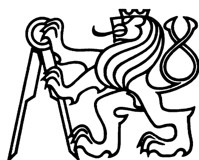
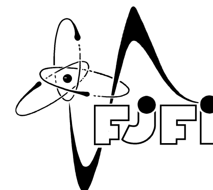


Czech Technical University in Prague
Faculty of Nuclear Sciences and Physical Engineering



Implementation of parallel algorithms for QR decomposition of real matrices in TNL library and their application

Implementace paralelních algoritmů pro QR rozklad reálných matic v knihovně TNL a jejich aplikace

Bachelor's Degree Project

Author: **Klára Přikrylová**

Supervisor: **Ing. Jakub Klinkovský**

Language advisor: **Mgr. Hana Čápová**

Academic year: 2022/2023

Katedra: matematiky

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	Klára Přikrylová
Studijní program:	Aplikovaná informatika
Název práce (česky):	Implementace paralelních algoritmů pro QR rozklad reálných matic v knihovně TNL a jejich aplikace
Název práce (anglicky):	Implementation of parallel algorithms for QR decomposition of real matrices in TNL library and their application

Pokyny pro vypracování:

1) Nastudujte algoritmy pro QR rozklad reálných matic založené na Gramm-Schmidtově ortogonalizaci, Householderových transformacích a Givensových rotacích.
2) Všechny zmíněné algoritmy implementujte pomocí datových struktur a funkcí dostupných v knihovně TNL.
3) Zkoumejte možnosti paralelizace zmíněných algoritmů pro vícejádrové procesory a grafické karty.
4) Porovnejte implementované algoritmy z hlediska výpočetní náročnosti a numerické přesnosti pro testovací matice relevantní pro řešení praktických úloh.
5) Aplikujte implementované algoritmy pro QR rozklad matic na řešení vybraných úloh (např. hledání vlastních čísel pomocí QR algoritmu).

Doporučená literatura:

1) W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, Numerical Recipes: The Art of Scientific Computing (3rd ed.), New York: Cambridge University Press, 2007, ISBN 978-0-521-88068-8.
2) M. Hoemmen, A communication-avoiding, hybrid-parallel, rank-revealing orthogonalization method. In 'IEEE International Parallel & Distributed Processing Symposium', IEEE, 2011.
3) J. Malard, C. C. Paige, Efficiency and scalability of two parallel QR factorization algorithms. In 'Proceedings of IEEE Scalable High Performance Computing Conference', IEEE, 1994.
4) John G. F. Francis, The QR transformation: a unitary analogue to the LR transformation—Part 1. The Computer Journal 4.3, 1961, 265-271.
5) John G. F. Francis, The QR transformation—part 2. The Computer Journal 4.4, 1962, 332-345.

Jméno a pracoviště vedoucího bakalářské práce:

Ing. Jakub Klinkovský
Katedra matematiky, FJFI ČVUT v Praze, Trojanova 13, 120 00 Praha 2

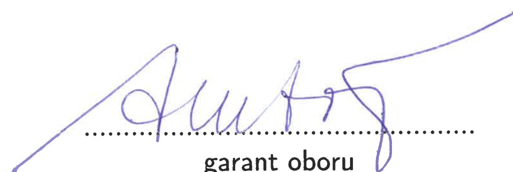
Jméno a pracoviště konzultanta:

Datum zadání bakalářské práce: 31.10.2022

Datum odevzdání bakalářské práce: 2.8.2023

Doba platnosti zadání je dva roky od data zadání.

V Praze dne 31.10.2022



....
garant oboru

....
vedoucí katedry

....
děkan

Acknowledgment:
I would like to thank Ing. Jakub Klinkovský for his expert guidance and express my gratitude to Mgr. Hana Čápová for her language assistance.

Author's declaration:
I declare that this Bachelor's Degree Project is entirely my own work and I have listed all the used sources in the bibliography.

Prague, August 2, 2023 Klára Přikrylová

Název práce:

Implementace paralelních algoritmů pro QR rozklad reálných matic v knihovně TNL a jejich aplikace

Autor: Klára Přikrylová

Obor: Aplikovaná informatika

Druh práce: Bakalářská práce

Vedoucí práce: Ing. Jakub Klinkovský, Katedra softwarového inženýrství, Fakulta jaderná a fyzikálně inženýrská, ČVUT

Abstrakt: Důležitou součástí matematiky je lineární algebra. QR rozklad se často používá k řešení výpočtu matic. Existuje několik metod pro výpočet QR rozkladu, jako je Gram-Schmidtův proces, Householderovy transformace a Givensovy rotace. Hlavním cílem této bakalářské práce je porovnat tyto tři metody QR rozkladu a zjistit, zda paralelizace výpočtu povede k rychlejším a přesnějším výsledkům. Tyto algoritmy jsou porovnávány pomocí unit testů a benchmarku. Tato studie neprokázala, že paralelizace výpočtů těchto tří metod rozkladu QR vede k rychlejším a přesnějším výsledkům. Dílčí výsledky však ukazují určité rozdíly mezi třemi přístupy.

Klíčová slova: Givensova rotace, Gram - Schmidtův proces, Hauseholderova matice, QR rozklad, Paralelizace, TNL knihovna

Title:

Implementation of parallel algorithms for QR decomposition of real matrices in TNL library and their application

Author: Klára Přikrylová

Abstract: An important part of mathematics is linear algebra. The QR decomposition is often used to solve the matrices. There are several methods for computing the QR decomposition such as the Gram-Schmidt process, Householder transformations and Givens rotations. The main aim of this bachelor thesis is to compare these three QR decomposition methods and find out whether parallelization of computation will lead to faster and more accurate results. These algorithms are compared via Unit tests and benchmark. This study didn't show that the computation parallelization of these three QR decompositions methods leads to faster and more accurate results. But partial results show some differences among three approaches.

Key words: Givens rotations, Gram-Schmidt process, Householder transformations, QR decomposition, Parallelization, TNL library

Contents

Introduction **10**

1 Basic terms **11**
1.1 Matrix 11
1.1.1 Complex and real matrices 11
1.1.2 Square and rectangular matrices 11
1.1.3 Triangular and diagonal matrices 12
1.1.4 Orthogonal matrix 12
1.1.5 Frobenius norm 12
1.2 QR decomposition 12
1.2.1 Gram-Schmidt orthogonalization process 12
1.2.2 Householder reflections 15
1.2.3 Givens rotations 16
1.3 TNL library 18
1.4 Language C++ 18
1.4.1 Use of the C++ language 18

2 Implementation of QR decomposition **19**
2.1 Functionality used from the TNL library 19
2.1.1 Horizontal operations 19
2.1.2 Vertical operations 19
2.1.3 Binding 20
2.2 Pseudocodes 20
2.2.1 Gram-Schmidt orthogonalization process 20
2.2.2 Householder reflections 21
2.2.3 Givens rotation 23

3 Code testing and benchmarking **25**
3.1 Unit tests 25
3.2 Performance benchmark 25
3.3 Results of benchmark 26
3.3.1 Theory of complexity 26
3.3.2 Measured complexity 27
3.3.3 Parallelization 29
3.3.4 Parallelized algorithms 30

4 Future applications **32**
4.1 QR decomposition 32
4.1.1 Parallelization 32
4.2 Use of Gram-Schmidt, Householder and Givens algorithms 32
4.3 Future work 33

Conclusion **34**

Bibliography **35**

Introduction

Sciences such as physics, chemistry, engineering and economy need modern technologies for their existence and progression. Modern technologies are all around us, they help us in everyday activities. The main basis of them is mathematics which is one of the most useful science in our lives, sometimes we even don't know about its presence. Modern technologies help with important and difficult mathematical procedures and calculations, without them everything would be slower and take a longer time. An important part of mathematics is linear algebra, which is used in most modern mathematics. It studies vectors, vector space, system of linear equations and linear transformation. The linear transformation can be written as a matrix. The QR decomposition is a decomposition of matrix into an orthogonal matrix Q and an upper triangular matrix R. The QR decomposition is often used to solve the linear least squares problem and is the basis for a particular eigenvalue algorithm, the QR algorithm. There are several methods for computing the QR decomposition such as the Gram-Schmidt process, Householder transformations and Givens rotations. Each of them has some advantages and disadvantages, main differences are in accuracy and calculation time. These algorithms are compared via unit tests tests and benchmark. In unit tests, the mathematical correctness of individual algorithms in which different testing approaches are tested. The benchmark examines not only the error rate of each algorithm but also the computational time. This test is performed successively on different number of cores. The main aim of this bachelor thesis is to compare these three QR decomposition methods and find out whether parallelization of computation will lead to faster and more accurate results.

