UNIVERZITA PARDUBICE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

DIPLOMOVÁ PRÁCE

2025 Bc. Erik Ženatý

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2024/2025

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Erik Ženatý**
Osobní číslo:	**I23287**
Studijní program:	**N0613A140007 Informační technologie**
Téma práce:	**Výpočet a užití QR rozkladu pro řídké matice**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

QR rozklad matice je způsob, jak zapsat čtvercovou matici jako součin dvou matic z nichž první je ortogonální a druhá je v horním trojúhelníkovém tvaru. Využívá při řešení soustav lineárních rovnic, výpočtu inverzních matic, nebo při výpočtu vlastních čísel matice. QR rozklad lze počítat několika způsoby. Cílem práce bude vytvoření aplikace pro výpočet QR rozkladu řídkých matic několika způsoby, srovnání časové a paměťové složitosti a aplikace QR rozkladu na konkrétní problémy.

Rozsah pracovní zprávy:
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná**

Seznam doporučené literatury:

VITÁSEK, Emil. *Numerické metody*. 1. vyd. Praha: SNTL, 1987, 512 s. JANG, Won-jong. *Applied numerical methods using MATLAB*. Hoboken: Wiley-Interscience, c2005, xiv, 509 s. ISBN 04-716-9833-4.

Vedoucí diplomové práce: **RNDr. Josef Rak, Ph.D.**
Katedra automatizace a matematiky

Datum zadání diplomové práce: **31. října 2024**
Termín odevzdání diplomové práce: **23. května 2025**

L.S.

prof. Ing. Petr Doležel, Ph.D. v.r.
děkan

prof. Ing. Antonín Kavička, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 29. listopadu 2024

Prohlašuji:

Práci s názvem Výpočet a užití QR rozkladu pro řídké matice jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 25. 08. 2025 Bc. Erik Ženatý

PODĚKOVÁNÍ

Na tomto místě bych rád poděkoval RNDr. Josefovi Rakovi, Ph.D za cenné rady, připomínky a pomoc při psaní diplomové práce. Jeho připomínky mi vždy pomohly práci vylepšit a díky tomu se mi práce psala mnohem lépe. Také bych chtěl poděkovat vyučujícím, kteří mě vzdělávali během magisterského studia za předání odborných znalostí, které využiji během následujících let.

ANOTACE

QR rozklad je způsob, jak vyjádřit čtvercovou matici jako součin ortogonální matice a horní trojúhelníkové matice. Tento postup se často využívá při řešení soustav lineárních rovnic, výpočtu inverzních matic nebo určení vlastních čísel. Existuje několik různých algoritmů pro výpočet QR rozkladu. Cílem této práce je vyvinout aplikaci pro výpočet QR rozkladu řídkých matic pomocí různých metod s různými formáty uložení řídkých matic. Cílem bude porovnat časovou a paměťovou složitost jednotlivých metod QR rozkladu na konkrétních příkladech.

KLÍČOVÁ SLOVA

Řídké matice, COO, CSR, CSC, QR rozklad

TITLE

Computation and Application of QR Decomposition for Sparse Matrices

ANNOTATION

QR decomposition is a way to express a square matrix as a product of an orthogonal matrix and an upper triangular matrix. This procedure is often used for solving systems of linear equations, calculating inverse matrices, or determining eigenvalues. There are several different algorithms for computing the QR decomposition. The goal of this work is to develop an application for computing the QR decomposition of sparse matrices using various methods with different sparse matrix storage formats. The aim will be to compare the time and memory complexity of individual QR decomposition methods on specific examples.

KEYWORDS

Sparse matrices, COO, CSR, CSC, QR decomposition

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

COO	Coordinate List
CSR	Compressed Sparse Row
CSC	Compressed Sparse Column
NNZ	Number of Non-Zeros

ÚVOD

Jedním ze základních nástrojů lineární algebry, využívaným například při řešení soustav lineárních rovnic, výpočtu inverzních matic a vlastních čísel matice, je QR rozklad, právě tímto rozkladem se bude zabývat celá práce.

Cílem této diplomové práce je implementace aplikace, která umožní výpočet QR rozkladu řídkých matic, tyto matice se ukládají v jiných formátech než matice husté. Pro navrženou aplikaci bude klíčové srovnání různých metod z hlediska časové a paměťové složitosti. Zvláštní důraz bude kladen na využití struktur řídkých matic a porovnání různých přístupů k výpočtu QR rozkladu. Aplikace bude navržena tak, aby bylo možné jednoduše měřit časovou a paměťovou náročnost jednotlivých algoritmů při aplikaci na konkrétní matice.

V teoretické části práce budou rozebrány základní koncepty vedoucí ke QR rozkladu, bude se jednat o základní pojmy lineární algebry. V této části budou nejprve uvedeny pojmy týkající se matic, poté se kapitola zaměří na ukládání matic a následně bude uveden LU rozklad. Po těchto konceptech bude dále teoretická část zaměřena na metody výpočtu QR rozkladu, konkrétně půjde o klasickou Grammovu-Schmidtovu metodu, modifikovanou Grammovu-Schmidtovu metodu, Householderovu metodu a Givensovu metodu výpočtu. Po rozebrání metod QR rozkladu bude nadále v práci popsána praktická část.

Praktická část práce se zabývá návrhem a implementací konzolové aplikace v jazyce C++ pro QR rozklad matic pomocí různých metod a formátů. Cílem je ověřit správnost implementace a porovnat časovou i paměťovou náročnost jednotlivých algoritmů. Před implementací byly definovány funkční a nefunkční požadavky. Výsledkem je aplikace doplněná o programátorskou a uživatelskou příručku. Správnost metod byla ověřena pomocí jednotkových testů v samostatném projektu. Závěr práce se věnuje analýze výpočetní efektivity na základě teorie i praktických měření.

V závěrečná část se zaměřuje na praktické aplikace QR rozkladu, jako je řešení lineárních soustav, výpočet inverze a pseudoinverze matic, výpočet vlastních čísel a kořenů polynomů. Uvedeny jsou i příklady z informatiky, kde se QR rozklad uplatňuje, s cílem ukázat jeho široké možnosti využití v praxi.

Matice

Obdélníkové schéma, které má m řádků a n sloupců a je ve tvaru $A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$ se nazývá matice. Toto obdélníkové schéma se skládá z prvků $(a_{ij})_{\substack{i=1,\dots\dots m \\ j=1\dots\dots n}}$, které mohou být reálné, respektive komplexní. Množinu všech matic určitého rozměru $m \times n$ lze označit jako $M^{m\times n}$. Prvky, které se nacházejí na pozicích $a_{11}, a_{22}, \dots, a_{mn}$ tvoří takzvanou hlavní diagonálu. [1]

1.1 Unární a binární operace s maticemi

Maticové operace lze rozdělit na unární a binární. Unární operace matic mají jen jediný operand, tedy je to operace, která je prováděna pouze s jednou maticí. Mezi unární operace se řadí transpozice a inverze matice. Oproti tomu binární operace obsahující dva operandy. Typickými příklady jsou sčítání a odčítání matic, násobení matice skalárem, násobení matice maticí a mnoho dalších. Dříve než budou vyjmenované operace popsány, je nutné nadefinovat rovnost matic. Matice A je rovna matici B právě tehdy pokud pro $\forall a_{ij} = b_{ij}$. [2]

Tabulka 1 – Unární a binární operace matic

Operace	**Definice operace**
Transpozice (A^T) $\boldsymbol{A \in M^{m\times n}}$	Je unární operace, kde pro $\forall i,j$ platí ${a^T}_{ij} = a_{ji}$
Sčítání matic ($C = A + B$) $\boldsymbol{A \in M^{m\times n}, B \in M^{m\times n}}$	Je binární operace, kde pro $\forall i,j$ platí $c_{ij} = a_{ij} + b_{ij}$
Odčítání matic ($C = A - B$) $\boldsymbol{A \in M^{m\times n}, B \in M^{m\times n}}$	Je binární operace, kde pro $\forall i,j$ platí $c_{ij} = a_{ij} - b_{ij}$
Násobení matice skalárem ($C = r \cdot A$) $\boldsymbol{A \in M^{m\times n}}$	Je binární operace, kde pro $\forall i,j$ platí $c_{ij} = r \cdot a_{ij}$
Násobení matice maticí ($C = A \cdot B$) $\boldsymbol{A \in M^{m\times s}, B \in M^{s\times n}}$	Je binární operace, kde pro $\forall i,j$ platí $c_{ij} = \sum_{k=1}^{s} a_{ik} \cdot b_{kj}$
Inverze (A^{-1}) $\boldsymbol{A \in M^{n\times n}}$	Je unární operace, pro kterou platí $A \cdot A^{-1} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

1.2 Typy matic

Jednotlivé matice lze kategorizovat do určitých skupin podle jejich prvků. Každou z těchto skupin lze pojmenovat a určit jaké matice do ní patří. V této části budou představeny nejčastější typy matic a jejich popis na matici A.

Tabulka 2 – Typy matic a jejich definice [1] [2] [3]

Název matice	**Popis matice**
Hustá	Je matice, kde většina $a_{ij} \neq 0$
Řídká	Je matice, kde většina $a_{ij} = 0$
Obdélníková	Je matice $m \times n$, kde $m \neq n$
Čtvercová	Je matice $m \times n$, kde $m = n$
Nulová	Je matice, kde $\forall a_{ij} = 0$
Jednotková	Je čtvercová matice I, kde všechny prvky hlavní diagonály jsou rovny 1
Transponovaná	Je matice, na kterou byla aplikována operace transpozice
Diagonální	Je čtvercová matice, kde jsou všechny prvky kromě hlavní diagonály rovny 0
Symetrická	Je čtvercová matice, pro kterou platí $A = A^T$
Inverzní	Je čtvercová matice, na kterou byla aplikována operace inverze
Regulární	Je čtvercová matice, pro kterou platí $\exists A^{-1}$
Singulární	Je čtvercová matice, pro kterou platí $\nexists A^{-1}$
Ortogonální	Je matice, kde platí $A \cdot A^T = I$
Horní trojúhelníková	Je matice, kde $\forall a_{ij} = 0$ pro $\forall i > j$
Dolní trojúhelníková	Je matice, kde $\forall a_{ij} = 0$ pro $\forall i < j$

V tabulce jsou zaznamenány nejčastější typy matic. Většina z těchto pojmů bude použita v následujících kapitolách diplomové práce. Nejdůležitějším typem matice pro praktickou část práce jsou pojmy řídké a husté matice. Důležité je zmínit, že tyto pojmy nejsou matematicky definované, a tudíž je nelze definovat přesně. [3]

1.3 Formáty ukládání matic

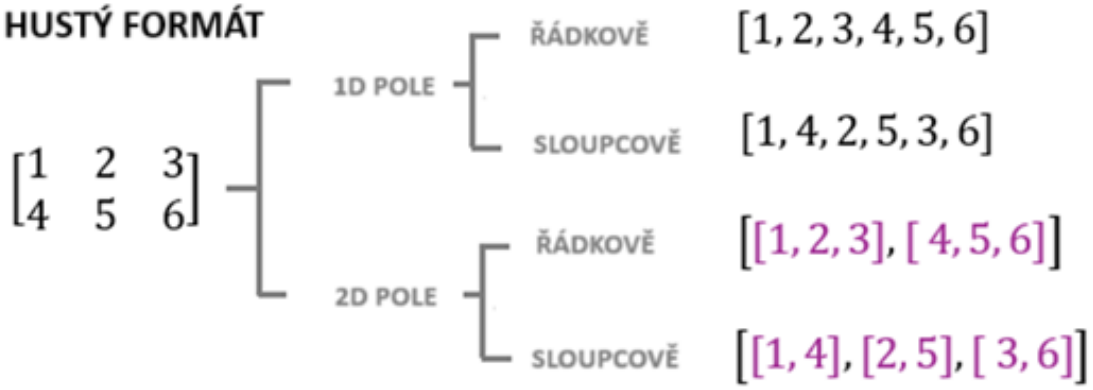
Cílem této podkapitoly je představit možnosti ukládání matic v paměti počítače. Nejčastějším způsobem ukládání je hustý formát. Tento formát je použitelný pro všechny typy matic, které byly popsány v předešlé části. Kromě hustého formátu existují formáty COO, CSR, CSC. Tyto formáty se používají především pro řídké typy matic. O formátu uložení matice může v určitých případech rozhodnout takzvaná hustota matice (density), která je definována jako počet nenulových prvků (nnz) vydělených celkovým počtem prvků. [4]

1.3.1 Hustý formát ukládání

Matice v hustém formátu lze ukládat do 1D pole, případně 2D pole. Každou z těchto dvou variant je možné uložit sloupcově či řádkově.

Matice A o rozměrech $m \times n$ je v případě 1D pole uložena do jednoho dlouhého pole o $m \cdot n$ prvcích. Matice je v tomto poli uložena po řádcích, případně po sloupcích. Při řádkovém ukládání se prvek matice a_{ij} nachází na pozici $i \cdot n + j$. Použije-li se sloupcové ukládání, bude prvek a_{ij} v 1D poli na pozici $j \cdot m + i$. Metoda ukládání do 1D pole umožňuje efektivnější využití paměti a lepší výkon, protože data jsou uložena souvisle, což zlepšuje práci s cache procesoru.

V případě uložení matice A s rozměrem $m \times n$ do 2D pole, bude mít pole opět $m \cdot n$ prvků. Tentokrát je způsob uložení jiný, jedná se o uložení „pole v poli“. Každý řádek, popřípadě sloupec, je uložen do pole, které je umístěno v hlavním poli. Pro přístup k prvku a_{ij} jsou použity dva indexy $A_{[i][j]}$. Tato implementace je lépe čitelná než 1D pole.

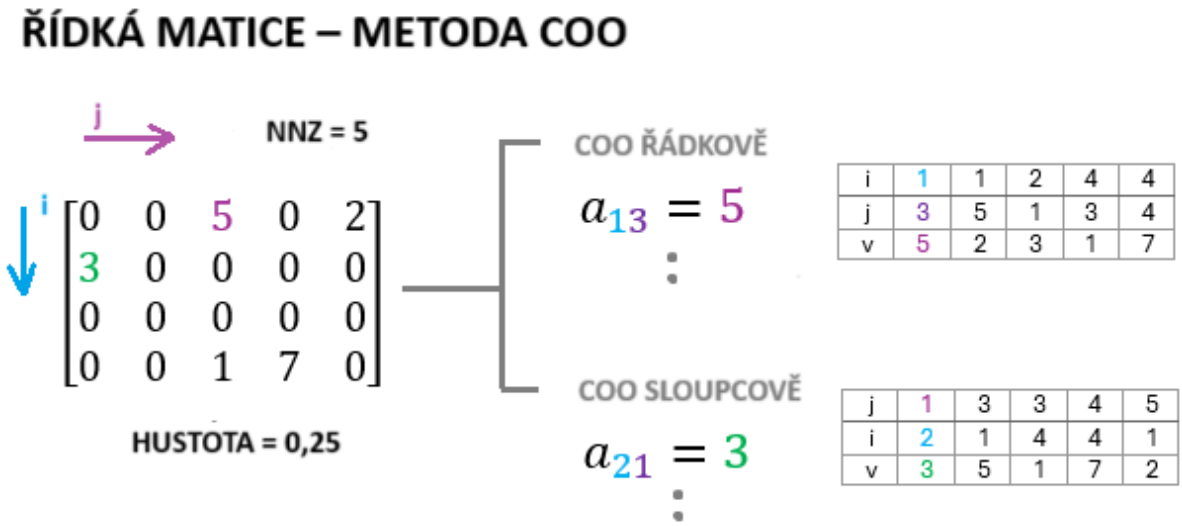


Obrázek 1 – Formáty ukládání hustých matic

1.3.2 Řídký formát matic

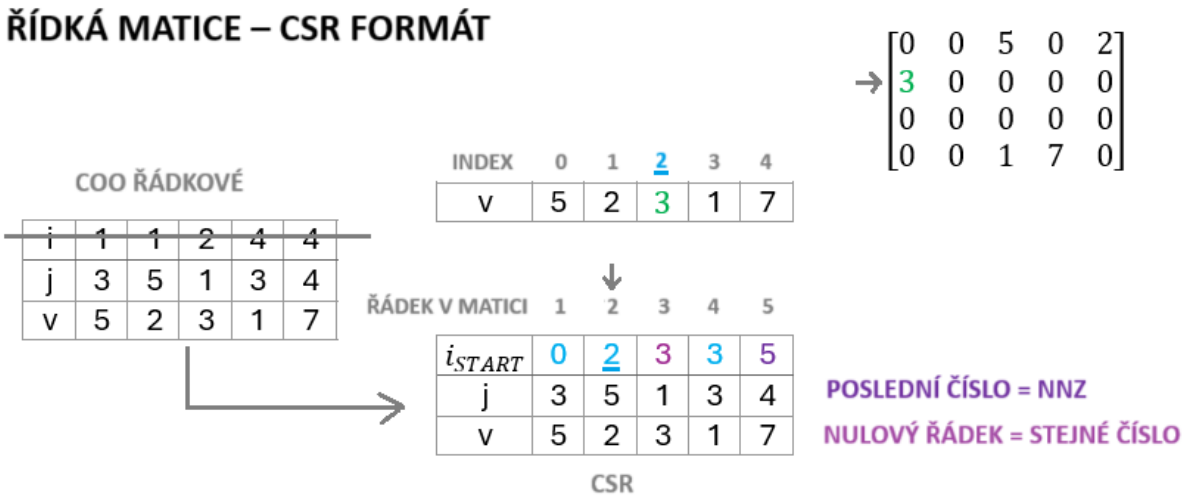
Existují i další možnosti ukládání matic, kterými jsou COO, CSR, CSC. Tyto formáty jsou výhodné především pro řídké matice. Všechny tyto formáty lze uložit do 1D pole, 2D pole, případně do tří separátních polí. Nejvhodnější metodou z pohledu čitelnosti je uložení do tří separátních polí.

Řídká matice A o rozměrech $m \times n$ lze uložit do COO (Coordinate List), neboli seznamu souřadnic. Tento způsob pro prvek a_{ij} ukládá trojici (i, j, v) do tří polí. Ve trojici i reprezentuje pozici řádku, j je pozice sloupce a v hodnota na dané pozici. Tyto trojice mohou být uloženy po řádcích (row-major) nebo po sloupcích (column-major). Výhodou tohoto formátu je dobrá čitelnost, avšak oproti jiným řídkým formátům horší paměťová náročnost. Z tohoto formátu vycházejí formáty CSR a CSC. U těchto formátů, platí $CSR(A) = CSC(A^T)$, případně $CSC(A) = CSR(A^T)$. [4]



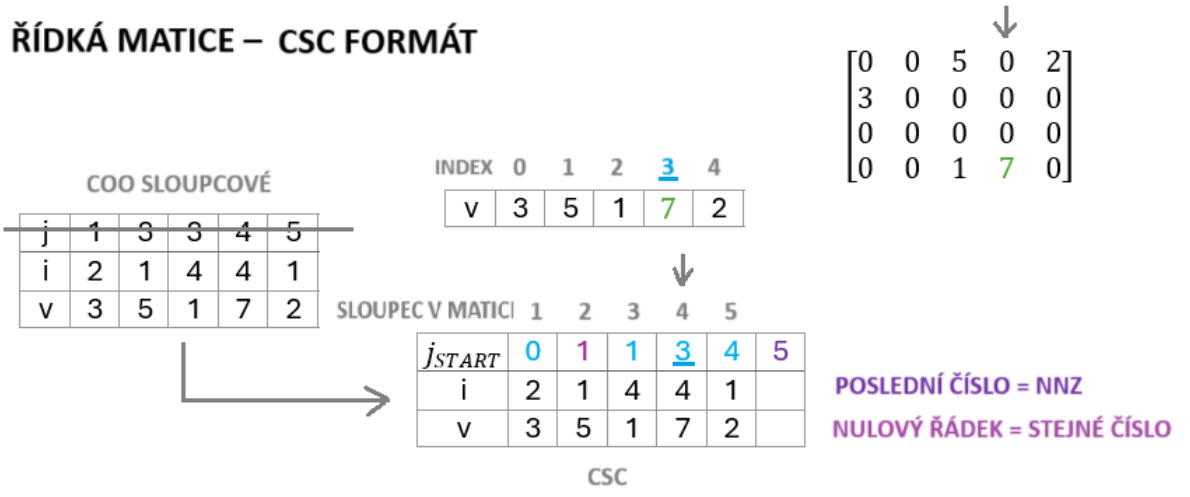
Obrázek 2 – COO metoda ukládání řídkých matic

Formát CSR (Compressed Sparse Row) je způsob uložení matice $A \epsilon M^{m \times n}$ založený na COO, který trojice ukládá po řádcích. Tento formát ukládá pro prvek a_{ij} trojice (i_{start}, j, v) opět do tří polí. Písmeno j určuje pozici sloupce, v je hodnotou na dané pozici stejně jako u COO. Rozdílem je i_{start}, který je indexem ve v kde začíná $i - tý$ řádek (např ve 2_{start} udává na jakém indexu v začíná druhý řádek matice). Posledním prvkem (m_{start}) je vždy počet nenulových prvků matice (nnz). V případě, že se v řádku nenachází žádná hodnota, je zde vložena hodnota následujícího řádku. [4]



Obrázek 4 – CSR metoda ukládání řídkých matic

Posledním popsaným formátem ukládání řídkých matic $A \epsilon M^{m \times n}$ bude formát CSC (Compressed Sparse Column). CSC je opět založený na COO a ukládání do tří polí, tentokrát se jedná o ukládání po sloupcích. Pro prvek a_{ij} se uloží trojice (i, j_{start}, v), u této trojice i značí pozici řádku, v hodnotu na dané pozici a j_{start} je index ve v kde začíná $j-tý$ sloupec. Například 3_{start} vyjadřuje na jakém indexu v začíná třetí sloupec. Poslední hodnotou n_{start} je opět nnz a v případě nulového sloupce se opět vloží hodnota následujícího sloupce. Výhodou tohoto přístupu je opět menší než u COO. [4]

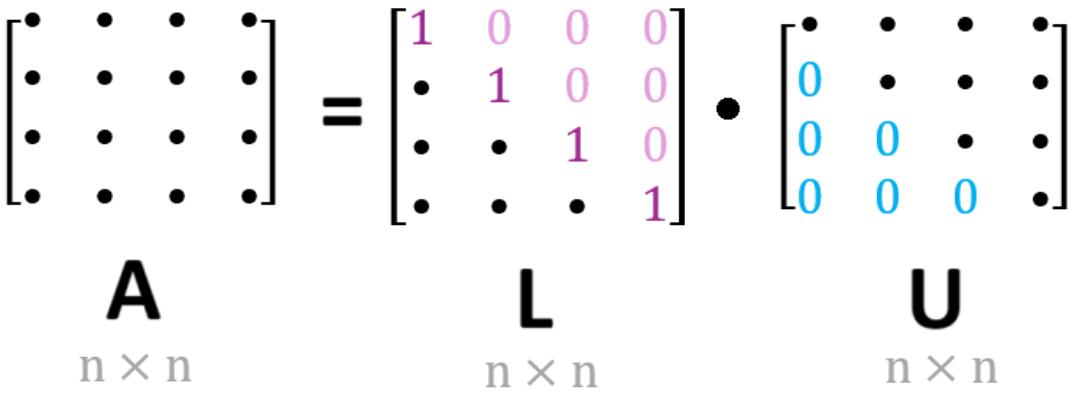


Obrázek 3 – CSC metoda ukládání řídkých matic

1.4 LU-rozklad

Pomocí LU-rozkladu se dají řešit soustavy lineárních rovnic, vypočítat determinant případně určit inverzní matici. Oproti přímé Gaussově eliminaci nabízí LU rozklad výhodu zejména při opakovaném řešení soustav s různými pravými stranami, kdy lze již jednou provedený rozklad opakovaně využít bez nutnosti dalších výpočtů. Na rozdíl od přímého výpočtu inverze matice, který je numericky méně stabilní, LU rozklad umožňuje řešit úlohy efektivněji. Při výpočtech determinantů poskytuje LU rozklad rychlý výpočet determinantu díky vlastnostem trojúhelníkových matic.

LU-rozklad je způsob zápisu čtvercové matice A ve formě součinu dvou čtvercových matic, a to L a U. Pro prvky matice L platí $l_{ij} = \begin{cases} l_{ij}, & pro\ i > j \\ 1, & pro\ i = j \\ 0, & pro\ i < j \end{cases}$ a pro prvky matice U platí $u_{ij} = \begin{cases} u_{ij}, pro\ i \leq j \\ 0, pro\ i > j \end{cases}$.



