Randomized LU Decomposition

Gil Shabat\textsuperscript{1}*, Yaniv Shmueli\textsuperscript{2} Amir Averbuch\textsuperscript{2}

\textsuperscript{1}School of Electrical Engineering, Tel Aviv University, Israel
\textsuperscript{2}School of Computer Science, Tel Aviv University, Israel

April 14, 2014

Abstract

We present a fast randomized algorithm that computes a low rank LU decomposition. The algorithm uses random projections type techniques to efficiently compute a low rank approximation of large matrices. The randomized LU algorithm can be parallelized and further accelerated by using sparse random matrices in its projection step. Several error bounds for the algorithm’s approximations are proved. To prove these bounds, recent results from random matrix theory related to subgaussian matrices are used. The algorithm, which can utilize sparse structures, is fully parallelized and thus can utilize efficiently GPUs. Numerical examples, which illustrate the performance of the algorithm and compare it to other decomposition methods, are presented.

Keywords. LU decomposition, matrix factorizations, random matrices, randomized algorithms, sparse matrices.

1 Introduction

Matrix factorizations and low rank approximations play a major role in many of today’s applications [40]. In mathematics, matrix decompositions are used for low rank approximations that often reveal interesting properties of a matrix. Matrix decompositions are used for example in solving linear equations and in finding least squares solutions. In engineering, matrix decompositions are used in computer vision [19], machine learning [32], collaborative filtering and Big Data analytics [26]. As the size of the data grows exponentially, analysis of large datasets has gained an increasing interest. Such an analysis can involve a factorization step of the input data given as a large sample-by-feature matrix or by a sample affinity matrix. Two main reasons for the difficulties in analyzing huge data structures are high memory consumption and the computational complexity of the factorization step. Recently, there is an on-going interest in applying

*Corresponding author: gil@eng.tau.ac.il
mathematical tools that are based on randomized algorithms to overcome these difficulties.

Some of the randomized algorithms use random projections, which project the matrix to a set of random vectors. Formally, given a matrix $A$ of size $m \times n$ (suppose $m \geq n$) and a random matrix $G$ of size $n \times k$, then the product $AG$ is computed to obtain a smaller matrix that potentially captures most of the data activities in $A$. In most of these applications, $k$ is set to be much smaller than $n$ to obtain a compact approximation for $A$.

Fast randomized matrix decomposition algorithms are used for tracking objects in videos [38], multiscale extensions for data [4] and detecting anomalies in network traffic for finding cyber attacks [13], to name some. There are randomized versions for many different matrix factorization algorithms [24], compressed sensing methods [16] and least squares problems [2].

In this paper, we develop a randomized version of the LU decomposition. Given an $m \times n$ matrix $A$, we seek a lower triangular $m \times k$ matrix $L$ and an upper triangular $k \times n$ matrix $U$ such that

$$
\|LU - PAQ\|_2 = C\sigma_{k+1},
$$

(1.1)

where $P$ and $Q$ are orthogonal permutation matrices, $\sigma_{k+1}$ is the $k + 1$ largest singular value of $A$ and $C$ is a constant depending on $m$, $n$ and $k$.

The interest in a randomized LU decomposition can be motivated (computationally wise) by three important properties of the classical LU decomposition: First, it can be applied efficiently to sparse matrices with computation time depending on the number of non zero elements. Second, LU decomposition with full pivoting on sparse matrices can generate large regions of zeros in the factorized matrices [14,15,37]. Third, LU decomposition can be fully parallelized that makes it applicable for running on Graphics Processing Units (GPU). GPUs are mostly used for computer games, graphics and visualization such as movies and 3D display. Their powerful computation capabilities can be used for fast matrix computations [25].

The contributions of the paper are the following: We develop a randomized version for LU decomposition. Such an algorithm does not appear in the literature, and we provide several error bounds for the error $\|LU - PAQ\|_2$. In addition, we present a sparse version of our randomized LU algorithm along with a full implementation on a standard GPU card. We present numerical results that compare our algorithm with other decomposition methods and show it superiority.

The paper is organized as follows: In Section 2, we overview related work on matrix decomposition and approximation using randomized methods. Section 3 reviews some mathematical facts that are needed for the development of the randomized LU. Section 4 presents the randomized LU algorithm and proves several error bounds on the approximation. We discuss the case of sparse matrices and also show how to solve rank deficient least squares problems using the randomized algorithm as an example. Section 5 presents numerical results on the approximation error, the computational complexity of the algorithm and
compares it with other methods. The performance comparison was done on random matrices, images and large sparse matrices.

2 Related Work

Efficient matrix decomposition serves as a basis for many studies and algorithms for data analysis applications. There is a variety of methods and algorithms that factorize a matrix into several matrices. Typically, the factorized terms have properties such as being triangular, orthogonal, diagonal, sparse or low rank. It is possible to have a certain control on the desired approximation error on a factorized matrix.

Rank revealing factorization uses permutation matrices on the columns and rows of $A$ so that the factorized matrices structure have a strong rank portion and a rank deficient portion. The most known example for approximating an $m \times n$ matrix $A$ by a low rank $k$ matrix is the truncated SVD. Other rank revealing factorizations can be used to achieve low rank approximations. For example, both QR and LU factorizations have rank revealing versions such as RRQR decomposition [8], strong RRQR [23] decomposition, RRLU decomposition [34] and strong RRLU decomposition [33].

Other matrix factorization methods such as Interpolative Decomposition (ID) [10] and CUR decomposition [18], use columns and rows of the original matrix $A$ in the factorization process. Such a property exposes the most important terms that construct $A$. An ID factorization of order $k$ of an $m \times n$ matrix $A$ consists of an $m \times k$ matrix $B$ whose columns consist of a subset of the columns of $A$, as well as a $k \times n$ matrix $P$, such that a subset of the columns of $P$ becomes a $k \times k$ identity matrix and $A \approx BP$ such that $\|A - BP\| \lesssim \mathcal{O}(n, \sigma_{k+1}(A))$. Usually, $k$ is chosen to be the numerical rank $k = \# \{ j : \sigma_j(A) \geq \delta \sigma_1(A) \}$ of $A$ up to a certain accuracy $\delta > 0$. This selection of $k$ guarantees that the columns of $B$ constitute a well conditioned basis to the range of $A$ [10].

Randomized version for many important algorithms have been developed in order to deal with computational complexity by approximating the solution to a desired rank. These include SVD, QR and ID factorizations [31], CUR decomposition as a randomized version [18] of the pseudo-skeleton decomposition, methods for solving least squares problems [2, 12, 35] and low rank approximations [1,12].

In general, randomization methods for matrix factorization have two steps. First, a low-dimensional space, which captures most of the “energy" of $A$, is found using randomization. Then, $A$ is projected into the retrieved subspace and projected matrix is factorized [24].

Several different selections exist for the random projection matrix, which is used in Step 1. For example, it can be a matrix of random signs $(\pm 1)$ [11,30]; a matrix of i.i.d Gaussian random variables with zero mean and unit variance [31]; a matrix whose columns are selected randomly from the identity matrix with either uniform or nonuniform probability [17,20]; a random sparse matrix designed to enable fast multiplication with a sparse input matrix $A$ [1,12]; ran-
dom structured matrices that use orthogonal transforms such as discrete Fourier transform, Walsh-Hadamard transform and so on ([2, 7, 35]). In our algorithm, we use Gaussian matrices in Step 1 as well as sparse Gaussian matrices (a special case of sub-Gaussian matrices) when factorizing sparse matrices.

