FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Maticová kalkulačka pro výpočet vlastních čísel
Student:	Matouš Bílek
Vedoucí:	doc. Ing. Ivan Šimeček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

1. Nastudujte formáty uložení řídkých matic[1].
2. Nastudujte restartovanou Arnoldiho metodu [1] pro výpočet několika vlastních čísel vstupní řídké matice.
3. Analyzujte knihovny s implementací restartované Arnoldiho metody, např. [2,3].
4. Navrhněte a implementujte kalkulačku několika vlastních čísel reálných nesymetrických matic. Implementujte pro ni rozhraní přes příkazovou řádku, společně s jednoduchým desktopovým rozhraním.
5. Výslednou implementaci produktu otestujte.

[1] Saad, Y.: Numerical Methods for Large Eigenvalue Problems. Society for Industrial and Applied Mathematics, 2011, doi:10.1137/1.9781611970739.
[2] Hernández, V.; Román, J.; Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Transactions on Mathematical Software (TOMS), ročník 31, č. 3, 2005: s. 351–362.
[3] Lehoucq, R. B.; Sorensen, D. C.; Yang, C.: ARPACK Users' Guide. Society for Industrial and Applied Mathematics, 1998, doi:10.1137/1.9780898719628.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 8. prosince 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Maticová kalkulačka pro výpočet vlastních čísel

Matouš Bílek

Katedra softwarového inženýrství
Vedoucí práce: doc. Ing. Ivan Šimeček, Ph.D.

9. května 2022

Poděkování

Děkuji svému vedoucímu práce, panu doc. Ing. Ivanu Šimečkovi, Ph.D., za možnost psát bakalářskou práci na toto pro mne zajímavé téma, za jeho čas a cenné rady.

Významné poděkování patří mým rodičům, kteří mi studium umožnili, motivovali mne, uznávali mne, pečovali o mne a povzbuzovali mne. Děkuji i své sestře a jejímu manželovi a celému zbytku rodiny za jejich podporu po celou dobu studia.

Tomu, který mne vedl a pomáhal mi.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo"), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu) licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2022 . . . . . .

Odkaz na tuto práci

Bílek, Matouš. *Maticová kalkulačka pro výpočet vlastních čísel*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Tato bakalářská práce se zabývá návrhem, implementací a otestováním kalkulačky několika největších vlastních čísel velkých řídkých nesymetrických reálných matic. Literární rešerše se proto zabývá popisem restartované Arnoldiho metody. Výpočetní část implementuje implicitně restartovanou Arnoldiho metodu. Kalkulačka je implementována v programovacím jazyce C++. Implementovaná kalkulačka poskytuje pro své řešení rozhraní přes příkazovou řádku společně s jednoduchým grafickým uživatelským rozhraním, které využívá framework Qt.

Klíčová slova kalkulačka, vlastní čísla, řídké matice, restartovaná Arnoldiho metoda, CLI, GUI, C++, Qt

Abstract

This bachelor's thesis deals with the design, implementation and testing of a calculator of a few largest eigenvalues of large sparse nonsymmetric real matrices. It hence deals with the description of the restarted Arnoldi method. The solver part implements the implicitly restarted Arnoldi method. The calculator is implemented in the C++ programming language. The implemented calculator provides for its solution a command line interface along with a simple graphical user interface which uses the framework Qt.

Keywords calculator, eigenvalues, sparse matrices, restarted Arnoldi method, CLI, GUI, C++, Qt

Obsah

Seznam obrázků

Seznam tabulek

Úvod

V informatice, matematice, jaderné fyzice a v mnoha dalších oborech nalézají své využití matice. Zacházení s těmito maticemi je často časově i paměťově náročné. Tato náročnost roste mnohdy i více než kvadraticky s velikostí těchto matic. Občas se ovšem stane, že má matice vzhledem ke své velikosti nezanedbatelný počet nulových prvků. A v takových případech nalézají své využití takové formáty uložení matic a takové metody pracující s takovými formáty, které tento fakt využívají.

