathbf{C}$'s QR decomposition: $\mathbf{C} = Q_1 R_1$,
2. Forming the product $\mathbf{D} \overset{\text{def}}{=} R_1\mathbf{B}$, and computing its QR decomposition: $\mathbf{D} = Q_2\mathbf{R}$,
3. Finally, forming the matrix $\mathbf{Q} \overset{\text{def}}{=} Q_1 Q_2$.

The resulting QR factorization has at least the same precision:

$$\|\mathbf{A} - \mathbf{QR}\| \leq \varepsilon. \tag{16}$$

Obtaining A's SVD: We can also easily get $\mathbf{A}$'s SVD through:
1. Computing $\mathbf{C}$'s QR decomposition: $\mathbf{C} = Q_1 R_1$,
2. Forming the product: $\mathbf{D} \overset{\text{def}}{=} R_1\mathbf{B}$ and computing its SVD: $\mathbf{D} = U_2\mathbf{\Sigma}\mathbf{V}^*$,
3. Finally, forming the product: $\mathbf{U} \overset{\text{def}}{=} Q_1 U_2$.

The resulting SV decomposition has at least the same precision:

$$\|\mathbf{A} - \mathbf{U\Sigma V}^*\| \leq \varepsilon. \tag{17}$$

Remark 2.5. *We could apply the same technique for other factorizations.*

2.2.2 Direct SVD

In the stage 2, a naive but efficient approach is to simply use a classic algorithm on $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$, and using the SVD as announced before yields this factorization scheme, which expresses visually the factorization trick to reduce computational complexity or storage space:

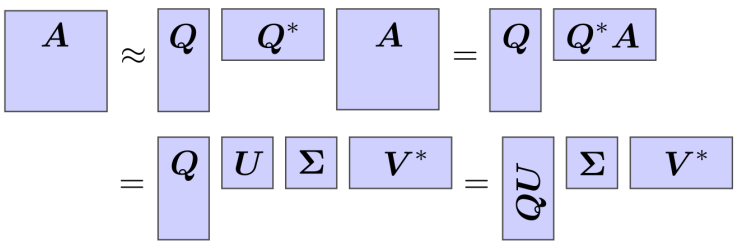

Figure 7: Stage 2 : Forming the SVD (figure from [Tro14, slide 41]).

This gives the following naive algorithm, in Figure 8:

Input: Given a matrix $\mathbf{A}$ and an orthonormal basis $\mathbf{Q}$ from stage 1.
1. Form the matrix $\mathbf{B} \overset{\text{def}}{=} \mathbf{Q}^*\mathbf{A}$,
2. Compute the SVD of the matrix $\mathbf{B} = \widetilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*$,
3. Form the orthonormal matrix $\mathbf{U} \overset{\text{def}}{=} \mathbf{Q}\widetilde{\mathbf{U}}$.

Figure 8: DIRECT SVD (Stage 2)

This algorithm relies on the stage 1 algorithm we chose to use, but we observe that if the stage 1 algorithm is precise with tolerance $\varepsilon > 0$, then this algorithm is as precise (thanks to (17)):

$$\|\mathbf{A} - \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*\| \leq \varepsilon. \tag{18}$$

This algorithm has a total complexity of $O(m \times n \times l + l^2 n)$ *flops* [Tro14, slide 41].

2.2.3 Other stage 2 algorithms?

We could detail other possibilities for stage 2 algorithms, but as explained above, the naive one is already working quite well in general. For more details, refer to [HMT11, section 5, 5.2 to 5.5].

---

3 Theoretical analysis

In this section, we first give a few preliminary linear algebra results, mainly on p.s.d. matrices, and standard Gaussian matrices. Then the lemmas are used to give and prove some error bounds on the algorithms presented before.

3.1 Preliminaries

Let us start by presenting some useful properties of positive semi-definite (p.s.d.) matrices [GE96, WLRT08, AM07]. For all this part, let $n \in \mathbb{N}^*$, and consider M a (square) p.s.d. matrix of size n.