Obrázek 5 – LU-rozklad

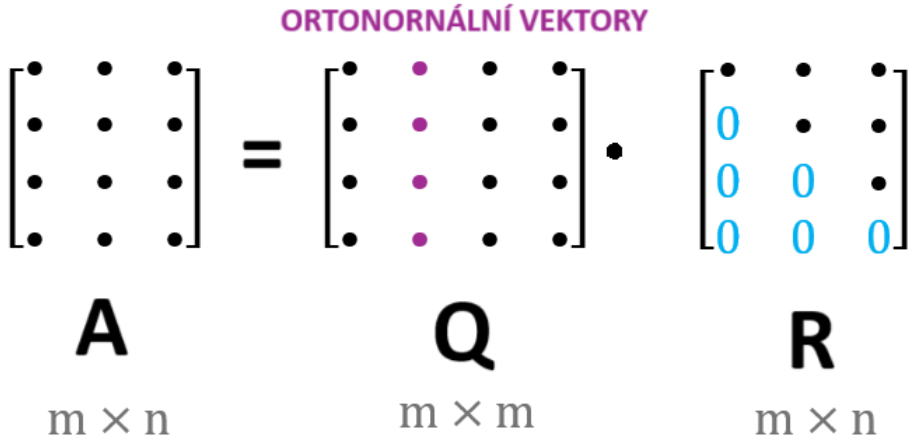
Tento rozklad lze dopočítat pomocí takzvaných elementárních matic, které vycházejí z ekvivalentních řádkových úprav soustav lineárních rovnic. Tyto matice jsou z většiny tvořeny z jednotkové matice, která je modifikována podle určité ekvivalentní úpravy soustavy lineárních rovnic. [5]

2 QR rozklad a metody jeho výpočtu

Tato kapitola se bude blíže zabývat QR rozkladem matic. Využijí se zde již známé věci z předešlé kapitoly a nastíní se způsoby, kterými lze QR rozklad spočítat. Než budou rozebrány metody QR rozkladu, bude nutno blíže specifikovat co QR rozklad je.

Nechť existuje matice A velikosti $m\ x\ n$, pak tato matice lze rozložit na součin $A = Q \cdot R$, kde Q je matice $m \times m$ obsahující m ortonormálních sloupcových vektorů a R je matice $m\ \times\ n$, která je v horním trojúhelníkovém tvaru. [6] [7]

V následujících podkapitolách bude spočítán QR rozklad různými metodami. Rozklad lze spočítat pomocí klasického Grammova-Schmidtova procesu, modifikovaného Grammova-Schmidtova procesu, Householderových transformací a Givensovy rotace. Tyto metody budou vždy nejprve představeny matematickou formou a následně vysvětleny na příkladě. K naprogramování jednotlivých metod je v této kapitole uveden algoritmus, který bude demonstrován na příkladu konkrétní matice. Pro výpočty týkající se Givensovy rotace a Householderovy transformace je nutné zmínit, že součin ortogonálních matic je také ortogonální matice. [8]



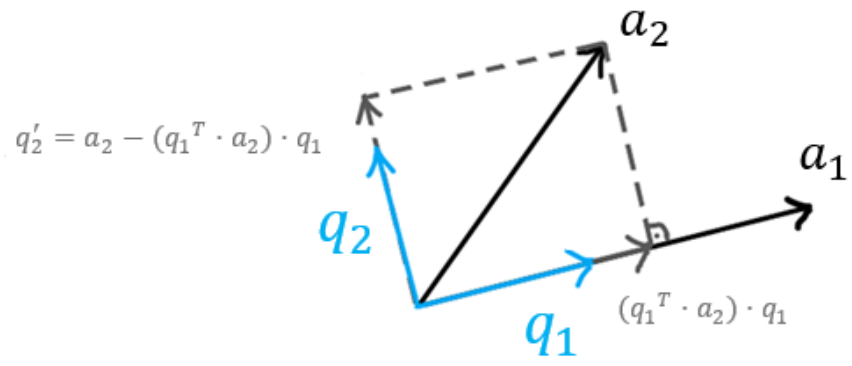
Obrázek 6 – QR rozklad

2.1 Klasický Grammův-Schmidtův proces

„Klasický Grammův-Schmidtův proces převede lineárně nezávislou sadu vektorů na ortonormální sadu vektorů se stejným lineárním obalem.“ [9 str. 29]

Nechť $\{a_j\}_{j=1}^{n}$ jsou sloupce matice A, kde platí $q_1 = \frac{a_1}{\|a_1\|}$. Právě tento vztah lze zobecnit jako $q_k = \frac{a_k - p_{k-1}}{\|a_k - p_{k-1}\|}$ kde $p_0 = 0$ a $p_{k-1} = \sum_{j=1}^{k-1} \langle q_j, a_k \rangle q_j$. Každé p_{k-1} je projekcí a_k na podprostor generovaný předchozími vektory $\{q_j\}_{j=1}^{k-1}$, proto residuální vektor projekce q'_k je roven vztahu $q'_k = a_k - p_{k-1}$. Normalizací q'_k je získána množina $\{q_j\}_{j=1}^{n}$, která obsahuje pouze vzájemně ortonormální vektory. [9]

Klasický ortogonalizační proces bude nyní popsán podrobněji na příkladu. Nechť existuje matice $A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ s lineárně nezávislými vektory $a_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$; $a_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ a $a_3 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$. Cílem je najít, vektory q_1 ; q_2 ; q_3, které patří do množiny navzájem ortonormálních vektorů $\{q_j\}_{j=1}^{k-1}$. Začíná se tím, že se vezme vektor a_1, ze kterého se vytvoří vektor q_1 dle vzorce $q_1 = \frac{a_1}{\|a_1\|}$. Pro aktuálně vybranou matici tento vektor vychází $q_1 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix}$. Pro vektor q_2 musí platit, že nesmí mít žádnou složku ve směru q_1, pokud tato složka existuje musí být od něj odečtena, vzorcem je to zapsáno jako $q'_2 = a_2 - (q_1^T \cdot a_2) \cdot q_1$. Výsledný vektor q'_2 je kolmý k q_1, avšak jeho délka není jednotková. Po aplikaci vzorce $q_2 = \frac{q'_2}{\|q'_2\|}$ jej již lze zařadit do množiny $\{q_j\}_{j=1}^{k-1}$. Právě popsaný vztah lze znázornit i graficky a je zaznamenán na následujícím obrázku.



Obrázek 7 – Tvorba vektoru q'_2

Pro zvolený příklad vychází vektor q_2 následovně $q_2 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix}$. Posledním hledaným vektorem z množiny $\{q_j\}_{j=1}^{n}$ je vektor q_3, který bude nalezen obdobně. Tentokrát je již nutné odečíst složky ve směru q_1 i q_2. Nejprve je nutné dopočítat vektor q'_3 podle následujícího vzorce $q'_3 = a_3 - (q_1^T \cdot a_3) \cdot q_1 - (q_2^T \cdot a_3) \cdot q_2$. Vzniklý vektor se znormuje na jednotkovou délku, čímž vzniká vektor $q_3 = \frac{q'_3}{\|q'_3\|}$ patřící do množiny ortonormálních vektorů. Pro aktuální příklad tedy $q_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$. Uvedeným způsobem byly získány vektory q_1 ; q_2 ; q_3, které jsou ortonormální[1]. Postup, kterým byly vektory vytvořeny se nazývá klasický Grammův-Schmidtův ortogonalizační proces. Pro tento proces bude není uveden algoritmus [10]

1. Inicializuj:
   - **a) Q := m × m nulová matice**
   - **b) R := m × n nulová matice**
   - **c) V := A (kopie matice A)**

2. Pro j = 0 do méně než min{m, n}:
   - **a) Pro k = 0 do méně než j:**
     - **R[k, j] := skalární součin q_k^T * v_j**
     - **v_j := v_j - R[k, j] * q_k**
   - **b) R[j, j] := $||v_j||_2$**
   - **c) Pokud R[j, j] < tolerance:**
     - **vyhoď chybu (nulový nebo lineárně závislý sloupec)**
   - **d) q_j:= v_j / R[j, j]**

3. Pokud m > n, doplň Q pro j = min{m, n} do méně než m:
   - **a) q_j := 1 + abs(i-j)**
   - **b) Pro k = 0 do méně než j:**
     - **q_j := q_j - (q_k^T * q_j) * q_k**
   - **c) q_j := q_j / $||q_j||_2$**

4. Pokud n > m, doplň R pro j = min{m, n} do méně než n:
   - **a) Pro k = 0 do min{m, n}:**
     - **R[k, j] := q_k^T * a_j**

Algoritmus 1 – QR dekompozice pomocí klasického Grammova-Schmidtova procesu [6] [7]

[1] Postup lze použít i když se nebude normovat délka na jednotkový vektor, potom budou vektory navzájem ortogonální.

Algoritmus klasického Grammova-Schmidtova procesu začíná inicializací nulových matic Q (rozměru $m \times m$), R (rozměru $m \times n$) a pomocné matice V, která je kopií původní matice A. Algoritmus pokračuje iterací přes sloupce j, kdy pro každý sloupec v_j se provádí ortogonalizace vůči předchozím ortonormálním vektorům q_k čímž se zároveň vyplňují prvky matice R. Jakmile jsou od vektoru v_j odečteny projekce je nutno tento vektor normalizovat na vektor q_j. Pokud při výpočtu normy nastane, že norma je příliš malá je vyhozena chyba. Pokud by se v tuto chvíli algoritmus nepřerušil, pokračoval by chybně. V tuto chvíli by algoritmus pro čtvercové matice skončil. Jelikož do této metody může přijít matice obdélníková, je nutno tuto skutečnost zahrnout. V případě, že matice A má více řádků než sloupců ($m > n$) je matice Q doplněna o vektor q_j. Naopak pokud má matice A více sloupců než řádků (tedy $n > m$) doplní se matice R o chybějící sloupce pomocí skalárních součinů původních sloupců matice A s vektory q_k.

Výpočet QR rozkladu pomocí klasického Grammova-Schmidtova algoritmu lze demonstrovat na následujícím příkladu. Nechť existuje matice $A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$ jejíž vektory jsou lineárně nezávislé. Podle postupu z předchozí části vychází $q_1 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix}$, $q_2 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix}$ a $q_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$. Matice Q je vytvořena právě z těchto vektorů a je rovna $Q = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{bmatrix}$. Matice R bude dopočítána vztahem $Q^T \cdot A = Q^T \cdot Q \cdot R = R$.

$$R = \begin{bmatrix} 1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 1/\sqrt{2} & 0 & -1/\sqrt{2} \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \sqrt{2} & 1/\sqrt{2} & \sqrt{2} \\ 0 & 1/\sqrt{2} & \sqrt{2} \\ 0 & 0 & 1 \end{bmatrix}.$$

Matici A lze vyjádřit pomocí QR rozkladu následovně $A = QR$:

$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{2} & 1/\sqrt{2} & \sqrt{2} \\ 0 & 1/\sqrt{2} & \sqrt{2} \\ 0 & 0 & 1 \end{bmatrix}$$

2.2 Modifikovaný Grammův-Schmidtův proces

Výpočet přes klasický Grammův-Schmidtův proces je číslicově nestabilní. Ve výpočtech QR rozkladu se v reálných případech objevují racionální a iracionální čísla, která je nutné vždy zaokrouhlit. Tímto vzniká chyba v nepřesnosti výsledné matice. Chybu lze částečně minimalizovat právě modifikovaným Grammovým-Schmidtovým procesem. [11]

Modifikovaný Grammův-Schmidtův proces je varianta klasického Grammova-Schmidtova procesu s drobnou úpravou, která je vidět v následujícím algoritmu.

1. **Inicializuj:**
   a) **Q := m × m nulová matice**
   b) **R := m × n nulová matice**
2. **Pro j = 0 do méně než min{m, n}:**
   a) **R[j, j] :=** $||v_j||_2$
   b) **Pro k = 0 do méně než j:**
      R[k, j] := $q_k^T * q_j$
      $q_j := q_j$ **- R[k, j] *** q_k
   c) **R[j, j] :=** $||q_j||_2$
   d) **Pokud R[j, j] < tolerance:**
      vyhoď chybu (nulový nebo lineárně závislý sloupec)
   c) $q_j :=$ **** q_j **/ R[j, j]**
3. **Pokud m > n, doplň Q pro j = min{m, n} do méně než m:**
   a) q_j **:= 1 + abs(i-j)**
   b) **Pro k = 0 do méně než j:**
      $q_j := q_j$ **-** $(q_k^T * q_j) * q_k$
   c) $q_j := q_j$ **/** $||q_j||_2$
4. **Pokud n > m, doplň R pro j = min{m, n} do méně než n:**
   a) **Pro k = 0 do min{m, n}:**
      R[k, j] := $q_k^T * a_j$

Algoritmus 2 – QR dekompozice pomocí modifikovaného Grammova-Schmidtova procesu [12] [7]

Algoritmus modifikované verze začíná inicializací nulových matic Q (velikosti $m \times m$) a R (velikosti $m \times n$). Pro každý sloupec j se vypočítá norma. Od tohoto sloupce se poté opakovaně odečítají projekce na předchozí ortonormální vektory q_k, aby byl získán ortogonální vektor q_j. Tento vektor se pak normalizuje. Pokud má norma q_j hodnotu menší než zvolená tolerance, algoritmus vyhodí chybu, protože je daný sloupec nulový nebo lineárně závislý. Pokud nastane možnost, že matice je typu $m > n$ je matice Q doplněna. Pokud se jedná o variantu $n > m$ doplní se matice R výpočtem skalárních součinů ortonormálních vektorů q_k.

Výpočet modifikovaného algoritmu bude opět demonstrován na matici $A = \begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$. Na vstupu jsou lineárně nezávislé vektory tedy $a_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$; $a_2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ a $a_3 = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}$. První krok zůstává stejný, je tedy nutné normalizovat vektor a_1 a tím vzniká q_1. Oproti klasické variantě je změna v tom, že se odečítá složka q_1 jakmile je to možné. Pro ostatní vektory musí platit, že neobsahují složku vektoru q_1, musí tedy platit $q'_2 = a_2 - (q_1^T \cdot a_2) \cdot q_1)$ a $q'_3 = a_3 - (q_1^T \cdot a_3) \cdot q_1)$. Pro zvolený příklad vychází $q_1 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix}$, $q'_2 = \begin{bmatrix} 1/2 \\ 0 \\ -1/2 \end{bmatrix}$ a $q'_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$. Vektor q_2' se normalizuje na $q_2 = \begin{bmatrix} 1/\sqrt{2} \\ 0 \\ -1/\sqrt{2} \end{bmatrix}$. Ostatní vektory opět nesmí obsahovat složku q_2. Platí tedy $q''_3 = q'_3 - (q_2^T \cdot q'_3) \cdot q_2$. V zadaném příkladě vychází $q''_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$, což je již normalizovaný vektor q_3. Z ortonormálních vektorů q_1 ; q_2 ; q_3 je vytvořena matice $Q = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{bmatrix}$. Matice R bude dopočítána stejně jako u předchozího způsobu, pomocí vztahu $Q^T \cdot A = Q^T \cdot Q \cdot R = R$

Matici A lze vyjádřit pomocí QR rozkladu následovně $A = QR$:

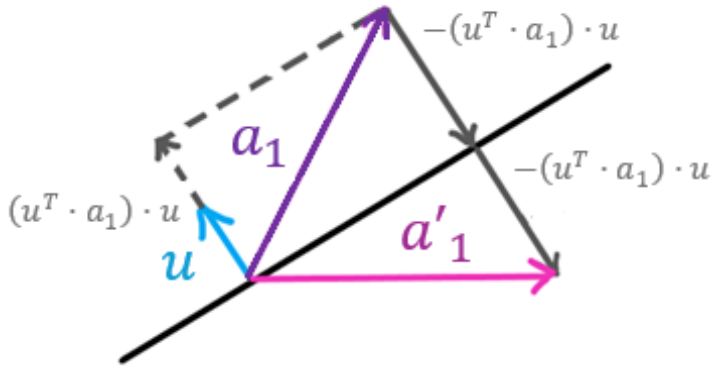
$$\begin{bmatrix} 1 & 1 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 0 & 0 & 1 \\ 1/\sqrt{2} & -1/\sqrt{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} \sqrt{2} & 1/\sqrt{2} & \sqrt{2} \\ 0 & 1/\sqrt{2} & \sqrt{2} \\ 0 & 0 & 1 \end{bmatrix}$$

2.3 Householderovy transformace

„Householderova transformace je lineární transformace, která zrcadlí vektor a_1 přes ortogonální doplněk u^T, pro některé specifikované u“ [9 str. 33]

Householderova transformace (Q_1) lze zapsat i maticově, pomocí vzorce $Q_1 = I - 2 \cdot u \cdot u^T$. Pro transformaci platí ${Q_1}^T \cdot Q_1 = I$, proto lze říct, že jsou Householderovy transformace ortonormální.

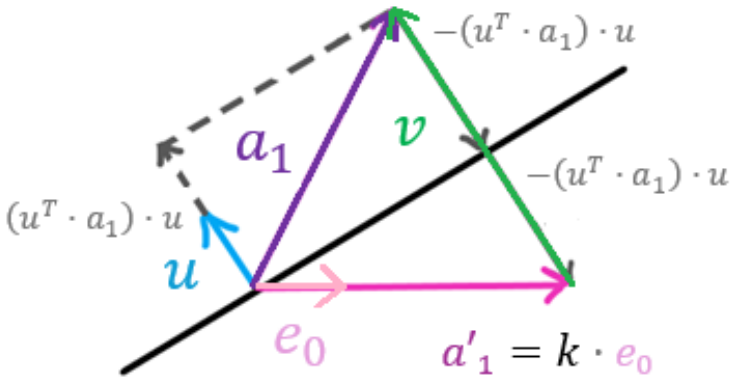
Pro demonstraci bude použit podprostor dimenze 1 (přímku) a vektor kolmý k tomuto podprostoru s délkou 1 (vektor u, který má jednotkovou velikost). Vektor u se vynásobí skalárem vektorů $u^T \cdot a_1$, tím vzniká projekce vektoru a_1 do vektoru u, nazvaná $u^T \cdot a_1 \cdot u$. Vynásobí-li se tento vektor (-1) vzniká vektor $-(u^T \cdot a_1 \cdot u)$, který je v opačném směru než vektor u. Pokud je tento vektor dán dvakrát za sebe, zjistí se koncový bod zrcadleného vektoru ${a'}_1$. Právě popsané vztahy jsou zaznamenány na následujícím obrázku.



Obrázek 8 – Householderova transformace – 1.část

Z popisu vyplývá, že vektor ${a'}_1$ je vyjádřen jako ${a'}_1 = a_1 - 2 \cdot u^T \cdot a_1 \cdot u$. Tento vztah lze přepsat pomocí matic jako ${a'}_1 = (I - 2 \cdot u \cdot u^T) \cdot a_1$. Vzorec $I - 2 \cdot u \cdot u^T$ se nazývá Householderova transformace.

Jak již ze vzorce vyplývá, tak pro výpočet je nutné znát normalizovaný vektor u. Pokud jsou známy vektory a_1 a a'_1 lze dopočítat vektor u pomocí vektoru $v = a_1 - a'_1$. Vektor v stačí pouze normovat a vychází $u = v/\|v\|$. Nyní tedy stačí vhodně zvolit a'_1, tak aby bylo možné dopočítat vektor u. Vektor a'_1 bude zvolen jako k-násobek jednotkového bázového vektoru (označen jako e_0). Vzorcem tuto skutečnost lze vyjádřit jako $a'_1 = k \cdot e_0$. Jelikož je $\|e_0\| = 1$, platí $|k| = \|a_1\|$. Tyto vztahy jsou zobrazeny na následujícím obrázku. [13]



Obrázek 9 – Householderova transformace – 2.část

Platí tedy $v = a_1 - a'_1 = a_1 - k \cdot e_0$. Tento vzorec lze rozepsat vektorově jako (využije se vztahu $|k| = \|a_1\|$) $v = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} - \begin{bmatrix} \pm\|a\| \\ 0 \\ \vdots \\ 0 \end{bmatrix}$. Znaménko $\pm$ se objevuje, protože v prostoru reálných čísel je možnost podprostor pro zrcadlení umístit dvěma způsoby. Z důvodu numerické stability se znaménko volí jako $-sign(a_{1,1})$. Pokud nastane případ $a_{1,1} = 0$, poté se volí kladné znaménko. Vzorec lze přepsat na $v = a_1 - (-sign(a_{1,1}) \cdot \|a_1\| \cdot e_0)$. Nyní z vektoru v bude vypočítán vektor u jako $u = v/\|v\|$. Jelikož je již možné spočítat vektor u, lze jej dosadit do vzorce pro Householderovu transformaci, čímž vyjde Householderova matice Q_1. [13]

Právě popsaná transformace lze využít pro výpočet QR rozkladu matice. U Grammova-Schmidtova procesu se pomocí série transformací uložených v matici R ortonormalizovala matice A. Při použití Householderovy transformace je to opačně, pomocí série ortonormálních transformací triangulizuje matici R. Algoritmus, kterým je toho docíleno je popsán na následující straně. [9]

1. **Inicializuj:**
   a) **Q := m × m jednotková matice**
   b) **R := A (kopie matice A)**
2. **Pro j = 0 do méně než min{m, n}:**
   a) **v := podvektor sloupce R[j:m, j]**
   b) **v[0] := v[0] −(-sign(R[j, j]) * ||v||₂)**
   c) **v := v / ||v||₂**
   d) **Pro k = j do méně než n:**
      w := 2 * (v^T * R[j:m, k])
      R[j:m, k] := R[j:m, k] − v * w
   e) **Pro i = 0 do méně než m**
      w := 2 * (Q[i, j:m] * v)
      Q[i, j:m] := (Q[i, j:m] − w * v^T

Algoritmus 3 – QR dekompozice pomocí Householderovy transformace [9], [13]

Algoritmus začíná inicializací matice Q, které je jednotková a rozměru $m \times m$ a matice R, která je kopií matice A. Pokračuje se iterací přes sloupce, kdy při každé iteraci je algoritmus zaměřen na vektor v. Tento vektor je tvořen z matice R od indexu j do m. Prvky vektoru v budou upraveny podle prvního prvku ve sloupci, znaménka prvku $R[j,j]$ a normy tohoto vektoru. Nově vzniklý vektor bude použit právě pro Householderovu transformaci, která je prováděna na maticích R a Q v bodech $2d$) a $2e$).

Při úpravě matice R je prováděna Householderova transformace sloupců i řádků od indexu j dále. Pro každý sloupec k se spočítá skalární násobek w, který odpovídá projekci sloupce R na vektor v. Tento skalár se poté použije pro aktualizaci sloupce v R odečtením části $v \times w$ čímž se vynulují prvky ve sloupci j. Stejná Householderova transformace se aplikuje i na matici Q zleva, tedy na její řádky. Pro každý řádek i se opět vypočítá skalár w což je projekce Q na vektor v. Segment v matici Q se upraví odečtením $w \times v^T$. Touto úpravou se matice Q udržuje ortogonální.

Při aktualizacích matic Q a R lze konstruovat elementární Householderovy matice Q_i podle popsaného způsobu. Tento přístup ale vyžaduje větší výpočetní náročnost než přímá aplikace na matice R a Q. Po ukončení všech operací je matice Q ortogonální s ortonormálními sloupci a matice R je horní trojúhelníková.

Postup konstrukce QR rozkladu pomocí elementárních Householderových matic bude popsán na následujícím příkladu. Obecně lze konstatovat, že matici R lze získat pomocí roznásobení Householderových matic, tedy podle vztahu $Q_n \cdot Q_{n-1} \cdot \ldots \cdot Q_1 \cdot A = R$. Matice Q je poté získána pomocí vzorce $\prod_{i=1}^{n} {Q_i}^T = Q$ kde pro jednotlivé elementární Householderovy matice Q_i platí vztah $Q_{i+1} = \begin{bmatrix} 1 & 0 \\ 0 & Q_i \end{bmatrix}$, který je použit při výpočtech.

Postup výpočtu QR rozkladu bude demonstrován matici $A = \begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix}$. Kde bude zvolen vektor a_1 jako $a_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$ a k němu příslušný jednotkový vektor $e_0 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$. Pomocí těchto dvou vektorů se dopočítá vektor v podle vzorce $v = a_1 - (-sign(a_{1,1}) \cdot \|a_1\| \cdot e_0)$. Pro zvolený příklad vychází $v = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \left(-sign(1) \cdot \sqrt{1^2 + 1^2 + 1^2 + 1^2} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 3 \\ 1 \\ 1 \\ 1 \end{bmatrix}$. Normalizací v vzniká $u = \begin{bmatrix} 3/\sqrt{12} \\ 1/\sqrt{12} \\ 1/\sqrt{12} \\ 1/\sqrt{12} \end{bmatrix}$.