All algorithms are programmed using the C++ language with its basic libraries. An additional library is Template Numerical Library (TNL), mathematical library containing a variety of already implemented functions. One of them, for example, is the function "dot", which performs a scalar product. This library increases the user-friendly aspect of the implementation. In addition, other tools such as Inscape and Gnuplot, which is used for graph creation, are used in the work.

The bachelor thesis consists from a theoretical and a practical part. The theoretical part contains all the essential concepts, which are important to understand the whole principle of QR decomposition. The study of individual algorithms and understanding of the basic principles of these algorithms is an integral part of the bachelor thesis. All the algorithms that are suitable for performing QR decomposition are described here. The practical part deals not only with the implementation of individual algorithms for QR decomposition but also with benchmarking and Unit testing. The implementation of individual algorithms is shown using pseudocodes. The results of each test are evaluated and shown in graphs.

Chapter 1

Basic terms

This first chapter is a review of the known definitions and general principles of QR decomposition that must be mentioned for the following chapters.

1.1 Matrix

A matrix A of n rows and m columns (matrix of type $n \times m$) is a set of rows (generally complex) written in the form.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{pmatrix}, \tag{1.1}$$

where $a_{jk} \in \mathbb{C}, j = 1, \dots, m, k = 1, \dots, n.$ [1] In the matrix, various mathematical operations can be performed. Then, the matrix can be added, subtracted (if they are of the same type) and multiplied (only if the number of columns of the matrix A is the same as the number of rows in the matrix B) [2].

1.1.1 Complex and real matrices

A complex matrix is a matrix that has some complex numbers among its elements. The complex matrix contains a complex/imaginary part, which is denoted by the letter i, and real numbers. Operations with complex matrices are equivalent to real matrices. An important operation with complex matrices is the search for the determinant. The determinant indicates whether the matrix has a solution at all [3]. On the other hand, a real matrix is a matrix whose entries are real numbers (a number that can be used to measure a continuous one-dimensional quantity) [4].

1.1.2 Square and rectangular matrices

"A matrix A is said to be square if it has the same number of rows and columns" [4]. The following example shows a square matrix:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tag{1.2}$$

On the other hand, a matrix in which the number of rows is not equal to the number of columns is called rectangular matrix. [4]

1.1.3 Triangular and diagonal matrices

A triangular matrix is a square matrix in which elements below and/or above the diagonal are all zeros. The triangular matrices are divided into two groups: upper triangular matrix and lower triangular matrix. An upper triangular square matrix is special type of matrix where all elements below the diagonal are zeros and a lower triangular matrix has all elements above the diagonal zero. [5] "A matrix A 1.2 is referred to as diagonal if all entries outside the main diagonal are zero" [6].

1.1.4 Orthogonal matrix

A square matrix A is orthogonal if and only if its transpose is the same as its inverse, i.e. $A^T = A^{-1}$, where A^T is the transpose of A and A^{-1} is the inverse of A [7].

1.1.5 Frobenius norm

The definition of the Frobenius norm is as follows

$$||A||_F = \sqrt{\sum_i^m \sum_j^n ||a_{ij}||^2} = \sqrt{\text{tr}(A^T A)} = \sqrt{\sum_{i=1}^{\min\{m,n\}} \sigma_i^2(A)}, \tag{1.3a}$$

where $\sigma_i(A)$ are the singular values of A and *tr* is abbreviation for *trace*. It denotes the sum of elements on the diagonal of this matrix, i.e. the sum of elements $a_{11}, a_{22}, \ldots, a_{nn}$, where n is the number of columns of the matrix A [8].

1.2 QR decomposition

The *QR* decomposition is one of the fundamental calculations and also very useful [9].
Definition: "A pair of matrices Q and R is called a *QR*-decomposition of matrix A if

$$A = QR, \tag{1.4}$$

where Q is the orthogonal matrix and R the upper triangular matrix"[10].

QR decomposition can reduce the difficulty of some numerical calculations, such as solving linear systems. In this work, the *QR* decomposition will be performed according to the following algorithms: the Gram-Schmidt Orthogonalization Process, Householder refelctions and Givens rotation. [10]

1.2.1 Gram-Schmidt orthogonalization process

The first method can be used for the *QR* matrix decomposition is the Gram-Schmidt orthogonalization process. This process is generally not computationally intensive. First, let us denote the input matrix as M. The matrix must be square. Then we denote the columns of matrix M as vectors, which we will denote by u_1, u_2, u_3, $...u_n$. In the next step, the set $U = \{u_1, u_2, u_3, ...u_n\}$ will be orthogonalized, that is, we are looking for a set $V = \{v_1, v_2, v_3, ...v_n\}$ for which it will be true that the vectors are mutually orthogonal and each vector from the set V can be written as a linear combination of vectors from the set U. [11]

In general, the Gram-Schmidt orthogonalization process can be performed in several steps.

1. We set the vector u_1 equal to the vector v_1. [11]

$$u_1 = v_1 \tag{1.5}$$

2. Using the vector u_2, which is from the set U, we then create a vector v_2, which is orthogonal to v_1, in the orthogonal set V. [11]

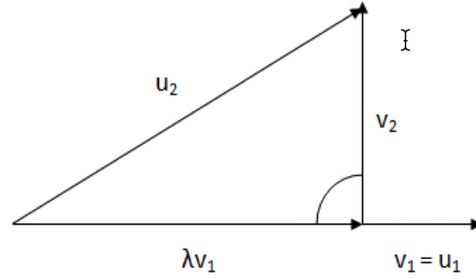


Figure 1.1: Gram-Schmidt orthogonalization process-setting the second vector of an orthogonal set

Figure 1.1 shows the vector u_1, which is identical to the vector v_1. The representation of the vector v_2 is using the vector u_2 and a multiple of the vector v_1. [11] The vector v_2 is then equal to:

$$v_2 = u_2 - \lambda v_1 \tag{1.6}$$

To get the vector v_2, we need to know the multiple of v_1, which we denote by the Greek letter λ. This is obtained using the scalar product, which is then adjusted. [11]

We use $u^T v$ to denote the product of vectors u and v. Because of the orthogonality of vectors v_1 and v_2, the following equation for the scalar product will hold [11]:

$$v_1^T v_2 = 0 \tag{1.7}$$

Next, we substitute equation (1.6) into (1.7) and obtain:

$$v_1^T v_2 = v_1^T (u_2 - \lambda v_1) \tag{1.8a}$$
$$v_1^T v_2 = v_1^T u_2 - \lambda (v_1^T v_1) \tag{1.8b}$$
$$0 = v_1^T u_2 - \lambda (v_1^T v_1) \tag{1.8c}$$
$$\lambda = \frac{v_1^T u_2}{v_1^T v_1} \tag{1.8d}$$
$$v_2 = u_2 - \frac{v_1^T u_1}{v_1^T v_1} v_1 \tag{1.8e}$$

3. Let us use the vector u_3 to find the vector v_3 orthogonality to both v_1 and v_2. [11]

   Figure 1.2 shows not only the vectors v_1 and v_3, which are orthogonal to each other, but also the vector v_3, which has the same property of the vectors v_1 and v_2. [11]

$$u_3 = \lambda_1 v_1 + \lambda_1 v_2 + v_3 \tag{1.9a}$$
$$v_3 = u_3 - \lambda_1 v_1 - \lambda_2 v_2 \tag{1.9b}$$