3 Preliminaries

In this section, we review the rank revealing LU (RRLU) decomposition and several singular values bounds on random matrices that will be used to prove the error bounds for the randomized LU algorithm. Throughout the paper, we use the following notation: for any matrix $A$, $\sigma_j(A)$ is the $j$th largest singular value and $\|A\|$ is the spectral norm (the largest singular value or $l_2$ operator norm). If $x$ is a vector then $\|x\|$ is the standard $l_2$ (Euclidean) norm. $A^\dagger$ denotes the pseudo-inverse of $A$.

3.1 Rank Revealing LU (RRLU)

The following theorem is adapted from [34] (Theorem 1.2):

**Theorem 3.1** ([34]). Let $A$ be an $m \times n$ matrix ($m \geq n$). Given an integer $1 \leq k < n$, then the following factorization

$$PAQ = \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{n-k} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix},$$

(3.1)

holds where $L_{11}$ is a unit lower triangular, $U_{11}$ is an upper triangular, $P$ and $Q$ are orthogonal permutation matrices. Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$ be the singular values of $A$, then:

$$\sigma_k \geq \sigma_{\text{min}}(L_{11}U_{11}) \geq \frac{\sigma_k}{k(n-k)+1},$$

(3.2)

and

$$\sigma_{k+1} \leq \|U_{22}\| \leq (k(n-k)+1)\sigma_{k+1}.$$  

(3.3)

Based on Theorem 3.1, we have the following definition:

**Definition 3.1** (RRLU Rank $k$ Approximation denoted $\text{RRLU}_k$). Given a RRLU decomposition (Theorem 3.1) of a matrix $A$ with an integer $k$ (as in Eq. 3.1) such that $PAQ = LU$, then the RRLU rank $k$ approximation is defined by taking $k$ columns from $L$ and $k$ rows from $U$ such that

$$\text{RRLU}_k(PAQ) = \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} (U_{11}U_{12}).$$

(3.4)

where $L_{11}, L_{21}, U_{11}, U_{12}, P$ and $Q$ are defined in Theorem 3.1

**Lemma 3.2** (RRLU Approximation Error). The error of the $\text{RRLU}_k$ approximation of $A$ is

$$\|PAQ - \text{RRLU}_k(PAQ)\| \leq (k(n-k)+1)\sigma_{k+1}.$$  

(3.5)
Proof. From Eqs. 3.1 and 3.4 we have
\[
\| \frac{PAQ - RRLU_k(PAQ)}{k} \| = \| \begin{pmatrix} L_{11} & 0 \\ L_{21} & I_{n-k} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} - \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix} \begin{pmatrix} U_{11} \\ U_{12} \end{pmatrix} \|
\]
\[
= \| U_{22} \| \leq (k(n-k) + 1)\sigma_{k+1}.
\]

The last inequality is derived from Eq. 3.3.
\[\square\]

Lemma 3.3 appears in [5], page 75:

**Lemma 3.3 ([5]).** Let \(A\) and \(B\) be two matrices and let \(\sigma_j(\cdot)\) denotes the \(j\)th singular value of a matrix. Then, \(\sigma_j(AB) \leq \|A\|\sigma_j(B)\) and \(\sigma_j(AB) \leq \|B\|\sigma_j(A)\).

Lemma 3.4 was taken from [31] and it is an equivalent formulation for Eq. 8.8 in [21].

**Lemma 3.4 ([31]).** Suppose that \(G\) is a real \(n \times l\) matrix whose entries are i.i.d Gaussian random variables with zero mean and unit variance and let \(m\) be an integer such that \(m \geq l, m \geq n, \gamma > 1\) and
\[
1 - \frac{1}{4(\gamma^2 - 1)\sqrt{\pi m\gamma^2}} \left( \frac{2\gamma^2}{e^{\gamma^2-1}} \right)^m \quad (3.7)
\]
is nonnegative. Then, \(\|G\| \leq \sqrt{2m\gamma}\) with probability not less than the value in Eq. 3.7.

### 3.2 Sparse Random Matrices

Sparse matrices have a significant importance in many applications. The computation of the degrees of separation between two individuals using Facebook 2011 connection matrix is a typical example. It requires to factorize a sparse matrix of size \(720,000,000 \times 720,000,000\) that has 69 billions connections. It means that only \(1.33 \times 10^{-5}\) percent of the matrix is non-zero [3]. The advantage of using sparse matrices is evident.

**Definition 3.2** (Sparse Gaussian matrix). \(A = (\xi_{ij})\) is a sparse Gaussian matrix if each entry is centered normally distributed with probability \(\rho\) and zero with probability \(1-\rho\). That is, \(\xi_{ij}\) is a random variable whose probability density function (PDF) is given by
\[
p(x) = (1 - \rho)\delta(x) + \frac{\rho}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}, \quad (3.8)
\]
where \(\delta(x)\) is the Dirac delta function, \(\sigma > 0\) and \(0 < \rho \leq 1\). If \(A\) is a sparse matrix whose non-zero entries are Gaussian variables, then we refer to \(\rho\) as the density of the matrix.
Definition 3.3. A real valued random variable $X$ is called subgaussian if there exists $b > 0$ such that for all $t > 0$, we have $\mathbb{E}e^{tX} \leq e^{bt^2/2}$ where $\mathbb{E}$ is the expectation.

Suppose $X$ is distributed as in Eq. 3.8 and $\mathbb{E}$ is the expectation. It can be easily verified that:

1. $\mathbb{E}X = 0$;
2. $\mathbb{E}X^2 = \rho \sigma^2$;
3. $\mathbb{E}|X|^3 = \frac{4\rho \sigma^3}{\sqrt{2\pi}}$;
4. $X$ is subgaussian.

We review several facts adapted from [29] and [36] about random matrices whose entries are subgaussian. We focus on the case where $A$ is a tall $m \times n$ matrix ($m > (1 + \frac{1}{\ln n})n$). Similar results can be found in [28] for square and almost square matrices.

Definition 3.4. Assume that $\mu \geq 1$, $a_1 > 0$ and $a_2 > 0$. $A(\mu, a_1, a_2, m, n)$ is the set of all $m \times n$ ($m > n$) random matrices $A = (\xi_{ij})$ whose entries are i.i.d real valued centered random variables satisfying the following conditions:

1. Moments: $\mathbb{E}|\xi_{ij}|^3 \leq \mu^3$;
2. Norm: $\mathbb{P} (\|A\| > a_1 \sqrt{m}) \leq e^{-a_2 m}$ where $\mathbb{P}$ is the probability function;
3. Variance: $\mathbb{E} \xi_{ij}^2 \geq 1$.

It is shown in [29] that if $A$ is subgaussian then $A \in A$. In particular, a Gaussian matrix whose entries are zero with probability $1 - \rho$ is also subgaussian. Hence, this model (Definition 3.4) can also be used for sparse Gaussian matrices with density $\rho$. For simplicity, we work with random matrices with unit variance.