Vlastní čísla vypovídají o některých vlastnostech dané matice. Někdy je potřeba vypočítat jen několik vlastních čísel z určité oblasti spektra řídké matice. Například pro odhad nejřidšího řezu grafu stačí vypočítat druhé nejmenší vlastní číslo Laplaceovy matice tohoto grafu. A v mnoha dalších oblastech se s prohlubujícími se znalostmi a s rostoucím výpočetním výkonem čím dál více začínají objevovat podobné problémy.

Restartovaná Arnoldiho metoda umožňuje aproximovat část spektra vstupní matice. Ve své základní verzi je nejefektivnější pro výpočet aproximací několika největších vlastních čísel vstupní řídké matice.

Tato práce tedy čtenáře seznamuje se základní verzí restartované Arnoldiho metody a potom navrhuje a implementuje kalkulačku několika vlastních čísel reálných nesymetrických matic s použitím této metody. K této kalkulačce pak navrhuje a implementuje rozhraní přes příkazovou řádku společně s grafickým uživatelským rozhraním.

Práce může být užitečná pro ostatní studenty píšící bakalářskou práci na podobné téma. Nebo pro studenty, kteří by na výsledek této práce chtěli navázat.

V kapitole cíl práce je popsán cíl této práce, společně s dílčími cíli.

V kapitole teoretický základ jsou nejprve definovány potřebné základní pojmy. Následně je zde čtenář seznámen se základními formáty uložení řídkých matic. Potom je zde popsána Arnoldiho metoda. V neposlední řadě je zde popsané její restartování.

V kapitole knihovny pro práci s maticemi jsou popsány vybrané knihovny, které poskytují implementaci maticových operací.

V kapitole přehled existujících řešení jsou popsány některé knihovny, které samy také implementují restartovanou Arnoldiho metodu.

V kapitole analýza požadavků jsou analyzovány funkční i nefunkční požadavky na produkt této bakalářské práce.

V kapitole návrh jsou zvoleny algoritmy pro realizaci výpočtu a vybrány technologie pro implementaci. V této kapitole je pak navržena architektura aplikace. Také je zde navržen vzhled grafického uživatelského rozhraní.

V kapitole realizace je popsána dokumentace implementovaného řešení a jeho adresářová struktura. Následně je zde popsána implementace výpočetní části, rozhraní přes příkazovou řádku a grafického uživatelského rozhraní.

V kapitole testování je otestováno splnění především funkčních požadavků na aplikaci. Je zde ověřeno splnění i nefunkčních požadavků.

V kapitole závěr je popsáno, jakým způsobem a do jaké míry byly splněny cíle této bakalářské práce. Obsahuje výhled několika možných rozšíření této práce do budoucna.

Cíl práce

Hlavním cílem této bakalářské práce je navrhnout a implementovat kalkulačku několika vlastních čísel reálných nesymetrických matic a implementovat pro ni rozhraní přes příkazovou řádku, společně s jednoduchým desktopovým rozhraním.

Cílem teoretické části práce je nastudovat restartovanou Arnoldiho metodu pro výpočet několika vlastních čísel vstupní řídké matice. Dílčím cílem je zavést potřebné teoretické pojmy. Navazujícím dílčím cílem je nastudovat formáty uložení řídkých matic. Dalším dílčím cílem je analyzovat knihovny s implementací restartované Arnoldiho metody. V neposlední řadě je cílem analyzovat funkční a nefunkční požadavky na implementovanou kalkulačku.

Cílem praktické části práce je navrhnout a implementovat kalkulačku několika vlastních čísel reálných nesymetrických matic, zpřístupnit ji přes příkazovou řádku a přes jednoduché grafické rozhraní. Dílčím cílem je vybrat vhodné technologie pro implementaci. V neposlední řadě je cílem výslednou implementaci produktu otestovat.