Vektor u bude použit pro výpočet Householderovy matice pomocí vzorce $Q_1 = I - 2 \cdot u \cdot u^T$. Matice vychází následovně:

$$Q_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} - 2 \cdot \begin{bmatrix} 3/\sqrt{12} \\ 1/\sqrt{12} \\ 1/\sqrt{12} \\ 1/\sqrt{12} \end{bmatrix} \cdot \begin{bmatrix} 3/\sqrt{12} & 1/\sqrt{12} & 1/\sqrt{12} & 1/\sqrt{12} \end{bmatrix} = \begin{bmatrix} -1/2 & -1/2 & -1/2 & -1/2 \\ -1/2 & 5/6 & -1/6 & -1/6 \\ -1/2 & -1/6 & 5/6 & -1/6 \\ -1/2 & -1/6 & -1/6 & 5/6 \end{bmatrix}$$

Matice v horním trojúhelníkovém tvaru postupně vzniká pomocí vzorce $A' = Q_1 \cdot A$.

$$A' = \begin{bmatrix} -1/2 & -1/2 & -1/2 & -1/2 \\ -1/2 & 5/6 & -1/6 & -1/6 \\ -1/2 & -1/6 & 5/6 & -1/6 \\ -1/2 & -1/6 & -1/6 & 5/6 \end{bmatrix} \cdot \begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} -2 & -3 & -2 \\ 0 & 10/3 & -4 \\ 0 & 10/3 & 0 \\ 0 & -5/3 & -2 \end{bmatrix}$$

Takto vypočítaná matice bude nyní zredukována na matici 3 x 2 tak, že se odstraní první řádek i sloupec. Matice A' se tedy aktualizuje pro další iteraci na $A' = \begin{bmatrix} 10/3 & -4 \\ 10/3 & 0 \\ -5/3 & -2 \end{bmatrix}$. Způsob výpočtu se nyní opakuje, zvolí se $a'_1 = \begin{bmatrix} 10/3 \\ 10/3 \\ -5/3 \end{bmatrix}$ a $e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$. Nyní se dopočítá potřebný vektor v' pomocí vzorce

$v' = a'_1 - (-sign(a'_{1,1}) \cdot \|a'_1\| \cdot e_1)$, ten vychází jako $v' = \begin{bmatrix} 25/3 \\ 10/3 \\ -5/3 \end{bmatrix}$, tento vektor se normuje na $u' = \begin{bmatrix} 5/\sqrt{30} \\ 2/\sqrt{30} \\ -1/\sqrt{30} \end{bmatrix}$. Householderova matice Q_2 vychází $Q_2 = \begin{bmatrix} -2/3 & -2/3 & 1/3 \\ -2/3 & 11/15 & 2/15 \\ 1/3 & 2/15 & 14/15 \end{bmatrix}$. Další část horní trojúhelníkové matice je dopočítána podle vzorce $A'' = Q_2 \cdot A'$, kde A'' je matice doplněna na jednotkovou matici.

$$A'' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -2/3 & -2/3 & 1/3 \\ 0 & -2/3 & 11/15 & 2/15 \\ 0 & 1/3 & 2/15 & 14/15 \end{bmatrix} \cdot \begin{bmatrix} -2 & -3 & -2 \\ 0 & 10/3 & -4 \\ 0 & 10/3 & 0 \\ 0 & -5/3 & -2 \end{bmatrix} = \begin{bmatrix} -2 & -3 & -2 \\ 0 & -5 & 2 \\ 0 & 0 & 12/5 \\ 0 & 0 & -16/5 \end{bmatrix}$$

Matice A“ bude zredukována na matici 2×1, tedy na $A'' = \begin{bmatrix} 12/5 \\ -16/5 \end{bmatrix}$. Definují se potřebné vektory $a''_1 = \begin{bmatrix} 12/5 \\ -16/5 \end{bmatrix}$ a $e_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ a použijí se na výpočet vektoru $v'' = a''_1 - (-sign(a''_{1,1}) \cdot \|a''_1\| \cdot e_2)$. Ze vzorce vyplývá, že pro zvolený příklad vyjde vektor v“ následovně $v'' = \begin{bmatrix} 32/5 \\ -16/5 \end{bmatrix}$. Normováním vektoru v“ vychází $u'' = \begin{bmatrix} 2/\sqrt{5} \\ -1/\sqrt{5} \end{bmatrix}$. Elementární Householderova matice Q_3 vychází $Q_3 = \begin{bmatrix} -3/5 & 4/5 \\ 4/5 & 3/5 \end{bmatrix}$. V tuto chvíli je nutné aktualizovat matici R vzorcem $A''' = Q_3 \cdot A''$.

$$A''' = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3/5 & 4/5 \\ 0 & 0 & 4/5 & 3/5 \end{bmatrix} \cdot \begin{bmatrix} -2 & -3 & -2 \\ 0 & -5 & 2 \\ 0 & 0 & 12/5 \\ 0 & 0 & -16/5 \end{bmatrix} = \begin{bmatrix} -2 & -3 & -2 \\ 0 & -5 & 2 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

Proces je nyní ukončen, protože iterace proběhla přes veškeré sloupce a byla získána horní trojúhelníková matice. Matice Q je získána součinem transponovaných Householderových matic, tedy $Q = {Q_1}^T \cdot {Q_2}^T \cdot \ldots \cdot {Q_n}^T$.

$$\begin{bmatrix} -1/2 & -1/2 & -1/2 & -1/2 \\ -1/2 & 5/6 & -1/6 & -1/6 \\ -1/2 & -1/6 & 5/6 & -1/6 \\ -1/2 & -1/6 & -1/6 & 5/6 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -2/3 & -2/3 & 1/3 \\ 0 & -2/3 & 11/15 & 2/15 \\ 0 & 1/3 & 2/15 & 14/15 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -3/5 & 4/5 \\ 0 & 0 & 4/5 & 3/5 \end{bmatrix} = \begin{bmatrix} -1/2 & 1/2 & -1/2 & -1/2 \\ -1/2 & -1/2 & 1/2 & -1/2 \\ -1/2 & -1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & 1/2 & 1/2 \end{bmatrix}$$

Tímto byl zjištěn výsledný QR rozklad pro zadaný příklad:

$$\begin{bmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{bmatrix} = \begin{bmatrix} -1/2 & 1/2 & -1/2 & -1/2 \\ -1/2 & -1/2 & 1/2 & -1/2 \\ -1/2 & -1/2 & -1/2 & 1/2 \\ -1/2 & 1/2 & 1/2 & 1/2 \end{bmatrix} \cdot \begin{bmatrix} -2 & -3 & -2 \\ 0 & -5 & 2 \\ 0 & 0 & -4 \\ 0 & 0 & 0 \end{bmatrix}$$

2.4 Givensovy rotace

„Obecná Givensova matice G(i, j, Θ) reprezentuje ortonormální transformaci, která rotuje rovinu určenou vektory e_i a e_j o Θ radiánů“ [9 str. 38]

Matice je ve tvaru $G(\mathrm{i,j,\Theta}) = \begin{bmatrix} I & 0 & 0 & 0 & 0 \\ 0 & c & 0 & s & 0 \\ 0 & 0 & I & 0 & 0 \\ 0 & -s & 0 & c & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}$. Jedná se o jednotkovou matici, kromě prvků na místech $g_{ii} = c;\ g_{ij} = s;\ g_{ji} = -s;\ g_{jj} = c.$

Právě zmíněná Givensova rotace lze použít pro výpočet QR rozkladu matice. Způsob, pomocí kterého je postupně tvořena matice R je popsán v následující části.

Pro matici $A(a_1, a_2, \dots\dots, a_n)$ musí platit, že při použití rotace G_1 bude $\|a_1\| = \|a'_1\|$, aby vznikla pod diagonálou nula. Tento vztah lze vyjádřit:

$$a_1 = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \\ \vdots \\ a_{n1} \end{bmatrix} \rightarrow G_1 \rightarrow a'_1 = \begin{bmatrix} \sqrt{a^2{}_{11} + a^2{}_{21}} \\ 0 \\ a_{31} \\ \vdots \\ a_{n1} \end{bmatrix}$$

Pro tuto transformaci bude volena matice $G_1 = \begin{bmatrix} c & s & & & \\ -s & c & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$. Ze vzorce $G_1 \cdot a_1 = a'_1$ vyplývá $c = \frac{a_{11}}{\sqrt{a^2{}_{11}+a^2{}_{21}}}$ a $s = \frac{a_{21}}{\sqrt{a^2{}_{11}+a^2{}_{21}}}$. Při použití jedné Givensovy rotace se vytvoří pouze jedna nula. Pro vynulování celého sloupce je nutno $(n-1)$ Givensových rotací. Pro transformaci matice na trojúhelníkový tvar bude zapotřebí $\sum_{k=1}^{n-1} k$ Givensových rotací. [14]

Algoritmus, který je aplikován na matici A a využívá Givensovy rotace při tvorbě QR rozkladu je zmíněn na další straně. Tento algoritmus opět triangulizuje matici R čímž je podobný Householderově transformaci

```
1. Inicializuj:
     a) Q := m ×m jednotková matice
     b) R := A
2. Pro j = 0 do méně než n:
     a) Pro i = j + 1 do méně než m:
          a := R[j, j]
          b := R[i, j]
          Pokud sqrt(a² + b²) = 0, pokračuj na další i
          c := a / sqrt(a² + b²)
          s := b / sqrt(a² + b²)
          Pro k = j do méně než n:
               t1 := R[j, k]; t2 := R[i, k]
               R[j, k] := c * t1 + s * t2
               R[i, k] := -s * t1 + c * t2
          Pro k = 0 do méně než m
               t1 := Q[r, j]; t2 := Q[r, i]
               Q[r, j] := c * t1 + s * t2
               Q[r, i] := -s * t1 + c * t2
```

Algoritmus 4 – QR dekompozice pomocí Givensovy rotace [9]

Algoritmus pro QR rozklad začíná inicializací jednotkové matice Q o velikosti $m \times m$ a matice R, která je rovna vstupní matici A. Algoritmus prochází sloupce matice a pro každý z nich se aplikuje rotace pro dvojici řádků j a i kde $i > j$ čímž se vynuluje prvek $R[i,j]$. Koeficienty Givensovy rotace c a s ((kosinus a sinus otočení) se spočítají na základě hodnot $a = R[j,j]$ a $b = R[i,j]$. Prvek $R[i,j]$ bude po rotaci roven nule.

Vypočítané koeficienty budou využity k aktualizaci matice R a Q. Alternativní způsob je sestavit matici G a tu aplikovat na matice Q a R. Tyto matice provádí otočení v dané rovině a mají za následek, že prvek r_{ij} bude vynulován. Matice G se aplikuje zleva na R a transponovaně zprava na Q. Pro získání QR rozkladu budou v příkladu využity následující vzorce $G^T{}_n \cdot G^T{}_{n-1} \cdot \ldots \ldots \cdot G^T{}_1 \cdot A = R$ a pomocí součinu $A = G_1 \cdot G_2 \cdot \ldots \ldots \cdot G_n \cdot R$ bude dopočítáno Q, tedy $\prod_{k=1}^{n} G_n = Q$. [7]

Postup výpočtu QR rozkladu bude demonstrován na následující matici $A = \begin{bmatrix} 0 & -1 & 1 \\ 4 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix}$.

Nejprve je nutné dopočítat hodnoty $c = \frac{a_{11}}{\sqrt{a^2{}_{11}+a^2{}_{21}}}$ a $s = \frac{a_{21}}{\sqrt{a^2{}_{11}+a^2{}_{21}}}$. Pro zvolený příklad vycházejí $c = 0$ a $s = 1$. Givensova matice $G^T{}_1$bude následující $G^T{}_1 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. Tato matice nyní bude použita v součinu $G^T{}_1 \cdot A = A'$. Matice A' vychází jako $\begin{bmatrix} 4 & 2 & 0 \\ 0 & -1 & 1 \\ 3 & 4 & 0 \end{bmatrix}$.

Pro Givensovu matici $G^T{}_2$ bude nutné opět dopočítat $c = \frac{a_{11}}{\sqrt{a^2{}_{11}+a^2{}_{31}}}$ a $s = \frac{a_{31}}{\sqrt{a^2{}_{11}+a^2{}_{31}}}$, které vychází následně $c = \frac{4}{5}$ a $s = \frac{3}{5}$. Matice $G^T{}_2$ bude rovna $\begin{bmatrix} 4/5 & 0 & 3/5 \\ 0 & 1 & 0 \\ -3/5 & 0 & 4/5 \end{bmatrix}$. Nyní se opět dopočítá matice $A'' = G^T{}_2 \cdot A'$, která vyjde $\begin{bmatrix} 5 & 4 & 0 \\ 0 & 1 & -1 \\ 0 & 2 & 0 \end{bmatrix}$. Vyčíslí se hodnota $c = \frac{a_{22}}{\sqrt{a^2{}_{22}+a^2{}_{32}}}$ a také hodnota $s = \frac{a_{32}}{\sqrt{a^2{}_{22}+a^2{}_{32}}}$, které pro zvolený příklad vychází $c = \frac{1}{\sqrt{5}}$ a $s = \frac{2}{\sqrt{5}}$. Givensova rotační matice vyjde následující $G^T{}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{5} & 2/\sqrt{5} \\ 0 & -2/\sqrt{5} & 1/\sqrt{5} \end{bmatrix}$. Posledním krokem je dopočítání matice $A''' = G^T{}_3 \cdot A''$, která je již v horním trojúhelníkovém tvaru, proto ji lze označit jako R. Tato matice R bude rovna $\begin{bmatrix} 5 & 4 & 0 \\ 0 & 5/\sqrt{5} & -1/\sqrt{5} \\ 0 & 0 & 2/\sqrt{5} \end{bmatrix}$. Jelikož je známa matice R, lze dopočítat dle vzorce matici Q nutnou pro QR rozklad a to vzorcem $\prod_{k=1}^{n} G_n = Q$. V příkladě vychází takto:

$$Q = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 4/5 & 0 & -3/5 \\ 0 & 1 & 0 \\ 3/5 & 0 & 4/5 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1/\sqrt{5} & -2/\sqrt{5} \\ 0 & 2/\sqrt{5} & 1/\sqrt{5} \end{bmatrix} = \begin{bmatrix} 0 & -\sqrt{5}/5 & \left(2\sqrt{5}\right)/5 \\ 4/5 & (-6\sqrt{5})/25 & \left(-3\sqrt{5}\right)/25 \\ 3/5 & \left(8\sqrt{5}\right)/25 & \left(4\sqrt{5}\right)/25 \end{bmatrix}.$$

Celkový QR rozklad matice A lze vyjádřit následovně:

$$\begin{bmatrix} 0 & -1 & 1 \\ 4 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\sqrt{5}/5 & (2\sqrt{5})/5 \\ 4/5 & (-6\sqrt{5})/25 & (-3\sqrt{5})/25 \\ 3/5 & (8\sqrt{5})/25 & (4\sqrt{5})/25 \end{bmatrix} \cdot \begin{bmatrix} 5 & 4 & 0 \\ 0 & 5/\sqrt{5} & -1/\sqrt{5} \\ 0 & 0 & 2/\sqrt{5} \end{bmatrix}$$

3 Tvorba aplikace

Cílem praktické části práce je vytvořit aplikaci umožňující provádět QR rozklad pomocí jednotlivých metod QR rozkladu v jednotlivých formátech matic a porovnat výsledky časové a paměťové složitosti aplikace. Při programování byla zvolena konzolová aplikace v C++. Před samotným programováním aplikace bylo nutné určit funkční a nefunkční požadavky. Jakmile programování započalo vznikala souběžně i programátorská příručka, která je uvedena v práci. Pro přívětivé ovládání aplikace byla vytvořena uživatelská příručka, podle které si může čtenář aplikaci vyzkoušet. Během programování aplikace byly vytvořeny i unit testy pro ověření správnosti QR rozkladu jako separátní projekt. Na závěr bylo nutné porovnat časovou a paměťovou složitost aplikace. Toho bylo docíleno formou asymptotické složitosti jednotlivých metod a konkrétním měřením na maticích. Právě popsané části budou nyní podrobněji popsány.

3.1 Funkční a nefunkční požadavky aplikace

Při tvorbě aplikace je důležité zjistit kladené požadavky na danou aplikaci, tyto požadavky jsou rozděleny na funkční (popisující funkčnost aplikace) a nefunkční (popisující, jak má aplikace fungovat). Tyto požadavky byly zaznamenány do následující tabulky.

Tabulka 3 – Funkční a nefunkční požadavky na aplikaci

FUNKČNÍ POŽADAVKY	NEFUNKČNÍ POŽADAVKY
Aplikace bude umožňovat zadání vlastní matice	Aplikace bude využívat jazyk C++
Aplikace bude umožňovat načíst matici ze souboru	Aplikace bude mít rozhraní typu CLI
Aplikace bude umožňovat uložit matici ve zvoleném formátu	Aplikace bude testována pomocí Google Tests
Aplikace bude umožňovat provést QR rozklad pomocí zvolené metody	Aplikace bude mít programátorskou příručku
Aplikace bude umožňovat uložení výsledků	Aplikace bude mít uživatelskou příručku
Aplikace bude zobrazovat výsledek výpočtu	Aplikace bude interaktivní
Aplikace bude zobrazovat odchylku výpočtů	
Aplikace bude měřit čas a paměť výpočtu	

3.2 Uživatelská příručka aplikace

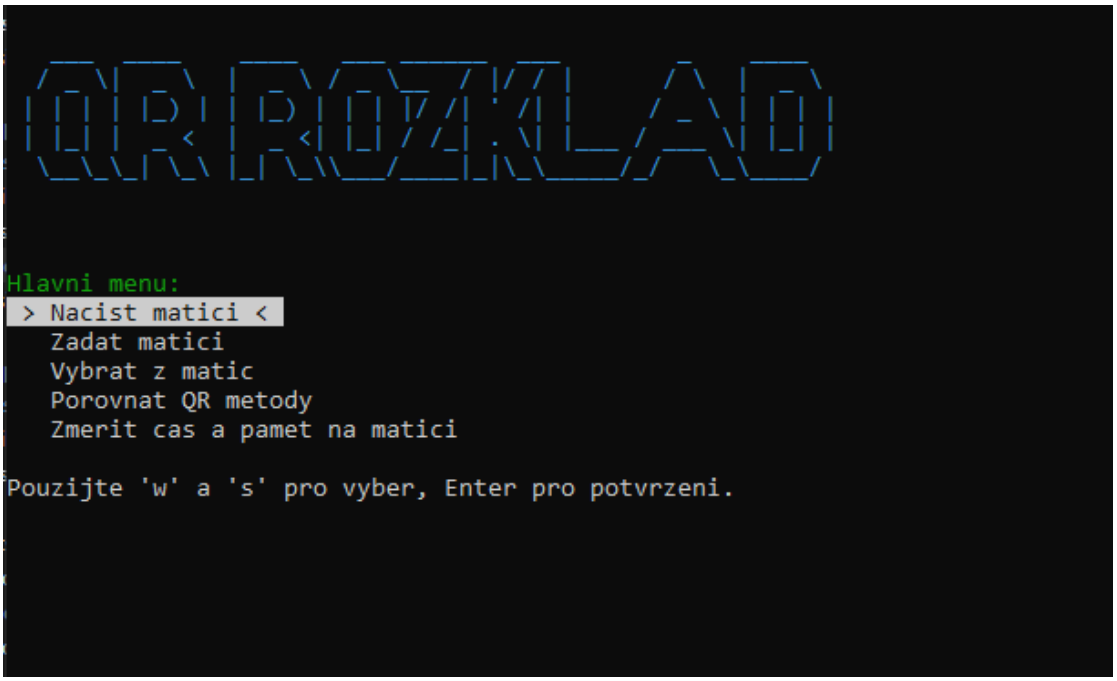
Aplikaci lze spustit dvěma způsoby a normálním způsobem, případně s argumenty příkazového řádku. Při spuštění s argumenty umožňuje vyhodnocení časové a paměťové náročnosti zadaného souboru. Jaké argumenty a v jakém pořadí je nutné je zadat je uvedeno na následujícím obrázku.

Command	$(TargetPath)
Command Arguments	**matice100 CSR Givens**
Working Directory	$(ProjectDir)
Attach	No
Debugger Type	Auto
Environment	
Merge Environment	Yes
SQL Debugging	No
Amp Default Accelerator	WARP software accelerator

Obrázek 10 – Argumenty příkazového řádku

Prvním argumentem je název souboru matice, tato matice musí být v požadovaném formátu umístěna do složky *Input* v projektu *QR_Decomposition_Runner*. Druhým argumentem je formát matice a třetím je metoda QR rozkladu. Tento způsob spouštění bude použit pro vyhodnocení paměťové a časové složitosti.

Uživatel pravděpodobně bude chtít spustit interaktivní aplikaci. Toho docílí pouhým kliknutím na tlačítko *Play* ve *Visual Studiu*. Po spuštění je zobrazeno hlavní menu jako na obrázku.



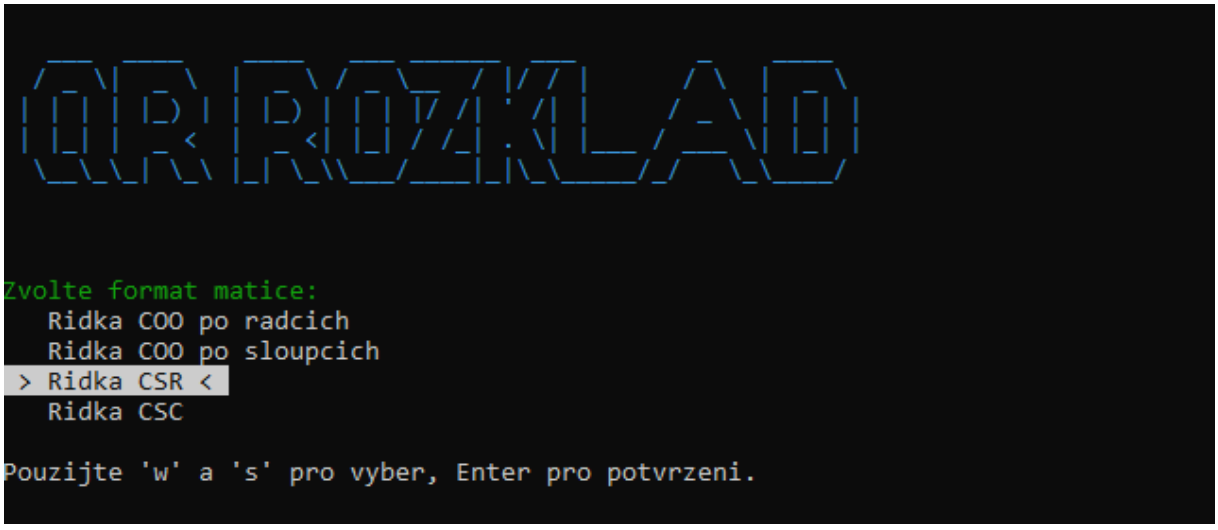
Obrázek 11 – Hlavní menu aplikace QR rozkladu

Uživatel nyní může zvolit pomocí kláves W a S požadovanou možnost a pomocí klávesy Enter ji potvrdit. V aplikaci jsou i již předdefinované matice, které lze pouze otestovat na vybrané metodě. Pokud bude chtít uživatel načíst matici je nutné ji umístit do složky $Input$ v daném projektu. Soubor matice musí být v testovém soboru ve formátu zobrazeným na následujícím obrázku.

```
4 4 3 9 3
4 3 2 5 2
4 2 8 2 6
```

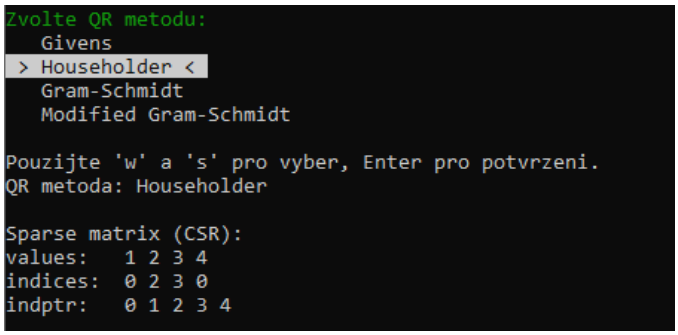
Obrázek 12 – Formát matic pro zadání do aplikace

Jakmile je vybrána matice, na které bude QR rozklad testován vyběhne nabídka, ve které uživatel zvolí, zda chce při výpočtech eliminovat malé hodnoty, toto nastavení je důležité hlavně u řídkých matic, protože ovlivňuje počet uložených prvků během výpočtů. Dalším důležitým krokem je volba formátu uložení matice, podporované formáty jsou zobrazeny na následujícím obrázku.