In the case of sparse Gaussian matrices, we set: $\sigma^2 = \frac{1}{\rho}$, then $\mu = \left( \frac{4}{\sqrt{2\pi \rho}} \right)^{\frac{1}{3}}$.

The following theorems are taken from Section 2 in [29]:

Theorem 3.5 ([29]). Every matrix $A$ of size $m \times n$ ($m \geq n$) whose entries are subgaussian with $\mu \geq 1$ and $a_2 \geq 0$ satisfies:

$$
\mathbb{P} (\|A\| \geq a_1 \sqrt{m}) \leq e^{-a_2 m}
$$

(3.9) with $a_1 = 6\mu \sqrt{a_2 + 4}$.

Theorem 3.5 provides an upper bound for the largest singular value that depends on the desired probability. Theorem 3.6 is used to bound from below the smallest singular value of sparse Gaussian matrices.
Theorem 3.6 ([29]). Let $\mu \geq 1$, $a_1, a_2 > 0$. Let $A$ be an $m \times n$ matrix with $m > (1 + \frac{1}{\ln n})n$. $m$ can be written as $m = (1 + \delta)n$. Suppose the entries of $A$ are independent centered random variables such that conditions 1, 2, 3 in Definition 3.4 hold. Then, there exist positive constants $c_1$ and $c_2$ such that:

$$
\mathbb{P}(\sigma_n(A) \leq c_1 \sqrt{m}) \leq e^{-m} + e^{-c''m/(2\mu^6)} + e^{-a_2m} \leq e^{-c_2m}. \quad (3.10)
$$

The exact values of constants $c_1, c_2$ and $c''$ are given by:

$$
c_1 = \frac{b}{e^2c_3} \left( \frac{b}{3e^2c_3a_1} \right) \frac{1}{2}, \quad (3.11)
$$

$$
c'' = \frac{27}{211}. \quad (3.12)
$$

Here, $c_3 = 4\sqrt{\frac{2}{\pi}} \left( \frac{2\mu^6}{a_1} + \sqrt{\frac{\pi}{\pi}} \right)$, $b = \min \left( \frac{1}{4}, \frac{c'}{5a_1\mu^6} \right)$ and $c' = \left( \frac{27}{211} \right)^{\frac{1}{2}}$. For the constant $c_2$, we need a small enough constant to satisfy the inequality in Eq. 3.10 and set it, for simplification, as

$$
c_2 = \min \left( 1, \frac{c'}{(2\mu^6)}, a_2 \right) - \frac{\ln 3}{m}. \quad (3.13)
$$

4 Randomized LU

In this section, we present the randomized LU algorithm (Algorithm 4.1) that computes the LU rank $k$ approximation of a full matrix. In addition, we present a version (Algorithm 4.2) that approximates a sparse matrix. Error bounds are proven for each algorithm.

The algorithm starts by projecting the input matrix on a random matrix. The resulting matrix captures most of the information of the input matrix. Then, we compute a triangular basis for this matrix and project the input matrix on it. Last, we find a second triangular basis for the projected columns and multiply it with the original basis. The product leads to a lower triangular matrix, $L$ and the $U$ matrix is the upper triangular matrix obtained from the second LU factorization.
Algorithm 4.1: Randomized LU Decomposition

**Input**: A matrix of size $m \times n$ to decompose; $k$ desired rank; $l$ number of columns to use.

**Output**: Matrices $P, Q, L, U$ such that $\|PAQ - LU\| \leq O(\sigma_{k+1}(A))$

where $P$ and $Q$ are orthogonal permutation matrices, $L$ and $U$ are the lower and upper triangular matrices, respectively.

1: Create a matrix $G$ of size $n \times l$ whose entries are i.i.d. Gaussian random variables with zero mean and unit standard deviation.
2: $Y \leftarrow AG$.
3: Apply RRLU decomposition (Theorem 3.1) to $Y$ such that $P Y Q_y = L_y U_y$.
4: Truncate $L_y$ and $U_y$ by choosing the first $k$ columns and the first $k$ rows, respectively, such that $L_y \leftarrow L_y(:, 1 : k)$ and $U_y \leftarrow U_y(1 : k, :)$.
5: $B \leftarrow L^\dagger_y PA$.
6: Apply LU decomposition to $B$ with column pivoting $BQ = L_b U_b$.
7: $L \leftarrow L_y L_b$.
8: $U \leftarrow U_b$.

**Remark 4.1.** The pseudo-inverse of $L_y$ in step 5 can be computed by $L^\dagger_y = (L^T_y L_y)^{-1} L^T_y$. This can be done efficiently when it is computed on platforms such as GPUs that can multiply matrices efficiently. Usually, the inversion is done on a small matrix since in many cases $k \ll n$ and it can be done by the application of Gaussian elimination.

**Remark 4.2.** In practice, it is sufficient to perform step 3 in Algorithm 4.1 using standard LU decomposition with partial pivoting instead of applying RRLU. The cases where $U$ grows exponentially are extremely rare (see section 3.4.5 in [22] and [41]).

We now present our main error bound for Algorithm 4.1:

**Theorem 4.3.** Given a matrix $A$ of size $m \times n$. Then, its randomized LU decomposition produced by Algorithm 4.1 with integers $k$ and $l$ $(l \geq k)$ satisfies:

$$\|LU - PAQ\| \leq \left(2\sqrt{2nl\beta^2 \gamma^2} + 1 + 2\sqrt{2nl\beta \gamma (k(n - k) + 1)}\right) \sigma_{k+1}(A),$$

(4.1)

with probability not less than

$$\xi \triangleq 1 - \frac{1}{\sqrt{2\pi(l - k + 1)}} \left(\frac{e}{(l - k + 1)\beta}\right)^{l-k+1} - \frac{1}{4(\gamma^2 - 1)\sqrt{\pi n \gamma^2}} \left(\frac{2\gamma^2}{e\gamma^2 - 1}\right)^n,$$

(4.2)

where $\beta > 0$ and $\gamma > 1$.

The proof of Theorem 4.3 is given in Section 4.2. To show that the success probability $\xi$ in Eq. 4.2 is sufficiently high, we present in Table 4.1 several calculated values of $\xi$. We omitted the value of $n$ from Table 4.1 since it does
not affect the value of $\xi$ due to the fact that the second term in Eq. 4.2 decays fast.

Table 4.1: Calculated values of the success probability $\xi$ (Eq. 4.2). The terms $l-k$, $\beta$ and $\gamma$ appears in Eq. 4.2.

<table>
<thead>
<tr>
<th>$l-k$</th>
<th>$\beta$</th>
<th>$\gamma$</th>
<th>$\xi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
<td>$1 - 6.8 \times 10^{-5}$</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
<td>$1 - 9.0 \times 10^{-8}$</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>5</td>
<td>$1 - 5.2 \times 10^{-16}$</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>5</td>
<td>$1 - 5.2 \times 10^{-8}$</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>5</td>
<td>$1 - 1.9 \times 10^{-12}$</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>5</td>
<td>$1 - 1.4 \times 10^{-24}$</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>10</td>
<td>$1 - 5.2 \times 10^{-8}$</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>10</td>
<td>$1 - 1.9 \times 10^{-12}$</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>10</td>
<td>$1 - 1.4 \times 10^{-24}$</td>
</tr>
</tbody>
</table>

In Section 5, we show that in practice, Algorithm 4.1 produces comparable results to other well known randomized factorization methods of low rank matrices such as randomized SVD and randomized ID.