Kapitola 1

Teoretický základ

V této kapitole jsou zavedeny základní teoretické pojmy, které se využívají v navazujících kapitolách.

1.1 Základní pojmy

Následující definice základních pojmů čerpá tato práce ze studijního textu ke předmětu Lineární algebra [1] na FIT ČVUT, jestliže není odkázáno jinam. Jsou zavedeny pro komplexní čísla, aby bylo později možné definovat plnou podobu Arnoldiho algoritmu. Tyto definice mají přímočaré analogie pro množinu reálných čísel, která je podmnožinou množiny komplexních čísel, jsou však zmíněny některé speciální případy.

1.1.1 Matice a vektory

Definice 1.1.1 *Nechť $m, n \in \mathbb{N}$, kde $\mathbb{N} = \{1, 2, \ldots\}$ je množina přirozených čísel. Uspořádaný soubor mn čísel zapsaný do tabulky o m řádcích a n sloupcích se nazývá* ***matice*** *typu $m \times n$. Matice se obvykle značí takto:*

$$\mathbb{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

kde a_{ij} jsou ***prvky*** *matice a označují se také jako $\mathbb{A}_{ij}$. Číslu i se říká* ***řádkový*** *a číslu j* ***sloupcový*** *index.* ***Množina všech matic typu*** *$m \times n$ s komplexními prvky se značí $\mathbb{C}^{m,n}$. Jako $\mathbb{A}_{i:} \in \mathbb{C}^{1,n}$ se značí* ***$i$tý řádek*** *matice $\mathbb{A}$:*

$$\mathbb{A}_{i:} = \begin{pmatrix} a_{i1} & a_{i2} & \cdots & a_{in} \end{pmatrix}.$$

Podobně $\mathbb{A}_{:j} \in \mathbb{C}^{m,1}$ *značí* j***tý sloupec*** *matice* $\mathbb{A}$*:*

$$\mathbb{A}_{:j} = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}.$$

Nechť dále $k, l, o, p \in \mathbb{N}$ *tak, že* $1 \leq k \leq l \leq m$ *a* $1 \leq o \leq p \leq n$*. Potom* $\mathbb{A}_{k:l,o:p} \in \mathbb{C}^{l-k+1,p-o+1}$ *značí*

$$\mathbb{A}_{k:l,o:p} = \begin{pmatrix} a_{k,o} & a_{k,o+1} & \cdots & a_{k,p} \\ a_{k+1,o} & a_{k+1,o+1} & \cdots & a_{k+1,p} \\ \vdots & \vdots & \ddots & \vdots \\ a_{l,o} & a_{l,o+1} & \cdots & a_{l,p} \end{pmatrix}.$$

Jestliže $k = l$*, může se v dolním indexu namísto* $k{:}l$ *psát pouze* k*, tedy* $\mathbb{A}_{k,o:p} \in \mathbb{C}^{1,p-o+1}$*, podobně pro případ* $o = p$*. Pokud* $k = 1 \wedge l = m$*, píše se obvykle pouze* $:$*, totiž* $\mathbb{A}_{:,o:p} \in \mathbb{C}^{m,p-o+1}$*, stejně tak když* $o = 1 \wedge p = n$*.*

Dvě matice se rovnají, pokud jsou stejného typu a mají shodné všechny odpovídající prvky.