Lemma 3.1 (Perturbation of inverses)**.** ***([HMT11, prop 8.2])***
Let $M \succeq 0$ (positive semi-definite). Then:

$$\mathbb{I} - (\mathbb{I} + M)^{-1} \preceq M.$$

Proof. We have: $\mathbb{I} - (\mathbb{I}+M)^{-1} = M(\mathbb{I}+M)^{-1} = M^{-1/2}(\mathbb{I}+M)^{-1}M^{-1/2} \preceq M$. Because $M \succeq 0$, $\mathbb{I}+M \succeq 1$, and so $(\mathbb{I}+M)^{-1} \preceq 1$. □

Remark 3.2. *Note: due to space constraints, we will not prove every result in this theoretical part. Most lemmas and properties can be found in [HMT11, section 8.1 and part III].*

Lemma 3.3. ***([HMT11, prop 8.3])*** *Let M be a p.s.d. matrix written in block form, with B and B^* on the anti-diagonal: $M = \begin{bmatrix} A & B \\ B^* & C \end{bmatrix}$. Then we can say that (for the spectral matrix norm $\|\cdot\|$): $\|M\| \leq \|A\| + \|C\|$.*

From now on, let P_M design the unique orthogonal projector such that $\mathrm{range}(M) = \mathrm{range}(P_M)$. If M has full range, we can write: $P_M = M\left(M^*M\right)^{-1}M^*$.

Lemma 3.4. *([HMT11, prop 8.4]) Let U be an unitary matrix, then $U^* P_M U = P_{U^* M}$.*

Proof. First, because U is unitary and P_M is a projector,

$$(U^* P_M U)(U^* P_M U) = U^*(P_M(UU^*)P_M)U = U^* P_M^2 U = U^* P_M U.$$

So $U^* P_M U$ is a projector, and is orthogonal since it is Hermitian ($(U^* P_M U)^* = U^* P_M^* (U^*)^* = U^* P_M U$). And we know that orthogonal projectors are fully determined by their range, so it suffices to determine the range of $U^* P_M U$, and this is quite direct:

$$\mathrm{range}(U^* P_M U) = U^* \mathrm{range}(P_M) = \mathrm{range}(P_{U^* M}).$$

And so, we have exactly what we announced: $U^* P_M U = P_{U^* M}$. □

We also have this last property, true for any orthogonal projector P (not necessarily P_M):

Proposition 3.5. *([HMT11, prop 8.6]) Let P be any orthogonal projector, M any matrix and $q \geq 0$.*

$$\forall q \geq 0, \quad \|PM\| \leq \|P(MM^*)^q M\|^{1/(2q+1)}. \tag{19}$$

Remark 3.6. *This result is supporting the idea used for the Power Iteration algorithm in section 2.1.3: working on $(AA^*)^q A$ increase the decay of the spectrum, geometrically in q.*

Now, we give a few results about (standard) Gaussian matrices[10] that are mainly coming from [HMT11, part 8] and presented in the shorter article [Tro12a].

Proposition 3.7. *([HMT11, prop 8.8]) Set S and T, and draw a Gaussian matrix G of same size. Then:*

$$\begin{cases} \mathbb{E}\left[\|SGT\|_F^2\right] &= \left(\|S\|_F \times \|T\|_F\right)^2. \\ \mathbb{E}\left[\|SGT\|\right] &\leq \|S\| \times \|T\|_F + \|T\| \times \|S\|_F. \end{cases} \tag{20}$$

Where $\|\cdot\|_F$ designs the $\ell_{2,2}$ Fröbenius matrix norm, and $\|\cdot\|$ is the spectral ℓ_2 matrix norm.