4.1 Computational Complexity Analysis

To compute the number of floating points operations in Algorithm 4.1, we evaluate the complexity of each step:

1. Generating an $n \times l$ random matrix requires $\mathcal{O}(nl)$ operations.

2. Multiplying $A$ by $G$ to form $Y$ requires $lC_A$ operations, where $C_A$ is the complexity of applying $A$ to an $n \times 1$ column vector.

3. Partial pivoting computation of LU for $Y$ requires $\mathcal{O}(ml^2)$ operations.

4. Selecting the first $k$ columns (we do not modify them) requires $\mathcal{O}(1)$ operations.

5. Computing the pseudo inverse of $L_y$ requires $\mathcal{O}(k^2m + k^3 + k^2m)$ operations and multiplying it by $A$ requires $kC_{AT}$ operations. Note that $P$ is a permutation matrix that does not modify the rows of $A$.

6. Computing the partial pivoting LU for $B$ requires $\mathcal{O}(k^2n)$ operations.

7. Computing $L$ requires $\mathcal{O}(k^2m)$ operations.

8. Computing $U$ requires $\mathcal{O}(1)$ operations.

By summing up the complexities of all the steps above, then Algorithm 4.1 necessitated

$$\mathcal{C}_{RandLU} = lC_A + kC_{AT} + \mathcal{O}(l^2m + k^3 + k^2n)$$

(4.3)
operations. Here, we used $C_A$ (and $C_{AT}$) to denote the complexity of applying $A$ (and $AT$) to a vector, respectively. For a general $A$, $C_A = C_{AT} = O(mn)$.

4.2 Bounds for the Randomized LU (Proof of Theorem 4.3)

In this section, we prove Theorem 4.3 and an additional complementary bound. This is done by finding a basis to a smaller matrix $AG$, which is achieved in practice by using RRLU. The assumptions are that $L$ is numerically stable so its pseudo-inverse can be computed accurately, that there exists a matrix $U$ such that $LU$ is a good approximation to $AG$ and that there exists a matrix $F$ such that $\|AGF - A\|$ is small. As for the numerical stability of $L$, it is always stable since it has a small condition number [39].

For the proof of Theorem 4.3, several lemmas are needed. Lemma 4.4 states that a given basis $L$ can form a basis for the columns $A$ by bounding the error $\|LL^\dagger A - A\|$. The lemma uses additional smaller matrices ($F$, $U$ and $G$). Later on, we use this lemma by using $G$ and $U$ from Algorithm 4.1 and $F$ from Lemma 4.5.

**Lemma 4.4.** Assume that $A$ is an $m \times n$ matrix, $L$ is an $m \times k$ matrix with rank $k$, $G$ is an $n \times l$ matrix, $U$ is a $k \times l$ matrix and $F$ is $l \times n$ ($k \leq m$) matrix. Then,

$$\|LL^\dagger A - A\| \leq 2\|AGF - A\| + 2\|F\|\|LU - AG\|.$$ (4.4)

**Proof.** By using the triangular inequality we get

$$\|LL^\dagger A - A\| \leq \|LL^\dagger A - LL^\dagger AGF\| + \|LL^\dagger AGF - AGF\| + \|AGF - A\|. \quad (4.5)$$

Clearly, the first term can be bounded by

$$\|LL^\dagger A - LL^\dagger AGF\| \leq \|LL^\dagger\|\|A - AGF\| \leq \|A - AGF\|. \quad (4.6)$$

The second term can be bounded by

$$\|LL^\dagger AGF - AGF\| \leq \|F\|\|LL^\dagger AG - AG\|. \quad (4.7)$$

Also,

$$\|LL^\dagger AG - AG\| \leq \|LL^\dagger AG - LL^\dagger LU\| + \|LL^\dagger LU - LU\| + \|LU - AG\|. \quad (4.8)$$

Since $L^\dagger L = I$, it follows that $\|LL^\dagger LU - LU\| = 0$ and that $\|LL^\dagger AG - LL^\dagger LU\| \leq \|AG - LU\|$. When combined with Eq. 4.8 we obtain:

$$\|LL^\dagger AG - AG\| \leq 2\|LU - AG\|. \quad (4.9)$$

By substituting Eq. 4.9 in Eq. 4.7 we get

$$\|LL^\dagger AGF - AGF\| \leq 2\|F\|\|LU - AG\|. \quad (4.10)$$

By substituting Eqs. 4.6 and 4.10 in Eq. 4.5 we get

$$\|LL^\dagger A - A\| \leq 2\|AGF - A\| + 2\|F\|\|LU - AG\|. \quad (4.11)$$

□
Lemma 4.5 appears in [31]. It uses a lower bound for the least singular value of a Gaussian matrix with zero mean and unit variance. This bound can be found in [9].

**Lemma 4.5 ([31]).** Assume that \(k, l, m\) and \(n\) are positive integers such that \(k \leq l, l \leq m\) and \(l \leq n\). Assume that \(A\) is a real \(m \times n\) matrix, \(G\) is \(n \times l\) whose entries are i.i.d Gaussian random variables of zero mean and unit variance, \(\beta\) and \(\gamma\) are real numbers, such that \(\beta > 0, \gamma > 1\) and the quantity

\[
1 - \frac{1}{\sqrt{2\pi(l-k+1)}} \left( \frac{e}{(l-k+1)\beta} \right)^{l-k+1} - \frac{1}{4(\gamma^2 - 1)\sqrt{\pi n\gamma^2}} \left( \frac{2\gamma^2}{e^{\gamma^2-1}} \right)^n
\]  

(4.12)

is nonnegative. Then, there exists a real \(l \times n\) matrix \(F\) such that

\[
\|AGF - A\| \leq \sqrt{2nl\beta^2\gamma^2 + 1}\sigma_{k+1}(A)
\]

(4.13)

and

\[
\|F\| \leq \sqrt{l}\beta
\]

(4.14)

with probability not less than the value in Eq. 4.12.

Lemma 4.6 rephrases Lemma 4.5 by utilizing the bounds that appear in Section 3.2. The proof is close to the argumentation that appear in the proof of Lemma 4.5.

**Lemma 4.6.** Let \(A\) be a real \(m \times n\) \((m \geq n)\) matrix. Let \(G\) be a real \(n \times l\) matrix whose entries are Gaussian i.i.d with zero mean and unit variance. Let \(k\) and \(l\) be integers such that \(l < m, l < n\) and \(l > (1 + \frac{1}{\ln k})k\). We define \(a_1, a_2, c_1\) and \(c_2\) as in Theorem 3.6. Then, there exists a real matrix \(F\) of size \(l \times n\) such that:

\[
\|AGF - A\| \leq \sqrt{\frac{a_2^2n}{c_1^2l} + 1}\sigma_{k+1}(A),
\]

(4.15)

and

\[
\|F\| \leq \frac{1}{c_1\sqrt{l}}
\]

(4.16)

with probability not less than \(1 - e^{-c_2l} - e^{-a_2n}\).