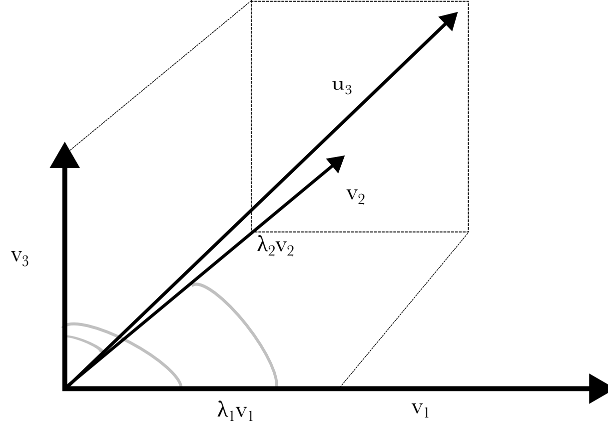


Figure 1.2: Gram-Smith orthogonalization process-setting the third vector of an orthogonal set

From the orthogonality conditions for vectors v_1, v_3 and v_3, v_2 we find the unknown multiples λ_1,λ_2. [11]

$$v_1^T v_3 = 0 \tag{1.10a}$$
$$v_3^T v_2 = 0 \tag{1.10b}$$

Then we evaluate $v_1^T v_3$, considering the vector v_3 of the form (1.9b).

$$v_1^T v_3 = (u_3 - \lambda_1 v_1 - \lambda_2 v_2)^T v_1 \tag{1.11a}$$
$$v_1^T v_3 = u_3^T v_1 - \lambda_1(v_1^T v_1) - \lambda_2(v_2^T v_1) \tag{1.11b}$$
$$0 = u_3^T v_1 - \lambda_1(v_1^T v_1) \tag{1.11c}$$
$$\lambda_1 = \frac{u_3^T v_1}{v_1^T v_1} \tag{1.11d}$$

Multiplying vector v_3 by vector v_2 we find λ_2.

$$v_3^T v_2 = (u_3 - \lambda_1 v_1 - \lambda_2 v_2)^T v_2 \tag{1.12a}$$
$$v_3^T v_2 = u_3^T v_2 - \lambda_1(v_1^T v_2) - \lambda_2(v_2^T v_2) \tag{1.12b}$$
$$0 = u_3^T v_2 - \lambda_2(v_2^T v_2) \tag{1.12c}$$
$$\lambda_2 = \frac{u_3^T v_2}{v_2^T v_2} \tag{1.12d}$$
$$v_3 = u_3 - \frac{u_3^T v_1}{v_1^T v_1} v_1 - \frac{u_3^T v_2}{v_2^T v_2} v_2 \tag{1.12e}$$

4. To work with other vectors from the set U up to u_n, we use a similar procedure. From the relations already mentioned, all vector multiplicities of the constructed orthogonal set V can be deduced and calculated. [11]

$$v_n = u_n - \frac{u_3^T v_1}{v_1^T v_1} v_1 - \frac{u_n^T v_2}{v_2^T v_2} v_2 - \ldots - \left[\sum_{k=1}^{n-1} \frac{u_n^T v_{n-1}}{v_{n-1}^T v_{n-1}} v_{n-1} \right] \tag{1.13}$$

1.2.2 Householder reflections

Theorem 1." Each matrix $A \in \mathbb{R}^{m\times n}$ can be decomposed by $s = \min\{n, m-1\}$ Householder matrix to QR product and the following expression applies." [10]

$$H_s \dots H_2 H_1 A = Q^T A = \begin{cases} \binom{R_1}{0} & m > n \\ (R_1, 0) & m < n \\ R & m = n \end{cases} \tag{1.14}$$

Then get the matrix $A \in \mathbb{R}^{m\times n}$

$$A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \tag{1.15}$$

As with the Gram-Schmidt orthogonalization process, the Householders reflections can be displayed through the following steps. [10]

1. Firstly, the Householder matrix H_1 must be set exactly so that $H_1 A$ has only zeros in the first column. The only exception is the position in the matrix $(1, 15)$. That is denoted by $*$. [10]

$$H_1 A = \begin{pmatrix} * & * & \dots & * \\ 0 & * & \dots & * \\ \vdots & \vdots & \ddots & \vdots \\ 0 & * & \dots & * \end{pmatrix}. \tag{1.16}$$

   For the following step we denote matrix $A^{(1)} := H_1 A$.

2. Let us construct the Householder matrix H_2 such that $H_2 A^{(1)}$ is 0 in the second column "below position (2,2)", observing the requirement of the first step, i.e. [10]

$$A^{(2)} := H_2 A^{(1)} = \begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & 0 & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \\ 0 & 0 & * & \dots & * \end{pmatrix} \tag{1.17}$$

   The matrix H_2 is obtained as follows: First, we construct a Householder matrix of size $(m-1) \times (n-1)$

$$\hat{H}_2 := I_{n-1} - \frac{2}{u_{n-1}^T u_{n-1}} \times (u_{n-1} u_{n-1}^T) \tag{1.18}$$

   such that

$$\hat{H}_2 \begin{pmatrix} c_{22} \\ c_{32} \\ \vdots \\ c_{m2} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{1.19}$$

And that is defined as

$$H_2 := \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & & & \\ \vdots & & \hat{H}_2 & \\ 0 & & & \end{pmatrix}. \tag{1.20}$$

This gives the matrix $A^{(2)} = H_2 A^{(1)}$. Analogously, the next steps follow.

3. Creation a general Householder matrix[10],

   For k ≤ s

$$\hat{H}_k := I_{n-k+1} - \frac{2}{u_{n-k+1}^T u_{n-k+1}} \times (u_{n-k+1} u_{n-k+1}^T) \tag{1.21}$$

   of the size $(m-k+1) \times (n-k+1)$ such that

$$\hat{H}_k \begin{pmatrix} c_{kk} \\ \vdots \\ c_{mk} \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}. \tag{1.22}$$

   Defined as

$$H_k := \begin{pmatrix} I_{k-1} & 0 \\ 0 & \hat{H}_k \end{pmatrix}, \tag{1.23}$$

   It could be evaluated $A^{(k)} = H_k A^{(k-1)}$.

In this manner, after s steps, the matrix $A^{(s)}$ is obtained that has the shape of the upper triangle, being just the matrix R. Because of

$$A^{(k)} = H_k A^{(k-1)} \quad k = 2, \dots, s, \tag{1.24}$$

It follows that

$$R = A^{(s)} = H_s A^{(s-1)} = H_s H_{s-1} A^{(s-2)} = \dots = H_s H_{s-1} \cdots H_2 H_1 A, \tag{1.25a}$$

$$Q^T = H_s H_{s-1} \cdots H_2 H_1. \tag{1.25b}$$

The required QR decomposition

$$R = Q^T A, \tag{1.26}$$

i.e.

$$A = QR. \tag{1.27}$$

1.2.3 Givens rotations

The last algorithm used is known as Givens rotation. Givens rotations are orthogonal transformations that provide the elimination of individual elements of the vector $x \in \mathbb{R}^n$. For a pair of indices i, j and

angle θ, this rotation is defined as follows [10]:

$$G(i,j,\theta) = \begin{pmatrix} 1 & & & & & & & & & & \\ & \ddots & & & & & & & & & \\ & & 1 & & & & & & & & \\ & & & \cos\theta & & & & \sin\theta & & & \\ & & & & 1 & & & & & & \\ & & & & & \ddots & & & & & \\ & & & & & & 1 & & & & \\ & & & -\sin\theta & & & & \cos\theta & & & \\ & & & & & & & & 1 & & \\ & & & & & & & & & \ddots & \\ & & & & & & & & & & 1 \end{pmatrix} \tag{1.28}$$

A plane is chosen in the space $\mathbb{R}^n$, which is defined by the$j$-th and i-th coordinate in $\mathbb{R}^n$ and in this space a rotation is created with the magnitude of the angle θ , which is clockwise. The vector $\boldsymbol{y}$ will be defined as the Givens rotation at coordinates i, j and angle θ , so that for the k-th position [10]:

Due to the shape of the matrix G, if we salt it with some vector

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}, \tag{1.29}$$

then only the i-th and j-th position of the vector x will change, the rest will remain unchanged. Subsequently, the renaming $s = \sin\theta$ and $c = \cos\theta$ will be used, where the following formula will apply [10]

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}} \tag{1.30}$$

$$s = \frac{x_j}{\sqrt{x_i^2 + x_j^2}} \tag{1.31}$$

It means that

$$cx_i + sx_j = \frac{x_i^2}{\sqrt{x_i^2 + x_j^2}} + \frac{x_j^2}{\sqrt{x_i^2 + x_j^2}} = \sqrt{x_i^2 + x_j^2} \tag{1.32}$$

and then

$$y_i = \sqrt{x_i^2 + x_j^2} \tag{1.33a}$$

$$y_j = 0 \tag{1.33b}$$

$$\theta = \arctan\left(-x_j/x_i\right) \tag{1.33c}$$

Givens rotations can be used to gradually eliminate all the elements below the diagonal to obtain QR decompostion. [10]

1.3 TNL library

TNL,Template Numberical Library, is a open-source library that is being developed at CTU. This library consists of several building blocks that facilitate the development of both efficient numerical solvers and High-Performance Computing (HPC) algorithms. The library is built on the C++ language, providing a flexible and user-friendly interface. [12]

1.4 Language C++

C++ is a multiparadigm programming language that was developed by Bjarne Stroustrup while working on his Ph.D thesis in 1979. This language was originally known as "C with Classes". The C++ language gained popularity due to fuatures like virtual function, and operator overloading, making it suitable for OOP (object oriented programming). Over the years, C++ received regular updates like libraries, such as the Standard Template Library (STL). It is a programming language that was designed to be able to work with hardware or to handle the complexity, which is in real an application.This language is not only an object-oriented language, but it also supports other programming styles such as procedural programming and generic programming. According to the American magazine TIOBE (the software quality company) [13], it is ranked third in popularity and usability for 2022. And this trend is not expected to change in 2023. [14]

1.4.1 Use of the C++ language

The use of the C++ language is quite varied. Its main use is in system programming. Other areas where C++ is used are antivirus programs, networking software, programming language compilers etc.

C++ is widely used in science to increase the speed of mathematical calculations, data analysis and scientific simulations. [15]

Chapter 2

Implementation of QR decomposition

This chapter is dedicated to the "pseudo-code" that served as the first code visualization for each of the compared algorithms, which are the Gram-Schmidt orthogonalization process, Householder reflections and Givens Rotations.

2.1 Functionality used from the TNL library

To simplify and visualize the code, some functions from `TNL` are used. In general, matrices can be written using vectors. This is why many templated classes from the `TNL` library can be used. First of all, dynamic vector are discussed. Dynamic vectors have tree template parameters:

- `Real` is type of data which is stored in the vector
- `Device` is the device where the vector is allocated
- `Index` is the type which is used for correct indexing the vector elements

The subsequent examples of operations can be performed on vectors. The dynamic vectors are divided into horizontal operation and vertical operations.

2.1.1 Horizontal operations

Horizontal operations can be considered as operations where one or more vectors are input but only one vector is output. In the **TNL** library, this operation is performed using the Expression Templates. It is metaprogramming technique which forms structures representing calculation at compile time. The expression is evaluated for individual elements rather than whole vectors, which avoids the allocation of temporary objects. Thus, this technique can save a large amount of memory. Horizontal operations are used primarily in basic mathematical operations such as multiplication, subtraction, additions and division. Other uses are scalar and vector multiplication. Futhermore, this method reduces the time needed to write a code, and improve the user experience. Without this method a for loop would be used. This kind of operation is used in each algorithm (the Gram-Schmidt orthogonalization process, Householder reflections, Givens rotation). Examples of horizontal operations are marked in the algorithms. [16] [17]

2.1.2 Vertical operations

Vertical operations can be considered as the operations where one vector expression is an input and one value as an output. These operations are used, for example with scalar product, normalization and/or

finding the minimum and/or maximum of a vector element. The example of the function where scalar product is used, is the `dot` function. Two vectors are multiplied by this function. [17]

2.1.3 Binding

The use of classical libraries, e.g. `"iostream"` and other libraries, does not allow the array to share data already allocated elsewhere. However, this can be achieved using the `ArrayView` structure, which does not deal with data allocation and deallocation, unlike `Array`. Therefore, we can use `ArrayView` to wrap data allocated elsewhere and split the array into subarrays[17]. This process where external data is "indicated" by external data is called binding. This method is called by using device kernels. The function `bind()` includes the following parameters:

- `data`: The data pointer bound to the array view.
- `size`: In the array views, the number of elements is the number of elements [17].

2.2 Pseudocodes

A pseudocode is a description of series of steps which must be done in a computer program for the correct function of the program. The algorithm is expressed in a formally-styled language then in a programming language. The pseudocode allows programmers to express the main algorithm ideas without having to follow the formalism of a specific programming language[18].

2.2.1 Gram-Schmidt orthogonalization process

Algorithm 1 The Gram-Schmidt function

```
Require: Matrix A
Ensure: A = QR
  Matrix Q = (m, n)                 ▷ The variable m represents the number of rows
  Matrix R = (m, n)                 ▷ The variable n represents the number of columns
  R=0
  v_0.bind (&A(0,0),m)              ▷ The function bind creates the vector v_0
  q_0.bind(&Q(0,0),m)               ▷ The function bind creates the vector q_0
  norm= a_1/|a_1|
  for i=1 to n do
      q_i=v_1
      for j=0 to i do
          d=TNL::dot(v_i, q_j)      ▷ Vertical operation
          R(j, i) = d
          q_i = q_i − d · q_j       ▷ Horizontal operation
      end for
      norm=TNL::l2Norm(q_i)         ▷ Vertical operation
      q_i=q_i/norm
      R(i, i)=norm
  end for
  return{Q, R}
```

This algorithm describes The Gram-Schmidt orthogonalization process. The input data is the matrix A, which will be decomposed by the algorithm into the product of matrices Q and R. The matrix Q is initially set to the zero matrix. The following vectors v_0 and q_0 are created with the function `bind` from the library **TNL**, which is explained in 2.1 section. The computed normalization of the first vector, is then stored in the `norm` variable. The selected **for** loop iterates through the columns from one to the n. In this loop value q_1 to v_1 is assigned. The inside `for` loop iterates through the columns from the zeros to the i-th element. The function `dot`, which is explained in 2.1 section, calculates and stores the scalar product of the vectors v_i and q_j. Then the value of $R(j,i)$ is set to the `d`. The vector q_i is normalized by function `l2Norm` and by means of its elements by the norm consecutively,which si explained in 2.1. The `norm` is set to the value of element $R(i,i)$. In the end, the program returns the matrices `Q` and `R` as the result.

2.2.2 Householder reflections

This algorithm describes the Householder reflections. The input data is the matrix A, which is decomposed by the algorithm into the product of matrices Q and R. The matrix Q is initially set to the identity matrix. Through the `for` loop, the Householder transformation is performed for each column of the matrix R. The matrix A is attributed to the matrix R. As explained in section 2.1.3, the vector r_i is set by `bind` function. Using the same process, the vector x is set for the truncated row by the value i, as explained in section 2.1.2. The Euclidean norm of vector is returned by `l2Norm` function and then the norm of columns$x$ is calculated. The vector x is assigned to vector u_i. The value x is assigned to 0. `If` loop finds where u_i is sharply greater than 0, then the value in element u_i is added to the norm variable and then the first element in the x array is assigned the value of the *norm* variable. Otherwise, the value of the *norm* variable is subtracted from the u_i element and then the first element in the x array is assigned the value of the *norm* variable. As explained in section 2.1.2, the vector u_i is normalized to unit length by `l2Norm` function. The subsequent `for` loop traverses the columns from the i-th +1. Vector r_j is set by the function `bind`. The vector y is set for the truncated row by the value i using the same method. The `dot` function enables an easier computation of vector multiplication, as described in section 2.1.2. The multiplication is stored in the variable d. The second `for` loop traverses the column which start from value 0. As explained in section 2.1.3, the vector k is set by the function `bind`. Using the same method, the vector e is set for the truncated row by the value i. As explained in section 2.1.2, the variable d represents the scalar product of vector u_i and e, which is obtained using the `dot` function of TNL library. The numerical formula e contains the numerical relation given by the equation (1.19). When the two for loops are finished, the matrix Q is transposed and returned in pair with matrix R.