Definice 1.1.2 *Nechť* $m \in \mathbb{N}$*. Prvky* $\mathbb{C}^{m,1}$ *se nazývají* m*prvkové* ***vektory*** *a namísto* $\mathbb{C}^{m,1}$ *se píše pouze* $\mathbb{C}^m$*. Jako* ***nulový vektor*** *se nazývá vektor z* $\mathbb{C}^m$*, jehož všechny prvky jsou nuly, a značí se* θ*. Prvky množiny* $\mathbb{C}$ *se nazývají* ***skaláry****.*

1.1.2 Matice se speciální strukturou

Definice 1.1.3 *Nechť* $n \in \mathbb{N}$*. Každý prvek z* $\mathbb{C}^{n,n}$ *se nazývá* ***čtvercová matice****.*

Definice 1.1.4 *Nechť* $n \in \mathbb{N}$*.* ***Diagonálou*** *čtvercové matice* $\mathbb{A} \in \mathbb{C}^{n,n}$ *se nazývá vektor*

$$\begin{pmatrix} \mathbb{A}_{11} \\ \mathbb{A}_{22} \\ \vdots \\ \mathbb{A}_{nn} \end{pmatrix} \in \mathbb{C}^n$$

Definice 1.1.5 *Nechť* $n \in \mathbb{N}$*.* ***Diagonální maticí*** *typu* $n \times n$ *se nazývá libovolná čtvercová matice* $\mathbb{A} \in \mathbb{C}^{n,n}$ *splňující pro* $\forall i, j \in \{1, 2, \ldots, n\}$*:*

$$i \neq j \implies \mathbb{A}_{ij} = 0,$$

tedy všechny prvky mimo diagonálu matice $\mathbb{A}$ *jsou nulové.*

Definice 1.1.6 ***Kroneckerovo delta*** *se definuje předpisem:*

$$\delta_{ij} = \begin{cases} 1, & \text{pokud } i = j, \\ 0, & \text{jinak} \end{cases}$$

Definice 1.1.7 *Nechť $n \in \mathbb{N}$.* ***Jednotková matice*** *typu $n \times n$ se definuje jako čtvercová matice $\mathbb{E} \in \mathbb{C}^{n,n}$ splňující pro $\forall i, j \in \{1, 2, \ldots, n\}$*

$$\mathbb{E}_{ij} = \delta_{ij}$$

Komplexní jednotková matice je rovna reálné jednotkové matici.

Definice 1.1.8 *Nechť $n \in \mathbb{N}$.* ***Trojúhelníkovou maticí*** *typu $n \times n$ se nazývá libovolná čtvercová matice $\mathbb{A} \in \mathbb{C}^{n,n}$ splňující pro $\forall i, j \in \{1, 2, \ldots, n\}$:*

$$i > j \implies \mathbb{A}_{ij} = 0,$$

tedy všechny prvky pod diagonálou matice $\mathbb{A}$ jsou nulové.

Následující dvě definice jsou převzaty z knihy [2].

Definice 1.1.9 *Nechť $n \in \mathbb{N}$.* ***Hessenbergovou maticí*** *typu $n \times n$ se nazývá libovolná čtvercová matice $\mathbb{H} \in \mathbb{C}^{n,n}$ splňující pro $\forall i, j \in \{1, 2, \ldots, n\}$:*

$$i > j + 1 \implies \mathbb{H}_{ij} = 0$$

Definice 1.1.10 *Nechť $n \in \mathbb{N}$.* ***Kvazi-trojúhelníkovou maticí*** *typu $n \times n$ se nazývá libovolná Hessenbergova matice $\mathbb{A} \in \mathbb{C}^{n,n}$ splňující navíc pro $\forall i \in \{1, 2, \ldots, n-1\}$:*

$$\mathbb{A}_{i+1,i} \neq 0 \implies ((i \geq 2 \implies \mathbb{A}_{i,i-1} = 0) \wedge (i \leq n-2 \implies \mathbb{A}_{i+2,i+1} = 0))$$