**Proof.** We begin by forming the SVD of \(A\)

\[
A = U\Sigma V^T,
\]

(4.17)

where \(U\) is orthogonal \(m \times m\) matrix, \(\Sigma\) is \(m \times n\) diagonal matrix with non-negative entries and \(V\) is orthogonal matrix \(n \times n\). Given \(V^T\) and \(G\), suppose that

\[
V^TG = \begin{pmatrix} H \\ R \end{pmatrix},
\]

(4.18)

where \(H\) is \(k \times l\) and and \(R\) is \((n-k) \times l\). Since \(G\) is a Gaussian i.i.d. matrix and \(V\) is an orthogonal matrix, \(V^TG\) is also a Gaussian i.i.d. matrix. Therefore, \(H\)
is a Gaussian i.i.d. matrix. Let us define $F = PV^T$, where $P$ is of size $l \times n$ such that

$$P = \begin{pmatrix} H^\dagger & 0 \end{pmatrix}.$$ 

Therefore,

$$F = (H^\dagger 0)V^T.$$  (4.19)

Computing $\|F\|$ using Theorem 3.6 gives:

$$\|F\| = \|PV^T\| = \|H\dagger\| = \|H^T(HH^T)^{-1}\| = \frac{1}{\sigma_k(H)} \leq \frac{1}{c_1\sqrt{l}}$$  (4.20)

with probability not less than $1 - e^{-c_2l}$. Now we can bound $\|AGF - A\|$. By using Eqs. 4.17, 4.18 and 4.19 we get

$$AGF - A = U\Sigma \begin{pmatrix} H \cr R \end{pmatrix} \begin{pmatrix} H^\dagger & 0 \end{pmatrix} - I) V^T.$$  (4.21)

We define $S$ to be the upper-left $k \times k$ block of $\Sigma$ and $T$ to be the lower-right $(n - k) \times (n - k)$ block. Then,

$$\Sigma \begin{pmatrix} H \cr R \end{pmatrix} \begin{pmatrix} H^\dagger & 0 \end{pmatrix} - I) = \begin{pmatrix} S & 0 \\
0 & T \end{pmatrix} \begin{pmatrix} 0 & 0 \\
0 & RH^\dagger \end{pmatrix} - I = \begin{pmatrix} 0 & 0 \\
TRH^\dagger & -T \end{pmatrix}.$$ 

The norm of the last term can be rewritten as:

$$\left\| \begin{pmatrix} 0 & 0 \\
TRH^\dagger & -T \end{pmatrix} \right\|^2 \leq \|TRH^\dagger\|^2 + \|T\|^2.$$  (4.22)

Therefore, by using Eqs. 4.21, 4.22 and the fact that $\|T\| = \sigma_{k+1}(A)$, we get

$$\|AGF - A\| \leq \sqrt{\|TRH^\dagger\|^2 + \|T\|^2} \leq \sqrt{\|H^\dagger\|^2\|R\|^2 + 1}\sigma_{k+1}(A).$$  (4.23)

Also we know that

$$\|R\| \leq \|V^T G\| = \|G\| \leq a_1 \sqrt{n}$$

with probability not less than $1 - e^{-a_2n}$. Combining Eq. 4.23 with the fact that $\|H^\dagger\| \leq \frac{1}{c_1\sqrt{l}}$ and $\|R\| \leq a_1 \sqrt{n}$ gives:

$$\|AGF - A\| \leq \sigma_{k+1}(A)\sqrt{\frac{a_1^2n}{c_1^2l}} + 1.$$  (4.24)

Remark 4.7. In contrast to Lemma 4.5 where $\|AGF - A\| = \mathcal{O}(\sqrt{n})$, Lemma 4.6 provides the bound $\|AGF - A\| = \mathcal{O}(\sqrt{\frac{n}{l}})$ that is tighter for large values of $l$.

\[\square\]
Remark 4.8. The condition \( l > \left( 1 + \frac{1}{\ln k} \right) k \) in Lemma 4.6 is satisfied without a dramatic increase of the computational complexity of Algorithm 4.1. However, there are bounds for the case where \( H \) is almost square \( (l \approx k) \) and square \( (l = k) \) and they are given in [28].

Proof of Theorem 4.3. The error is given by the expression \( \|LU - PAQ\| \) where \( L, U, P \) and \( Q \) are the outputs of Algorithm 4.1 whose inputs are the matrix \( A \), integers \( k \) and \( l \). From Steps 7 and 8 in Algorithm 4.1 we have

\[
\|LU - PAQ\| = \|L_yL_bU_b - PAQ\|. \tag{4.25}
\]

Here, \( L_y \) is the \( m \times k \) matrix in step 4. By using the fact that \( BQ = L_bU_b = L_y^\dagger PAQ \), we get

\[
\|LU - PAQ\| = \|L_yL_bU_b - PAQ\| = \|L_yL_y^\dagger PAQ - PAQ\|. \tag{4.26}
\]

Applying Lemma 4.4 gives that

\[
\|LU - PAQ\| = \|L_yL_y^\dagger PAQ - PAQ\| \leq 2\|PAQ\tilde{G}F - PAQ\| + 2\|F\|\|L_yU_y - PAQ\tilde{G}\|. \tag{4.27}
\]

Here, \( U_y \) is the \( k \times n \) matrix in step 4 in Algorithm 4.1. This holds for any matrix \( \tilde{G} \). In particular, for a matrix \( \tilde{G} = GQ_y \), where \( G \) is a random Gaussian i.i.d. matrix. \( G \) is in fact \( \tilde{G} \) after row and columns permutations. Therefore, the last term can be reformulated as \( \|L_yU_y - PAQ\tilde{G}\| = \|L_yU_y - PAGQ_y\| \) where \( G \) is the random matrix in Algorithm 4.1. By applying Lemmas 3.2 and 3.3 to \( \|L_yU_y - PAQ\tilde{G}\| \) we get

\[
\|L_yU_y - PAQ\tilde{G}\| = \|L_yU_y - PAGQ_y\| \leq (k(n-k)+1)\sigma_{k+1}(AG) \tag{4.28}
\]

\[
\leq (k(n-k)+1)\|G\|\sigma_{k+1}(A).
\]

Lemma 4.5 gives that \( \|PAQ\tilde{G}F - PAQ\| \leq \sqrt{2nl\beta^2\gamma^2 + 1}\sigma_{k+1}(A) \) and \( \|F\| \leq \sqrt{l}\beta \). By combining Lemmas 4.5 and 3.4 we get

\[
\|LU - PAQ\| \leq \left( 2\sqrt{2nl\beta^2\gamma^2 + 1} + 2\sqrt{2nl\beta\gamma (k(n-k)+1)} \right) \sigma_{k+1}(A), \tag{4.29}
\]

which completes the proof. \( \square \)

Remark 4.9. The error in Theorem 4.3 may appear large, especially for the case where \( k \approx \frac{n}{2} \) and \( n \) is large. Yet, we performed extensive numerical experiments showing that the actual error was much smaller when using Gaussian elimination with partial pivoting. Note that the error can decrease by increasing \( k \). This is applicable to certain applications. Numerical illustrations appear in section 5.
We now present an additional error bound that relies on [29]. Asymptotically, this is a sharper bound for large values of $n$ and $l$, since it contains the term $\sqrt{\frac{n}{l}}$, which is smaller than the term $\sqrt{n}l$ appear in Theorem 4.3. See also Remark 4.7.