Algorithm 2 The Householder reflections function

```
Require: Matrix A
Ensure: A = QR
  Matrix Q = (m,n)                ▹ The Variable m represents the number of rows
  Matrix R = (m,n)                ▹ The Variable n represents the number of columns
  Q=identity matrix
  R=A
  for i=O to n do
    r_i.bind(&R(0,i),m)           ▹ The function bind creates the vector r_i
    x.bind(&r_i[i],m-i)           ▹ The function bind creates the vector x
    norm=TNL::l2Norm(x)           ▹ Vertical operation
    u_i = x
    x = 0
    if u_i [0]>0 then
      u_i[0]=u_i[0]+norm
      x[0]=norm
    else
      u_i[0]=u_i[0]-norm
      x[0]=norm
    end if
    u_i=u_i/TNL::l2norm(u_i)      ▹ Vertical operation
    for j=i+1 to n do
      r_j.bind(&R(0,j),m)         ▹ The function bind creates the vector r_j
      y.bind(&r_j[i],m-i)         ▹ The function bind creates the vector y
      d =TNL::dot(u_i,y)          ▹ Vertical operation
      y = y − 2d · u_i            ▹ Horizontal operation
    end for
    for j=0 to n do
      k.bind($Q(0,j).m)           ▹ The function bind creates the vector k
      e.bind(&k[i],m-i)           ▹ The function bind creates the vector e
      d=TNL::dot(u_i,e)           ▹ Vertical operation
      e = e − 2d · u_i            ▹ Horizontal operation
    end for
  end for
  return{Q^T,R}
```

2.2.3 Givens rotation

Algorithm 3 Givens rotation function

Require: Matrix A
Ensure: A=QR

```
Matrix Q = (m,n)                      ▷ Variable m represents the rows
Matrix R = (m,n)                      ▷ Variable n represents the columns
Q=identity matrix
R=A
for i = 0 to m do
    for i = i+1 to n do
        if a_j == 0 then
            c = 1
            s = 0
        else if |a_j| ≥ |a_i| then
            t = a_i/a_j
            s = 1/√(1 + t^2)
            c = s · t
        else
            t = a_j/a_i
            c = 1/√(1 + t^2)
            s = c · t
        end if
        for p = i to n do
            new_i = R(i, p)
            new_j = R(j, p)
            R(i, p) = c · new_i + s · new_j
            R(j, p) = −s · new_i + c · new_j
        end for
        for p = 0 to m do
            new_i = Q(p, i)
            new_j = Q(p, j)
            Q(p, i) = c · new_i + s · new_j
            Q(p, j) = −s · new_i + c · new_j
        end for
    end for
end for
return{Q,R}
```

This algorithm describes the Givens Rotation. The input data is the matrix A, which is decomposed by algorithm into the product of matrix Q and R. The matrix Q is initially set to the identity matrix. A copy of matrix A is stored as matrix R. The first `for` loop iterates through the rows and the inside `for` loop iterates through the columns, where the index is i+1. If a_j is equal to 0 in the loop then c is assigned the value 1 and s is assigned the value 0. If a_j is greater that or equal to a_i, the following operations are performed. The result of the division of a_i by a_j is stored in the variable t. The inverse of the square root of $1 + t^2$ is stored in variable s. The multiplication of variables c and t is assigned to the variable s. If the conditions previously stated are not valid, the following operations are performed. The result

of the division of a_j by a_i is stored in the variable t. The inverse of the square root of $1 + t^2$ is stored in variable c. The multiplication of variables c and t is assigned to the variable s. The `for` loop performs transformations for all rows of the matrix. The transformations are as follows:

- The variable new_i is assigned to the matrix R in row i and column p
- The variable new_j is assigned to the matrix R in row j and column p
- The element $R(i, p)$ is equated to the relation $c \cdot new_i + s \cdot new_j$
- The element $R(j, p)$ is equated to the relation $-s \cdot new_i + c \cdot new_j$

The inside `for` loop transforms all elements in column i of matrix Q and column j of matrix Q.

- The variable new_i is assigned to the matrix Q in the row p and column i
- The variable new_j is assigned to the matrix Q in the row p and columns j
- The element $Q(p, i)$ is equated to the relation $c \cdot new_i + s \cdot new_j$
- The element $Q(p, j)$ is equated to the relation $-s \cdot new_i + c \cdot new_j$

After the completion of the inside and outside **for** loop cycle, the matrices Q and R are returned.

Chapter 3

Code testing and benchmarking

In this chapter, the error rate of algorithm is tested by unit tests and the measurement of each algorithm is tested by benchmark. As mentioned previously, three algorithms are implemented to decompose the matrices, namely Householder reflections, Gram-Schmidt process and Givens rotations. Our study emphasizes the comparison of each algorithm. The subject of investigation is the computation time and also the accuracy of the computations. For the attainment of relevant results, extensive calculations with different data are performed. To compare the result, each algorithm is tested with the same dataset. The unit test verifies the functioning and consistency of the implemented algorithms.

3.1 Unit tests

Unit tests are used to automatically test and verify both the whole algorithm and its subparts[19]. The C++ language is used within the Google test library, which allows testing and verification of the mentioned algorithms correctness. Embedded header files containing the implemented code of the algorithms are included at the beginning of the unit test. All three algorithms are tested according to the following steps.

1. The creation of matrix A and subsequent initialization of matrices Q and R were subsequently realized.
2. The function Givens/Householder/Gram-Schmidt is called.
3. The correct dimensions of the matrices Q and R (whose number of rows and columns correspond to the original matrix A) is verified.
4. An auxiliary matrix B, which calculates the product of the matrices Q and R, is created.
5. The matrix B is compared with the matrix A.
6. In the auxiliary matrix E, the difference between matrix A and matrix B is stored.
7. The Frobenius norm is calculated from the matrix E.

3.2 Performance benchmark

Benchmark is an algorithm used to measure the running time of a program and to compare the performance of individual algorithms. This benchmark is programmed in the C++ programming language. Libraries such as TNL/timer (measures the running time of algorithms) and STL Random library

(generates random matrices) are used. Embedded header files containing the implemented code of the algorithms are included at the beginning of the benchmark. All three algorithms are tested according to the following steps.

1. Implementation of the `Frobenius` function which calculates the Frobenius norm.
2. Implementation of the `measureQRTime` function that measures the computation time of algorithms.
3. Implementation of the `RandomMatrix` function that generates a random matrix with dimensions "rows" and "columns".
4. The "main" function performs the benchmark in the following steps:
   (a) The dimensions of the matrix n is entered by the user.
   (b) Random matrix A is generated by `RandomMatrix` function.
   (c) A QR decomposition is performed for each algorithm and stored in variables. The measurement of calculation time of individual algorithms is base on this analysis.
   (d) In each algorithm, an auxiliary matrix B that computes the product of matrices Q and R is created.
   (e) For each algorithm, the difference between original matrix A and matrix B is calculated. The `Frobenius` norm of the difference for each of these matrices as the measure of the error of the QR decomposition is evaluated.
5. The results of the individual algorithms are compared with the output and their accuracy and computation time are listed on the output.

3.3 Results of benchmark

In this section, the measured values during benchmark testing are examined for each algorithm. The main objective of this project is to test the complexity of individual algorithms and to verify the alignment of theoretical knowledge with the measured values.

3.3.1 Theory of complexity

Theory of complexity includes algorithmic complexity which describes the velocity of a specific algorithms performance. The complexity is characterized as a numerical function T(n) - time depending on the input size n. The aim is to determine the duration of the algorithm without relying on the specifics of its execution.