1.1.3 Maticové operace

Definice 1.1.11 *Nechť $m, n \in \mathbb{N}, \alpha \in \mathbb{C}$ a $\mathbb{A} \in \mathbb{C}^{m,n}$ je komplexní matice. Pak* ***součin čísla*** *α* ***s maticí*** *$\mathbb{A}$ se definuje jako*

$$\alpha\mathbb{A} := \begin{pmatrix} \alpha\mathbb{A}_{11} & \alpha\mathbb{A}_{12} & \cdots & \alpha\mathbb{A}_{1n} \\ \alpha\mathbb{A}_{21} & \alpha\mathbb{A}_{22} & \cdots & \alpha\mathbb{A}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha\mathbb{A}_{m1} & \alpha\mathbb{A}_{m2} & \cdots & \alpha\mathbb{A}_{mn} \end{pmatrix}$$

Definice 1.1.12 *Nechť $m, n \in \mathbb{N}, \alpha \in \mathbb{C}$ a $\mathbb{A}, \mathbb{B} \in \mathbb{C}^{m,n}$ jsou komplexní matice.* ***Součet matic*** *$\mathbb{A}$ a $\mathbb{B}$ se definuje takto:*

$$\mathbb{A} + \mathbb{B} := \begin{pmatrix} \mathbb{A}_{11} + \mathbb{B}_{11} & \mathbb{A}_{12} + \mathbb{B}_{12} & \cdots & \mathbb{A}_{1n} + \mathbb{B}_{1n} \\ \mathbb{A}_{21} + \mathbb{B}_{21} & \mathbb{A}_{22} + \mathbb{B}_{22} & \cdots & \mathbb{A}_{2n} + \mathbb{B}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{A}_{m1} + \mathbb{B}_{m1} & \mathbb{A}_{m2} + \mathbb{B}_{m2} & \cdots & \mathbb{A}_{mn} + \mathbb{B}_{mn} \end{pmatrix}$$

Ihned lze pozorovat komutativitu i asociativitu maticového sčítání vyplývající z komutativity a asociativity sčítání komplexních čísel.

Definice 1.1.13 *Nechť $m, n, p \in \mathbb{N}$ a $\mathbb{A} \in \mathbb{C}^{m,n}, \mathbb{B} \in \mathbb{C}^{n,p}$ jsou komplexní matice.* ***Součinem matic*** *$\mathbb{A}$ a $\mathbb{B}$ je matice $\mathbb{D} \in \mathbb{C}^{m,p}$, pro kterou pro $\forall i \in \{1, 2, \ldots, m\}$ a pro $\forall j \in \{1, 2, \ldots, p\}$ platí*

$$\mathbb{D}_{ij} = \sum_{k=1}^{n} \mathbb{A}_{ik}\mathbb{B}_{kj},$$

což se značí $\mathbb{D} = \mathbb{A}\mathbb{B}$.

Vícenásobný součin toutéž maticí $\mathbb{A}\mathbb{A}\cdots\mathbb{A}$ se zkráceně značí $\mathbb{A}^k$, kde k je počet činitelů v tomto součinu.

S využitím této definice pak lze vyjádřit ijtý prvek součinu $\mathbb{D} = \mathbb{A}\mathbb{B}$ jako součin $i$tého řádku matice $\mathbb{A}$ a $j$tého sloupce matice $\mathbb{B}$, totiž $\mathbb{D}_{ij} = \mathbb{A}_{i:}\mathbb{B}_{:j}$.

Na rozdíl od sčítání, maticové násobení není komutativní, ale pouze asociativní. Neutrálním prvkem vůči maticovému součinu je jednotková matice.

Definice 1.1.14 *Nechť $m, n \in \mathbb{N}$ a $\mathbb{A} \in \mathbb{C}^{m,n}$ je komplexní matice.* ***Transpozicí*** *matice $\mathbb{A}$ se nazývá matice z $\mathbb{C}^{n,m}$, jejíž prvek v $j$tém řádku a $i$tém sloupci je roven $\mathbb{A}_{ij}$. Tato matice se značí $\mathbb{A}^T$. Tedy*

$$\mathbb{A}^T = \begin{pmatrix} \mathbb{A}_{11} & \mathbb{A}_{21} & \cdots & \mathbb{A}_{m1} \\ \mathbb{A}_{12} & \mathbb{A}_{22} & \cdots & \mathbb{A}_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{A}_{1n} & \mathbb{A}_{2n} & \cdots & \mathbb{A}_{mn} \end{pmatrix}$$