Proposition 3.8. *([HMT11, prop 8.9]) Let G be a $k \times (k+p)$ Gaussian matrix with $p, k \geq 2$. Then:*

$$\begin{cases} \mathbb{E}\left[\|G^\dagger\|_F^2\right] &= k/(p-1). \\ \mathbb{E}\left[\|G^\dagger\|\right] &= \mathrm{e}\sqrt{k+p}/p. \end{cases} \tag{21}$$

Finally, we have this concentration inequality, which is needed for the proofs of the probabilistic error bounds for the Range Finder algorithm (theorem 3.13):

Proposition 3.9 (Concentration inequality). *Let h be a L-Lipschitz function (on matrices), and G a standard Gaussian matrix. Then:*

$$\forall t \in \mathbb{R}, \quad \mathbb{P}\Big\{h(G) \geq \mathbb{E}\left[h(G) + Lt\right]\Big\} \leq e^{-t^2/2}. \tag{22}$$

Proof. It comes quite quickly from the generalized Hoeffding's inequality. □

[10] *I.e.* random matrices drawn from a multi-variate normal distribution, $\mathcal{N}(\mu, \Sigma)$ in $\mathbb{R}^d$.

3.2 Error bounds

The goal of this part is to provide some theoretical warranties on the efficiency and precision of some of algorithms presented before in section 2. A more complete analysis for some algorithms can be found in [HMT11]. In all this part, let $\mathbf{A} \in \mathbb{M}_{m,n}(\mathbb{R})$ be fixed. We use the notations from the algorithms presented above in section 2. We will focus on controlling the *backward error*: $\|\mathbf{A} - \mathbf{BC}\|$ with a **tolerance** $\texttt{tol} \overset{\text{def}}{=} \varepsilon$ if $\mathbf{A}$ is factored as $\mathbf{BC}$.

Let us first rewrite $\mathbf{A}$'s SVD by splitting is singular values matrix Σ into the first k values in Σ_1 and the $n-k$ others in Σ_2:

$$\mathbf{A} = \mathbf{U}\begin{pmatrix}\Sigma_1 & \mathbf{0}\\ \mathbf{0} & \Sigma_2\end{pmatrix}\begin{pmatrix}V_1^*\\ V_2^*\end{pmatrix}. \tag{23}$$

and so we also have the same "splitting" for Ω: $\Omega_1 = V_1^*\Omega$ and $\Omega_2 = V_2^*\Omega$.

Now, for the Y matrix (the "compressed" version of A), as used in the algorithms, we have:

$$\mathbf{Y} \overset{\text{def}}{=} \mathbf{A\Omega} = \mathbf{U}\begin{pmatrix}\Sigma_1^*\Omega_1\\ \Sigma_2^*\Omega_2\end{pmatrix}. \tag{24}$$

And finally, by writing $P_{\mathbf{Y}}$ explicitly thanks to lemma 3.4, we have this major result, used in all the theorem that follow: $\|\mathbf{A} - \mathbf{QQ}^*\mathbf{A}\| = \|(I - P_{\mathbf{Y}})\mathbf{A}\|$.

3.2.1 Error bound for the Proto-Algorithm

The most important theorem for the Proto-Algorithm is the one that follows.

Theorem 3.10 (Deterministic Error Bound for the Proto-algorithm)**.** ***([HMT11, theorem 9.1])*** *Assuming that Ω_1 has full row rank:*

$$\|(I - P_{\mathbf{Y}})\mathbf{A}\|^2 \leq \|\Sigma_2\|^2 + \|\Sigma_2\Omega_2\Omega_1^\dagger\|. \tag{25}$$

Proof. Let: $\tilde{\mathbf{A}} \overset{\text{def}}{=} U^*\mathbf{A}$ and $\tilde{Y} = \tilde{\mathbf{A}}\Omega$. We get: $\|(\mathbb{I} - P_{\mathbf{Y}})\mathbf{A}\| = \|U^*(\mathbb{I} - P_{\mathbf{Y}})U\tilde{\mathbf{A}}\| = \|(\mathbb{I} - P_{\tilde{Y}})\tilde{\mathbf{A}}\|$. There are two cases:

(1) On one hand, if Σ_1 is not strictly positive, then $\Sigma_2 = 0$ because of the singular value ordering: $\text{range}(\tilde{A}) = \text{range}(\begin{pmatrix}\Sigma_1\Omega_1\\ 0\end{pmatrix}) = \text{range}(\tilde{Y})$. This degenerate case implies that: $\|(\mathbb{I} - P_{\tilde{Y}})\tilde{A}\| = 0$.