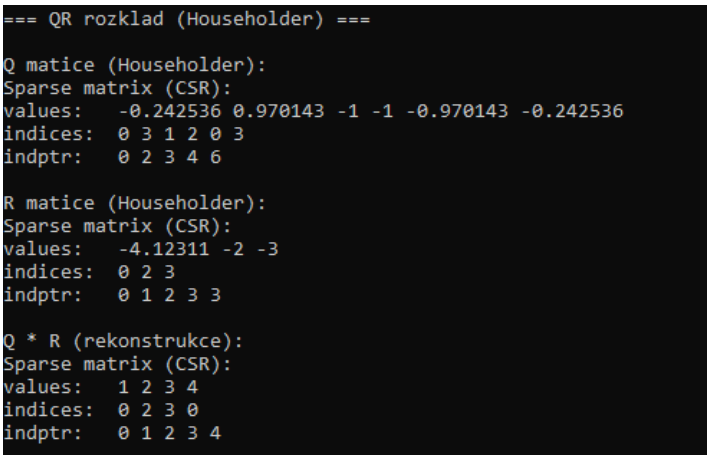
Obrázek 13 – Podporované formáty uložení

Po volbě formátu matice je nutné zvolit metodu výpočtu QR rozkladu, na výběr jsou metody Grammova-Schmidtova metoda v klasické a modelované verzi, Householderova transformace a Givensova rotace. Jakmile uživatel zvolí metodu je nejprve zobrazena zvolená matice ve zvoleném formátu.



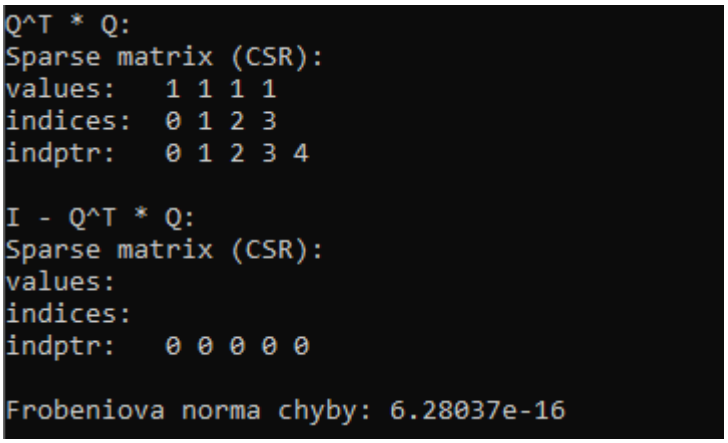
Obrázek 15 – Podporované metody QR rozkladu

Poté je uveden QR rozklad matice vypočítaný podle zvolené metody QR rozkladu a uveden ve stejném formátu jako zvolená matice. Správnost výpočtu je ověřena roznásobením matice Q a R a uvedením daného výsledku, který se shoduje s původní maticí.



Obrázek 14 – Zobrazení výsledků QR rozkladu

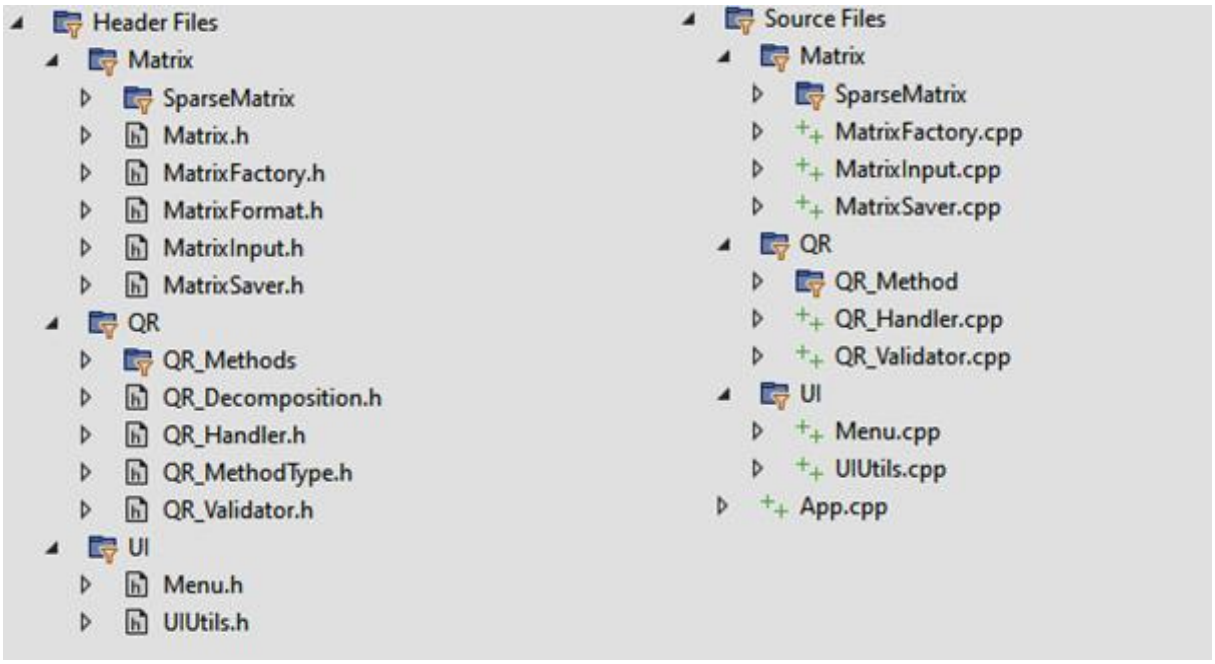
Pro určení spolehlivosti výpočtu byla zavedena kontrola ortogonality ve formě $Q \cdot Q^T = I$. Vychází-li jednotková matice je výpočet správný. Pro ověření přesnosti výpočtu je uvedena i Frobeniova norma chyby ortogonality matice Q, která je počítána z nulové matice $(I - Q \cdot Q^T)$ a to jako odmocnina součtu čtverců všech jejich prvků.



Obrázek 16 – Ověření výsledků výpočtu

3.3 Programátorská příručka aplikace

Předtím, než bude popsána časová a prostorová složitost jednotlivých QR algoritmu je vhodné se seznámit se strukturou projektu. Projekt je uspořádán do tří filtrů s názvy $Matrix$, QR a UI. Jako je vidět na obrázku níže. Většina ze souborů obsahuje i hlavičkové soubory, které jsou strukturovány do stejné hierarchie. V této podkapitole bude nyní uveden stručný popis jednotlivých částí projektu.



Obrázek 17 – Struktura projektu

Jednotlivé filtry budou nyní rozebrány, jako jeden z nejdůležitějších filtrů je filtr $Matrix$ ve kterém se nachází podfiltr $SparseMatrix$, ten obsahuje třídy jednotlivých formátů matic. Déle je ve filtru $Matrix$ třída $MatrixFactory$ jejíž úkolem je vytvářet jednotlivé matice. $MatrixInput$ třída zajišťuje zadávání a výběr matic pro výpočet a $MatrixSaver$ je odpovědný za ukládání matic do $.txt$. Dále je v projektu filtr QR ve kterém je podfiltr $QR\_Methods$ obsahující jednotlivé algoritmy QR rozkladu. Kromě toho se ve filtru QR nachází třída $QR\_Handler$, která je zodpovědná za volbu metody QR rozkladu a zpracování výsledků. Poslední třídou ve filtru QR je $QR\_Validator$, jehož úkolem je ověřit, zda jsou výsledky správné, jeho využití je převážně v testech. Posledním filtrem projektu je UI ve kterém je třída $Menu$ zajišťující výběrové menu a třída $UIUtils$, která je zodpovědná za výpis do konzole. Základním pilířem projektu je $main$, který se nachází ve třídě App.cpp.

3.3.1 Filtr *Matrix*

Tento filtr obsahuje třídy týkající se formátů řídkých matic ty jsou rozděleny do podfiltrů. V této části budou zde popsány hlavičkové a zdrojové soubory současně.

- **Hlavičkový soubor *Matrix.h***

  Jedná se o abstraktní základní třídu, která definuje rozhraní pro práci s maticemi libovolného formátu. Uchovává počet řádků a sloupců a poskytuje virtuální metody pro přístup k jednotlivým prvkům, nastavení hodnot, transpozici, násobení matic, výpis dat a klonování. Zahrnuje také metody pro aplikaci Givensových a Householderových transformací, získání prvků z řádku nebo sloupce a vytvoření Householderova vektoru. Tato třída obsahuje statickou hodnotu epsilon udávající toleranci při výpočtech.

- **Hlavičkový soubor *MatrixFormat.h***

  Tento hlavičkový soubor obsahuje výčtová typ, který specifikuje konkrétní způsob uložení dat v paměti. Jedná se o řídké formáty jako COO, CSR a CSC. Tento výčtový typ slouží k jednotné identifikaci struktury a formátu matice v programu.

- **Podfiltr *SparseMatrix***

  Tento podfiltr obsahuje třídy reprezentující jednotlivé formáty matic, konkrétně se jedná o třídy SparseMatrixCOOColumnMajor, SparseMatrixCOORowMajor, SparseMatrixCSC a SparseMatrixCSR. V jednotlivých třídách jsou vždy metody umožňující čtení, klonování, transpozici, násobení a aplikace Givensových rotací a Householderových transformací. Ke každé z těchto tříd existuje i hlavičkový soubor. Stejně tak existuje i hlavičkový soubor pro *SparseMatrix*.

- ***MatrixInput.h* a *MatrixInput.cpp***

  MatrixInput poskytuje statické metody pro načítání matic. Uživatel má možnost získat matici několika způsoby. Jedná se o výběr předdefinované matice, ruční zadání matice, načtení matice ze souboru, porovnání QR metod na matici a změření času a paměti na matici. Pokud je zvolena metoda načtení ze souboru je nutné umístit danou matici v požadovaném formátu do složky *Input*. Požadovaným formátem je textový soubor, který má řádky matice separátně na řádcích a sloupce jsou oddělené mezerou. Takováto matice je poté zpracována.

- ***MatrixFactory.h* a *MatrixFactory.cpp***

  Tato třída slouží pro vytváření jednotlivých matic v závislosti na specifikovaném formátu. Obsahuje metodu *createAndFill*, která vytvoří matici v určitém formátu a naplní ji prvky. Kromě této metody je zde metoda *createIdentityMatrix* sloužící pro vytvoření jednotkové matice daných rozměrů se zadaným formátem. Poslední metodou je createZeroMatrix, jenž vytváří nulovou matici.

- ***MatrixSaver.h* a *MatrixSaver.cpp***

  Třída *MatrixSaver* je určená pro ukládání matic do souboru. Hlavním účelem této třídy je zajištění interakce s uživatelem týkající se uložení matic do textových souborů. Cílem metody *askToSaveInputMatrix* je zjistit, zda uživatel chce zadanou matici uložit. Pokud ano bude tato matice uložena do složky Input a následně je ji možné načíst. Podobný cíl má i metoda *askToSaveMatrices* jejíž cílem je zjistit, zda uživatel má zájem o uložený výsledku QR rozkladu do souboru. Tyto soubory budou uloženy do složky *Output*. Samotné ukládání jednotlivých matic je realizováno metodami *saveInputMatrixToFile* a *saveMatrixToFile*, které zapisují odpovídající matici do souboru.

3.3.2 Filtr QR

Tento filtr obsahuje třídy související s výpočtem QR rozkladu a jeho validace. Filtr se opět nachází jak u hlavičkových souborů, tak u zdrojových. Tyto soubory budou popsány současně.

- **Hlavičkový soubor $QR\_Decomposition.h$**

  Tato abstraktní třída slouží pro reprezentaci QR rozkladu matice. Obsahuje tři metody čistě virtuální metody $compute$, $getQ$ a $getR$. Tato třída je využita jako rozhraní pro různé konkrétní implementace QR rozkladu.

- **Hlavičkový soubor $QR\_MethodType.h$**

  Tento soubor obsahuje výčtový typ, který má čtyři hodnoty: Givens, Householder, GramSchmidt a ModifiedGramSchmidt

- **$QR\_Handler.h$ a $QR\_Handler.cpp$**

  $QR\_Handler$ slouží jako statická třída pro zpracování QR rozkladu matice. Obsahuje metodu $performQR$, která vybírá patřičný algoritmus QR rozkladu na základě výčtového typu $QR\_MethodTyp$, metodu $handleQRResult$, která zpracovává výsledky QR rozkladu, konkrétně vypisuje jednotlivé matice a určuje chybu výpočtu. Kromě těchto metod jsou zde i metody pro porovnání časové a paměťové složitosti.

- **Podfiltr $QR\_Methods$**

  Tento filtr obsahuje jednotlivé metody QR rozkladu. Jedná se o třídy QR_GramSchmidt, QR_ModifiedGramSchmidt, QR_Householder, QR_Givens. Každá z těchto tříd obsahuje i hlavičkový soubor.

- **$QR\_Validator.h$ a $QR\_Validator.cpp$**

  Třída $QR\_Validator$ obsahuje metody pro ověřování vlastností a správnosti QR rozkladu matice. Metoda $reconstructProduct$ slouží k rekonstrukci původní matice pomocí součinu matic Q a R. Metoda $computeQtQ$ vypočítá součin $Q^{\mathrm{T}} \cdot Q$, který by měl v ideálním případě odpovídat jednotkové matici. Metoda $computeOrthogonalityError$ vrací matici $I - Q^{\mathrm{T}} \cdot Q$ a zároveň vypočítá Frobeniovu normu této chyby. Funkce $isUpperTriangular$ kontroluje, zda je matice R horní trojúhelníková v rámci zadané tolerance. Poslední metoda, $hasDependentColumns$, ověřuje, zda má matice A lineárně závislé sloupce, opět na základě dané tolerance.

3.3.3 Filtr *UI*

Ve filtru *UI* jsou veškeré třídy týkající se uživatelského rozhraní. Tento filtr je opět jak v hlavičkových souborech, tak ve zdrojových proto bude popsán současně.

- ***UIUtils.h* a *UIUtils.cpp***

  Třída *UIUtils* slouží pro vizuální prezentaci matic pro uživatelského rozhraní. Obsahuje definice barev, metodu pro vyčištění konzolové aplikace a vykreslení záhlaví. Dále obsahuje metodu pro zarovnání textu na střed, formátování desetinných čísel bez zbytečných nul a výpočet šířky sloupců pro konzistentní zobrazení matice. Kromě toho je zde metoda *printMatrixPreview* sloužící k náhledu matice při výběru. Dále je zde metoda *printMatrixInputInstructions*, která uživateli ukazuje, jakým způsobem má správně zadat matici pro třídu *MatrixInput*.

- ***Menu.h* a *Menu.cpp***

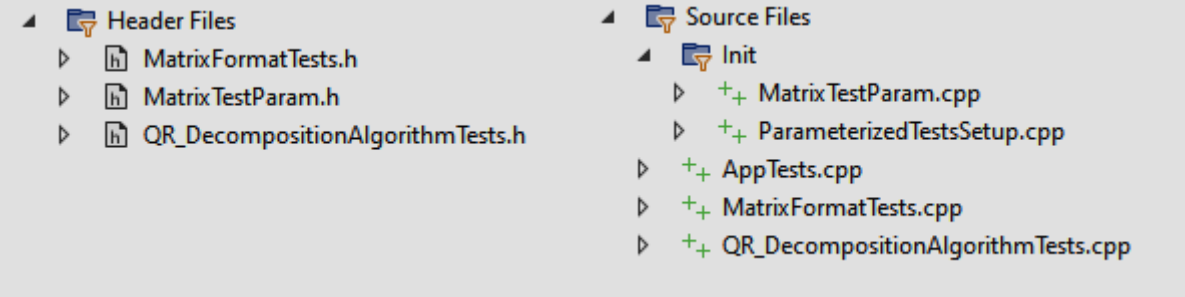
  Tato třída obsahuje metody pro zobrazování výběrových nabídek v aplikaci. Metoda *showInternal* slouží pro vykreslení nabídku možností a umožňuje pohyb pomocí kláves 'w' a 's', zobrazí náhled vybrané možnosti. Metody *show* a *showWithPreview* volají *showInternal* kde se volí, zda bude nabídka v konzoli zobrazena s náhledem matice nebo bez něj. Metoda *selectMatrix* slouží k výběru vstupní matice. Nachází se zde i metoda, která řeší zpracování příkazů z příkazového řádku.

3.3.4 Třída *App.cpp*

Tato třída je *mainem* celé aplikace. Ve třídě se nejprve zvolí, jakým způsobem bude matice načtena, poté v jakém formátu a jakou metodou QR rozkladu bude zpracována. Po volbě algoritmu bude vyvolána nabídka, zda chce uživatel během eliminovat velmi malé hodnoty (tedy hodnoty menší než 1e-12). Vybraná matice je pak zobrazena a podle zvoleného formátu vytvořena pomocí návrhového vzoru Factory. Pokud je matice menší, než 50x50 bude zobrazena v konzoli. Nakonec se provede QR rozklad pomocí zvolené metody. Celý tento proces probíhá interaktivně skrze textové menu.

3.4 Testování výpočtů aplikace

Při programování projektu bylo nutné provádět jednotkové testy pro ověření výpočtů a správnosti. Testování bylo provedeno pomocí parametrických unit testů pomocí GoogleTests. Testy bylo nutné do projektu nejprve integrovat formou vytvoření nového projektu v daném řešení. Tento projekt slouží jako ověření funkčnosti předešlé aplikace. V této části bude nyní rozebráno, jaké testy při tvorbě aplikace byly vytvořeny. Jelikož se jedná o parametrické testy je v projektu možnost otestovat správnost výpočtu i na určené matici. Pro pochopení, jakým způsobem je projekt tvořen je nutné nyní stručně popsat funkcionalitu jednotlivých tříd. Struktura tohoto projektu je zaznamenána na následujícím obrázku.



Obrázek 18 – Struktura projektu s testy

Ve zdrojových souborech se nachází pouze jeden filtr s názvem $Init$. V tomto filtru jsou nastavení týkající se testů včetně konkrétních matic pro testování. Pro testování byly použity různé typy matic a pro ověření, že je výpočet správný bylo použito mnoho metod ze třídy $QR\_Validator$ z minulého projektu. Na jednotkové testy byly využity dvě třídy $MatrixFormatTests$, která testuje správné uložení matice do daného formátu, a $QR\_DecompositionAlgorithmTests$, která testuje, zda byl QR rozklad proveden korektně pomocí zadaného algoritmu QR rozkladu. Jednotlivé třídy mají své hlavičkové soubory, které budou stejně jako u minulého projektu rozebírány postupně se zdrojovými soubory. Pro spuštění testů je nejdůležitější část projektu $main$, který se nachází ve třídě $AppTest.cpp$. Jednotlivé třídy uvedené na předchozím obrázku budou nyní popsány podrobněji.

- ***MatrixTestParam.h* a *MatrixTestParam.cpp***

  V této třídě se definuje $testParams$, který obsahuje sadu testovacích matic. Jsou zde zahrnuty jsou různé typy matic pro účely testování, jako čtvercové matice, jednotková matice, nulová matice, horní a dolní trojúhelníkové matice, singulární matice, matice různých rozměrů, řídká matice, permutační matice, ale i matice s jedním řádkem nebo jedním sloupcem. Kromě již nadefinovaných matic zde uživatel může přidat svoji konkrétní matici a vyhodnotit správnost výpočtu přímo na ní.

- ***ParametrizedTestsSetup.cpp***

  Jedná se o třídu, která nastaví framework Google Tests. Instancují se zde dvě sady testů $MatrixFormatTests$ a $QR\_DecompositionTests$. Tyto sady testů jsou vyhodnocovány na $testParams$ zadaných v $MatrixTestParam.cpp$. První sadou testů se ověřuje správnost struktury uložení matice, zatímco druhá testuje implementaci QR dekompozice.

- ***MatrixFormatsTests.h* a *MatrixFormatsTests.cpp***

  Tato třída ověřuje správnost implementací matic v jednotlivých formátech. Pomocí $expectMatrixEquals$ je pak ověřeno, zda tato matice obsahuje očekávané hodnoty. Pomocí této třídy jsou pak otestovány jednotlivé matice nacházející se v $MatrixTestParam$.

- ***QR_DecompositionAlgorithmTests.h* a *QR_DecompositionAlgorithmTests.cpp***

  V dané třídě se nacházejí jednotkové testy, které testují, zda byl QR rozklad podle dané metody a podle daného formátu proveden korektně. Testování je provedeno formou součinu Q · R pro danou matici s odchylkou, poté je ověřena ortogonalita matice Q a pomocí Gaussovy eliminace se ověří, zda je matice R horní trojúhelníková.

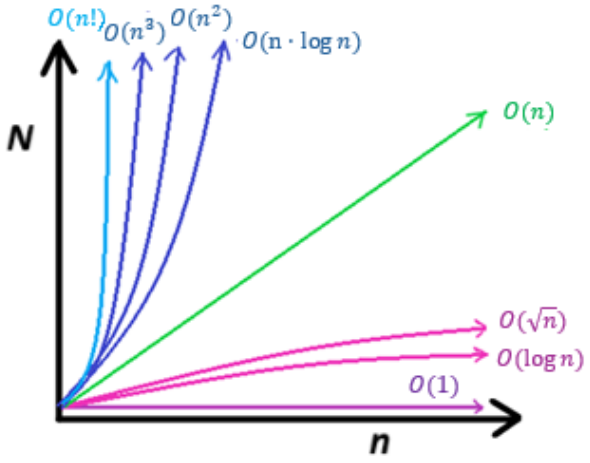
- **Třída *AppTests.cpp***

  Právě zmíněná třída je tou nejdůležitější z celého testovacího projektu, jedná se o $main$ aplikace pro testování. Díky ní se spouští veškeré dříve popsané testy.

3.5 Asymptotická složitost algoritmů QR rozkladu v aplikaci

Algoritmy popsané v práci se navzájem liší, tudíž je vhodné se na ně podívat z pohledu výpočetní složitosti a zvolit tak ten nejvhodnější pro danou matici. Tento výběr se obvykle opírá o dvě hlavní metriky. První z nich je časová složitost, která udává dobu potřebnou k výpočtu vzhledem k velikosti vstupních dat. Druhou je paměťová složitost, jež popisuje množství využité paměti během běhu algoritmu. Jelikož přesné určení složitosti bývá často obtížné a závisí na konkrétní implementaci i použitém hardwaru, používá se pro zjednodušení asymptotická složitost. Ta hodnotí, jak rychle roste počet operací algoritmu při zvětšujícím se rozsahu vstupních dat. Tato složitost ignoruje nižší řády a konstanty, a umožňuje tak efektivní porovnání rychlosti algoritmů. [15]

Existují i další typy složitosti, které pomáhají lépe porozumět chování algoritmů. Mezi nejčastěji používané patří horní odhad (označovaný jako O), který vyjadřuje maximální hranici počtu operací, jež algoritmus nikdy nepřekročí. Poté dolní odhad (označovaný jako Ω), jenž udává minimální počet operací, pod který algoritmus při daném vstupu vykoná. Mezi těmito odhady je asymptotická těsná mez (Θ), která představuje přesnější odhad, kdy horní i dolní odhad jsou stejné až na konstantu. Tato složitost pracuje s průměrnou dobou běhu algoritmu jako s náhodnou veličinou. Amortizovaná složitost, která vyjadřuje průměrnou časovou náročnost posloupnosti operací, čímž nabízí kompromis mezi nejhorším a průměrným případem. Nejčastější příklady složitostí horního odhadu jsou uvedeny na obrázku. [15]



Obrázek 19 – Základní asymptotické složitosti

3.5.1 Analýza výpočetní složitosti metod maticových formátů

Po nadefinování základních pojmů je vhodné přistoupit k praktické analýze aplikace. Jelikož se v aplikaci nacházejí různé maticové formáty, které jsou reprezentovány pomocí tříd, budou se lišit i jejich implementace v jednotlivých třídách. Při výpočtu QR rozkladu budou používány metody jednotlivých maticových tříd. Časová a paměťová náročnost těchto metod může být odlišná, proto je nutné provést analýzu. Po důkladném zkoumání kódu byly zjištěny časové a paměťové složitosti jednotlivých metod pro určité maticové formáty. Tyto složitosti byly zaznamenány do tabulek níže. Pro lepší orientaci v tabulce bylo vytvořeno značení, které je uvedeno na následujícím obrázku a poté je podrobněji rozebráno.

konstantní nebo lineární složitost	$O(nnz_{row})$ $O(nnz_{col})$
lineární složitost	$O(m)$ $O(n)$ $O(v)$
lineární nebo kvadratická složitost	$O(nnz)$

Obrázek 20 – Značení asymptotické složitosti

Pokud se ve složitosti vyskytuje písmeno m závisí tato složitost na počtu prvků v řádku (jedná o lineární složitost). Podobně tomu je pro n, které značí počet prvků ve sloupci a tato závislost je opět lineární. V některých složitostech se objevuje písmeno v, které označuje velikost vektoru dané metody. Pokud je ve složitosti uvedeno nnz (počet nenulových prvků), tato složitost může být lineární případně kvadratická. Kromě nnz se vyskytují i složitosti nnz_{row} a nnz_{col}, tyto složitosti značí počet nenulových prvku v určitém řádku případně sloupci. U těchto složitostí může být konstantní časová složitost, pokud matice v daném řádku/sloupci nemá žádný nenulový prvek. V ostatních případech, kdy se v matici vyskytují nenulové prvky se jedná o lineární složitost.