Definice 1.1.15 *Nechť $m, n \in \mathbb{N}$ a $\mathbb{A} \in \mathbb{C}^{m,n}$ je komplexní matice. Matice z $\mathbb{C}^{n,m}$, jejíž prvek v $j$tém řádku a $i$tém sloupci je roven $\overline{\mathbb{A}_{ij}}$, kde $\overline{z}$ pro $z \in \mathbb{C}$ představuje komplexně sdružené číslo k číslu z, se nazývá* ***hermitovsky sdruženou*** *maticí k matici $\mathbb{A}$. Tato matice se pak značí $\mathbb{A}^H$. Tedy*

$$\mathbb{A}^H = \begin{pmatrix} \overline{\mathbb{A}_{11}} & \overline{\mathbb{A}_{21}} & \cdots & \overline{\mathbb{A}_{m1}} \\ \overline{\mathbb{A}_{12}} & \overline{\mathbb{A}_{22}} & \cdots & \overline{\mathbb{A}_{m2}} \\ \vdots & \vdots & \ddots & \vdots \\ \overline{\mathbb{A}_{1n}} & \overline{\mathbb{A}_{2n}} & \cdots & \overline{\mathbb{A}_{mn}} \end{pmatrix}$$

Hermitovsky sdružená matice k reálné matici se rovná její transpozici.

1.1.4 Matice se speciálními vlastnostmi

Definice 1.1.16 *Nechť $n \in \mathbb{N}$.* ***Symetrickou maticí*** *se nazývá libovolná čtvercová matice $\mathbb{A} \in \mathbb{C}^{n,n}$ splňující*

$$\mathbb{A} = \mathbb{A}^T$$

Definice 1.1.17 *Nechť $n \in \mathbb{N}$.* ***Hermitovskou maticí*** *se nazývá libovolná čtvercová matice $\mathbb{A} \in \mathbb{C}^{n,n}$ splňující*

$$\mathbb{A} = \mathbb{A}^H$$

Ihned lze zpozorovat, že reálná matice je hermitovská právě tehdy, když je symetrická.

Definice 1.1.18 *Nechť $n \in \mathbb{N}$, $\mathbb{A} \in \mathbb{C}^{n,n}$. Jestliže $\exists \mathbb{B} \in \mathbb{C}^{n,n}$ tak, že platí*

$$\mathbb{A}\mathbb{B} = \mathbb{E} \wedge \mathbb{B}\mathbb{A} = \mathbb{E},$$

matice $\mathbb{A}$ se nazývá ***regulární*** *a matice $\mathbb{B}$ se nazývá* ***inverzní maticí*** *k matici $\mathbb{A}$. To se značí $\mathbb{B} = \mathbb{A}^{-1}$. Jestliže $\mathbb{A}$ není regulární, nazývá se matice $\mathbb{A}$* ***singulární****.*

Lze ukázat, že inverze regulární matice je určena jednoznačně a že je možné ekvivalentně nahradit v předchozí definici konjunkci disjunkcí, totiž že stačí ověřit pouze jednu rovnost.

Zdrojem následujících dvou definic je kniha [3].

Definice 1.1.19 *Nechť $n \in \mathbb{N}$ a $\mathbb{E} \in \mathbb{R}^{n,n}$ je jednotková matice.* ***Ortogonální maticí*** *se nazývá libovolná matice $\mathbb{Q} \in \mathbb{R}^{n,n}$ splňující*

$$\mathbb{Q}^T\mathbb{Q} = \mathbb{E}$$

Definice 1.1.20 *Nechť $n \in \mathbb{N}$ a $\mathbb{E} \in \mathbb{C}^{n,n}$ je jednotková matice.* ***Unitární maticí*** *se nazývá libovolná matice $\mathbb{Q} \in \mathbb{C}^{n,n}$ splňující*

$$\mathbb{Q}^H\mathbb{Q} = \mathbb{E}$$

Ihned si lze povšimnout toho, že reálná matice je unitární právě tehdy, když je ortogonální.