(2) On the other hand, if Σ_1 is strictly positive, then let $W \overset{\text{def}}{=} \begin{pmatrix}\Sigma_1\Omega_1\\ 0\end{pmatrix}$. Since Ω_1 has full row-rank: $\text{range}(W) = \text{range}(\begin{pmatrix}\mathbb{I}_k\\ 0\end{pmatrix})$. And so its orthogonal projector will be $P_W = \begin{pmatrix}0 & 0\\ 0 & \mathbb{I}\end{pmatrix}$.

We then construct the matrix Z by flattening the top of $\tilde{Y}$: $Z = \tilde{Y}\Omega_1^*\Sigma_1^{-1}$. By construction, $\text{range}(\mathbf{Z}) \subset \text{range}(\tilde{\mathbf{Y}})$. Hence, we have that: $\|(\mathbb{I} - P_{\tilde{Y}}\tilde{A})\| \leq \|(\mathbb{I} - P_Z\tilde{A})\|$. And so, we get: $\|(\mathbb{I} - P_{\tilde{Y}}\tilde{A})\| \leq \|\Sigma^*(\mathbb{I} - P_Z)\Sigma\|$.

To conclude the proof, two applications of the lemma 3.3 suffice. For all the details, see pages 51–54 of [HMT11]. □

3.2.2 Error bounds for the range finder algorithm

We keep the same notations as before. This second theorem gives a deterministic error bound for the range finder algorithm Figure 2.

Theorem 3.11 (Average Fröbenius and spectral norm errors)**.** ***([HMT11, theorems 10.5 and 10.6])*** *If $k, p \geq 2$ and $k + p \leq \min(m, n)$, we have for the Fröbenius norm:*

$$\mathbb{E}\left[Error_F\right] = \mathbb{E}\left[\|(I - P_{\mathbf{Y}})\mathbf{A}\|_F\right] \leq \left(1 + \frac{k}{p-1}\right)^{1/2} \left(\sum_{j>k} \sigma_j^2(\mathbf{A})\right)^{1/2}. \tag{26}$$

And for the spectral norm:

$$\mathbb{E}\left[Error\right] = \mathbb{E}\left[\|(I - P_{\mathbf{Y}})\mathbf{A}\|\right] \leq \left(1 + \frac{k}{p-1}\right)\sigma_{k+1}(\mathbf{A}) + \frac{\mathrm{e}\sqrt{k+p}}{p}\left(\sum_{j>k} \sigma_j^2(\mathbf{A})\right)^{1/2} \tag{27}$$

Remark 3.12. *The second inequality (27) is too loose because it is derived from the inequality on the Fröbenius norm (26). We can see also that the minimal error we can get is the one we get with* truncated exact *SVD.*

We conclude this section with this other result, which gives a probabilistic error bounds, again for both the Fröbenius and spectral norms.

Theorem 3.13 (Probabilistic Fröbenius and spectral norm errors)**.** ***([HMT11, theorem 10.6])*** *Now for $p \geq 4$ and $t \geq 1$, with probability at least $2t^{-p} + \mathrm{e}^{-u^2/2}$, we have for the Fröbenius norm:*

$$\|(I - P_{\mathbf{Y}})\mathbf{A}\|_F \leq \left(1 + t\sqrt{\frac{3k}{p+1}}\right)\left(\sum_{j>k} \sigma_j^2\right)^{1/2} + ut \cdot \frac{\mathrm{e}\sqrt{k+p}}{p} \cdot \sigma_{k+1}(\mathbf{A}) \tag{28}$$

