WARSAW UNIVERSITY OF TECHNOLOGY

High Performance Computing

Faculty of Mathematics and Information Science

Project 3

- Final Report -

By **Elie SAAD**

June 8, 2020

Contents

1 Introduction

1.1 Problem Statement

The following problem statement has been given:

"Parallel ILLT for sparse symmetric matrix divided by columns. We remember only non-zeros of the matrix."

1.2 Problem Description

Based on the problem statement above, we are required to consider the incomplete Cholesky factorizations (ILLT) for the iterative solution of sparse symmetric positive definite (SPD) $N \times N$ linear systems

$$Au = b.$$

Incomplete Cholesky factorizations are commonly described with the help of Cholesky version of Gaussian elimination, which amounts to compute a lower triangular matrix L such that $A = LL^T$ where L is a factor of A produced by exact elimination. Whereas the incomplete factorization is obtained by introducing approximation - or lack thereof - into the elimination process in an attempt to avoid non-zeros fill ins (non-zeros would replace the elements in A where they themselves are zeros). Keeping the factors as close to being sparse like A, or as close to being dense like the factors computed by their exact elimination counterpart. This process is notably used as a preconditioner for an iterative method for a sparse SPD linear system. Thus, we are required to apply the ILLT algorithm, on a sparse SPD, over multiple processes.

2 Definitions

Definition 2.1 (Positive Definiteness)**.** Let A be an $n \times n$ matrix. A is called a **positive definite** matrix if $x^T Ax > 0$ for every nonzero vector x [1][2].

Definition 2.2 (Symmetric Matrix)**.** Let A be an $n \times n$ matrix. A is called a **symmetric matrix** if and only if $A = A^T$ [1].

Definition 2.3 (Sparse Matrix)**.** A matrix where the majority of the values are zero is called a **sparse matrix**.

Definition 2.4 (Cholesky Factorization)**.** A symmetric positive definite ($n\times n$)-matrix A can be decomposed as $A = R^TR$ where R, the Cholesky factor, is a upper triangular matrix with positive diagonal elements [3]. It can also be written as $A = LL^T$ where L is the lower triangular matrix with positive diagonal elements. Both forms of the solution are unique.

Definition 2.5 (Preconditioners)**.** A **preconditioner** $P \approx A$ where A is an $n \times n$ matrix, such that $P^{-1}A$ has a smaller condition number than A. Where P is used to speed up the iterative solution for $Ax = b$. [4]

Definition 2.6 (Incomplete Cholesky Factorization)**.** The incomplete Cholesky decomposition is a modification of the original Cholesky algorithm. If an element a_{ij} off the diagonal of A is zero where A is an $n \times n$ matrix, the corresponding element $r_{ij} \in R$ is set to zero where R is the Cholesky factor. The factor returned, R, has the same distribution of non-zeros as A above the diagonal.

3 Preliminaries and Notation

The following technical notation and assumptions are used:

- $[i, j] = \{i, i + 1, ..., j\}$ stands for the ordered set of integers ranging from i to j.
- I stands for identity matrix and O for zero matrix (matrix with all entries being zero).
- For any vector v, $||v||$ is its Euclidean norm. For any matrix C, the induced matrix norm is

$$||C|| = \max_{v\neq 0} \frac{||Cv||}{||v||}.$$

- For any SPD matrix D, $\lambda_{\max}(D)$ and $\lambda_{\min}(D)$ is, respectively, its largest and its smallest eigenvalues. Since $\lambda_{\min}(D) > 0$, the spectral condition number $\kappa(D) = \frac{\lambda_{max}(D)}{\lambda_{\min}(D)}$ is well defined.
- For any $n \times n$ block matrix $E = (E_{i,j})$ and any $1 \leq i \leq k \leq n$,

$$E_{i:k,j} = (E^T_{i,j}...E^T_{k,j})^T,$$

and, for any $1 \leq j \leq m \leq n$,

$$E_{i:k,j:m} = (E^T_{i:k,j}...E^T_{i:k,m})^T.$$

- Let $A \in \mathbb{R}^{n \times n}$ be a sparse symmetric positive definite matrix.
- Let $v_i \in A$ be a column in A for $i = 1, ..., n$.
- Let Σ be the set of all sets of columns in the columns space of A.
- Let P be the set of all processes, where $|P| = m$.
- Let $p_i \in P$ be a process where $i = 1, ..., m$.