Po rozebrání značení asymptotické složitosti je možné přistoupit k časové analýze jednotlivých metod maticových tříd což je uvedeno v následující tabulce.

Tabulka 4 – Časová složitost metod pro jednotlivé maticové formáty (viz obrázek 13 a jeho popis)

	COOColMajor	COORowMajor	CSC	CSR
constructor	O(1)	O(1)	O(n)	O(m)
clone()	O(nnz)	O(nnz)	O(nnz)	O(nnz)
get()	O(nnz)	O(nnz)	$O(nnz_{col})$	$O(nnz_{row})$
set()	O(nnz)	O(nnz)	O(nnz)	O(nnz)
getElemsInRow()	O(nnz)	O(nnz)	O(nnz)	$O(nnz_{row})$
getElemsInCol()	O(nnz)	O(nnz)	$O(nnz_{col})$	O(nnz)
getHousVector()	O(nnz)	O(nnz)	$O(nnz_{col})$	O(nnz)
applyHousLeft()	O(n·v·nnz)	O(n·v·nnz)	O(n·v·nnz)	O(n·v·nnz)
applyHousRight()	O(m·v·nnz)	O(m·v·nnz)	O(m·v·nnz)	O(m·v·nnz)
applyGivRows()	O(nnz)	O(nnz)	O(nnz)	O(nnz)
applyGivCols()	O(nnz)	O(nnz)	O(nnz)	O(nnz))

Z tabulky vyplývá, že u konstruktoru je nejlepší časová složitost u matic formátu COO. Naklonování řídkých matic má složitost $O(nnz)$. Metoda set má časovou složitost vždy $O(nnz)$ oproti tomu metoda get má složitost u formátů CSR $O(nnz_{col})$ a u CSC $O(nnz_{row})$. Metody $getElementsInRow$, $getElementsInCol$ a $getHouseholderVector$ mají povětšinou lineární složitost případně kvadratickou. Poslední čtyři metody souvisejí s aplikováním Householderovy transformace a Givensovy rotace na matice R a Q. Tyto metody jsou zásadní pro výpočet v algoritmech. V tabulce je zřejmé, že Householderovy metody mají kubickou závislost závislou na velikosti vektoru v. Oproti tomu metody provádějící Givensovu rotaci mají u formátu COO lineární případně kvadratickou závislost.

Kromě časové analýzy byla provedena i paměťová analýza jednotlivých metod maticových tříd. Výsledky jsou zpracovány do následující tabulky.

Tabulka 5 – Paměťová složitost metod pro jednotlivé maticové formáty (viz obrázek 13 a jeho popis)

	COOColMajor	COORowMajor	CSC	CSR
constructor	O(1)	O(1)	O(n)	O(m)
clone()	O(nnz)	O(nnz)	O(nnz)	O(nnz)
get()	O(1)	O(1)	O(1)	O(1)
set()	O(nnz)	O(nnz)	O(1)	O(1)
getElemsInRow()	$O(nnz_{row})$	$O(nnz_{row})$	O(n)	$O(nnz_{row})$
getElemsInCol()	$O(nnz_{col})$	$O(nnz_{col})$	$O(nnz_{col})$	O(m)
getHousVector()	$O(nnz_{col})$	$O(nnz_{col})$	$O(nnz_{col})$	O(m)
applyHousLeft()	O(n)	O(n)	O(1)	O(n)
applyHousRight()	O(m)	O(m)	O(m)	O(m)
applyGivRows()	O(n)	O(n)	O(1)	$O(nnz_{row})$
applyGivCols()	O(m)	O(m)	$O(nnz_{col})$	O(1)

Z paměťové analýzy uvedené v tabulce vyplývá, že konstruktor má konstantní paměťovou složitost v COO formátech, lineární v CSR a CSC. Metoda get má vždy konstantní paměťovou složitost, oproti tomu metoda set ji má v COO formátech lineární. Metody $getElementsInRow$, $getElementsInCol$ a $getHouseholderVector$ mají vždy lineární časovou složitost. Metoda $applyHouseholderLeft$ má nevýhodnější paměťovou složitost u formátu CRC kde je konstantní, ostatní formáty počítají s lineární závislostí. Tato skutečnost již neplatí pro metody $applyHouseholderRight$, kde je ve všech případech lineární závislost. Metody provádějící Givensovu na řádky mají konstantní složitost u hustých formátů a u formátu CSC, ostatní formáty ji mají lineární. Poslední metodou je $applyGivensRotationCols$, která je u formátu CSR konstantní. Formáty COO a CSC mají tuto složitost lineární.

Z analýzy jednotlivých metod vyplývá, že časové a paměťové složitosti se v jednotlivých řídkých formátech liší.

3.5.2 Vyhodnocení asymptotické složitosti metod QR rozkladu

Cílem této podkapitoly je analyzovat časovou a prostorovou složitost jednotlivých algoritmů. Jelikož jsou jednotlivé algoritmy napsány genericky bude záviset vždy na implementaci jednotlivé metody v třídě matice, případně v třídě $MatrixFactory$. Časová a prostorová složitost jednotlivých metod využitých v těchto algoritmech byla popsána v předešlé části. Nyní je nutné vyhodnotit časovou a prostorovou složitost metody používaných pro QR rozklad ve třídě $MatrixFactory$. Konkrétně to budou metody $createAndFill$, $createIdentityMatrix$ a $createZeroMatrix$. Tyto metody jsou využívány v samém počátku algoritmů a jejich časová a prostorová složitost je klíčová. V práci bude nejprve rozebrána časová složitost, která je zaznamenána do následující tabulky.

Tabulka 6 – Časová složitost metod třídy MatrixFactory (viz obrázek 13 a jeho popis)

	COOColMajor	COORowMajor	CSC	CSR
createAndFill()	O(n·m·nnz)	O(n·m·nnz)	O(n·m·nnz)	O(n·m·nnz)
createIdentityMatrix()	O(k·nnz)	O(k·nnz)	O(k·nnz)	O(k·nnz)
createZeroMatrix()	O(1)	O(1)	O(n)	O(m)

Z dat v tabulce je zřejmé, že metoda $createAndFill$ má u řídkých matic kubickou složitost. Jelikož se tato metoda v algoritmech nepoužívá není na ni brán takový zřetel při analýze časové složitosti. Oproti tomu metody $createIdentityMatrix$ a $createZeroMatrix$ jsou v QR rozkladu využívány mnohem více. Tyto metody slouží pro vytvoření vstupních matic algoritmů. U metody $createZeroMatrix$ je ve formátu COO konstantní časová složitost, tato metoda se používá hlavně u Grammových-Schmidtových metod. Oproti tomu metoda $createIdentityMatrix$ je využívána hlavně u Householderovy a Givensovy metody. Složitost této metody je může být kvadratická.

Po rozebrání časové složitost třídy $MatrixFactory$ se nyní práce zaměří na prostorovou složitost. Jednotlivé asymptotické složitosti jsou uvedeny v následující tabulce.

Tabulka 7 – Časová složitost metod třídy MatrixFactory (viz obrázek 13 a jeho popis)

	COOColMajor	COORowMajor	CSC	CSR
createAndFill()	O(nnz)	O(nnz)	O(nnz)	O(nnz)
createIdentityMatrix()	O(k)	O(k)	O(n)	O(m)
createZeroMatrix()	O(1)	O(1)	O(n)	O(m)

Z uvedených dat je patrné že metoda $createAndFill$ může mít složitost lineární případně kvadratickou, je to totiž dáno počtem nenulových prvků v matici. Tato metoda opět není tolik zastoupena v metodách QR rozkladu, tudíž je její paměťová složitost méně významná. Metoda $createIdentityMatrix$ je využívána u Householderovy transformace a Givensovy rotace. Složitost této metody je lineární. Poslední metodou je $createZeroMatrix$, která je využívána u Grammových-Schmidtových metod QR rozkladu. Tato metoda má lineární případně konstantní. Nejlepší výsledky paměťové složitosti jsou zaznamenány u formátu COO.

Po rozebraní časové a paměťové složitosti metod třídy $MatrixFactory$ je na čase se zaměřit na konkrétní algoritmy QR rozkladu. Tyto algoritmy budou v práci vždy uvedeny a poté bude popsán jejich rozbor. Jelikož je cílem práce se zaměřit na čtvercové matice bude se tedy analyzovat časová a prostorová složitost pouze čtvercových matic. Na následujících obrázcích bude zaznamenána časová složitost pro každý řádek ve formě komentářů za kódem. Kromě toho zde bude i a prostorová složitost pouze u těch řádků kde je relevantní. V případech, kdy záleží časová a prostorová složitost na implementaci v jednotlivé třídě bude uvedena složitost jako znak $O(!)$. Tyto složitosti bude poté nutné porovnat z předchozími tabulkami.

První analyzovanou metodou bude klasická Grammova-Schmidtova metoda, jejíž kód je vidět na následujícím obrázku.

```
void QR_GramSchmidt::compute(const Matrix& A) {                    // time  |  space
    int m = A.getRows();                                           // O(1)  |  O(1)
    int n = A.getCols();                                           // O(1)  |  O(1)

    Q = MatrixFactory::createZeroMatrix(m, m, A.getFormat());      // O(!)  |  O(!)
    R = MatrixFactory::createZeroMatrix(m, n, A.getFormat());      // O(!)  |  O(!)

    std::unique_ptr<Matrix> V(A.clone());                          // O(!)  |  O(!)
    int limit = std::min(m, n);                                    // O(1)  |  O(1)

    for (int j = 0; j < limit; j++) {                              // O(n;m)
        for (int k = 0; k < j; k++) {                              // O(n)
            double rkj = 0.0;                                      // O(1)  |  O(1)
            for (int i = 0; i < m; i++) {                          // O(m)
                rkj += Q->get(i, k) * V->get(i, j);                // O(!)
            }
            R->set(k, j, rkj);                                     // O(!)
            for (int i = 0; i < m; i++) {                          // O(m)
                V->set(i, j, V->get(i, j) - rkj * Q->get(i, k));   // O(!)
            }
        }
        double norm = 0.0;                                         // O(1)  |  O(1)
        for (int i = 0; i < m; i++) {                              // O(m)
            double val = V->get(i, j);                             // O(!)  |  O(1)
            norm += val * val;                                     // O(1)
        }
        double rjj = std::sqrt(norm);                              // O(1)  |  O(1)
        R->set(j, j, rjj);
        if (rjj < TOLERANCE) {                                     // O(1)
            throw std::runtime_error("nulovy nebo linearne zavisly sloupec");
        }
        for (int i = 0; i < m; i++) {                              // O(m)
            Q->set(i, j, V->get(i, j) / rjj);                      // O(!)
        }
    }
```

Obrázek 21 – Časová a paměťová složitost Grammova-Schmidtova algoritmu

Algoritmus začíná zjištěním počtu řádků a sloupců rozkládané matice tato část má konstantní časovou i prostorovou složitost. Následuje vytvoření nulových matic a naklonování matice A do matice V. Prostorová i časová složitost záleží na implementaci v jednotlivých třídách, je tedy $O(!)$. Následuje inicializace proměnné limit, která je pro čtvercovou matici rovna n s konstantními složitostmi. Algoritmus nyní obsahuje tři vnořené smyčky, které mají časovou složitost $O(n^3)$. Do algoritmu vstupuje i časová složitost get a set metod, které jsou uvedeny v jednotlivých implementacích. Celkově lze říct, že časová složitost je ovlivněna implementací ve třídách a paměťová složitost je ovlivněna formátem matic.

Druhou metodou je Modifikovaná verze Grammova-Schmidtova procesu. Časová a paměťová náročnost kódu je zaznamenána na následujícím obrázku.

```
void QR_ModifiedGramSchmidt::compute(const Matrix& A) {          // time  |  space
    int m = A.getRows();                                         // O(1)  |  O(1)
    int n = A.getCols();                                         // O(1)  |  O(1)

    Q = MatrixFactory::createZeroMatrix(m, m, A.getFormat());    // O(!)  |  O(!)
    R = MatrixFactory::createZeroMatrix(m, n, A.getFormat());    // O(!)  |  O(!)
    int limit = std::min(m, n);                                  // O(1)  |  O(1)

    for (int j = 0; j < limit; j++) {                            // O(n;n)
        for (int i = 0; i < m; i++) {                            // O(m)
            Q->set(i, j, A.get(i, j));                           // O(!)
        }
        for (int k = 0; k < j; k++) {                            // O(n)
            double rkj = 0.0;                                    // O(1)  |  O(1)
            for (int i = 0; i < m; i++) {                        // O(m)
                rkj += Q->get(i, k) * Q->get(i, j);              // O(!)
            }
            R->set(k, j, rkj);                                   // O(!)
            for (int i = 0; i < m; i++) {                        // O(m)
                Q->set(i, j, Q->get(i, j) - rkj * Q->get(i, k)); // O(!)
            }
        }
        double norm = 0.0;                                       // O(1)  |  O(1)
        for (int i = 0; i < m; i++) {                            // O(m)
            norm += Q->get(i, j) * Q->get(i, j);                 // O(!)
        }
        double rjj = std::sqrt(norm);                            // O(1)  |  O(1)
        R->set(j, j, rjj);                                       // O(!)
        if (rjj < TOLERANCE) {                                   // O(1)
            throw std::runtime_error("nulovy nebo linearne zavisly sloupec");
        }

        for (int i = 0; i < m; i++) {                            // O(m)
            Q->set(i, j, Q->get(i, j) / rjj);                    // O(!)
        }
    }
```

Obrázek 22 – Časová a paměťová složitost Modifikovaného Grammova-Schmidtova algoritmu

Oproti minulé metodě zde již není použita matice V, tudíž se snížila režie algoritmu. Časová a prostorová složitost tvorba nulových matic opět závisí na implementaci a je tedy $O(!)$. Stejně tak implementace get a set metod. Algoritmus má opět tři vnořené smyčky, jejichž časová složitost je $O(n^3)$. Celková časová a prostorová složitost algoritmu je stejně jako v minulém případě závislá na třídách a prostorová složitost je dána formátem matic.

Další metodou bude Householderova transformace, která je již od minulých metod více odlišná. Její kód je zaznamenán na následujícím obrázku spolu s vyhodnocením složitostí.

```
void QR_Householder::compute(const Matrix& A) {                                    // time   |  space
    int m = A.getRows();                                                           // O(1)   |  O(1)
    int n = A.getCols();                                                           // O(1)   |  O(1)

    Q = MatrixFactory::createIdentityMatrix(m, m, A.getFormat());                  // O(!)   |  O(!)
    R.reset(A.clone());                                                            // O(!)   |  O(!)
    int limit = std::min(m, n);                                                    // O(1)   |  O(1)
    for (int j = 0; j < limit; j++) {                                              // O(n;m)
        double squaredNorm = 0.0;                                                  // O(1)   |  O(1)
        std::vector<std::pair<int, double>> v_sparse = R->getHouseholderVector(j, squaredNorm);  // O(!)   |  O(!)
        if (squaredNorm == 0.0) {
            v_sparse.clear();                                                      // O(1)   |  O(1)
            v_sparse.push_back({ 0, 1.0 });                                        // O(1)   |  O(1)
        }
        else {
            double alpha = (R->get(j, j) > 0) ? -std::sqrt(squaredNorm) : std::sqrt(squaredNorm);  // O(!)   |  O(1)
            double v0 = R->get(j, j) - alpha;                                      // O(!)   |  O(1)
            bool found = false;                                                    // O(1)   |  O(1)
            for (auto& elem : v_sparse) {                                          // O(m)
                if (elem.first == 0) {                                             // O(1)
                    elem.second = v0;                                              // O(1)
                    found = true;                                                  // O(1)
                    break;
                }
            }
            if (!found) {
                v_sparse.push_back({ 0, v0 });                                     // O(1)   |  O(1)
            }
            double norm = 0.0;                                                     // O(1)   |  O(1)
            for (const auto& elem : v_sparse)                                      // O(m)
                norm += elem.second * elem.second;                                 // O(1)
            norm = std::sqrt(norm);                                                // O(1)
            for (auto& elem : v_sparse)                                            // O(m)
                elem.second /= norm;                                               // O(1)
        }
        R->applyHouseholderLeft(j, v_sparse);                                      // O(!)   |  O(!)
        Q->applyHouseholderRight(j, v_sparse);                                     // O(!)   |  O(!)
    }
}
```

Obrázek 23 – Časová a paměťová složitost Householderovy transformace

Zjištění počtu řádků a sloupců vstupní matice je opět konstantní složitost. Tentokrát je již vytvářena jednotková matice, která může mít v implementaci jinou složitost než vytváření nulové matice, proto je časová i prostorová složitost rovna $O(!)$. V algoritmu se nacházejí dvě vnořené smyčky jejich časová složitost je tedy $O(n^2)$. Jelikož se jedná o řídký Householderův vektor je složitost $O(n \cdot nnz_v)$. Výslednou časovou i prostorovou složitost ovlivní i metody $applyHouseholder$ v jednotlivých implementacích. Aktuálně lze konstatovat, že složitost může být odlišná pro různé formáty matic.

Poslední analyzovanou metodou je Givensova rotace. V kódu je uvedena časová a paměťová náročnost pro jednotlivé řádky algoritmu.

```
void QR_Givens::compute(const Matrix& A) {                                  // time   | space
    int m = A.getRows();                                                    // O(1)   | O(1)
    int n = A.getCols();                                                    // O(1)   | O(1)
    Q = MatrixFactory::createIdentityMatrix(m, m, A.getFormat());           // O(!)   | O(!)
    R.reset(A.clone());                                                     // O(!)   | O(!)
    for (int j = 0; j < n; ++j) {                                           // O(n)
        auto colElements = R->getElementsInCol(j);                          // O(!)   | O(!)
        for (const auto& [i, b] : colElements) {                            // O(!)
            if (i <= j) continue;                                           // O(1)
            double a = R->get(j, j);                                        // O(!)   | O(1)
            if (a == 0.0 && b == 0.0) continue;                             // O(1)
            double denom = std::sqrt(a * a + b * b);                        // O(1)   | O(1)
            if (denom == 0.0) continue;                                     // O(1)
            double c = a / denom;                                           // O(1)   | O(1)
            double s = b / denom;                                           // O(1)   | O(1)
            R->applyGivensRotationRows(j, i, c, s);                         // O(!)   | O(!)
            Q->applyGivensRotationCols(j, i, c, s);                         // O(!)   | O(!)
        }
    }
}
```

Obrázek 24 – Časová a paměťová složitost Givensovy rotace

Kód začíná inicializací proměnných m a n s konstantní složitostí. Stejně jako u minulé metody je tvořena jednotková matice se složitostí $O(!)$. V hlavní části algoritmu se nacházejí dvě vnořené smyčky, které mají časovou složitost $O(n \cdot !)$. Dalším důležitým bodem je vyhodnocení časové složitosti get metody, která se liší v implementaci. Výpočty a alokace proměnných pro Givensovu rotaci mají v algoritmu konstantní časovou i prostorovou složitost. Ke konci se volají metody $applyGivens$, které jsou implementovány pro jednotlivé formáty matic odlišně.

Veškeré metody QR rozkladu závisí na konkrétní implementaci v jednotlivých třídách. Ve třídách je logika jednotlivých metod odlišná, tudíž jejich časová a prostorová složitost může být též odlišná. Po rozebrání časové a paměťové složitosti všech generických algoritmů QR rozkladu je nyní nutné přistoupit k analýze pro jednotlivé maticové formáty a následnému porovnání výsledků analýzy. Tyto výsledky budou zpracovány ve formě tabulek a uvedeny v následující podkapitole diplomové práce.

3.5.3 Porovnání asymptotické složitosti jednotlivých metod

Takto část bude obsahovat vyhodnocení jednotlivých metod a jejich porovnání mezi sebou z pohledu časové a prostorové složitosti. V následující tabulce jsou zaznamenány odhady časové složitosti pro jednotlivé maticové formáty a související metody QR rozkladu.

Tabulka 8 – Porovnání časové a prostorové složitosti v aplikaci pro jednotlivé metody (viz obrázek 13 a jeho popis)

	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas	paměť	čas	paměť	čas	paměť	čas	paměť
GramSchmidt	$O(n^3)$	$O(nnz_{qrv})$	$O(n^3)$	$O(nnz_{qrv})$	$O(n^3)$	$O(nnz_{qrv})$	$O(n^3)$	$O(nnz_{qrv})$
ModifiedGramSchmidt	$O(n^3)$	$O(nnz_{qr})$	$O(n^3)$	$O(nnz_{qr})$	$O(n^3)$	$O(nnz_{qr})$	$O(n^3)$	$O(nnz_{qr})$
Householder	$O(n^2 \cdot v)$	$O(nnz_{qr}+v)$	$O(n^2 \cdot v)$	$O(nnz_{qr}+v)$	$O(n^2 \cdot v)$	$O(nnz_{qr}+v)$	$O(n^2 \cdot v)$	$O(nnz_{qr}+v)$
Givens	$O(n^2 \cdot nnz)$	$O(nnz_{qr})$	$O(n^2 \cdot nnz)$	$O(nnz_{qr})$	$O(n^2 \cdot nnz)$	$O(nnz_{qr})$	$O(n^2 \cdot nnz)$	$O(nnz_{qr})$

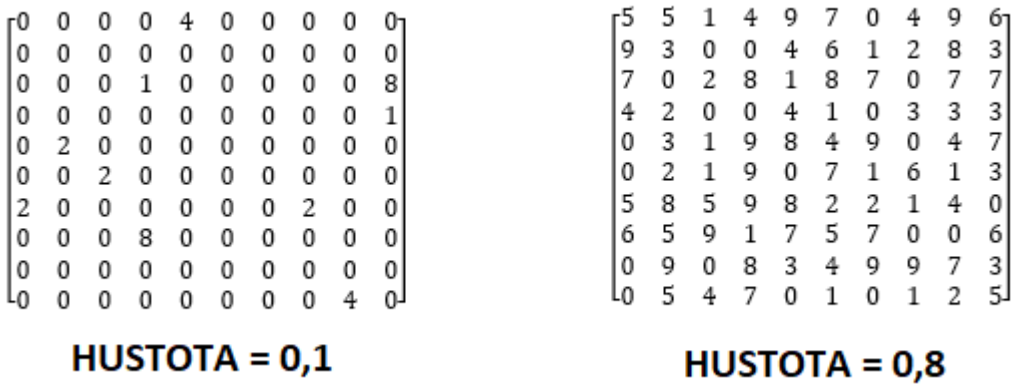
Pro jednotlivé formáty se u Grammových-Schmidtových metod objevuje kubická časová složitost. U metody Householder se do časové složitosti započítává i délka Householderova vektoru v, která je pak zpracována metodou $applyHouseholder$, Podobně tomu je i u Givense, kde se předpočítají jednotlivé proměnné a ty se poté aplikují v metodě $applyGivens$. Z tabulky je patrné, že Grammovy-Schmidtovy metody dosahují v aplikaci horší časové složitosti než metody Householder a Givens. Kromě časové složitosti byla zkoumána i prostorová složitost aplikace ze které vyplývá, že Grammova-Schmidtova metoda dosahuje nejhorší paměťové složitosti z důvodu nutnosti alokovat místo pro tři matice (Q, R, V), oproti tomu modifikovaná verze již nepotřebuje matici V. U Householderovy metody do alokací paměti vstupuje vektor v. Poslední metodou je Givensova metoda, která má stejně jako Householder alokuje místo pouze pro dvě matice Q a R.

Celkově lze konstatovat, že v aplikaci je časová složitost horší než paměťová složitost. Po této analýze bude provedeno měření na konkrétních maticích, které by mělo potvrdit právě zkoumané asymptotické složitosti.