And for the spectral norm:

$$\|(I - P_{\mathbf{Y}})\mathbf{A}\| \leq \left[\left(1 + t\sqrt{\frac{3k}{p+1}}\right)\sigma_{k+1}(\mathbf{A}) + t\frac{\mathrm{e}\sqrt{k+p}}{p}\left(\sum_{j>k} \sigma_j^2(\mathbf{A})\right)^{1/2}\right] + ut \cdot \frac{e\sqrt{k+p}}{p}\sigma_{k+1}(\mathbf{A}) \tag{29}$$

Proof. For both theorems, see [HMT11, sections 10.2 and 10.3]. □

3.3 Comments on the error bounds

Here we should precise that these bounds are asymptotic. In practice, it should not be surprising to obtain a better performance than the pessimistic bound. Additionally, one should keep in mind that the various stage 1 algorithms presented above in subsection 2.1 are all specific to a certain type of matrices. For more errors bounds on these algorithms, all the details can be found in the reference paper [HMT11]. For example, for the power scheme for the randomized range finder (algorithms from Figure 4 and Figure 5), a simple error bound is presented in [HMT11, section 10, theorem 8].

We said that this algorithm does not work well if $\mathbf{A}$'s spectrum is not fast-decaying, and we can observe that the bounds for the Range Finder algorithm depend on $\sum_{j>k} \sigma_j^2(\mathbf{A})$, the "tail" of $\mathbf{A}$'s spectrum. For the spectral norm bound (29), an emphasis is put on the $(k+1)$-th singular value $\sigma_{k+1}(\mathbf{A})$, which comes from L.Mirsky's theorem given above in (7).

4 Implementation and numerical experiments

In this section, we present shortly what was implemented, how and why, and then two numerical experiments on artificial data are presented. An emphasis was put on the reproducibility of these experiments, not on the optimization of my implementation. Both experiments confirmed the theoretical bounds and complexities proved in the previous section.

4.1 Quick overview of our implementation

For our implementation, we chose to use GNU Octave, but all the programs are compatible with MATLAB, and should work on any platform. The code can be found on-line at `http://lbo.k.vu/pcs2016`, in the `src/` folder. We relied on the Numerical Tours MATLAB toolboxes `signal` and `general`, from [Pey11][11]. A few programs are only here for the second experiment, but the most interesting part is the implementation of our stage 1 algorithms. The only stage 2 algorithm considered in practice is the naive Direct SVD, even if I also tried with l-truncated SVD (see `QuickDirectSVD.m`).

4.2 A first experiment

The first experiment[12] is quite small and simple. We simply generated some random matrices A of size $n \times n$, with $n = 100, 200, 400, 800$, dense at first, and then sparse with various sparsity parameter. We checked that all the algorithms presented above were working "as expected", in the sense that: they all produce a valid SV decomposition $U, \Sigma, V^\star$, they are not slower than regular SVD (the stage 1 is usually very quick), and even if they are highly based on random column sampling, they worked on every tests I performed. We compared them on their norm errors $\|A - U^{(i)}\Sigma^{(i)}V^{(i)\star}\|$ (of the order of 10^{-12}), and on their (decaying) spectra $\tilde{\sigma}_j^{(i)}$.

4.3 Second experiment on a large sparse matrix

We present here the second experiment[13], reproducing the results of the second application from the main reference paper [HMT11, part 7.2].

This example involves a large matrix that arises in image processing. Some image processing algorithms uses the geometry of the image for tasks such as *denoising* and *inpainting*. They use a normalized graph Laplacian to represent the geometry of the image.

We worked a small sub-image, to keep our execution time small, and so a small gray-scaled patch of the standard image Lena[14], of size 50×50 is considered. As usual, each pixel is represented by an integer value between 0 and 255. Then, to construct a similarity graph for this image, each pixel x is represented by a 5×5 patch $\hat{x}$ around this pixel[15].