4 ILLT Algorithm

To begin tackling the issue at hand we will first begin by explaining the ILLT algorithm running traditionally on a single process. To do that we will prove the correctness of the ILLT algorithm, then we will discuss the algorithm itself running on a single processor and discuss its rime complexity and show why parallelizing the ILLT algorithm is better.

4.1 Proof of Correctness

Lemma 4.1. Assume that $A \in \mathbb{R}^{(n+1)\times(n+1)}$ is positive definite and symmetric. Write

$$A = \begin{pmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

with $a_{11} \in \mathbb{R}$, $A_{12} \in \mathbb{R}^{1\times n}$, $A_{21} = A_{12}^T$, and $A_{22} \in \mathbb{R}^{n\times n}$, and define the Schur-complement of A with respect to a_{11} as

$$S := A_{22} - \frac{1}{a_{11}} A_{21} A_{12}.$$

Then also S is positive definite and symmetric.

Proof. It is obvious that the matrix S is symmetric. We therefore have only to show that it is positive definite. Let therefore $x \in \mathbb{R}^n \setminus \{0\}$ and let $y \in \mathbb{R}^{n+1}$ be defined as

$$y = \begin{pmatrix} -\frac{1}{a_{11}} A_{12} x \\ x \end{pmatrix}.$$

Moreover,

$$\begin{aligned}
y^TAy &= \begin{pmatrix} -\frac{1}{a_{11}}A_{12}x & x^T \end{pmatrix} \begin{pmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} -\frac{1}{a_{11}}A_{12}x \\ x \end{pmatrix} \\
&= \begin{pmatrix} -\frac{1}{a_{11}}A_{12}x & x^T \end{pmatrix} \begin{pmatrix} -A_{12}x + A_{12}x \\ -\frac{1}{a_{11}}A_{21}A_{12}x + A_{22}x \end{pmatrix} \\
&= \begin{pmatrix} -\frac{1}{a_{11}}A_{12}x & x^T \end{pmatrix} \begin{pmatrix} 0 \\ -\frac{1}{a_{11}}A_{21}A_{12}x + A_{22}x \end{pmatrix} \\
&= -\frac{1}{a_{11}}x^TA_{21}A_{12}x + x^TA_{22}x \\
&= x^TSx.
\end{aligned}$$

□

Theorem 4.1. An invertible matrix $A \in \mathbb{R}^{n\times n}$ admits a Cholesky factorization $A = LL^T$ with a lower triangular matrix $L \in \mathbb{R}^{n\times n}$, if and only if A is symmetric and positive definite.

Proof. f. Assume that $A = LL^T$. Then

$$A^T = (LL^T)^T = L^TL = A,$$

proving that A is symmetric. Moreover, if $x \in \mathbb{R}^n \setminus \{0\}$, then

$$x^Tx = x^TLL^Tx = (L^Tx)^TL^Tx = ||L^Tx||_2^2.$$

Since A is assumed to be invertible, so is the matrix L and therefore also L^T (this follows from the fact that $0 \neq det(A) = det(LL^T) = det(L)^2$). Since $x \neq 0$, this implies that also $L^Tx \neq 0$, and consequently $||L^Tx||_2 > 0$, proving that A is positive definite.
In order to show that, conversely, every symmetric and positive definite matrix has a Cholesky factorization, we apply induction over the dimension n of the matrix.

- $n = 1$: A 1×1 matrix A is positive definite, if and only if it has the form $A = (a)$ with $a > 0$. Defining $\ell := \sqrt{a}$ and $L = (\ell)$, we see immediately that L is a Cholesky factor of A.
- $n \mapsto n+1$: Assume that we have shown that every positive definite, symmetric $n \times n$ matrix has a Cholesky factorization. We write the matrix A in the form
$$A = \begin{pmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

with $a_{11} \in \mathbb{R}$, $A_{12} \in \mathbb{R}^{1\times n}$, $A_{21} = A_{12}^T$, and $A_{22} \in \mathbb{R}^{n\times n}$. Define now

$$S := A_{22} - \frac{1}{a_{11}} A_{21} A_{12},$$

the Schur-complement of A with respect to a_{11}. According to Lemma 4.1, The matrix $S \in \mathbb{R}^{n\times n}$ is symmetric and positive definite. Using the induction hypothesis, we therefore conclude that it has a Cholesky decomposition. That is, we can write $S = L_S L_S^T$ for some lower triangular matrix $L_S \in \mathbb{R}^{n\times n}$.
Define now

$$L := \begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}} A_{21} & L_S \end{pmatrix}.$$