3.6 Vyhodnocení na konkrétních maticích

Tato část práce se bude zabývat vyhodnocením časové a paměťové náročnosti na konkrétních příkladech matic. Měření bude probíhat pomocí C++ knihovny $chrono$ a při měření paměti bude využito Windows API, a to konkrétně funkce $WorkingSetSize$. Při měření bude vyhodnocena pouze metoda $compute$ pro jednotlivé metody QR rozkladu a to pomocí $runneru$, díky kterému se budou spouštět jednotlivé matice. Při měření bude zahrnuta hustota matice. Jak hustota matice ovlivňuje čas a použitou paměť u jednotlivých metod QR rozkladu bude v této podkapitole blíže specifikováno.

V první podkapitole bude vyhodnoceno měření na 100 maticích, které jsou rozměru 10×10 s různou hustotou, která bude až na první hustotu rozčleněna podle 0,1. Jak hustota ovlivňuje počet prvků nenulových prvků matice je zobrazeno na následujícím obrázku.



Obrázek 25 – Matice 10×10 *s odlišnou hustotou*

Druhá podkapitola se zaměří na časové a paměťové porovnání matic se stejnou hustotou ale s odlišnými rozměry. V této části bude viditelný časový a paměťový nárust na základě rozměrů matice. Pro měření byly voleny matice rozměrů 10×10, 20×20, 40×40, 80×80, 160×160 a 320×320. Pro jednotlivá měření bylo použito opět 100 matic, kromě pár měření, která budou zapsány šedě. Pro tyto měření bylo použito pouze 5 matic z důvodu dlouhého běhu programu.

3.6.1 Vyhodnocení metod na základě hustoty na matici 10 × 10

V této podkapitole budou vyhodnoceny jednotlivé metody QR rozkladu na 100 maticích s rozměrem 10 × 10 s různou hustotou. Vyhodnocován bude čas a paměť výpočtu.

První z metod je klasická Grammova-Schmidtova metoda, jelikož se jedná o metodu, která je numericky nestabilní při výskytu nulových sloupců je zde zaznamenán i počet úspěšných měření na 100 maticích. Veškerá naměřená data jsou zaznamenána do následující tabulky.

Tabulka 9 – Časová a paměťová složitost klasické Grammovy-Schmidtovy metody

HUSTOTA	Počet měření	COO Řádkově		COO Sloupcově		CSR		CSC	
		čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
0.01	0	–	–	–	–	–	–	–	–
0.1	0	–	–	–	–	–	–	–	–
0.2	2	0.779	24	0.815	16	0.286	32	0.279	32
0.3	55	1.322	25.74	1.430	17.74	0.472	29.09	0.459	32.43
0.4	85	1.477	23.62	1.708	18.4	0.511	27.01	0.492	30.58
0.5	99	1.543	23.31	1.801	18.50	0.537	26.26	0.508	30.62
0.6	100	1.595	23.88	1.817	19.32	0.552	25.96	0.546	31.28
0.7	100	1.609	23.24	1.912	19.6	0.521	26.76	0.490	30.28
0.8	100	1.584	24	1.880	20.72	0.501	27.04	0.474	30.48
0.9	100	1.629	24.92	1.855	20.96	0.491	27.04	0.462	30.64
1	100	1.617	24.84	1.862	20.44	0.468	27.12	0.460	30.4

Z tabulky vyplývá, že klasická Grammova-Schmidtova metoda začíná být výhodná až při vyšší hustotě vstupní matice. Tato metoda je pro hustotu do 0.2 velice numericky nestabilní díky výskytu nulových sloupců. Proto pro výpočet QR rozkladu v řídkém formátu není ideální. Stabilní začíná být až při hustotě 0.5 kde úspěšně proběhlo 99 ze 100 měření.

Další metodou QR rozkladu, která bude zkoumána na matici 10×10 je právě modifikovaná verze předešlé metody. Jelikož již předchozí metoda je nestabilní při výskytu nulových sloupců bude zde opět uveden počet úspěšných měření. Tyto měření budou zaznamenána do tabulky.

Tabulka 10 – Časová a paměťová složitost modifikované Grammovy-Schmidtovy metody

HUSTOTA	Počet měření	COO Řádkově		COO Sloupcově		CSR		CSC	
		čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
0.01	0	–	–	–	–	–	–	–	–
0.1	0	–	–	–	–	–	–	–	–
0.2	2	0.767	36	0.813	30	0.27	40	0.306	40
0.3	55	1.232	35.63	1.442	30.25	0.415	39.34	0.426	39.49
0.4	85	1.410	32.18	1.686	28.8	0.446	36.04	0.433	35.76
0.5	99	1.505	30.42	1.842	27.47	0.478	33.61	0.457	33.85
0.6	100	1.485	30.32	1.849	27.92	0.485	33.24	0.459	33.36
0.7	100	1.55	30.4	1.934	28.32	0.478	33.52	0.466	33.76
0.8	100	1.558	30.12	1.962	28.64	0.491	33.52	0.463	33.12
0.9	100	1.605	30.08	1.971	28.84	0.489	33.48	0.465	33.68
1	100	1.579	30.68	2.00	28.48	0.481	33.88	0.466	34.08

Z měřených dat vyplývá, že počet úspěšných měření je pro klasickou Grammovu-Schmidtovu metodu stejný jako pro modifikovanou verzi. Vyskytuje se zde opět vysoká numerická nestabilita, a tudíž je tato metoda až po hustotu 0,2 vysoce nevhodná. Stabilita výpočtů přichází až při hustotě 0,5, kde bylo zaznamenáno 99 úspěšných měření z celkových 100. Obecně lze konstatovat, že Grammovy-Schmidtovy metody se pro výpočet QR rozkladu v řídkém formátu nehodí z důvodu numerické nestability, proto bude vhodné použít jinou metodu, kterou může být Householderova transformace případně Givensova rotace. Tyto metody budou vyhodnoceny v textu na následujících stranách.

Příští metodou, která bude vyhodnocena je Householderova transformace, data z měření byla zaznamenána do tabulky. Měření opět probíhalo na 100 maticích rozměru 10×10, tentokrát již nebylo nutné uvádět počet úspěšných měření, protože ve všech případech byl 100.

Tabulka 11 – Časová a paměťová složitost Householderovy metody

HUSTOTA	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
0,01	0.315	58.12	0.312	45.84	0.107	44.4	0.096	48.52
0,1	0.643	58.8	0.622	46.56	0.199	44.76	0.185	49.36
0,2	1.740	59.4	1.732	45.84	0.344	40.72	0.338	44.44
0,3	3.669	63.16	3.697	51.28	0.482	41.08	0.470	45.08
0,4	4.845	67.68	4.920	56.4	0.527	41.36	0.508	45.6
0,5	5.665	68.72	5.722	56.24	0.543	41.36	0.535	45.08
0,6	6.204	70.16	6.299	58	0.552	41.28	0.546	45.84
0,7	6.682	72.44	6.727	59.36	0.560	42.12	0.539	45.88
0,8	6.900	74.64	6.964	59.64	0.550	42.2	0.535	46.4
0,9	7.338	75.6	7.342	62.04	0.556	42.44	0.527	46.28
1	7.378	74.92	7.537	60.32	0.526	41.56	0.513	45.92

Z dat vyplývá, že Householderova transformace je oproti Grammovým-Schmidtovým metodám vhodnější metodou pro QR rozklad v řídkém formátu. Z tabulky je patrné, že čas výpočtu s nabývající hustotou matice mírně roste, stejně tomu je u paměti. Jelikož se jedná o rozměr 10×10 nebude nárust času a paměti tak radikální. Z měření je důležité podotknout, že při srovnání časové náročnosti matic s hustotou 0,01 a hustotou 1 je velký rozdíl v době výpočtu. Z hlediska paměti je vyšší rozdíl v hustotě u metod COO, oproti tomu u formátů CSR a CSC tento rozdíl není patrný.

Poslední měřenou metodou QR rozkladu na maticích 10×10 bude Givensova rotace, která bude zpracována opět do tabulky. Při výpočtu nebylo nutné zavádět počet úspěšných pokusů, protože úspěšně prošlo všech 100 pokusů. Díky tomuto faktu se Givensova metoda nabízí pro výpočet QR rozkladu v řídkém formátu.

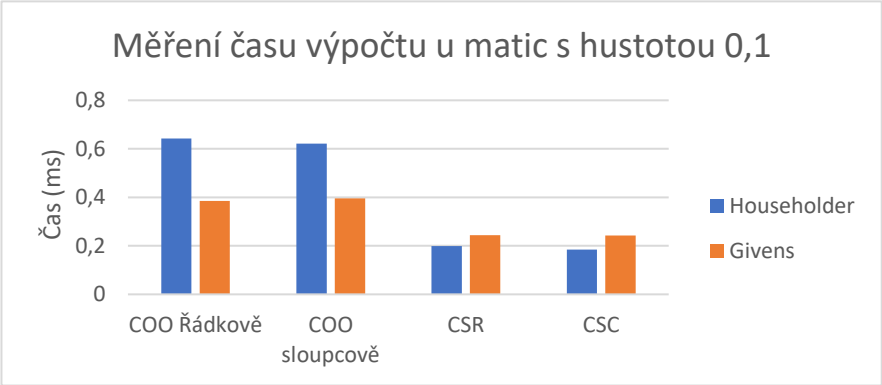
Tabulka 12 – Časová a paměťová složitost Givensovy metody

HUSTOTA	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
0,01	0.081	31.04	0.078	25.8	0.066	35.2	0.065	32.2
0,1	0.385	39.76	0.396	47.44	0.244	51.72	0.243	43.56
0,2	1.356	43.64	1.346	36.68	0.672	47.48	0.648	39.6
0,3	3.958	44.96	3.903	37.04	1.637	47.44	1.610	39.84
0,4	6.055	45.68	6.058	38.28	2.433	48.72	2.269	41.36
0,5	7.753	47.44	7.639	40.12	2.810	48.16	2.837	40.96
0,6	8.754	48.36	8.848	39.96	3.186	47.6	3.160	41.48
0,7	9.483	48.04	9.662	39.52	3.487	26.76	3.390	30.28
0,8	10.28	48.04	10.224	40.32	3.656	48.64	3.549	42.4
0,9	10.980	48.92	10.913	40.84	3.994	47.24	3.785	41.96
1	11.050	47.48	11.06	39.68	3.967	46.76	3.769	42.36

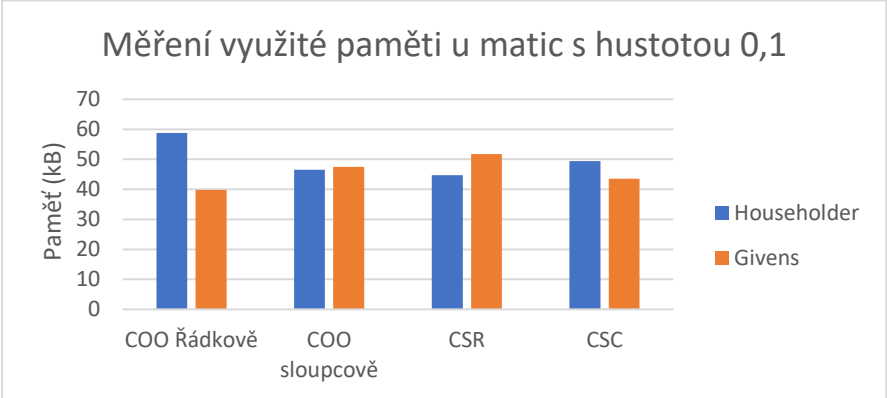
Z dat je viditelné, že Givensova rotace je vhodnější metodou pro řídký QR rozklad než Grammovy-Schmidtovy metody, protože je numericky stabilnější. Tabulka udává nárust času při zvyšující se hustotě matice. Při srovnání měření matice, která má hustotu 0,01 a matice s hustotou 1 je velký rozdíl v čase výpočtu. Srovnání ohledně použité paměti je patrnější u COO metod než u CSR a CSC.

Data z přechozích měření budou nyní vyhodnocena ve formě grafu, který bude uveden pro matici s hustotou 0,1 a matici s hustotou 0,8. Cílem vyobrazených grafů je porovnat jednotlivé metody QR rozkladu pro určité formáty uložení matice. Při porovnání bude analyzován čas výpočtu a použitá paměť.

Jako první budou vyhodnoceny metody QR rozkladu na maticích 10×10 s hustotou 0,1 a to ve formě dvou grafů udávající čas měření a použitou paměť.



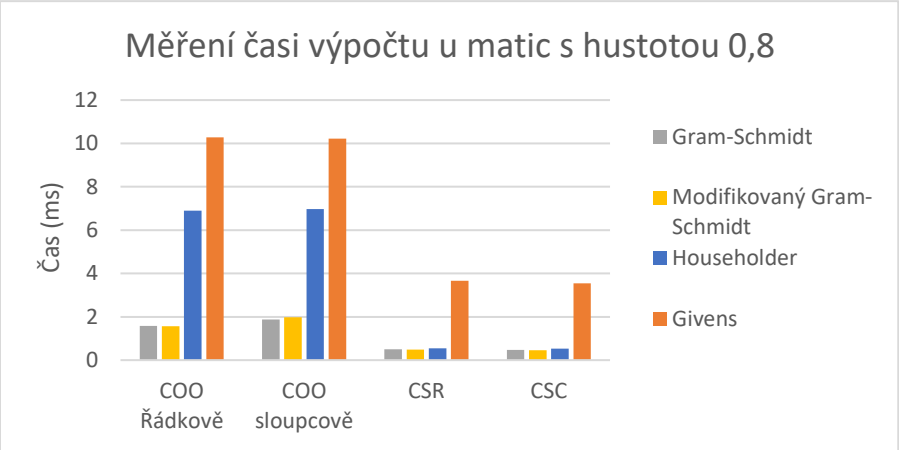
Obrázek 26 – Porovnání času výpočtu matic 10×10 *s hustotou 0,1*



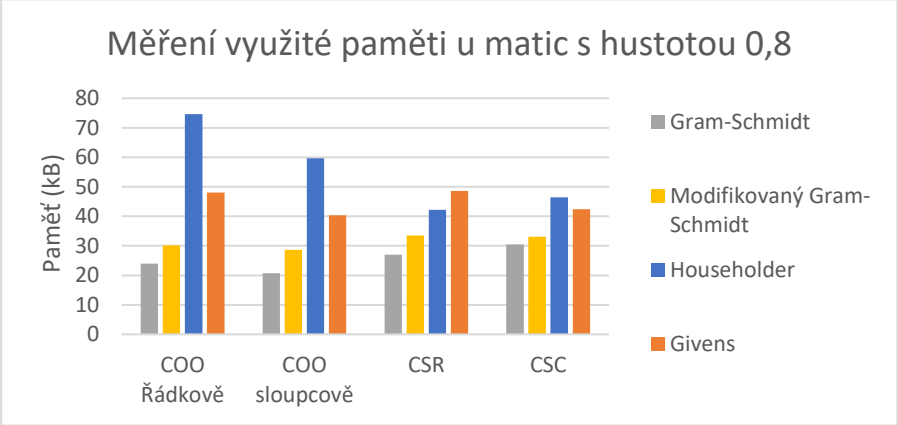
Obrázek 27 – Porovnání použité paměti matic 10×10 *s hustotou 0,1*

Z prvního grafu je zřejmé, že Householderova metoda má ve srovnání s Givensovou metodou vyšší časovou složitost. Tento rozdíl se projevuje zejména u matic ve formátu COO, kde je rozdíl v době výpočtu mezi oběma metodami nejmarkantnější. Naopak u formátů CSR a CSC je tento rozdíl méně výrazný. Druhý graf, který zobrazuje paměťovou náročnost jednotlivých metod, ukazuje, že u matic o velikosti 10×10 nejsou mezi použitými metodami zaznamenány žádné výraznější rozdíly v paměťové spotřebě. To naznačuje, že rozdíly v nárocích na paměť se mohou projevit až u větších matic, zatímco pro malé vstupy zůstávají metody z hlediska paměti srovnatelné.

Vyhodnocení metod QR rozkladu bude pokračovat vyhodnocením na 100 maticích rozměru 10×10 a hustotou 0,8. Tyto data se opět opírají o předchozí hodnoty zaznamenané do tabulek. Vyhodnocení proběhne formou dvou grafů udávajících čas výpočtu a využitou paměť.



Obrázek 28 – Porovnání času výpočtu matic 10×10 *s hustotou 0,8*



Obrázek 29 – Porovnání využité paměti u matic 10×10 *s hustotou 0,8*

Z předložených grafů je zřejmé, že při zvýšení hustoty matice na hodnotu 0,8 dochází k výraznému nárůstu doby výpočtu zejména u Givensovy metody. Tento nárůst je ve srovnání s ostatními testovanými metodami nejvýraznější, což naznačuje, že Givensova metoda je při vyšší hustotě méně efektivní z hlediska časové náročnosti. Druhá největší časová náročnost byla zaznamenána u Householderovy metody, přičemž tento nárůst je nejpatrnější v případě použití formátu COO. Z druhého grafu, který zachycuje spotřebu paměti jednotlivých metod, vyplývá, že Householderova metoda vykazuje obecně vyšší paměťové nároky ve srovnání s ostatními metodami. Největší nárůst spotřeby paměti byl zaznamenán opět při použití formátu COO. To naznačuje, že kombinace Householderovy metody a COO formátu je z hlediska paměťové náročnosti méně vhodná, zejména při vyšší hustotě matic.

3.6.2 Vyhodnocení metod na základě rozměru matice

Tato podkapitola se zabývá vyhodnocením časové a paměťové složitosti na základě rozměru matic. Pro měření bude použito 100 matic s hustotou 0,01 a 0,02. Jelikož některá měření trvají déle, budou vyhodnoceny pouze na 5 maticích. Tyto měření budou označena šedě a jsou pouze orientační z důvodu nízkého počtu matic použitých při měření.

První z vyhodnocovaných metod budou Householderovy transformace, které budou vyhodnoceny formou tabulek udávající čas výpočtu a využitou paměť.

Tabulka 13 – Časová a paměťová složitost Householderovy transformace matic s hustotou 0,01

VELIKOST	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
10 × 10	0.324	56.84	0.316	46.8	0.109	44.2	0.099	48.28
20 × 20	0.718	52.12	0.749	39.52	0.228	39.8	0.192	43.76
40 × 40	2.709	53.28	2.638	42.72	0.645	40.24	0.523	44.08
80 × 80	22.682	60.4	22.825	48.32	2.946	41.52	2.472	45.6
160 × 160	8549.7	375.6	8757.0	370.9	149.655	163.36	198.854	168.56
320 × 320	–	–	–	–	25385.2	1744.96	29864.2	1767.12

Tabulka 14 – Časová a paměťová složitost Householderovy transformace matic s hustotou 0,02

VELIKOST	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
10 × 10	0.337	53.76	0.335	44.6	0.113	41.72	0.103	46.72
20 × 20	0.894	52.96	0.827	40.56	0.252	40.48	0.215	44.36
40 × 40	4.805	58.52	4.788	46.48	0.980	40.64	0.854	44.84
80 × 80	366.93	190.84	389.99	179.96	18.559	70.96	20.924	74.28
160 × 160	199046	1158.4	201843	1295.2	1734.4	546.76	2133.29	547.96
320 × 320	–	–	–	–	52074.6	2854.4	62416.7	2178.4

Z tabulky je zřejmé, že Householderova metoda dosahuje vyšší časové náročnosti u formátů COO než u formátů CSR a CSC. Z hlediska využití paměti jsou opět formáty COO náročnější na paměť. Některé hodnoty nejsou vyplněny z důvodu dlouhého vyhodnocení.

Druhou a poslední metodou vyhodnocenou z hlediska času výpočtu a použité paměti bude Givensova rotace. Měření proběhne stejným způsobem jako u Householderovy metody a bude zpracováno znovu formou tabulky.

Tabulka 15 – Časová a paměťová složitost Givensovy rotace matic s hustotou 0,01

VELIKOST	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
10 × 10	0.085	33.4	0.084	27.52	0.072	37.2	0.069	33.84
20 × 20	0.185	39.04	0.182	31.48	0.141	42.92	0.136	35.4
40 × 40	0.560	41.72	0.547	33.6	0.402	44.2	0.390	36.08
80 × 80	2.514	44.96	2.431	36.48	1.764	46.32	1.674	38.2
160 × 160	61.514	80.04	63.557	71.6	24.019	65	26.555	60.16
320 × 320	–	–	–	–	27883	1679.56	36154.7	1743.56

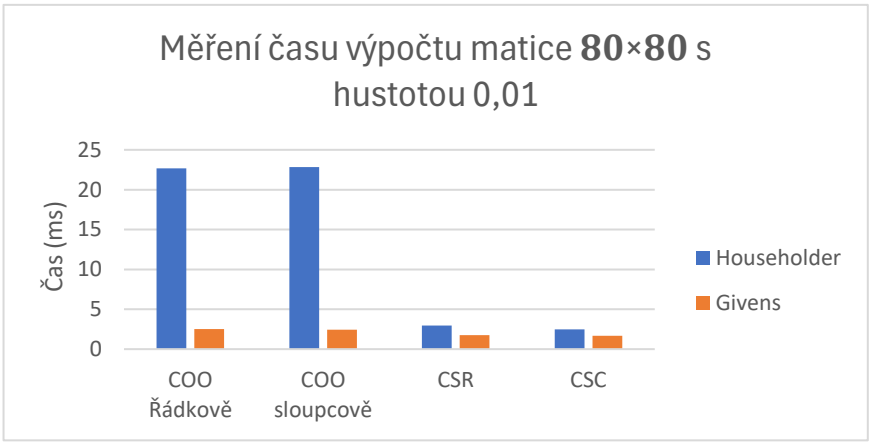
Tabulka 16 – Časová a paměťová složitost Givensovy rotace matic s hustotou 0,02

VELIKOST	COO Řádkově		COO Sloupcově		CSR		CSC	
	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)	čas(ms)	paměť(kB)
10 × 10	0.104	33	0.103	26.44	0.082	36.96	0.080	31.96
20 × 20	0.274	40.76	0.278	32.76	0.195	43.96	0.193	36.6
40 × 40	1.238	45.6	1.257	37.8	0.797	45.84	0.779	37.44
80 × 80	16.09	60.6	16.12	51.72	7.434	55.2	7.626	49.04
160 × 160	36381.7	528	43612.6	565.6	2485.47	447.2	3154.09	420.24
320 × 320	–	–	–	–	167419	2787.2	196753	2408.8

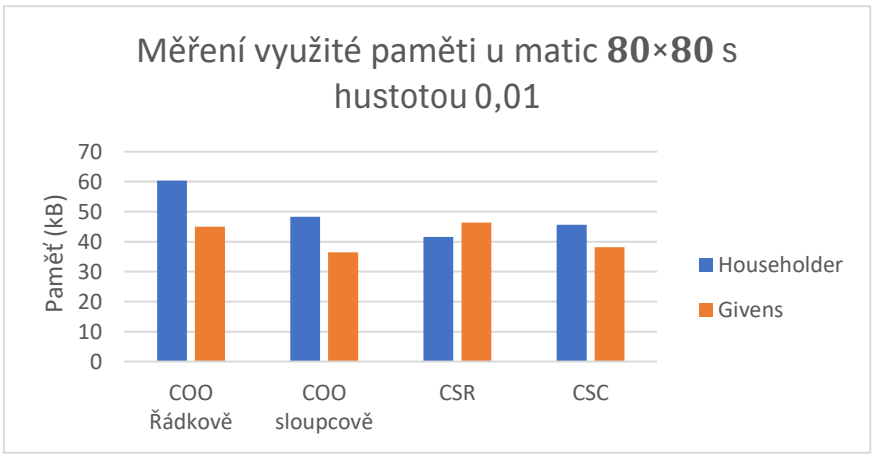
Z dat v tabulce vyplývá, že formát COO je časově náročnější než formáty CSR a CSC. Paměťová náročnost Givensovy metody je u formátů COO a CSR a CSC vcelku srovnatelná. Jednotlivá měření budou nyní vyhodnocena formou grafů.

Z předchozích měření nyní bude zpracováno porovnání ve formě grafů. Jelikož se čas výpočtu při vyhodnocení výrazně navyšoval nebylo možné provést detailnější analýzu. Porovnání jednotlivých metod QR rozkladu bude probíhat na datech z předchozích měření a na maticích s rozměrem 80×80 a 160×160 s hustotou 0,01.

Nejprve bude porovnána časová a paměťová náročnost matic 80×80. Toto porovnání proběhne pro metody Householderova transformace a Givensova rotace a je vyobrazeno na následujících grafech.