**Theorem 4.10.** Given a matrix $A$ of size $m \times n$, integers $k$ and $l$ such that $l > (1 + \frac{1}{\ln k})k$ and $a_2 > 0$. By applying Algorithm 4.1 with $A, k$ and $l$ as its input parameters, we get a randomized LU decomposition that satisfies

$$
\|LU - PAQ\| \leq \left( 2\sqrt{\frac{a_1^2n}{c_1^2l}} + 1 + \frac{2a_1\sqrt{n}}{c_1\sqrt{l}} (k(n - k) + 1) \right) \sigma_{k+1}(A),
$$

with probability not less than $1 - e^{-a_2n} - e^{-c_2l}$. The value of $c_1$ is given in Eq. 3.11 and the value of $c_2$ is given in Eq. 3.13. Both values depend on $a_2$.

**Proof.** By using steps 5, 6, 7 and 8 in Algorithm 4.1, we get that

$$
\|LU - PAQ\| = \|L_yL_y^\dagger PAQ - PAQ\|.
$$

Then, from Lemma 4.4,

$$
\|L_yL_y^\dagger PAQ - PAQ\| \leq 2\|PAQ\tilde{G}F - PAQ\| + 2\|F\|\|L_yU_y - PAQ\tilde{G}\|.
$$

From Lemma 4.6 we get that

$$
\|PAQ\tilde{G}F - PAQ\| \leq \sqrt{\frac{a_1^2n}{c_1^2l}} + 1\sigma_{k+1}(A).
$$

Using the same argumentation given in Theorem 4.3, we get

$$
\|L_yU_y - PAQ\tilde{G}\| = \|L_yU_y - PAGQ_y\| \leq (k(n - k) + 1) \|G\|\sigma_{k+1}(A)
$$

where $G$ is the matrix used in Algorithm 4.1 Step 1. Combining Eqs. 4.32, 4.33, 4.34 and since $\|F\| \leq \frac{1}{c_1\sqrt{l}}$, $\|G\| \leq a_1\sqrt{n}$ (see Lemma 4.6 and Theorem 3.5, respectively), we get that

$$
\|LU - PAQ\| \leq 2\sqrt{\frac{a_1^2n}{c_1^2l}} + 1\sigma_{k+1}(A) + \frac{2a_1\sqrt{n}}{c_1\sqrt{l}} (k(n - k) + 1) \sigma_{k+1}(A).
$$

Here, $\mu = \left( \frac{4}{\sqrt{2\pi}} \right)^{\frac{\mu}{2}}$, $a_1$ is given by Theorem 3.5 and $c_1$ is given by Eq. 3.11. 

**4.3 Randomized LU for Sparse Matrices**

Assume that $A$ is a sparse matrix. We want to compute its approximated $LU$ factorization by applying Algorithm 4.1 to $A$. If the random matrix $G$, which is used in step 1 in Algorithm 4.1, is sparse, then $AG$ can be done by applying a sparse matrix multiplication. If this product is also sparse, then its LU
decomposition can be computed by using sparse LU, which is more efficient. Moreover, the resulted matrices $L$ and $U$ are also sparse. This observation can help to accelerate even more the randomized LU on large sparse matrices. In this section, we derive the randomized LU approximation error bound using the tools introduced in Section 3.2. As can be seen from Theorem 4.3, the error depends on both the largest and the smallest singular values of the random matrix $G$ which is a sparse random matrix. Therefore, we compute the error bounds for the singular values of $G$ when $G$ is a sparse random matrix with density $\rho$. We present the Randomized LU algorithm for sparse matrices in Algorithm 4.2 and its error bound in Theorem 4.11.

Algorithm 4.2: Randomized LU Decomposition for Sparse Matrices

**Input:** A sparse matrix of size $m \times n$ to decompose; $k$ desired rank; $l$ number of columns to use; $\rho$ - random matrix density.

**Output:** Matrices $P, Q, L$ and $U$ such that $\|PAQ - LU\| \leq O(\sigma_{k+1}(A))$ where $P$ and $Q$ are orthogonal permutation matrices and $L$ and $U$ are sparse lower and upper triangular matrices, respectively.

1. Apply Algorithm 4.1 where $G$ is a sparse random Gaussian matrix, with density $\rho$ and its non-zero entries are $g_{ij} \sim N(0, 1)$.

Algorithm 4.2 finds an LU rank $k$ decomposition of $A$ since it is based on Algorithm 4.1. However, the error bound proof (Theorem 4.10) for a sparse Gaussian $G$ is not applicable here since it relies on the fact that $GU$ is also a Gaussian i.i.d ($U$ is an orthogonal matrix), which is not the case when $G$ is a sparse Gaussian matrix. Therefore, we present a modified error bound (Theorem 4.11) for Algorithm 4.2.

**Theorem 4.11.** Given a matrix $A$, integers $k$ and $l$ such that $l > \left(1 + \frac{1}{\ln k}\right)k$, $k \approx \frac{l}{2}$, $a_2 > 0$ and $0 < \rho < 1$. By applying Algorithm 4.2 with $A, k, l, \rho$ as its input parameters, the resulted randomized LU decomposition satisfies

\[
\|LU - PAQ\| \leq \left(2\sqrt{\frac{a_2^2 n}{c_1^2 l}} + 1 + \frac{2a_1\sqrt{n}}{c_1 \sqrt{l}} (k(n - k) + 1)\right) \sigma_{k+1}(A),
\]

with probability not less than $1 - e^{-a_2 n} - e^{-c_2 l}$. The values of $c_1$ and $c_2$ (which depend on $\rho$) are given in Eqs. 3.11 and 3.13, respectively.

To prove the error bound for Algorithm 4.2, we present Conjecture 4.12 that bounds the $k$th singular value of a sparse subgaussian matrix multiplied by an orthogonal matrix.

**Conjecture 4.12.** Let $G$ be an $n \times l$ sparse Gaussian matrix with density $\rho \ll 1$. Assume that $l$ and $k$ are integers such that $k \approx \frac{l}{2}$. $Q$ is an $n \times n$ orthogonal matrix. Define $G_1$ as a matrix with the $k$ top rows of $G$ and $B_1$ as a matrix with the $k$ top rows of $QG$. Then, for $n$ and $l$ sufficiently large, $\sigma_k(G_1) \leq \sigma_k(B_1)$ with high probability.
To verify that Conjecture 4.12 holds experimentally, we calculated the failure probability \( \mathbb{P}(\sigma_k(G_1) > \sigma_k(B_1)) \) by computing the \( k \)th singular value of \( B_1 \) and \( G_1 \) 10,000 times for different values of \( n, l, k \) and \( \rho \). The results are presented in Table 4.2.

Table 4.2: Probability \( \mathbb{P} \) of the failure of Conjecture 4.12. The Average value of \( \sigma_k(G_1) \) was computed 10,000 times for different values of \( n, l, k \) and \( \rho \).