Note here that the positive definiteness of A implies that $a_{11} > 0$; thus the definition of L actually makes sense. It is easy to see that L is lower triangular. Moreover, using the fact that $A_{21}^T = A_{12}$

$$\begin{aligned} LL^T &= \begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}} A_{21} & L_S \end{pmatrix} \begin{pmatrix} \sqrt{a_{11}} & \frac{1}{\sqrt{a_{11}}} A_{21} \\ 0 & L_S^T \end{pmatrix} \\ &= \begin{pmatrix} a_{11} & A_{12} \\ A_{21} & \frac{1}{a_{11}} A_{21} A_{12} + L_S L_S^T \end{pmatrix} \\ &= \begin{pmatrix} a_{11} & A_{12} \\ A_{21} & \frac{1}{a_{11}} A_{21} A_{12} + S \end{pmatrix} \\ &= A. \end{aligned}$$

Thus we have constructed a Cholesky factorization of A, which concludes the induction step.

$\square$

The Cholesky factorization can also be applied to complex matrices. We recall that the adjoint of a matrix $A \in \mathbb{C}^{n\times n}$ is defined as

$$A* := \bar{A}^T.$$

That is the ij-th entry of A^* is the complex conjugate of the ji-th entry of A. Or, if $\langle \cdot, \cdot \rangle$ denotes the usual Euclidean inner product on $\mathbb{C}^n$, then A^* is the unique matrix satisfying

$$\langle x, Ay \rangle = \langle A^* x, y \rangle$$

for all vectors $x, y \in \mathbb{C}^n$. We now say that a Cholesky factorization of a complex matrix A is a factorization of the form $A = LL^*$, where L is a lower triangular matrix (in the complex case we have to define it using the adjoint of L and not its transpose).

Now recall that a matrix A is Hermitian if $A^* = A$, and positive definite if $\langle x, Ax\rangle > 0$ for all $x \in \mathbb{C}^n \setminus \{0\}$. Using the same proof as above (but replacing each transpose by an adjoint), one can now show that a complex invertible matrix A has a Cholesky factorization, if and only if it is positive definite Hermitian.

The proof of the ILLT algorithm follows directly from the proof of the Cholesky factorization as an approximation of the matrix A.

4.2 Computation and Time Complexity

The Cholesky factorization can be computed by a form of Gaussian elimination that takes advantage of the symmetry and definiteness. Equating (i, j) elements in the equation $A = R^T R$ gives

$$i = j : a_{ii} = \sum_{k=1}^{i} r_{ki}^2,$$

$$j > i : a_{ij} = \sum_{k=1}^{i} r_{ki} r_{kj}.$$

These equations can be solved to yield R a column at a time, according to the following algorithm:

```
for $j \leftarrow 1$ to $n$ do
    for $i \leftarrow 1$ to $j-1$ do
        $r_{ij} \leftarrow (a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj})/r_{ii}$
    end for
    $r_{jj} \leftarrow (a_{jj} - \sum_{k=1}^{i-1} r_{kj}^2)^{1/2}$
end for
```

And thus the implementation of the ILLT algorithms is a simple iteration of the above algorithm as follows:

```
for $j \leftarrow 1$ to $n$ do
    for $i \leftarrow 1$ to $j-1$ do
        if $a_{ij} \neq 0$ then
            $r_{ij} \leftarrow (a_{ij} - \sum_{k=1}^{i-1} r_{ki} r_{kj})/r_{ii}$
        **end if**
    **end for**
    $r_{jj} \leftarrow (a_{jj} - \sum_{k=1}^{i-1} r_{kj}^2)^{1/2}$
**end for**
```

The positive definiteness of A guarantees that the argument of the square root in this algorithm is always positive and hence that R has a real, positive diagonal. The algorithm requires $\frac{n^3}{3} + O(n^2)$ flops and n square roots, where a flop is any of the four elementary scalar arithmetic operations.

5 Parallel ILLT

In what follows we will describe and explain our take in parallelizing the above discussed ILLT algorithm. Beginning first by explaining how the columns of A will be divided between all the processes. Then we will discuss the topology and communication method between each process. And finally we will explain how the Parallel ILLT algorithm works for each process.