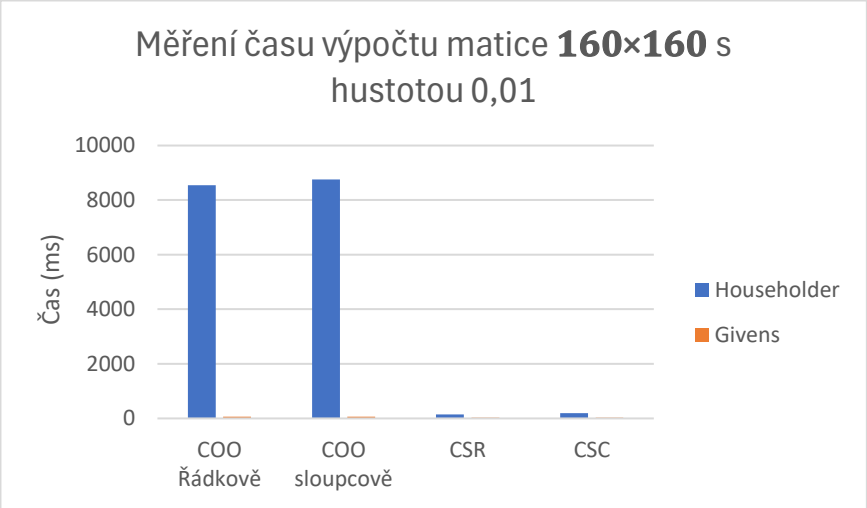
Obrázek 30 – Porovnání času výpočtu u matic 80×80 s hustotou 0,01



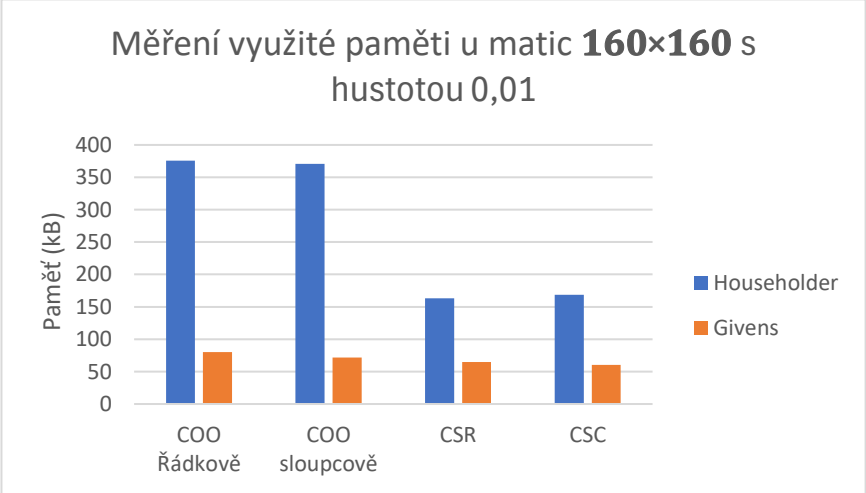
Obrázek 31 – Porovnání využité paměti u matic 80×80 s hustotou 0,01

První graf udává, že doba výpočtu Householderovy metody QR rozkladu je u rozměru 80×80 je výrazně vyšší u formátů COO než CSR a CSC. V porovnání s Givensovou rotací je Householderova transformace časově náročnější. Z hlediska využité paměti je u formátu COO větší využití paměti než Givensovy metody. U formátů CSR a CSC je využitá paměť porovnatelná.

Druhé porovnání proběhne na maticích 160×160 s hustotou 0,01. Tyto hodnoty byly převzaty z měření uvedeného v tabulkách. Srovnání je provedeno pro Householderovy transformace a Givensovy rotace ve formě dvou následujících grafů.



Obrázek 32 – Porovnání času výpočtu u matic 160×160 *s hustotou 0,01*



Obrázek 33 – Porovnání využité paměti u matic 160×160 *s hustotou 0,01*

Z prvního grafu je patrné, že doba výpočtu při použití Householderovy transformace na matici o rozměrech 160×160 výrazně narůstá čas výpočtu zejména v případě COO formátu. Tento nárůst je mnohem výraznější než u Givensovy rotace. Rozdíl mezi oběma metodami je zřejmý nejen z hlediska časové náročnosti, ale i u využití paměti. Householderova transformace zde vykazuje vyšší paměťové nároky. Celkově tedy z grafu vyplývá, že pro operace s řídkými maticemi, zejména ve formátu COO, je Givensova rotace časově i paměťově výhodnější.

4 Využití QR rozkladu

Tato kapitola bude zaměřena na jednotlivé aplikace QR-rozkladu. Jak je již z minulé kapitoly patrné, QR-rozklad lze provést více metodami. V této kapitole budou nastíněny možnosti využití QR-rozkladu, včetně demonstrace na konkrétním příkladě.

První z aplikací QR-rozkladu je řešení lineárních soustav rovnic, kde bude ukázáno, jakým způsobem se pomocí tohoto rozkladu a zpětné substituce dopočítá lineární soustava rovnic. Následovat bude výpočet inverze pomocí QR-rozkladu. Jelikož se práce zabývá i obdélníkovými maticemi, bude zde představen pojem pseudoinverze, která je na ně uplatnitelná. Další možnou aplikací QR-rozkladu bude představení algoritmu pro výpočet vlastních čísel, který slouží hlavně pro matice velkých rozměrů. Díky tomuto algoritmu bude možné vypočítat i takzvaný SVD rozklad. Kromě této aplikace bude daný algoritmus využit i u výpočtu kořenů polynomu. V závěru této podkapitoly budou uvedeny praktické příklady z oblasti informatiky, ve kterých se QR-rozklad může uplatnit. Cílem této části je poukázat v jakých oblastech se tento rozklad může využít a nastínit tak další možnosti jeho využití.

Z právě popsaných využití QR-rozkladu je možné některé z nich počítat i metou LU rozkladu. V následující tabulce je uvedeno porovnání LU-rozkladu s QR-rozkladem.

Tabulka 17 – Porovnání QR rozkladu s LU rozkladem

Vlastnost	**LU rozklad**	**QR rozklad**
Typ matice	Čtvercová	Obdélníková
Numerická stabilita	Nižší	Vyšší
Rychlost výpočtu	Rychlejší	Pomalejší
Použití	Soustavy lin. rovnic Výpočet inverze matice Výpočet determinantu Výpočet vlastních čísel	Soustavy lin. rovnic Výpočet pseudoinverze matice Výpočet vlastních čísel Výpočet SVD rozkladu Výpočet kořenů polynomu

Z tabulky je patrné, že QR-rozklad lze použít pro obdélníkové matice. Oproti tomu LU rozklad nelze. Při porovnání těchto metod lze říct, že QR-rozklad je výpočetně obtížnější, ale numericky stabilnější než LU rozklad, který je méně numericky stabilní a výpočtově méně náročný. [16]

4.1 Řešení soustav lineárních rovnic

Nechť existuje lineární soustava rovnic ve tvaru $Ax = b$. Matice A lze podle předešlé podkapitoly rozložit na $A = QR$. Lineární soustava rovnic lze tedy přepsat jako $QRx = b$. Tuto rovnici lze zleva vynásobit Q^T , čímž vychází $Q^TQRx = Q^Tb$. Pokud je matice Q ortogonální, lze součin Q^TQ vypustit. Vychází soustava rovnic $Rx = Q^Tb$. Pro dopočítání x bude využita zpětná substituce. [17]

Mějme systém lineárních rovnic:

$$-x_2 + x_3 = 0$$
$$4x_1 + 2x_2 = 4$$
$$3x_1 + 4x_2 = 3$$

Systém lineárních rovnic lze vyjádřit jako $Ax = b$. Matice $A = \begin{bmatrix} 0 & -1 & 1 \\ 4 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix}$ a $b = \begin{bmatrix} 0 \\ 4 \\ 3 \end{bmatrix}$. Pro matici A vychází QR rozklad následovně.

$$\begin{bmatrix} 0 & -1 & 1 \\ 4 & 2 & 0 \\ 3 & 4 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\sqrt{5}/5 & (2\sqrt{5})/5 \\ 4/5 & (-6\sqrt{5})/25 & (-3\sqrt{5})/25 \\ 3/5 & (8\sqrt{5})/25 & (4\sqrt{5})/25 \end{bmatrix} \cdot \begin{bmatrix} 5 & 4 & 0 \\ 0 & 5/\sqrt{5} & -1/\sqrt{5} \\ 0 & 0 & 2/\sqrt{5} \end{bmatrix}$$

Nejprve je nutné vyřešit pravou stranu, tedy Q^Tb:

$$\begin{bmatrix} 0 & 4/5 & 3/5 \\ -\sqrt{5}/5 & (-6\sqrt{5})/25 & (8\sqrt{5})/25 \\ (2\sqrt{5})/5 & (-3\sqrt{5})/25 & (4\sqrt{5})/25 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$$

Takto získaný vektor nyní stačí aplikovat na levou stranu na které je Rx.

$$\begin{bmatrix} 5 & 4 & 0 \\ 0 & 5/\sqrt{5} & -1/\sqrt{5} \\ 0 & 0 & 2/\sqrt{5} \end{bmatrix} \cdot \mathrm{x} = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$$

Pomocí zpětného chodu jsou dopočítány jednotlivé složky vektoru x. Pro zadaný příklad vycházejí rovnice $\frac{2}{\sqrt{5}} \cdot x_3 = 0$; $\frac{5}{\sqrt{5}} \cdot x_2 - \frac{1}{\sqrt{5}} \cdot x_3 = 0$ a $5 \cdot x_1 + 4 \cdot x_2 = 5$ ze kterých je patné, že $x_1 = 1$; $x_2 = 0$; $x_3 = 0$. Řešením soustavy lineárních rovnic je tedy vektor $x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$.

4.2 Výpočet pseudoinverzních matic

Nechť existuje matice A. Pokud je tato matice čtvercová a její determinant je různý od nuly, lze zjistit její inverzi A^{-1}. Pojem inverze lze zobecnit na tzv. pseudoinverzi, která se používá i pro singulární matice nebo pro obdélníkové matice. V takovém případě mluvíme o Moore–Penroseově pseudoinverzi. Pro pseudoinverzi A^+ musí platit relace:

$$A \cdot A^+ \cdot A = A$$
$$A^+ \cdot A \cdot A^+ = A^+$$
$$(A \cdot A^+)^T = A \cdot A^+$$
$$(A^+ \cdot A)^T = A^+ \cdot A.$$

Pseudoinverzní matice splňující tyto relace je jedinečná a vždy existuje. Pro matice, které lze invertovat standartní cestou je pseudoinverzní matice shodná s inverzní. [18]

Při výpočtu pseudoinverzní matice A^+ pomocí QR rozkladu matice $A = QR$ je nutno nejprve tento součin poupravit na vhodný tvar pomocí aplikace pseudoinverze na obě strany, vychází tedy $A^+ = (QR)^+$. Nyní je nutné odstranit závorku pomocí následujícího vzorce $(A \cdot B)^+ = B^+ \cdot A^+$ díky tomu vzniká $A^+ = R^+ \cdot Q^+$. Jelikož je matice R čtvercová a její determinant není roven nule platí $R^+ = R^{-1}$. Pro matici Q platí, že je ortogonální, tudíž platí $Q^+ = Q^T$. Tímto přeznačením vzniká součin $A^+ = R^{-1} \cdot Q^T$ podle kterého je vypočítána pseudoinverze. [19]

Inverzní matice R^{-1} se dopočítá pomocí zpětné substituce ve tvaru $R \cdot r'_i = e_i$, kde i je index sloupce. Takto získané vektory r'_i budou tvořit inverzní matici R^{-1}.

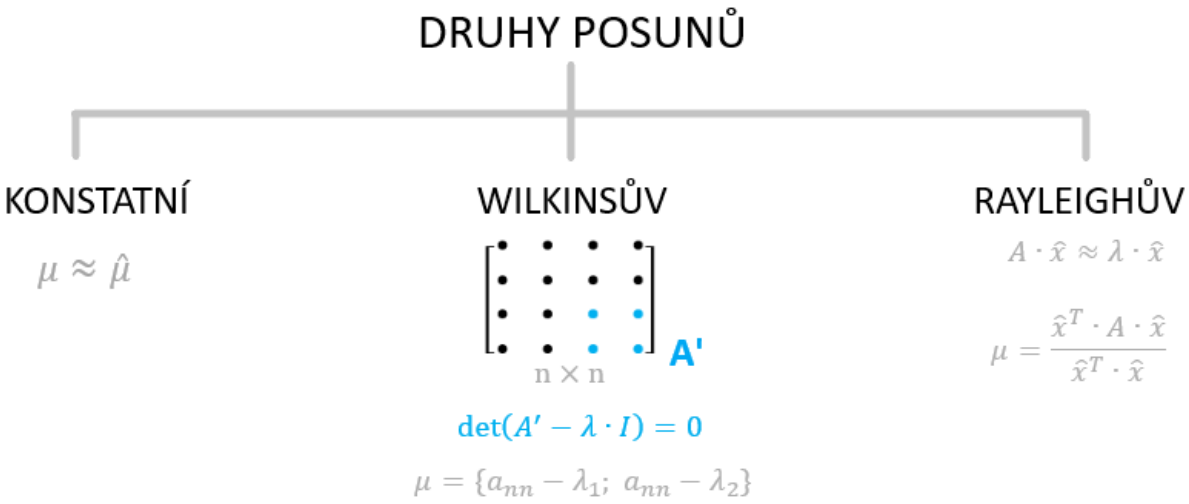
Na ukázku bude volena matice $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$ u které je nutno spočítat redukovaný QR-rozklad $\begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{6} \\ 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & \sqrt{2/3} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{2} & 1/\sqrt{2} \\ 0 & \sqrt{3/2} \end{bmatrix}$. Nyní bude provedena transpozice matice Q, která vyjde $Q^T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{6} & -1/\sqrt{6} & \sqrt{2/3} \end{bmatrix}$. Matice R^{-1} se dopočítá pomocí zpětné substituce a vyjde $R^{-1} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & \sqrt{2/3} \end{bmatrix}$. Nyní už stačí dopočítat pseudoinverzi vzorcem $A^+ = R^{-1} \cdot Q^T$. Po dosazení vychází $\begin{bmatrix} 1/3 & 2/3 & -1/3 \\ 1/3 & -1/3 & 2/3 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{6} \\ 0 & \sqrt{2/3} \end{bmatrix} \cdot \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{6} & -1/\sqrt{6} & \sqrt{2/3} \end{bmatrix}$. Takto vypočítaná matice je pseudoinverzi zadané matice.

4.3 Výpočet vlastních čísel a vlastních vektorů matice

Tato podkapitola bude věnována vlastním číslům matice, jelikož pojem vlastní číslo v kontextu obdélníkových matic neexistuje, proto bude tato část věnovat pouze čtvercovým maticím.

Při výpočtu vlastních čísel při využití QR-rozkladu se nejprve pro zadanou matici A najde QR-rozklad, tedy $A_0 = Q_0 \cdot R_0$. Dále se najde matice A_1 pomocí součinu $A_1 = R_0 \cdot Q_0$, pro tuto matici se opět hledá QR-rozklad $A_1 = Q_1 \cdot R_1$ a poté se najde matice A_2 obdobným způsobem jako A_1. Po několika iteracích tento algoritmus může konvergovat k matici, která obsahuje vlastní čísla na diagonále. Pokud tomu tak je, budou veškeré hodnoty pod hlavní diagonálou rovny téměř nule. Avšak existují případy, kdy konvergence nefunguje, proto se využívají pomocné metody jako deflace a posun pro zajištění správné a efektivní funkce algoritmu. [19]

Pomocná metoda posunu spočívá v odečtení násobku jednotkové matice od matice A před provedením QR-rozkladu, zapsáno $A_i - \mu I = Q_i R_i$. Hodnota μ se nazývá posun a její hodnota je volena různými strategiemi (např. Konstantní posun, Wilkinsův posun, Rayleighův posun atd). Při další iteraci je k násobek matice přičten zpět k nové matici $A_{i+1} = R_i \cdot Q_i + \mu I$. [20]



Obrázek 34 – Druhy posunů u výpočtu vlastních čísel přes QR-rozklad

Metoda deflace redukuje matici na menší části, pokud se pod diagonálou vyskytne téměř nulový prvek. Jelikož je nalezeno vlastní číslo, je možné provést deflaci daného úseku. Matice je poté zmenšena a algoritmus pokračuje dále. [21]

Pro demonstraci je volena matice $A_0 = \begin{bmatrix} 5 & 1 \\ 1 & 5 \end{bmatrix}$ s QR rozkladem $\begin{bmatrix} 5/\sqrt{26} & 1/\sqrt{26} \\ -1/\sqrt{26} & 5/\sqrt{26} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{26} & 5\sqrt{2/13} \\ 0 & 12\sqrt{2/13} \end{bmatrix}$. Tyto matice se využijí ve výpočtu $A_1 = R_0 \cdot Q_0$, která vyjde $\begin{bmatrix} 60/13 & 38/13 \\ -12/13 & 60/13 \end{bmatrix}$. QR-rozklad matice A_1 je $\begin{bmatrix} 5/\sqrt{26} & -1/\sqrt{26} \\ 1/\sqrt{26} & 5/\sqrt{26} \end{bmatrix} \cdot \begin{bmatrix} 12\sqrt{2/13} & 5\sqrt{2/13} \\ 0 & \sqrt{26} \end{bmatrix}$. Matice A_2 je dopočítána $A_2 = R_1 \cdot Q_1$ a vychází $\begin{bmatrix} 5 & 1 \\ 1 & 5 \end{bmatrix}$. Algoritmus aktuálně stagnuje, protože vygeneroval matici A_2 stejnou s maticí A_0, je tedy nutné provést posun s hodnotou μ, která lze volit podle různých strategií.

Nejprve bude demonstrován konstantní posun s hodnotou $\mu = 4{,}5$, tato hodnota by měla být co nejblíže některému z vlastních čísel. Matici A_2 je nutné upravit dle vzorce $A_2 - \mu I = Q_2 R_2$. Matice po odečtení vyjde $\begin{bmatrix} 1/2 & 1 \\ 1 & 1/2 \end{bmatrix}$ a její QR-rozklad je $\begin{bmatrix} 1/\sqrt{5} & 2/\sqrt{5}1 \\ 2/\sqrt{5} & -1/\sqrt{5} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{5}/2 & 2/\sqrt{5} \\ 0 & 3/(2\sqrt{5}) \end{bmatrix}$. Následuje vzorec $A_3 = R_2 \cdot Q_2 + \mu I$, tedy $A_3 = \begin{bmatrix} 13/10 & 3/5 \\ 3/5 & -3/10 \end{bmatrix} + \begin{bmatrix} 45/10 & 0 \\ 0 & 45/10 \end{bmatrix} = \begin{bmatrix} 58/10 & 3/5 \\ 3/5 & 42/10 \end{bmatrix}$. Proces se nyní opakuje, než bude docílena požadovaná přesnost.

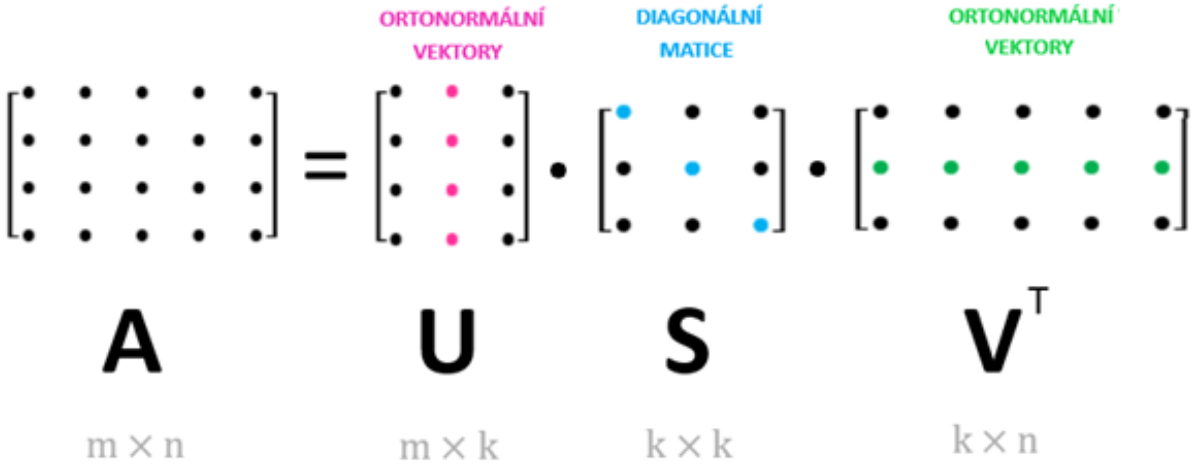
Další variantou je Wilkinsonův posun, který se volí jako minimum rozdílu hodnoty vlastních čísel spodního diagonálního bloku o velikosti 2x2 od hodnoty a_{nn}. Jelikož je zvolená matice 2x2 vypočítají se její vlastní čísla klasickou metodou dle vzorce $\det(A - \lambda \cdot I) = 0$. Tyto vlastní čísla vychází $\lambda_1 = 6$ a $\lambda_2 = 4$, algoritmus tímto krokem končí. Pokud by se jednalo o větší matici, vybere se vhodné vlastní číslo podle vzorce $\mu = \min\{a_{nn} - \lambda_1;\ a_{nn} - \lambda_2\}$. Matice se poté upraví příslušným vzorcem $A_2 - \mu I = Q_2 R_2$. Vypočítá se QR-rozklad a vytvoří se matice $A_3 = R_2 \cdot Q_2 + \mu I$ a pokračuje se dalšími iteracemi.

Poslední variantou, která zde bude uvedena, je Rayleighův posun. Hodnota μ volena dle vzorce $\mu = \frac{\hat{x}^T \cdot A \cdot \hat{x}}{\hat{x}^T \cdot \hat{x}}$. Zde je nutné zvolit vhodný odhad vektoru $\hat{x}$. Pro zvolený příklad bude jeho hodnota volena jako první sloupec matice A_2 tedy $\hat{x} = \begin{bmatrix} 5 \\ 1 \end{bmatrix}$. Hodnota μ vychází $\mu = \frac{[5 \quad 1] \cdot \begin{bmatrix} 5 & 1 \\ 1 & 5 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 1 \end{bmatrix}}{[5 \quad 1] \cdot \begin{bmatrix} 5 \\ 1 \end{bmatrix}} = \frac{70}{13}$ a bude použita pro úpravu matice A_2 tedy $A_2 - \mu I = Q_2 R_2$. Po dosazení vychází matice $\begin{bmatrix} -5/13 & 1 \\ 1 & -5/13 \end{bmatrix}$ jejíž QR rozklad je $\begin{bmatrix} -5/\sqrt{194} & 13/\sqrt{194} \\ 13/\sqrt{194} & 5/\sqrt{194} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{194}/13 & -5\sqrt{2/97} \\ 0 & (72\sqrt{2/97})/13 \end{bmatrix}$. Nyní je nutné provést výpočet $A_3 = R_2 \cdot Q_2 + \mu I$, tedy $A_3 = \begin{bmatrix} 5460/1261 & 72/97 \\ 72/97 & 7150/1261 \end{bmatrix}$. Algoritmus bude pokračovat, než bude docílena potřebná přesnost.

Jednotlivé metody posunu ukazují, že pro zadaný příklad při použití více iterací vycházejí vlastní čísla rovny $\lambda_1 = 6$ a $\lambda_2 = 4$. Při výpočtu lze použít i deflaci, která je vhodná spíše pro matice větších rozměrů. [22]

4.4 Výpočet SVD rozkladu matice

Každá matice A o m řádcích a n sloupcích lze rozložit na singulární rozklad (tzv. SVD), tento rozklad je ve tvaru $A = U \cdot S \cdot V^T$ kde U a V^T jsou reálné ortogonální matice. Matice U je rozměru $m \times m$ se skládá z levých singulárních vektorů a matice V, která je rozměru $n \times n$,naopak z pravých singulárních vektorů. Matice S v diagonále obsahuje nezáporná čísla (nazývaná singulární čísla), která jsou seřazena sestupně.