[11] Free and open-sourced. Our implementation is also open-sourced, see section A.3.

[12] For this first experiment, the reference program is `experiment1.m`.

[13] For this second experiment, the reference program is `experiment2.m`. It is well commented and gives a lot of details about the progression of the simulation. See the file `experiment2.txt` for a complete log file of its output.

[14] We use hereafter the Lena image for the example, but the same experiment and the same result were observed for other gray-scaled image (*e.g.* boat, mandrill). I did not tried on color-scaled image though.

[15] Note that all these parameters are included in the program as constant defined in the beginning, it would be easy to try other values – which we did. Increasing the size of the sub-image or the size of the patches only increased the running time, but we observed the same result.

We calculate the weight matrix $\tilde{\mathbf{W}}$, reflecting the similarity between the patches with the usual exponential similarity (of proximity parameter σ) on the Euclidean norm[16]

$$\widetilde{w_{ij}} \overset{\text{def}}{=} \exp\left(\frac{-\|\hat{x}_i - \hat{x}_j\|^2}{\sigma^2}\right).$$

In practice, we chose $\sigma = 50$. By zeroing all the entries of the weight matrix $\tilde{\mathbf{W}}$ except the $s = 7$ largest ones in each row, we construct our large sparse matrix $\mathbf{W}$ (of size 1296×7 here). We can then construct the graph Laplacian matrix: $\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{1/2}$.

Here, we are interested in the eigenvalues of this matrix $\mathbf{A}$, and on their decay: $\mathbf{A} = \mathbf{D}^{1/2}\mathbf{W}\mathbf{D}^{-1/2}$ which decay very slowly, as the figure 9 shows.

The blue plot describes the first algorithm (*i.e.* $q = 0$) ("Randomized Range Finder", from Figure 2). This algorithm runs faster than the others – around 7 seconds – but it is less accurate. The green plot describes the power iteration[17] algorithm, with $q = 4$ ("Randomized Power Iteration", from Figure 4). It takes around 12 seconds to compute but it is more accurate. The last plot in red represents the SRFT method ("Fast Randomized Range Finder", from Figure 6). In a shorter time (about 10 seconds), the last algorithm yields similar results to the power iteration algorithm.

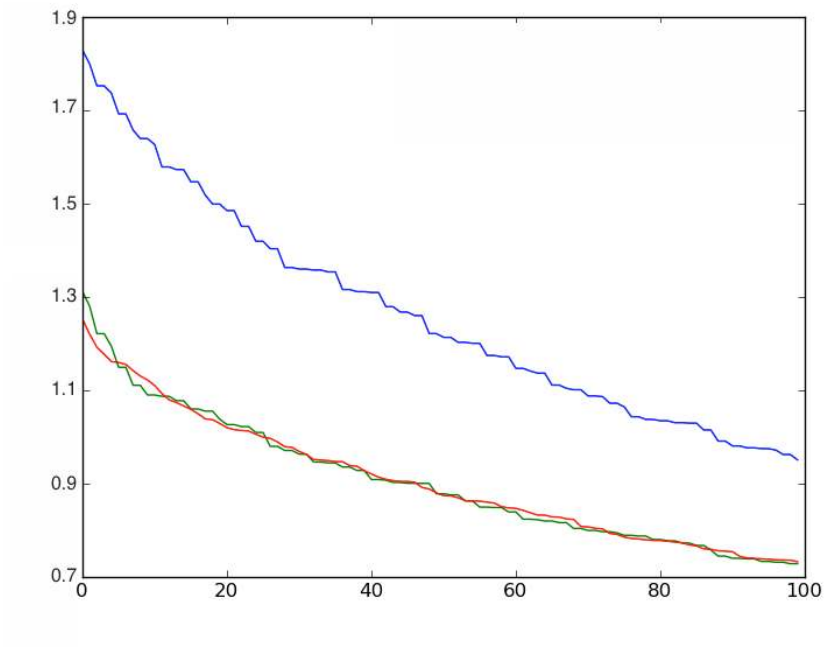

Figure 9: Comparison of the 3 main stage-A algorithms, decay of the first 100 singular values of $\mathbf{A}$. **Randomized Range Finder** is quicker but less precise, while **Randomized Power Iteration** and **Fast Randomized Range Finder** are optimal.