5.1 Data Division

To divide the columns of A between all the processes, we have decided to separate the columns between each process evenly except for the last process where it might handle one or more extra columns than the other processes. The set of columns $S \in \Sigma$ from the matrix A for a process p_i in the set of processes P, is chosen based as follows

$$S_i = \left\{ v_{(i-1)\lfloor \frac{r}{m} \rfloor + 1}, v_{(i-1)\lfloor \frac{r}{m} \rfloor + 2}, ..., v_{i\lfloor \frac{r}{m} \rfloor} \right\},$$

where we have the range $r = \text{range}(A)$, and the number of processes $m = |P|$, for $i = 1, ..., m-1$. As for the last process, the column division will follow the following

$$S_n = \left\{ v_{(m-1)\lfloor \frac{r}{m} \rfloor + 1}, v_{(m-1)\lfloor \frac{r}{m} \rfloor + 2}, ..., v_n \right\}.$$

This means that the processes have the number of columns as described by

$$\begin{cases} |S_i| & = \lfloor \frac{r}{m} \rfloor \text{ for } p_i \text{ where } i = 1, ..., m-1, \\ |S_m| & = \lfloor \frac{r}{m} \rfloor + n - (m-1)\lfloor \frac{r}{m} \rfloor = m\lfloor \frac{r}{m} \rfloor + n. \end{cases}$$

5.2 Topology and Communication

We will be using the graph topology, where the process p_i sends the data of its calculations to the processes $p_{i+1}, ..., p_m$, and receives from $p_1, ..., p_{i-1}$. Except for the last process where it does not send any data to any other process, it only receives from all the processes $p_1, ..., p_{m-1}$. And also, for the first process where it does not receive from any other process but sends its calculations to all other processes $p_2, ..., p_m$. The data that the processes are sending and receiving are the values of the calculations that are done, along with the indexes of those calculations in the matrix A.

5.3 Algorithm

Each process p_i for $i = 1, ..., m$ follows the following algorithm which is a simple iteration of the ILLT Algorithm which we have already discussed and proven.

```
$rank \leftarrow$ rank of the current process
$m \leftarrow$ number of processes
$r \leftarrow$ range$(A)$
$R \leftarrow$ the zero matrix where $R \in \mathbb{R}^{n \times n}$
for $j \leftarrow (rank - 1)\lfloor \frac{r}{m} \rfloor + 1$ to $rank\lfloor \frac{r}{m} \rfloor$ do
    for $i \leftarrow j$ to $n$ do
        if Not received $r_{i,1}, ..., r_{i,(rank-1)\lfloor \frac{r}{m} \rfloor + 1}$ then
            Receive $r_{i,1}, ..., r_{i,(rank-1)\lfloor \frac{r}{m} \rfloor + 1}$
        end if
        if $i = j$ then
            $r_{jj} \leftarrow (a_{jj} - \sum_{k=1}^{j} r_{kj}^2)^{1/2}$
        else
            if $a_{ij} \neq 0$ then
                $r_{ij} \leftarrow (a_{ij} - \sum_{k=1}^{j} r_{ik} r_{jk})/r_{ii}$
            end if
        end if
        Send $r_{ij}$ to $p_{rank+1}, ..., p_m$
    end for
end for
```

We do note that the process p_1 runs the above algorithm without the receiving line, and the process p_m also runs the above algorithm but without the sending line.

6 Program Documentation

In what follows we will explain the program that we came up with in detail. We begin by explaining some general information, in regards with the libraries and solutions that have been used to realize the project. Then we move on to describing the various methods used to fill in the gaps in the algorithms described earlier and the rest of the program's code. Then finally we will show a running example instance of the program.

6.1 General Information

The program has been coded in the language C, which is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions.

The libraries that have been used are

- "stdio" which is used for all reading and writing to text files of the Matrix Market format and also outputting the results of the calculation to the terminal,
- "stdlib" which is used for dynamic memory allocation of arrays and also take the absolute values of numbers in the calculations,
- "string" which is used to compare strings from the terminal to see if they match those that have been hard coded for different modes in the program,
- "math" which is used to take the power of 0.5 of numbers in the calculations since that is the definition of a square root,
- "stddef" which is used to give the offset in bytes of the structure members of "data" used in the communication between the processes,
- "mpi" which is used as the message passing standard based on the consensus of the MPI Forum,
- "mmio" which is used to facilitate the exchange of matrix data between the "matrix.txt" file and the program itself.