<table>
<thead>
<tr>
<th>( n )</th>
<th>( l )</th>
<th>( k )</th>
<th>( \rho )</th>
<th>Average ( \sigma_k(G_1) )</th>
<th>( \mathbb{P}(\sigma_k(G_1) &gt; \sigma_k(B_1)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>3000</td>
<td>200</td>
<td>100</td>
<td>0.03</td>
<td>0.611</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>3000</td>
<td>300</td>
<td>200</td>
<td>0.03</td>
<td>1.324</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>3000</td>
<td>400</td>
<td>200</td>
<td>0.02</td>
<td>2.602</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>3000</td>
<td>700</td>
<td>400</td>
<td>0.02</td>
<td>4.165</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>3000</td>
<td>900</td>
<td>400</td>
<td>0.03</td>
<td>9.215</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>4000</td>
<td>300</td>
<td>100</td>
<td>0.03</td>
<td>3.663</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>4000</td>
<td>500</td>
<td>300</td>
<td>0.02</td>
<td>3.356</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>4000</td>
<td>700</td>
<td>400</td>
<td>0.03</td>
<td>5.718</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>4000</td>
<td>700</td>
<td>300</td>
<td>0.01</td>
<td>0.923</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>4000</td>
<td>700</td>
<td>300</td>
<td>0.02</td>
<td>6.578</td>
<td>(&lt; 10^{-4})</td>
</tr>
<tr>
<td>4000</td>
<td>700</td>
<td>300</td>
<td>0.03</td>
<td>6.738</td>
<td>(&lt; 10^{-4})</td>
</tr>
</tbody>
</table>

Conjecture 4.12 is used to prove Lemma 4.13, which is similar to Lemma 4.6.

**Lemma 4.13** (Based on Conjecture 4.12). Let \( A \) be a real \( m \times n \) \((m \geq n)\) matrix. Let \( G \) be a real \( n \times l \) sparse Gaussian matrix with density \( \rho \ll 1 \), and i.i.d. entries with unit variance. Let \( k \) and \( l \) be integers such that \( l < m \), \( l < n \), \( l > (1 + \frac{1}{\ln k}) k \) and \( k \approx l^2 \). Let \( F \) be a real matrix of size \( l \times n \). We define \( a_1, a_2, c_1, c_2 \) as in Theorem 3.6. Then,

\[
\|AGF - A\| \leq \sqrt{\frac{a_2^2 n}{c_1^2 l}} + 1\sigma_{k+1}(A),
\]

and

\[
\|F\| \leq \frac{1}{c_1 \sqrt{l}}
\]

with probability not less than \( 1 - e^{-c_2 l} - e^{-a_2 n} \).

**Proof.** The proof is almost identical to the proof of Lemma 4.6. Here, in order to prove Eq. 4.20 for \( G \) as a sparse Gaussian matrix, we use Conjecture 4.12 and take \( Q = V^T \). Therefore, \( \sigma_k(H) \geq \sigma_k(G_1) \) where \( G_1 \) is the upper \( k \times l \) block of \( G \). By combining Conjecture 4.12 and Theorem 3.6 we get

\[
\|F\| = \frac{1}{\sigma_k(H)} \leq \frac{1}{\sigma_k(G_1)} \leq \frac{1}{c_1 \sqrt{l}}
\]

where \( c_1 \) depends on \( \mu = \left( \frac{4}{\sqrt{2\pi \rho}} \right)^{\frac{1}{3}} \). \( \square \)
Proof of Theorem 4.11. The proof is almost identical to the proof of Theorem 4.10, except that we use Lemma 4.13 instead of Lemma 4.6 that is applicable to sparse Gaussian random matrices. The bound stays the same as in Theorem 4.10, except that $a_1$, $c_1$ and $c_2$ depend on $\mu = \frac{4}{\sqrt{2\pi\rho}}$. \hfill \qed

Remark 4.14. In practice, step 3 in Algorithm 4.1 can be done using standard LU decomposition with either partial or full pivoting, when it is called by Algorithm 4.2. By using full pivoting we can sparsify $L$ even more.

### 4.4 Rank Deficient Least Squares

In this section, we present an application that uses the randomized LU and show how it can be used to solve efficiently the Rank Deficient Least Squares (RDLS) problem.

Assume that $A$ is an $m \times n$ matrix ($m \geq n$) with $\text{rank}(A) = k$, $k < n$ and $b$ is a column vector of size $m \times 1$. We want to minimize $\|Ax - b\|$. Because $A$ is a rank deficient matrix, then the problem has an infinite number of solutions since if $x$ is a minimizer and $z \in \text{null}(A)$, then $x + z$ is also a minimizer (i.e. a valid solution). We now show that the complexity of the solution depends on the rank of $A$ and that the problem is equivalent to solving the following two problems: a full rank Least Square (LS) problem of size $m \times k$ and a simplified undetermined linear system of equations that requires a matrix inversion of size $k \times k$.

The solution is derived by the application of Algorithm 4.1 to $A$ to get

$$\|Ax - b\| = \|P^T LUQ^T x - b\| = \|LUQ^T x - Pb\|,$$

(4.40)

where $L$ is an $m \times k$ matrix, $U$ is a $k \times n$ matrix and both $L$ and $U$ are of rank $k$. Let $y = UQ^T x$ and $c = Pb$. Then, the problem is reformulated as

$$\min \|Ly - c\|.$$

(4.41)

Note that $L$ is a full rank matrix and the problem to be solved becomes a standard full rank LS problem. The solution is given by $y = L^T c$. Next, we solve

$$Uz = y,$$

(4.42)

where $z = Q^T x$. Since $U$ is a $k \times n$ matrix, Eq. 4.42 is an underdetermined system. Assume that $U = [U_1 \ U_2]$ and $z = [z_1 \ z_2]^T$, where $U_1$ is a $k \times k$ matrix, $z_1$ is a $k \times 1$ vector and $z_2$ is a $(n-k) \times 1$ vector. Then, the solution is given by setting any value to $z_2$ and solving

$$U_1 z_1 = y - U_2 z_2.$$

(4.43)

For simplicity, we choose $z_2 = 0$. Therefore, we get that $z_1 = U_1^{-1} y$. The final solution is given by $x = Qz$. This procedure is summarized in Algorithm 4.3 that finds the solution to the deficient least squares problem that uses Algorithm 4.1.
Algorithm 4.3: Solving Rank Deficient Least Squares with Randomized LU

**Input:** matrix $A$ of size $m \times n$ with rank $k$; $l$ integer such that $l \geq k$; $b$ vector of size $m \times 1$.

**Output:** solution $x$ that minimizes $\|Ax - b\|$.

1: Apply Algorithm 4.1 to $A$ with parameters $k$ and $l$.
2: $y \leftarrow L^\dagger Pb$.
3: $z_1 \leftarrow U_1^{-1}y$.
4: $z \leftarrow \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$, where $z_2$ is an $n-k$ zero vector.
5: $x \leftarrow Qz$.

The complexity of Algorithm 4.3 is equal to the randomized LU complexity (Algorithm 4.1) with an additional inversion cost of the matrix $U_1$ in Step 3, which is of size $k \times k$. Note that the solution given by Algorithm 4.3 is sparse in the sense that $x$ contains at most $k$ non-zero entries.

5 Numerical Results

In order to evaluate Algorithm 4.1, we present the numerical results by comparing between the performances of several randomized low rank approximation algorithms. We tested the algorithms and compared between them by applying them to random and sparse matrices and to images. All the results were computed using the standard MATLAB libraries on a machine with two Intel Xeon CPUs X5560 2.8GHz that contains an nVidia GPU GTX TITAN card.