[16] This is classical, see for instance Michal Valko's 2nd lecture on "Graphs for Machine Learning", and this lab report.

[17] For the iteration parameter $q \geq 0$, the smaller the quicker, but the higher the more we increase the decay of small singular values. We tried higher values for q, but apparently staying at $q = 2, 3, 4$ is always sufficient.

5 Conclusion

The framework presented here is quite complete, and mainly comes from [HMT11]. We tried to present a few algorithms, adapted to each case (as described in the abstract). The main issue was finding a low rank approximation through randomization (stage 1), to reduce the computational time complexity of classical techniques. It could be completed with more detailed sampling methods for other structured matrices and by detailing the corresponding error bounds. Efforts should also be made to try and derive more sophisticated tools and obtain more powerful error estimations, if possible. As we have seen, the bounds given here are very large, compared with the numerical results[18]. One goal could be to solve the fixed precision problem more efficiently for sparse and very sparse matrices. Otherwise, one also can improve this framework by determining a more precise way to get the oversampling parameter, in an adaptive approach for the SRFT matrices (last stage 1 algorithm).

[18]But this is quite usual in applied mathematics, for instance the Lai and Robbins upper bound for Multi-Arm Bandits Problem (MAB) is optimal (1985), but very large in practice (for usual MAB problems, some algorithms are very much more efficient than the theoretical bound).

A Appendix

A.1 Acknowledgments

I would like to thank Gabriel Peyré my project advisor, as he replied quickly to my few queries and provided useful direction of research. Thanks also to my MVA comrade Basile Clement for proofreading my project report and slides.

A.2 Personal feeling about the project

I enjoyed working on this small project, and as usual for this kind of maths, I liked the different aspects we touched with this project: algorithms, complexity proofs, implementation, numerical simulation, proof of inequalities and theorems, linear algebra etc. With more time, I would have liked to try to apply the algorithms presented here on a real-world problem, or to try to explore more on the idea of an adaptive oversampling parameter for SRFT matrices.

A.3 References

[Alb72] Arthur Albert (1972). *Regression and the Moore-Penrose pseudo-inverse*. Elsevier.

[AM07] Dimitris Achlioptas and Frank Mcsherry (2007). *Fast Computation of low-rank Matrix Approximations*. *Journal of the ACM (JACM)*, 54(2):9. URL `http://dl.acm.org/citation.cfm?id=1219097`.

[BG13] Christos Boutsidis and Alex Gittens (2013). *Improved matrix algorithms via the subsampled randomized Hadamard transform*. *SIAM Journal on Matrix Analysis and Applications*, 34(3):1301–1340. URL `http://epubs.siam.org/doi/abs/10.1137/120874540`.

[Cip00] Barry Cipra (May 2000). *The Best of the 20th Century: Editors Name Top 10 Algorithms*. *SIAM News*, 33(4):1.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2009). *Introduction to algorithms third edition*. The MIT Press.

[Cou08] Pierre Courrieu (2008). *Fast computation of Moore-Penrose inverse matrices*. *arXiv preprint arXiv:0804.4809*.

[CT65] James W. Cooley and John W. Tukey (1965). *An algorithm for the machine calculation of complex Fourier series*. *Mathematics of computation*, 19(90):297–301. URL `http://www.jstor.org/stable/2003354`.

[DMM08] Petros Drineas, Michael W. Mahoney, and S. Muthukrishnan (2008). *Relative-error CUR matrix decompositions*. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881. URL `http://arxiv.org/abs/0708.3696v1`.