Complexity of each algorithm:

- Gram-Schmidt orthogonal process: $2n^3$ flops
- Householder reflection: $\frac{8}{3}n^3$ flops
- Givens rotation: $4n^3$ flops

The complexity of the algorithms Householder and Givens include also complexity of computing the matrix Q by matrix products. Meanwhile, the Gram-Schmidt algorithm computes the matrix Q without matrix products.

3.3.2 Measured complexity

n	Givens rotation		Gram-Schmidt process		Householder reflection	
	norm	time[s]	norm	time[s]	norm	time[s]
100	7.54E-05	0.0008	1.74E-05	0.0010	6.82E-05	0.0007
200	2.19E-04	0.0071	4.89E-05	0.0120	1.80E-04	0.0139
400	6.17E-04	0.0338	1.39E-04	0.0342	3.61E-04	0.0547
600	1.09E-03	0.0874	2.55E-04	0.1083	6.17E-04	0.1655
800	1.69E-03	0.2295	3.92E-04	0.2160	8.86E-04	0.3894
1000	2.38E-03	0.2953	5.47E-04	0.3859	1.14E-03	0.7233
1200	3.12E-03	0.7091	7.19E-04	0.5965	1.54E-03	1.1786
2000	6.59E-03	5.0132	1.55E-03	2.3832	3.11E-03	4.6402
2500	9.33E-03	10.6160	2.16E-03	4.5163	4.24E-03	8.8375
3000	1.21E-02	18.7938	2.84E-03	7.3198	5.62E-03	14.3757
3500	1.53E-02	30.2631	3.56E-03	11.1714	7.09E-03	21.9541
4000	1.88E-02	65.5368	4.33E-03	16.6865	8.54E-03	31.6138
4500	2.17E-02	117.9170	5.14E-03	22.8485	1.02E-02	44.8669
5000	2.56E-02	177.2025	5.97E-03	31.3516	1.21E-02	59.4051

Table 3.1: Table of values which show "norm" and "time[s]" of the algorithms

Table 3.1 shows the measured data for three algorithms. The measured data for the individual matrices sizes are in the columns $\mathtt{n}$. The rows show the uniform algorithms that contain two subsets -"norm" and "time". The norm shows the computation of the Frobenius norm that gives the overfitting of the algorithm. The time line is used to indicate the uniform computation times of the algorithms.

According to the theory of complexity, the Gram-Schmidt process is the fastest procedure for QR decomposition following by Householder reflection and Givens rotations as the slowest one. Our study confirms this assumption but the Householder reflection and the Givens rotations seem to be even more time consuming than the theory of complexity supposes.

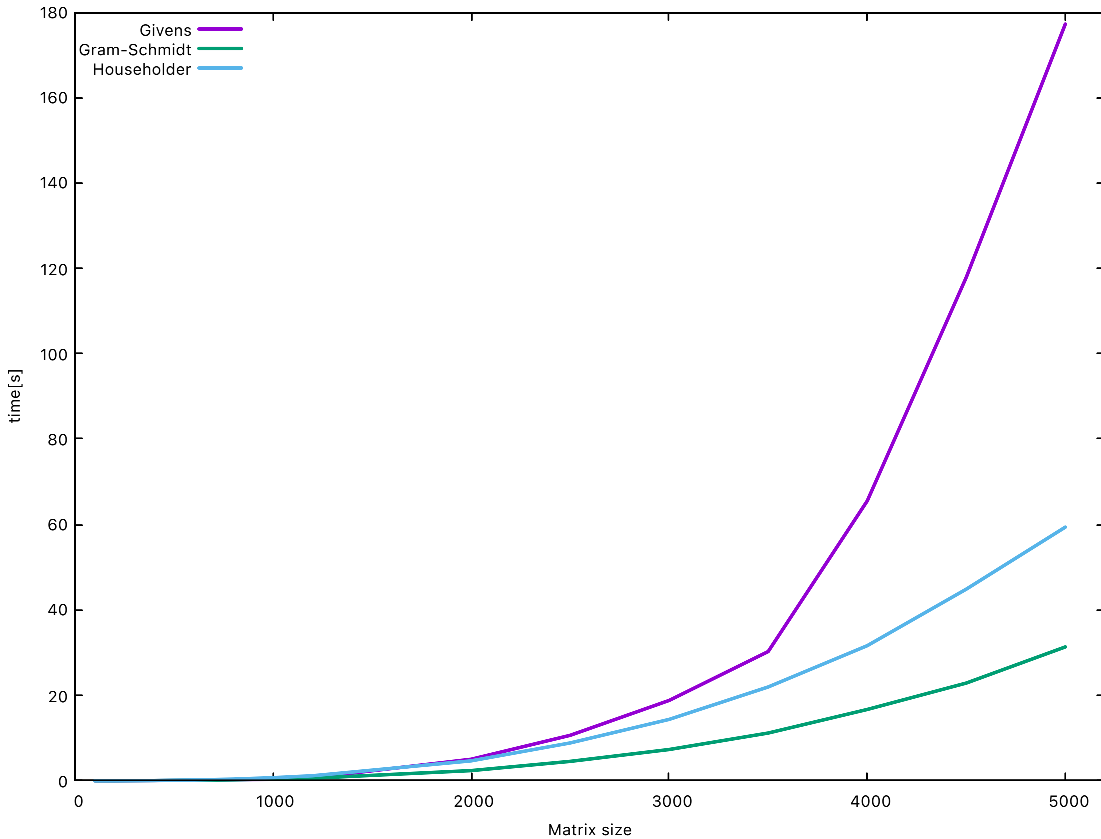


Figure 3.1: Graph of matrix size dependence on computational power

The graph in Figure 3.1 shows the comparison of the algorithms Gram-Schmidt, Householder reflection and Givens rotation with the computation time. The x-axis represents matrix sizes (n) and the y-axis represents the computation power, which is measured in seconds. The graph shows that as the size of the matrices increases, the computation time of all algorithms increases at the same time. This increase in the computational time is mainly due to the higher computational complexity of the individual algorithms. It is visible from the graph, that the the Gram-Schmidt algorithm has the shortest computation time which depends on the size of the matrix, in contrast to the Givens rotation which has the longest computation time.