5.1 Error Rate and Computational Time Comparisons

The performance of the randomized LU (Algorithm 4.1) was tested and compared to a randomized SVD and to a randomized ID (see [24, 31]). The tests compare between the normalized (relative) error of the low rank approximation obtained by the examined methods. In addition, the computational time of each method was measured. If $A$ is the original matrix and $\hat{A}$ is a low rank approximation of $A$, then the relative approximation error is given by:

$$
\text{err} = \frac{\|A - \hat{A}\|}{\|A\|}.
$$

(5.1)

We compared between the low rank approximation achieved by the application of the randomized SVD, randomized ID and randomized LU with different ranks $k$. Throughout the experiments, we chose $l = k + 3$ and the test matrix was a random matrix of size $3000 \times 3000$ with exponentially decaying singular values. The computations of the algorithms were done in a single precision. The results are presented in Fig. 5.1. The experiment shows that the error of the randomized ID is significantly larger than the error obtained from both the randomized SVD and the randomized LU (Algorithm 4.1), which are almost identical. In
addition, we compared between the execution time of these algorithms. The re-
sults are presented in Fig. 5.2. The results show that the execution time of the
randomized LU (Algorithm 4.1) is significantly lower than the execution time
of the randomized SVD and the randomized ID algorithms. The LU factoriza-
tion has a parallel implementation (see [22] section 3.6). To see the impact of
a parallel LU factorization implementation, the execution time for computing
a randomized LU of a matrix of size $3000 \times 3000$ was measured on an nVidia
GTX TITAN GPU device and it is shown in Fig. 5.3. The execution time on
the GPU was up to $\times 10$ faster than running it on an eight cores CPU. Thus,
the algorithm scales well. For larger matrices ($n$ and $k$ are large) the differences
between the performances running on CPU and on GPU are more significant.

![Figure 5.1](image_url)

**Figure 5.1:** Comparison between the low rank approximation error of different
algorithms: Randomized SVD, Randomized ID and Randomized LU. Randomized LU achieves the lowest error.
Figure 5.2: Comparison between the execution times of the same algorithms as in Fig. 5.1 running on a CPU. Randomized LU achieved the lowest execution time.

Figure 5.3: Comparison between the execution times from running Algorithm 4.1 on different computational platforms: CPU with 8 cores and GPU. Randomized LU achieved the lowest execution time.
5.2 Image Matrix Factorization

Algorithm 4.1 was applied to images given in a matrix format. The factorization error and the execution time were compared with the performances of the randomized SVD and the randomized ID. We also added the SVD error computation and execution time as benchmark computed using Lanczos bidiagonalization [22] implemented in the PROPACK package [27]. The image size was $2124 \times 7225$ pixels and it has 256 gray levels. The parameters were $k = 200$ and $l = 203$. The approximation quality (error) was measured in PSNR defined by:

$$\text{PSNR} = 20 \log_{10} \frac{\max_A \sqrt{N}}{\|A - \hat{A}\|_F} \quad (5.2)$$

where $A$ is the original image, $\hat{A}$ is the approximated image (the result of Algorithm 4.1), $\max_A$ is the maximal pixel value of $A$ and $N$ the total numbers of pixels. $\| \cdot \|_F$ is the Frobenius norm.

Figure 5.4: The original input image of size $2124 \times 7225$ that was factorized by the randomized LU, randomized ID and randomized SVD algorithms.

Figure 5.5: The reconstructed image from the randomized LU factorization with $k = 200$ and $l = 203$.

Figures 5.4 and 5.5 show the original and the reconstructed images, respectively. The image approximation quality (measured in PSNR) related to rank
$k$ is shown in Fig. 5.6 where for the same $k$ the PSNR of Algorithm 4.1 is higher than the randomized ID and almost identical to the randomized SVD. Also, the PSNR is close to the result achieved by the application of the Lanczos SVD which is the best possible rank $k$ approximation. The execution time of each algorithm is shown in Fig. 5.7. All the computations were done in double precision. Here, the randomized LU is significantly faster than all the other compared methods making it applicable for real time applications.

![Figure 5.6: Comparison between the PSNR values from image reconstruction application using randomized LU, randomized ID, randomized SVD and Lanczos SVD algorithms.](image-url)
5.3 Sparse Matrix Factorization

In order to test Algorithm 4.1 on sparse matrices, we used a binary sparse matrix of size 862,664 × 862,664 with 19,235,140 non-zero elements ($\rho = 2.58 \times 10^{-5}$) that contains the results of crawling the .eu domain. The eu-2005 matrix was generated and studied in [6]. Each edge in the eu-2005 graph represents a link between two websites. The approximation error of each algorithm applied to the eu-2005 matrix is shown in Fig. 5.8, and the execution time is shown in Fig. 5.9.
Figure 5.8: Comparison between the approximation error of the randomized LU, randomized ID and randomized SVD algorithms, executed on the sparse matrix eu-2005.

Figure 5.9: Comparison between the execution time of the randomized LU, randomized ID and randomized SVD algorithms, executed on the sparse matrix eu-2005.
To test the factorization for large $k$ values, we applied Algorithms 4.1 and 4.2 to a $100,000 \times 100,000$ random generated sparse matrix $A$ with density $\rho = 10^{-6}$. Applying Algorithm 4.1 to this matrix using $k = 10,000$ requires a large memory size and CPU resources. This resulted in a very long execution time. A similar experiment was conducted by applying Algorithm 4.2 to the same matrix $A$ with $k$ ranging from 13,000 to 25,000. The random matrix $G$, selected in Step 1 of Algorithm 4.2, was a random generated sparse matrix with two possible densities $\rho = 10^{-4}$ and $\rho = 3 \times 10^{-4}$. The approximation error for each density-based execution is shown in Fig. 5.10 and the execution time is shown in Fig. 5.11. The graphs represent an averaging from successive application of the algorithm.

![Figure 5.10: Approximation error from the application of Algorithm 4.2 to the matrix $A$ with two different densities.](image-url)
The approximation error in Fig. 5.10 depends on the sparsity $\rho$ of the random matrix $G$. The error increases as $G$ became sparser. Figure 5.11 shows the dependency between execution time and sparsity. A sparser matrix reduces the algorithm execution time. This trade-off enables the user to choose between speed and accuracy by selecting a proper $\rho$ value.

**Conclusion**

In this work, we presented a randomized algorithm for computing an LU rank $k$ decomposition. Given an integer $k$, the algorithm finds an LU decomposition such that both $L$ and $U$ are of rank $k$ with negligible failure probability. We constructed error bounds for the approximation of the input matrix and proved that they are proportional to the $k + 1$ singular value. We also compared the performance of the algorithm with regard to the error rate and to the computational time. We compared the results to randomized SVD, randomized ID and to the application of Lanczos SVD running on sparse matrices. We also showed that our algorithm can be parallelized since it consists mostly of matrix multiplication and pivoted LU. The results on GPU show that it was possible to accelerate the computational time significantly even by using only the standard MATLAB libraries.
Acknowledgment

This research was partially supported by the Israel Science Foundation (Grant No. 1041/10), by the Israeli Ministry of Science & Technology (Grants No. 3-9096, 3-10898), by US - Israel Binational Science Foundation (BSF 2012282) and by a Fellowship from Jyväskylä University. The authors would like to thank Yoel Shkolnisky for the helpful discussions.

References