The program has two modes, the matrix generation mode and the ILLT calculation mode. To initiate the matrix generation mode, the user should type in the terminal the following command "./pr_3_ElieSaad.o -gm" where the flag "-gm" indicates to the program that it must generate a new matrix. The new matrix will be generated in a file called "matrix.txt" which will be in the Matrix Market format. To initiate the calculation mode, the user should type in the terminal the following command "mpirun -np 3 ./pr_3_ElieSaad.o". The calculation mode assumes that a file called "matrix.txt" which is in the matrix market format exists. Therefore - to run this mode - the user must either generate a new matrix in the matrix generation mode or provide a file that is in the Matrix Market format named "matrix.txt".

6.2 The Code

The program makes two assumptions which - if undesirable - can be changed within the code then recompiled for the desired outcome. The first assumption that the program does is that the dimension of the matrix is 10, the second one is that the file name to be generated or read from is named "matrix.txt".

The code consists of only one function called "random_at_most" which is defined as

long random_at_most(long max)

This function returns a random value between 0 and the value of max. This function takes in a number of datatype long. This function is used to generate the elements of the matrices in the first part of the program.

The second part of the program - the matrix calculation part - begins by declaring the matrices A and R are of dimension dim, and are set to be completely filled up with zeros. Then the values of the variables J, I, val, and A are then set based on the content of the file in Matrix Market format "matrix.txt". We then use the variable matrix *binary_matrix* as a binary matrix indicating which calculations have already been done by the process and the previous processes. The values of the elements of the variable binary matrix *binary_matrix* are set to 1 if the calculation for the specific element has been done, and 0 if the calculation of the specific element has not yet been done. The structure *data* is initialized to have three elements

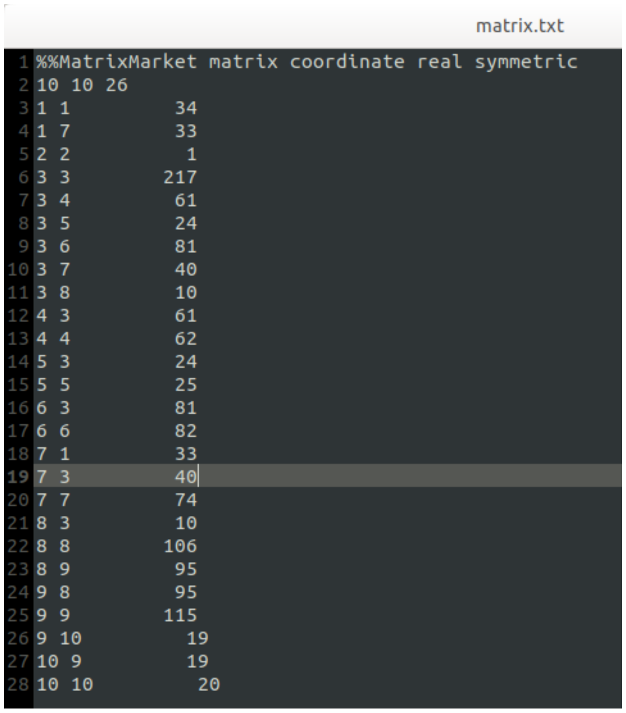
- *value* which holds the value of the calculation that has been made by the process,
- *row* which holds the value of the row index of the element that the calculation has been made on by the process,

- *column* which holds the value of the column index of the element that the calculation has been made on by the process.

The structure member *comm_data* which stands for "communication data" is used as a variable to hold the values to be communicated and used in the calculations. The data is divided exactly as described in the theoretical part above, with the variable $first\_index$ and *last_index* holding the index of the first column that the process will begin its calculations on, and the index of the last column that the process will end its calculations on. Each process - as described in the theoretical part - begins by receiving the necessary data from all previous processes. Each time the process receives a structure from a previous process, the value with the indexes of the elements received are updated in the matrix R with the appropriate value, and the value of the element with the received indexes is set to 1 in the binary matrix *binary_matrix*. The *binary_matrix* is then iterated upon to check whether the process has received all of the values necessary to complete the current calculations. If the process has all of the necessary missing data from the previous processes, it then moves on to do its own calculations described in the theoretical part, then sends its calculations to all the processes after it for them to be able to complete their part of the calculations. The last process then finally prints the matrix R and R^T to the terminal after completing its part of the calculations.