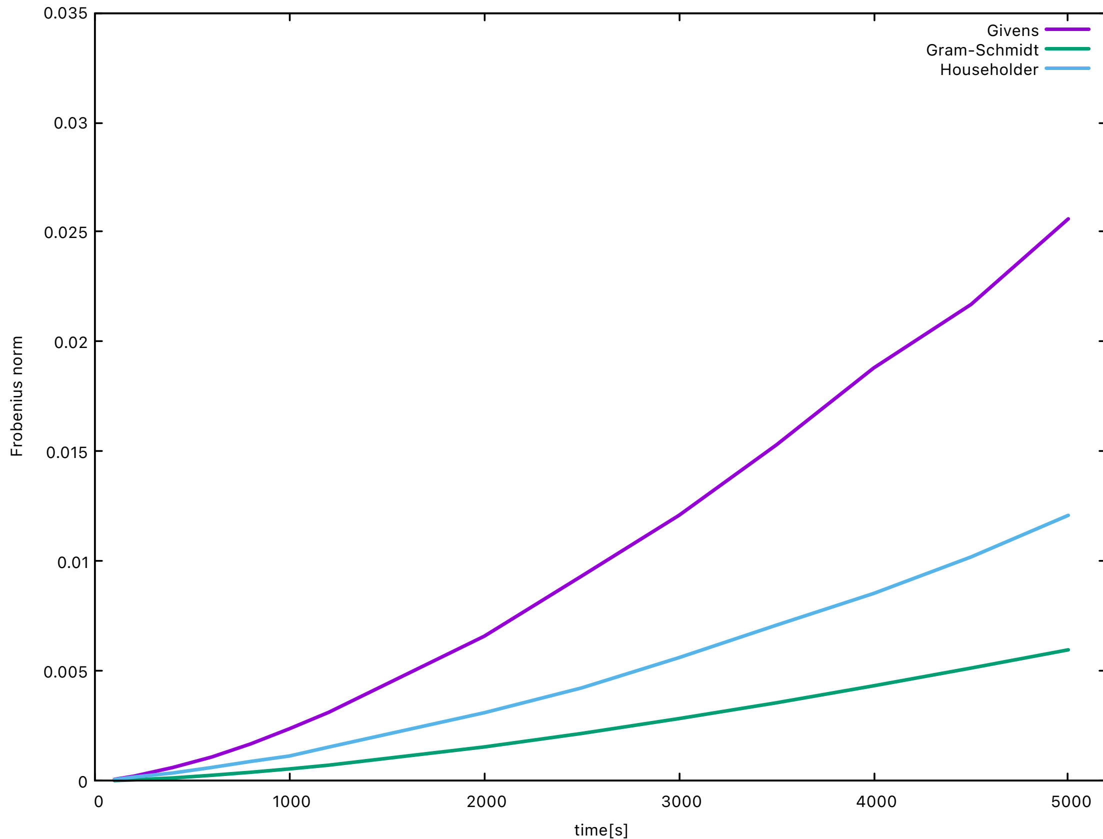


Figure 3.2: Graph of error norms for different matrix sizes

The graph in Figure 3.2 shows the comparison of the algorithms Gram-Schmidt, Householder reflection and Givens rotation with the Frobenius norm. The x-axis represents matrix sizes (n) and the y-axis represents the Frobenius norm. The data demonstrate, that regardless of the algorithm type,the error rate increases with increasing the matrix size. This tendency indicates that with increasing matrix size the computational complexity decreases for all algorithms. Unlike the Givens rotation, which performs relatively high algorithm error rates, the Gram-Schmidt process performs relatively low.

3.3.3 Parallelization

Parallelization is a process in which computation is split into multiple concurrent components that can be virtual and physical. For instance, virtual Graphics Processing Unit (GPU) threads and physical Central Processing Unit (CPU) are suggested. Parallelization offers many advantages such as increase in processing power to solve decomposition of large matrices, or full utilization of CPU and GPU hardware [20]. Using already implemented algorithms, operations can be divided into those that are suitable for parallelization and those that are sequential.

Unlike the Householder reflection and Gram-Schmidt orthogonal process, Givens rotation is not suitable for parallelization, because individual operations depend on the previous ones.

3.3.3.1 Parallelized operation

In case of the Householder reflection and Gram-Schmidt orthogonal process, some operations can be parallelized. One of the operation that is part of Householder's algorithm, is the calculation of the vector for each column of the matrix R, which is performed independently. Similarly, the calculation of the column of the matrix R is performed independently. The following operations are parallelized in the Gram-Schmidt algorithm.

- The calculation of the scalar product of vector
- Normalisation of vector q_i
- Application of orthogonal transformations to matrices Q
- Validity calculation of vector v_0

3.3.4 Parallelized algorithms

In this part of the study, the implementation of the algorithm is analyzed in parallel. The macOS Monterey system with M1 chip, which supports high performance (4 threads available) and energy efficiency (4 threads available), is used. The algorithms are designed for parallelization on multiple threads. First of all, the algorithms are tested on a single thread, whose benchmark results are located in chapter 3.3.2. The algorithm is tested for one and four threads.

n	Givens rotation		Gram-Schmidt process		Householder reflection	
	one core	four cores	one core	four cores	one core	four cores
100	0.0008	0.0018	0.0010	0.0015	0.0007	0.0020
200	0.0071	0.0063	0.0120	0.0120	0.0139	0.0142
400	0.0338	0.0401	0.0342	1.2008	0.0547	0.7030
600	0.0874	0.1010	0.1083	4.0085	0.1655	4.1239
800	0.2295	0.3311	0.2160	7.0672	0.3894	11.4638
1000	0.2953	0.3103	0.3859	11.6640	0.7233	22.4726
1200	0.7091	0.7159	0.5965	15.9757	1.1786	35.0741
2000	5.0132	5.1028	2.3832	44.3141	4.6402	113.7870
2500	10.6160	10.5667	4.5163	73.0379	8.8375	186.2050
3000	18.7938	19.5969	7.3198	102.8590	14.3757	282.6100
3500	30.2631	31.3132	11.1714	140.5550	21.9541	383.4850
4000	65.5368	65.5623	16.6865	183.7850	31.6138	626.0220
4500	117.9170	116.6370	22.8485	248.5960	44.8669	695.2690
5000	177.2025	176.1060	31.3516	292.4920	59.4051	844.6290

Table 3.2: The table of values which show computation time [s] for one and four threads.

The table 3.2 shows the measured data using four threads, when algorithms are employed. The matrix size is displayed on the columns. The rows depicted the uniform algorithm that contain two subsets-"one core and "four cores". The "one core" represents the computation of computation time for one thread. The "four cores" line is used to indicate the uniform computation times for four threads of the algorithms.

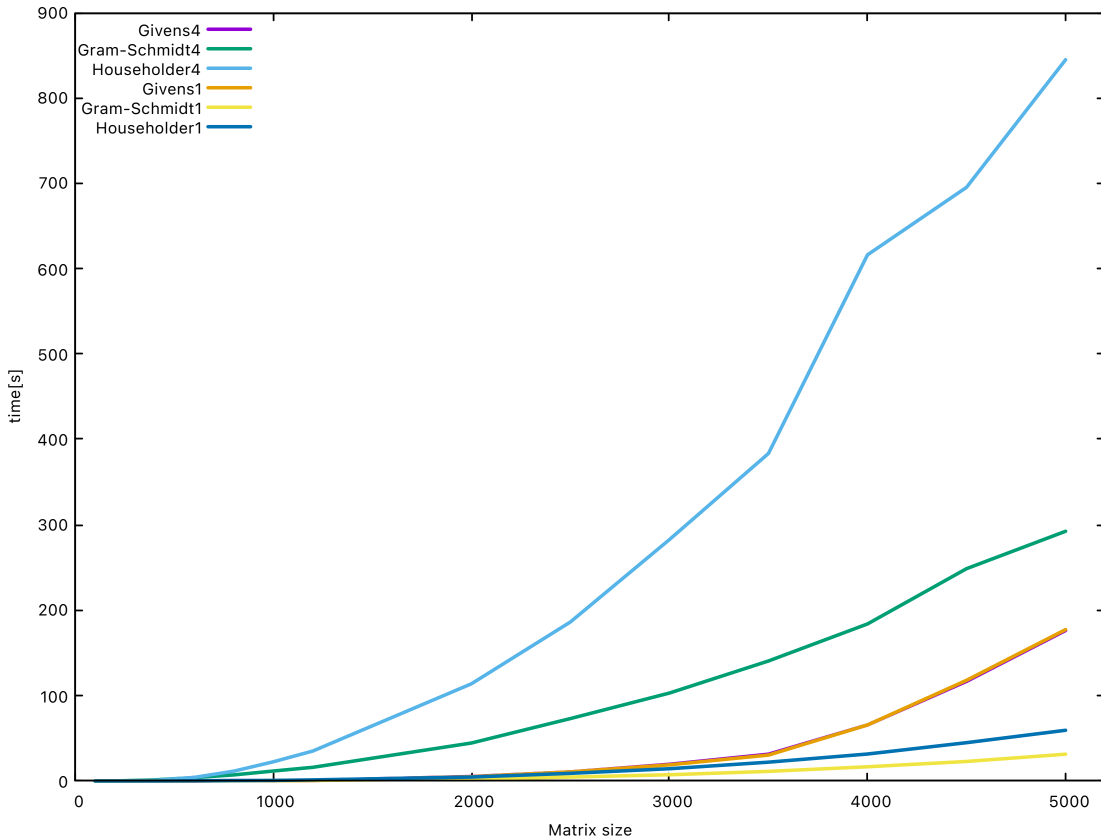


Figure 3.3: Graph of matrix size dependence on computational power

The graph in Figure 3.3 shows the comparison of the mentioned algorithms with the computation time using one and four threads. The x-axis represents matrix size (n) and the y-axis represents the computation power, which is measured in seconds. The graph shows that as the size of the matrices increases, the computation time of all algorithm increases.