Obrázek 35 – SVD rozklad

Postup pro provedení SVD rozkladu je následující:

1. Vytvoří se matice $A^T \cdot A$
2. Nalezne se následující rozklad $A^T \cdot A = V \cdot D \cdot V^T$ (tento rozklad se nazývá spektrální)
   - pro matici $A^T \cdot A$ budou spočítána vlastní čísla a vlastní vektory
   - vlastní vektory matice $A^T \cdot A$ tvoří matici V (nutno najít vhodné pořadí a normovat)
   - vlastní čísla matice $A^T \cdot A$ jsou seřazena sestupně na diagonále matice D
3. Získají se singulární hodnoty do matice S (odmocněním vlastních čísel matice D)
4. Vytvoří se matice $A \cdot V$
5. Vyřeší se rovnice $U \cdot S = A \cdot V$ (neznámá je U)
   - tuto rovnici lze řešit pomocí QR-rozkladu (platí Q = U)
   - pokud nesedí rozměr matice je nutné ji doplnit
6. Sestavení $A = U \cdot S \cdot V^T$

Z daného postupu je patrné, že QR-rozklad v tomto případě lze použít při výpočtu vlastních čísel a při řešení rovnice $U \cdot S = A \cdot V$. Některé akce v tomto algoritmu mohou být nestabilní, a tudíž se používá jejich stabilnější verze založená na bidiagonalizaci. [23]

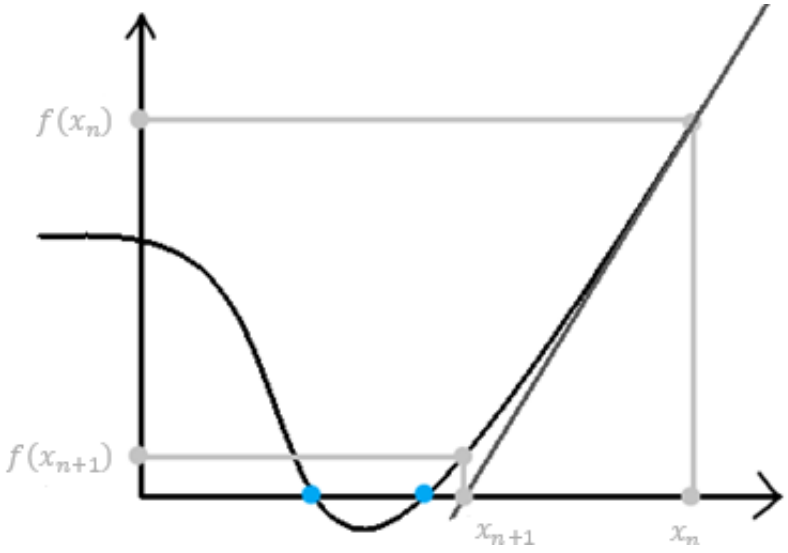
Pro demonstraci SVD rozkladu byla převzata matice a postup výpočtu ze zdroje [24], konkrétně se jednalo o matici $A = \begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & -2 \end{bmatrix}$. Nejprve bylo nutné dopočítat matici $A^T \cdot A$, která vychází $\begin{bmatrix} 5 & 1 \\ 1 & 5 \end{bmatrix}$. Dalším krokem je spektrální rozklad této matice. Pro zjištění matice D a V budou potřeba vypočítat vlastní čísla a vlastní vektory matice $\begin{bmatrix} 5 & 1 \\ 1 & 5 \end{bmatrix}$. Tento postup je popsán v předchozí podkapitole. Vlastní vektory vycházejí $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ a $v_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ a vlastní čísla pro zadanou matici vycházejí $\lambda_1 = 6$ a $\lambda_2 = 4$. Dané vlastní vektory se znormují na $v_1 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$ a $v_2 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$. Matice spektrálního rozkladu vycházejí následovně $D = \begin{bmatrix} 6 & 0 \\ 0 & 4 \end{bmatrix}$; $V = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$ a $V^T = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$. Singulární hodnoty matice S vycházejí $\delta_1 = \sqrt{6}$ a $\delta_1 = 2$. Nyní je nutné vypočítat součin $A \cdot V$, kterým vyjde matice $\begin{bmatrix} \sqrt{2} & \sqrt{2} \\ \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{2} \end{bmatrix}$. Nyní je nutné vyřešit soustavu lineárních rovnic $U \cdot S = A \cdot V$ pomocí QR rozkladu. Pro výpočet byla použita Grammova-Schmidtova metoda. QR-rozklad vychází následovně $\begin{bmatrix} \sqrt{2} & \sqrt{2} \\ \sqrt{2} & 0 \\ -\sqrt{2} & \sqrt{2} \end{bmatrix} = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} \\ 1/\sqrt{3} & 0 \\ -1/\sqrt{3} & 1/\sqrt{2} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 2 \end{bmatrix}$. První dva sloupce matice U tedy vycházejí $\begin{bmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ -1/\sqrt{3} \end{bmatrix}$ a $\begin{bmatrix} 1/\sqrt{2} \\ 0 \\ 1/\sqrt{2} \end{bmatrix}$. V tuto chvíli je třeba dopočítat třetí vektor, pro zvolený případ lze použít vektorový součin, případně využít Grammova-Schmidtův proces. Vektorový součin $u_1 \times u_2 = \begin{vmatrix} u_{31} & u_{32} & u_{33} \\ 1/\sqrt{3} & 1/\sqrt{3} & -1/\sqrt{3} \\ 1/\sqrt{2} & 0 & 1/\sqrt{2} \end{vmatrix}$. Z vektorového součinu vyplývá, že zbývající vektor $u_3 = \begin{bmatrix} -1/\sqrt{6} \\ 2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix}$. Po doplnění do matice U tedy vychází $\begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & -1/\sqrt{6} \\ 1/\sqrt{3} & 0 & 2/\sqrt{6} \\ -1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{6} \end{bmatrix}$.

Pro zadaný příklad vychází SVD následovně:

$$\begin{bmatrix} 2 & 0 \\ 1 & 1 \\ 0 & -2 \end{bmatrix} = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{2} & -1/\sqrt{6} \\ 1/\sqrt{3} & 0 & 2/\sqrt{6} \\ -1/\sqrt{3} & 1/\sqrt{2} & 1/\sqrt{6} \end{bmatrix} \cdot \begin{bmatrix} \sqrt{6} & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ 1/\sqrt{2} & -1/\sqrt{2} \end{bmatrix}$$

4.5 Výpočet kořenů polynomu

Je-li dán polynom $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ lze pro tento polynom sestavit přidruženou matici (tzv Frobeniovu matici) ve tvaru $P = \begin{bmatrix} 0 & 0 & \dots & 0 & -a_0/a_n \\ 1 & 0 & \dots & 0 & -a_1/a_n \\ 0 & 1 & \dots & 0 & -a_2/a_n \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -a_{n-1}/a_n \end{bmatrix}$. Kořeny polynomu odpovídají vlastním číslům přidružené matice. Pro přesnější výpočet kořene polynomu se přidává více iterací v algoritmu, případně se použije Newtonova metoda, tato metoda využívá vzorec $x_{n+1} = x_n - \frac{p(x)}{p'(x)}$, kde x_{n+1} je přesnější hodnota daného kořene. Na obrázku níže je interpretována vizuálně. [25] [26]



Obrázek 36 – Newtonova metoda zpřesnění kořenu polynomu

Z obrázku vyplývá, že původní hodnota kořene x_n se po aplikaci vzorce přiblíží k opravdovému kořeni polynomu označeného modrou tečkou.

Pro demonstraci výpočtu jednotlivých kořenů pomocí QR-rozkladu bude volen následující polynom $x^3 - 6x^2 + 11x - 6$, u kterého bude třeba najít jeho kořeny. Nejprve se sestaví příslušná přidružená matice $P = \begin{bmatrix} 0 & 0 & 6 \\ 1 & 0 & -11 \\ 0 & 1 & 6 \end{bmatrix}$. Na tuto Frobeniovu matici bude aplikován algoritmus pro hledání vlastních čísel matice popsaný v práci dříve. Po několika iteracích bez posunu vychází matice s vlastním čísly $\lambda_1 = 3$; $\lambda_2 = 2$ a $\lambda_3 = 1$, které odpovídají kořenům zadaného polynomu. Tyto hodnoty byly zpřesněny pomocí Newtonovy metody.

4.6 Možnosti použití QR rozkladu v oboru IT

QR-rozklad je základní numerická metoda lineární algebry s uplatněním různých oblastech oboru IT. V této sekci práce bude demonstrováno několik možných využití v oblasti AI, zpracování obrazu a videa, animací, a nakonec v kryptografii.

Oblast umělé inteligence

QR-rozklad lze podle zdroje [27] použít v hlubokých neuronových sítí jako algoritmus pro redukci zapomínání dat při učení nových dat. Kromě této aplikace lze QR-rozklad využít při backpropagaci v neuronových sítích, který se využívá v řadě aplikací v oblasti strojového učení, touto problematikou se zabývá zdroj [28]. Ve zdroji [29] je uvedeno jakým způsobem lze pomocí QR-rozkladu redukovat dimenzionalitu datasetu MNIST. Poslední uvedený zdroj [30] využívá opět dataset MNIST, tentokrát je představena optická neuronová síť využívající QR-rozklad a zároveň SVD rozklad.

Oblast zpracování obrazu a videa, animace

V této oblasti se podle zdroje [31] dá QR-rozklad použít pro detekci pozadí v sekvenci snímků. Zdroj [32] uvádí, že se QR-rozklad dá využít při detekci střihů a přechodů ve videu. Další aplikací je auto kalibrace kamery ve scéně pomocí SVD rozkladu matice, jak uvádí zdroj [33]. V oblasti animací lze QR-rozklad využít pro tvorbu animace obličeje při řeči. Pro tvorbu animace je nutné, aby daný subjekt měl na obličeji vytvořené značky, ze kterých bude animace vytvořena, více informací uvádí zdroj [34].

Oblast kryptografie

Ve zdroji [35] je uvedena QR-Crypt šifra, která je spjata s protokolem využívající QR-rozkladu. Kromě této aplikace lze zde zdroje [36] implementovat šifrovací algoritmus, který pomocí QR-rozkladu a modula 68 šifruje a dešifruje poslanou zprávu. Tento algoritmus má i vzorový příklad na kterém jsou dané akce znázorněny. Ve zdroji [37] je uveden způsob jakým lze pomocí QR-rozkladu šifrovat otisk prstu pomocí nové asymetrické šifry. Posledním uvedeným zdrojem z této oblasti bude možnost vkládání barevného vodoznaku do barevného obrázku za využití QR rozkladu, touto problematikou se zabývá zdroj [38].

Z této části textu je patrné, že se QR-rozklad dá využít v mnoha oblastech oboru IT. V textu práce byly zmíněny byly pouze vybrané příklady. V dnešní době QR-rozklad představuje robustní numerickou metodu, která je jedním z klíčových nástrojů pro mnoho algoritmů v IT.

5 ZÁVĚR

Tato diplomová práce se zaměřila na QR rozklad řídkých matic. V teoretické části byly popsány klíčové pojmy lineární algebry, způsoby reprezentace matic, zmíněn zde byl i LU rozklad. Důležitou částí práce bylo představení jednotlivých metod výpočtu QR rozkladu, mezi které patřila klasická a modifikovaná Grammova-Schmidtova metoda, Householderova transformace a Givensova rotace.

V rámci praktické části byla vyvinuta konzolová aplikace v jazyce C++, jejímž cílem bylo umožnit provádění QR rozkladu pomocí různých metod pro různé formáty řídkých matic. Před samotnou implementací byly stanoveny funkční a nefunkční požadavky, které sloužily jako základ pro návrh a realizaci aplikace. Během vývoje byla průběžně vytvářena programátorská příručka. Pro snadnější ovládání a orientaci v aplikaci byla vytvořena také uživatelská příručka, jež umožňuje čtenáři snadné spuštění. Správnost implementovaných metod byla ověřena pomocí unit testů, které byly vytvořeny jako samostatný projekt. V závěrečné fázi byly jednotlivé metody QR rozkladu porovnány z hlediska časové a paměťové složitosti. Toto porovnání bylo provedeno jak teoreticky pomocí analýzy asymptotické složitosti, tak i experimentálně na konkrétních vstupech v podobě matic různých rozměrů. Z této části vyplývá, že Householderova transformace u měřených matic vykazuje vyšší časovou náročnost než Givensova rotace, zejména při použití formátu COO, kde dochází k výraznému nárůstu výpočetního času s rostoucím rozměrem matice. Naproti tomu formáty CSR a CSC jsou z hlediska času výrazně efektivnější. Z hlediska paměťové náročnosti je opět nejméně výhodný formát COO, a to především u Householderovy metody. Naopak Givensova rotace vykazuje nižší paměťové nároky, přičemž u formátů CSR a CSC jsou rozdíly mezi metodami minimální.

V závěru práce byla představena řada praktických aplikací QR rozkladu jako řešení soustav lineárních rovnic, výpočet inverzních matic, výpočet vlastních čísel či numerické hledání kořenů polynomů, v závěrečné části je ukázka konkrétních problémů z oblasti IT.

POUŽITÁ LITERATURA

1. ROKYTA, Mirko. *Matice a determinanty.* [online] Praha : Matematická sekce Matematicko-fyzikální fakulta Univerzita Karlova, [citace 10.07.2025]. Dostupné z https://www.karlin.mff.cuni.cz/~rokyta/vyuka/1011/ls/F_apl_mat/ApMat_Kap_8_beamer.pdf.

2. PAJEROVÁ, Nikola. *Matematika 1.* [online] Praha : ČVUT, Fakulta strojní, [citace 11.07.2025]. Dostupné z https://users.fs.cvut.cz/nikola.pajerova/M1_matice.pdf.

3. MAI, Thanh Quang. *Efektivní násobení řídkých matic.* [online] Praha : ČVUT, Fakulta informačních technologii, [citace: 11.07.2025]. Dostupné z https://dspace.cvut.cz/bitstream/handle/10467/83579/F8-BP-2019-Mai-Thanh%20Quang-thesis.pdf?sequence=-1&isAllowed=y.

4. MERTA, Michal. *Numerická lineární algebra 1 Řídké matice, grafika v Matlabu.* [online] Ostrava : Technická univerzita Ostrava, [citace 20.07.2025]. Dostupné z https://homel.vsb.cz/~mer126/NLA1/Lectures/3/NLA1_3.pdf.

5. DONALDSON, Neil. *Elementary Matrix Operations and Systems of Linear Equations.* [online] California : University of California, Department of Mathematics, , [citace 15.07.2025]. Dostupné z https://www.math.uci.edu/~ndonalds/math121a/3elementary.pdf.

6. GARETH, Williams. *Linear algebra with applications.* Burlington : Jones & Bartlett Learning, 2014. ISBN: 978-1-4496-7954-5.

7. MERTA, Michal. *Numerická lineární algebra 1 QR rozklad.* [online] Ostrava : VŠB – Technická univerzita Ostrava, [citace: 23.03.2025]. Dostupné z https://homel.vsb.cz/~mer126/NLA1/Lectures/10/NLA1_11.pdf.

8. HLADÍK, Milan. *Matice.* [online] Praha : Univerzita Karlova Fakulta Aplikované matematiky, [citace 22.08.2025]. Dostupné z https://kam.mff.cuni.cz/~hladik/LA3/text_la3.pdf.

9. OBERHUBER, Tomáš. *QR algoritmus.* [online] Praha : ČVUT, Fakulta jaderná a fyzikálně inženýrská, [citace 26.04.2025]. Dostupné z https://geraldine.fjfi.cvut.cz/~oberhuber/data/vyuka/num-1/14-qr-rozklad.pdf.

10. STRANG, Gilbert. *Linear algebra and its applications.* 4th ed. Belmont : Thomson Brooks/Cole, 2006. ISBN: 0030105676.

11. KWOK, Anthony. *Detailed Explanation of QR Decomposition by Classical & Modified Gram-Schmidt Method.* [online] In : Medium, [citace: 16. 03. 2025]. Dostupné z https://kwokanthony.medium.com/important-decomposition-in-linear-algebra-detailed-explanation-on-qr-decomposition-by-classical-3f8f5425915f.

12. GANDER, Walter. *Algorithms for the QR decomposition.* Zürich : Technische Hochschule, 1980. Dostupné z https://people.inf.ethz.ch/gander/papers/qrneu.pdf.

13. KWOK, Anthony. *Detailed Explanation of QR Decomposition by Householder Transformation.* [online] In : Medium, [citace 23.03.2025]. Dostupné z https://kwokanthony.medium.com/detailed-explanation-with-example-on-qr-decomposition-by-householder-transformation-5e964d7f7656.

14. KWOK, Anthony. *Detailed Explanation of QR Decomposition by Givens Rotation.* [online] In : Medium, [citace 24.03.2025]. Dostupné z https://kwokanthony.medium.com/detailed-explanation-with-example-on-qr-decomposition-by-givens-rotation-6e7bf664fbdd.

15. RICHTA, Karel. *Složitost algoritmů.* [online] Praha : ČVUT, [citace 11.08.2025]. Dostupné z https://cw.fel.cvut.cz/old/_media/courses/b6b36dsa/dsa-3-slozitostalgoritmu.pdf.

16. CÍCHA, Jakub. *Maticové rozklady a jejich použití.* [online] Brno : Masarykova univerzita, Přírodovědecká fakulta, [citace 10.07.2025]. Dostupné z https://is.muni.cz/th/nasgc/DP_Cicha.pdf.

17. ZAJÍČKOVÁ, Markéta. *QR rozklad a jeho využití.* [online] Olomouc : Univerzita Palackého, Katedra matematické analýzy a aplikací matematiky, [citace: 01.07.2025]. Dostupné z https://library.upol.cz/arl-upol/en/csg/?repo=upolrepo&key=84330042910.

18. ŠTRONER, Martin. *Teorie chyb a vyrovnávací počet 1.* [online] Praha : ČVUT, Fakulta stavební, [citace: 16.03.2025]. Dostupné z https://k154.fsv.cvut.cz/~stroner/TCH1/tch_pred_6.pdf.

19. ZEMÁNEK, Petr. *QR-rozklad.* [online] Brno : Masarykova Univerzita, [citace 02.07.2025]. Dostupné z https://www.math.muni.cz/~zemanekp/files/QR-rozklad_%5Bseminarni_prace_-_Petr_Zemanek%5D.pdf.

20. KONECNY, Jan. *Algoritmy pro rozsáhlá data.* [online] Olomouc : Univerzita Palackého, Katedra informatiky, 2021. Dostupné z https://phoenix.inf.upol.cz/~konecnja/vyuka/2021W/ALS1/L10.pdf.

21. CIRBUS, Jan. *Implicitní QR algoritmus s násobnými shifty.* [online] Praha : Matematicko-fyzikální fakulta Univerzita Karlova, [citace 04.07.2025]. Dostupné z https://dl1.cuni.cz/pluginfile.php/1987032/mod_resource/content/1/BCprace_QRalgoritmus.pdf.

22. PERSSON, Per-Olof. *The QR Algorithm II.* [online] California : University of California,Department of Mathematics, [citace 07.07.2025]. Dostupné z https://dspace.mit.edu/bitstream/handle/1721.1/75282/18-335j-fall-2006/contents/lecture-notes/lec16.pdf.

23. MERTA, Michal. *Numerická lineární algebra 1 Singulární rozklad.* [online] Ostrava : VŠB – Technická univerzita Ostrava, [citace 03.07.2025]. Dostupné z https://homel.vsb.cz/~mer126/NLA1/Lectures/13/NLA1_13.pdf.

24. BARTO, Libor. *Cvičení k přednášce NMAG112 Lineární algebra 2.* [online] Praha : Matematická sekce Matematicko-fyzikální fakulta Univerzita Karlova, [citace 03.07.2025]. Dostupné z https://www2.karlin.mff.cuni.cz/~barto/LinAlg/CV_LA2_LS2122_11.pdf.

25. STAROSTA, Štěpán. *Mocninná metoda a QR algoritmus.* [online] Praha : ČVUT, Fakulta informačních technologií, [citace 02.07.2025]. Dostupné z https://kam.fit.cvut.cz/deploy/mi-mpi/mi-mpi-prednaska-23-vlastni-cisla.pdf.

26. ZLÁMALOVÁ, Lenka. *Numerické metody pro hledání.* [online] Brno : Masarykova univerzita, Přírodovědecká fakulta, 2006. Dostupné z https://is.muni.cz/th/i2va1/bakalarka.pdf.

27. JONGHONG, Kim. *Incremental Learning for Online Data Using QR Factorization on Convolutional Neural Networks.* [online] In : MDPI, [citace 09.07.2025]. Dostupné z https://www.mdpi.com/1424-8220/23/19/8117.

28. ROBERTS, Denisa A.O. *QR and LQ Decomposition Matrix Backpropagation Algorithms for Square, Wide, and Deep -- Real or Complex -- Matrices and Their Software Implementation.* [online] New York : Cornell University, [citace 09.07.2025]. Dostupné z https://arxiv.org/abs/2009.10071.

29. MOHSIN, Ali. *Investigation of Neural Network Parameters for MNIST Using QR Decomposition Algorithm and Principal Component Analysis.* [online] In : IEEE Xplore, [citace 09.07.2025]. Dostupné z https://ieeexplore.ieee.org/abstract/document/10486346.

30. LIN, Jian. *A Robust MZI-Based Optical Neural Network Using QR Decomposition.* [online] In : IEEE Xplore, [citace 09.07.2025]. Dostupné z https://ieeexplore.ieee.org/abstract/document/10707275.

31. AMINTOOSI, Mahmood. *QR Decomposition-Based Algorithm for Background Subtraction.* [online] In : IEEE Xplore, [citace 09.07.2025]. Dostupné z https://ieeexplore.ieee.org/abstract/document/4217274.

32. AMIRI, Ali. *Video Shot Boundary Detection Using QR-Decomposition.* [online] Iran : Computer Engineering Department, University of Science and Technology, [citace 09.07.2025]. Dostupné z https://link.springer.com/content/pdf/10.1155/2009/509438.pdf.

33. LOURAKIS, Manolis I.A. *Camera Self-Calibration Using the Singular Value Decomposition.* [online] In : Sophia-Antipolis Cedex, [citace 09.07.2025]. Dostupné z https://www.researchgate.net/publication/2441156_Camera_Self-Calibration_Using_the_Singular_Value_Decomposition_of_the_Fundamental_Matrix.

34. LUCERO, Jorge C. *Data-driven facial animation of speech using a QR.* [online] Brasília : Department of Mathematics, University of Brasília, [citace 09.07.2025]. Dostupné z https://www.researchgate.net/profile/Jorge-Lucero-2/publication/228950596_Data-driven_facial_animation_of_speech_using_a_QR_factorization_algorithm/links/02bfe513368746d395000000/Data-driven-facial-animation-of-speech-using-a-QR-factorization-a.

35. QAYUM, Abdul. *QR decomposition-based cryptography: Via image generation (QR-CRYPT).* [online] In : IEEE Xplore, [citace 09.07.2025]. Dostupné z https://ieeexplore.ieee.org/abstract/document/6470939.

36. KUMAR, Aparna Ashok. *Encryption and Decryption of messages using QR Decomposition.* [online] In : IEEE Xplore, [citace 09.07.2025]. Dostupné z https://ieeexplore.ieee.org/abstract/document/10563267.

37. MEHRA, Isha. *Fingerprint image encryption using phase retrieval algorithm in gyrator wavelet transform domain using QR decomposition.* [online] In : ScienceDirect, [citace 09.07.2025]. Dostupné z https://www.sciencedirect.com/science/article/abs/pii/S003040182300010X.

38. SU, Qingtang. *An improved color image watermarking algorithm based on QR decomposition.* [online] In : Springer Nature, [citace 09.07.2025]. Dostupné z https://link.springer.com/article/10.1007/s11042-015-3071-x.

39. BRADLEY, Tai-Danae. *Understanding Entanglement With SVD.* [online] In : Math3ma, [citace 04.07.2025]. Dostupné z https://www.math3ma.com/blog/understanding-entanglement-with-svd.

40. MAŘÍK, Robert. *Soustavy lineárních rovnic.* [online] Brno : Mendelova univerzita, Lesnická a dřevařská fakulta, [citace 18.07.2025]. Dostupné z https://user.mendelu.cz/marik/mtk/mat-slidy/soustavy/.

41. ROKYTA, Mirko. *Lineární vektorové prostory.* [online] Praha : Matematické sekce Matematicko-fyzikální fakulta Univerzita Karlova, [citace: 19.07.2025]. Dostupné z https://www.karlin.mff.cuni.cz/~rokyta/vyuka/1011/zs/F_apl_mat/ApMat_Kap_7_beamer.pdf.

42. STRANG, Gilbert. *Elimination Matrices and Inverse Matrices.* [online] Massachusett : Institute of Technology, Department of Mathematics, [citace: 21.07.2025]. Dostupné z https://math.mit.edu/~gs/linearalgebra/ila6/ila6_2_2.pdf.

43. TICHÝ, Petr. *LU rozklad a jeho numerická analýza.* [online] Praha : Matematická sekce Matematicko-fyzikální fakulta Univerzita Karlova, [citace 15.07.2025]. Dostupné z https://www.karlin.mff.cuni.cz/~ptichy/blogs/nan/NA_05_prednaska.pdf.

44. TŮMA, Jiří. *Gaussova eliminace.* [online] Praha : Matematická sekce Matematicko-fyzikální fakulta Univerzita Karlova, [citace 17.07.2025]. Dostupné z https://www.karlin.mff.cuni.cz/~tuma/2003/NNLinalg2.pdf.

SEZNAM PŘÍLOH