[DS00] Jack Dongarra and Francis Sullivan (February 2000). *Top Ten Algorithms of the Century*. *Computing in Science and Engineering*, 2(1):22–23. URL `http://people.sc.fsu.edu/~jburkardt/fun/misc/algorithms_dongarra.html`.

[DV90] Pierre Duhamel and Martin Vetterli (1990). *Fast Fourier transforms: a tutorial review and a state of the art*. *Signal processing*, 19(4):259–299.

[GE96] Ming Gu and Stanley C. Eisenstat (1996). *Efficient Algorithms for Computing a strong Rank-Revealing QR Factorization*. *SIAM Journal on Scientific Computing*, 17(4):848–869. URL `http://epubs.siam.org/doi/abs/10.1137/0917055`.

[Giv01] Dan Givoli (2001). *The Top 10 Computational Methods of the 20th Century*. *IACM Expressions*, 11:5–9. URL http://people.sc.fsu.edu/~jburkardt/fun/misc/algorithms_givoli.html.

[GJ79] Michael R. Garey and David S. Johnson (1979). *Computers and Intractability: a guide to the theory of NP-completeness*. W. H. Freeman, New York.

[GR13] Jean-Philippe Grivet and Magali Ribot (DL 2013, cop. 2013). *Méthodes numériques appliquées pour le scientifique et l'ingénieur (in French)*. Grenoble Sciences. EDP sciences, Les Ulis, 2nd edition. URL http://laboutique.edpsciences.fr/produit/9782759803866.

[HMT11] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp (2011). *Finding Structure with Randomness: Probabilistic algorithms for constructing Approximate Matrix Decompositions*. *SIAM review*, 53(2):217–288. URL http://arxiv.org/abs/0909.4061.

[JL84] William B. Johnson and Joram Lindenstrauss (1984). *Extensions of Lipschitz mappings into a Hilbert space*. *Contemporary Mathematics*, 26(189-206):1.

[MT+00] George Marsaglia, Wai Wan Tsang, *et al.* (2000). *The Ziggurat Method for Generating Random Variables*. *Journal of Statistical Software*, 5(8):1–7. URL http://www.jstatsoft.org/v05/i08/.

[Pey11] Gabriel Peyré (2011). *The Numerical Tours of Signal Processing*. *Computing in Science & Engineering*, 13(4):94–97. URL http://www.numerical-tours.com/.

[Rak04] Medhat A. Rakha (2004). *On the Moore-Penrose generalized inverse matrix*. *Applied Mathematics and Computation*, 158(1):185–200.

[SS08] Daniel A. Spielman and Nikhil Srivastava (2008). *Graph Sparsification by Effective Resistances*. *CoRR*, abs/0803.0929. URL http://arxiv.org/abs/0803.0929.

[Tro12a] Joel A. Tropp (December 2012). *User-friendly tools for Random Matrices*. *Neural Information Processing Systems (NIPS), Stateline*. URL http://users.cms.caltech.edu/~jtropp/talks.html.

[Tro12b] Joel A. Tropp (December 2012). *User-friendly tools for Random Matrices*. URL https://www.youtube.com/watch?v=YSupQSKVV7w, tutorial slides, http://users.cms.caltech.edu/~jtropp/slides/Tro12-User-Friendly-Tutorial-NIPS.pdf.

[Tro14] Joel A. Tropp (June 2014). *Finding Structure with Randomness*. Tutorial slides, http://users.cms.caltech.edu/~jtropp/slides/Tro14-Finding-Structure-ICML.pdf.

[WLRT08] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert (2008). *A Fast Randomized Algorithm for the Approximation of Matrices*. *Applied and Computational Harmonic Analysis*, 25:335–366.

[Zha15] Zhihua Zhang (November 2015). *Randomized Numerical Linear Algebra (RNLA): review and progresses*. Tutorial slides, http://bcmi.sjtu.edu.cn/~zhzhang/papers/rnla.pdf.

(*Note:* a more detailed bibliography is available on-line, in HTML, PDF and BibTEX.)

License?