From the graph, it is readable that the number of running threads does not matter for the Givens algorithm, and therefore, parallelization does not have any influence on the computation time. The effect of parallelization on the shortening computation time is not discovered for the remaining two algorithms. Unexpectedly, the more threads are used, the longer computation time is occurred. In comparing to other algorithms, the algorithm of Householder with four threads appears to be the most computational demanding, while Gram-Schmidt using a single thread seems to be as the computational fastest.

Chapter 4

Future applications

The using of modern of technology needs creation of more and more algorithms which is used in mathematics and other sciences. Already programmed algorithms can do the computation faster and user friendly. Programmed algorithms allow user to work with a much larger data than before.

4.1 QR decomposition

The use of QR decomposition has many useful applications in mathematics, physics and other sciences. One of them can be the signal analysis where signal can be represented by the matrix QR decomposition. Many finite element solvers use QR decomposition to solve eigenvalue problems.

4.1.1 Parallelization

Not every algorithm is suitable for parallelization. The right method choose for parallelisation is crucial for the computing. The results can be different for example because of using Gram - Schmidt modified transformation which is not suitable for parallelization, but more accurate in computing results. And vice versa usual Gram - Schmidt transformation is more suitable for parallelization, but leads to less accurate computing results. Remaining two methods might be modified for better use in parallelization.

4.2 Use of Gram-Schmidt, Householder and Givens algorithms

Programmed algorithms (Gram-Schmidt orthogonalisation process, Householder reflection, Givens rotation) can serve as quite powerful tools in linear algebra and numerical calculations.

As mentioned above, all tree algorithms are quite efficient tools for **QR decomposition**. However, this is not the only use of these algorithms.

The Gram-Schmidt orthogonalization algorithm has a primary purpose: to orthogonalize a vector in space. This algorithms can be part of solving linear equations or approximating functions, but it can be part of many other mathematical problems that require an orthogonal base. Other applications of this algorithm are found in linear regression, signal filtering or compression algorithms.

The Householder reflection is used to reduce the matrix to a triadiagonal form. Similar to the Gram-Schmidt orthogonalization process algorithm, this algorithm is used in the solution to linear equations, but not only in it. Another method can be function approximation, where the function is approximated by polynomials. This procedure is commonly used to approximate relatively complex functions to simplify ones. The algorithm is used primarily in the fields of physics (the signal filtering and i the mage analysis) and mathematics (linear regression).

In particular, the rotation of the vector in space is the main use of the Givens rotation. Another application may be in the calculation of condition systems of equations or least squares approximation.

4.3 Future work

This work could then be furthered through testing on high-performance graphics cards. One of the potential benefits of this testing would be to verify the magnitude of the computation speed increase due to parallel processing. For more exact results, other testing of this algorithm should be provided by solution some mathematical examples. One of these could be searching for custom number using *QR* algorithm.

The future usage could be implementation into TNL library, which can help scientist with mathematical examples. The algorithms can be modified and subsequently used on modern supercomputers for further use for the better algorithm efficiency.

Conclusion

Linear algebra is one of the main part of modern mathematics and is used in most science and fields of engineering. Computing is often very difficult and complicated, takes a long time and needs the newest PC equipment. To find an optimal way how to make computing most efficient and accurate is an important aim of current and future studies. This study didn't shown that using of three QR decompositions methods (the Gram-Schmidt process, Householder transformations and Givens rotations) in parallelisation of computation leads to faster and more accurate results.

The performance and complexity of three algorithms were compared using the benchmark. The object of the investigation was to find the Frobenius norm of computation time for different matrix sizes. Both the Frobenius norm and the measured computation time increased in value with the size of the matrices. A relatively lower error rate and computation time were observed for the Gram-Schmidt algorithm compared to Givens rotation and Householder reflection. For all algorithms, the computation times increased depending on their theoretical complexity. Results of the compared tests did not confirm the consistency of theoretical knowledge with practical measurements.

Unexpectedly, the more threads are used, the longer computation time is occurred. The number of running threads does not matter for the Givens algorithm, and therefore, parallelization does not have any influence on the shortening of the computation time. The effect of parallelization on the shortening computation time is not discovered for the remaining two algorithms as well. In comparing to other algorithms, the algorithm of Householder with four threads appears to be the most computational demanding, while Gram-Schmidt using a single thread seems to be as the computational fastest. Unexpected results could be caused by using Gram-Smidt modified transformation which is not suitable for parallelization, but more accurate in computing results. And vice versa usual Gram - Schmidt transformation is more suitable for parallelization, but leads to less accurate computing results. Remaining two methods might be modified for better use in parallelization.

Bibliography

1. KREJČIŘÍK, David. *Lineární algebra*. 2020. Available also from: http://nsa.fjfi.cvut.cz/david/other/la.pdf.
2. FIEDLER, Miroslav. *Special matrices and their applications in numerical mathematics*. Courier Corporation, 2008.
3. HUANG, Yongwei; ZHANG, Shuzhong. Complex matrix decomposition and quadratic programming. *Mathematics of Operations Research*. 2007, vol. 32, no. 3, pp. 758–768.
4. ANDRILLI, Stephen; HECKER, David. *Elementary linear algebra*. Academic Press, 2022.
5. *IBM Corporation*. 2019. Available also from: https://www.ibm.com/docs/en/essl/6.2?topic=matrices-triangular-matrix.
6. Wikimedia Foundation, 2023. Available also from: https://en.wikipedia.org/wiki/Square_matrix.
7. CUEMATH.COM. *Matrices - solve, types, meaning, examples: Matrix definition*. [N.d.]. Available also from: https://www.cuemath.com/algebra/solve-matrices/.
8. WEISSTEIN, Erik W. *Frobenius Norm*. 2023. Available also from: https://mathworld.wolfram.com/FrobeniusNorm.html.
9. GOODALL, Colin R. 13 Computation using the QR decomposition. In: *Computational Statistics*. Elsevier, 1993, vol. 9, pp. 467–508. Handbook of Statistics. ISSN 0169-7161. Available from DOI: https://doi.org/10.1016/S0169-7161(05)80137-3.
10. ZEMÁNEK, Petr. *QR rozklad*. 2006. Available also from: https://www.math.muni.cz/~zemanekp/files/QR-rozklad_%5Bseminarni_prace_-_Petr_Zemanek%5D.pdf.
11. LAZNOVÁ, Veronika. *QR-rozklad matice*. 2015. Available also from: https://otik.zcu.cz/bitstream/11025/19796/1/Bakalarska%20prace.pdf.
12. OBERHUBER, Tomáš; KLINKOVSKÝ, Jakub; FUČÍK, Radek. TNL: Numerical library for modern parallel architectures. *Acta Polytechnica*. 2021, vol. 61, no. SI, pp. 122–134.
13. *TIOBE*. 2023. Available also from: https://www.tiobe.com/.
14. FINEGAN, Edward. 1 "The Design and Evolution of C++". 2005.
15. CHAN, Terrence. *Unix system programming using C++*. Prentice-Hall, Inc., 1996.
16. VELDHUIZEN, Todd. Expression templates. *C++ Report*. 1995, vol. 7, no. 5, pp. 26–31.
17. KLINKOVSKÝ, Jakub. *TNL Documentation*. 2023. Available also from: https://tnl-project.gitlab.io/tnl/md_UsersGuide_users_guide.html#UsersGuide.
18. GILBERG, Richard F; FOROUZAN, Behrouz A. *Data structures: A pseudocode approach with C++*. Brooks/Cole Publishing Co., 2001.

19. OLAN, Michael. Unit testing: test early, test often. *Journal of Computing Sciences in Colleges*. 2003, vol. 19, no. 2, pp. 319–328.
20. JEREB, Borut; PIPAN, Ljubo. Measuring parallelism in algorithms. *Microprocessing and Microprogramming*. 1992, vol. 34, no. 1-5, pp. 49–52.