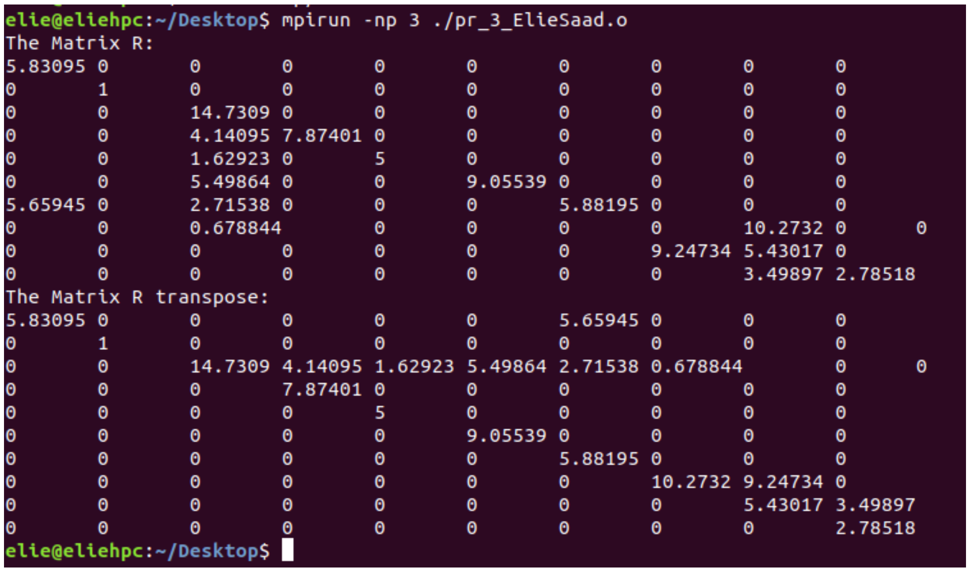
6.3 Running Example

In what follows, we show a running example of the code. The first image shows the output file from the program called "matrix.txt", which is also the input file used for the output of the second image:



matrix.txt

```
%%MatrixMarket matrix coordinate real symmetric
10 10 26
1 1         34
1 7         33
2 2          1
3 3        217
3 4         61
3 5         24
3 6         81
3 7         40
3 8         10
4 3         61
4 4         62
5 3         24
5 5         25
6 3         81
6 6         82
7 1         33
7 3         40
7 7         74
8 3         10
8 8        106
8 9         95
9 8         95
9 9        115
9 10         19
10 9         19
10 10         20
```



```
elie@eliehpc:~/Desktop$ mpirun -np 3 ./pr_3_ElieSaad.o
The Matrix R:
5.83095 0       0       0       0       0       0       0       0       0
0       1       0       0       0       0       0       0       0       0
0       0       14.7309 0       0       0       0       0       0       0
0       0       4.14095 7.87401 0       0       0       0       0       0
0       0       1.62923 0       5       0       0       0       0       0
0       0       5.49864 0       0       9.05539 0       0       0       0
5.65945 0       2.71538 0       0       0       5.88195 0       0       0
0       0       0.678844        0       0       0       0       10.2732 0       0
0       0       0       0       0       0       0       9.24734 5.43017 0
0       0       0       0       0       0       0       0       3.49897 2.78518
The Matrix R transpose:
5.83095 0       0       0       0       0       5.65945 0       0       0
0       1       0       0       0       0       0       0       0       0
0       0       14.7309 4.14095 1.62923 5.49864 2.71538 0.678844        0       0
0       0       0       7.87401 0       0       0       0       0       0
0       0       0       0       5       0       0       0       0       0
0       0       0       0       0       9.05539 0       0       0       0
0       0       0       0       0       0       5.88195 0       0       0
0       0       0       0       0       0       0       10.2732 9.24734 0
0       0       0       0       0       0       0       0       5.43017 3.49897
0       0       0       0       0       0       0       0       0       2.78518
elie@eliehpc:~/Desktop$ 
```

References

[1] Massachusetts Institute of Technology. Symmetric matrices and positive definiteness. Fall 2011. URL `https://ocw.mit.edu/courses/mathematics/18-06sc-linear-algebra-fall-2011/positive-definite-matrices-and-applications/symmetric-matrices-and-positive-definiteness/MIT18_06SCF11_Ses3.1sum.pdf`. [20/4/2020], pages 1 – 2.

[2] D. Kincaid and W. Cheney. Numerical Analysis: Mathematics of Scientific Computing. 3rd Edition, (American Mathematical Society, Providence, RI, 2002), page 145.

[3] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 1996.

[4] Massachusetts Institute of Technology. *Lec 15 — MIT 18.086 Mathematical Methods for Engineers II*. URL `https://www.youtube.com/watch?v=LtNVodIs1dI&t=764s`. time 2:00 – 13:32.