Je dobré zmínit, že inverzní matice k unitární matici existuje a je rovna jejímu hermitovskému sdružení.

1.1.5 Vlastní čísla a vlastní vektory

Definice 1.1.21 *Nechť $\mathbb{A} \in \mathbb{C}^{n,n}$, kde $n \in \mathbb{N}$, je čtvercová matice. Komplexní číslo $\lambda \in \mathbb{C}$ se nazývá* ***vlastním číslem matice*** *$\mathbb{A}$, právě když existuje nenulový vektor $\mathbb{x} \in \mathbb{C}^n$ splňující*

$$\mathbb{A}\mathbb{x} = \lambda\mathbb{x}.$$

Takovýto vektor $\mathbb{x}$ se pak nazývá ***vlastním vektorem matice*** *$\mathbb{A}$* ***příslušejícím vlastnímu číslu*** *λ. Množina všech vlastních čísel matice $\mathbb{A}$ se nazývá* ***spektrem matice*** *$\mathbb{A}$ a označuje se $\sigma(\mathbb{A})$.*

Je vhodné poznamenat, že hermitovská matice má reálné spektrum.

1.2 Formáty uložení řídkých matic

Tato kapitola čerpá z knihy [2], jestliže není uvedeno jinak.

Protože se tato práce primárně zabývá výpočtem několika vlastních čísel řídkých reálných matic, jsou definice a pojmy vztahující se především k řídkým maticím zavedeny nad reálnými čísly.

1.2.1 Definice řídké matice

Definice 1.2.1 *Nechť $m, n \in \mathbb{N}$ a $\mathbb{A} \in \mathbb{R}^{m,n}$. Počet nenulových prvků matice $\mathbb{A}$ je definován takto:*

$$nnz(\mathbb{A}) := \#\{(i,j) \mid i \in \{1,\dots,m\}, j \in \{1,\dots,m\}, \mathbb{A}_{ij} \neq 0\},$$

kde $\#M$ znamená počet prvků množiny M. Jestliže je zřejmé, o jakou matici se jedná, pak lze použít značení $NNZ := nnz(\mathbb{A})$.

NNZ je zkratkou pro anglické *Number of Nonzeros*.

Existuje několik různých definic řídké matice.

Wilkinson definoval řídkou matici pragmaticky jako matici, u které se vyplatí využít přítomnosti nulových prvků, ať už pro jejich procentuální zastoupení, nebo pro jejich rozložení [4, str. 191]. Tuto definici využívá i tato bakalářská práce.

Matice, která není řídká, se potom označuje jako hustá.

1.2.2 COO

Souřadnicový formát, anglicky *Coordinate format*, je jedním z nejjednodušších formátů vnitřního uložení řídkých matic. Jeho datová struktura se skládá z následujících tří polí velikosti NNZ:

- AA: neseřazené pole obsahující všechny reálné *nenulové prvky* matice $\mathbb{A}$
- JR: obecně neseřazené pole přirozených čísel představujících *čísla řádků* patřičných, od začátku pole stejně vzdálených, prvků pole AA
- JC: obecně neseřazené pole přirozených čísel představujících *čísla sloupců* patřičných, od začátku pole stejně vzdálených, prvků pole AA

Příklad. Matici

$$\mathbb{A} = \begin{pmatrix} 1{,}1 & 0 & 0 & 2{,}7 & 0 \\ 3{,}2 & 0 & 4{,}5 & 0 & 0 \\ 0 & 0 & 5{,}6 & 0 & 6{,}8 \\ 0 & 7{,}3 & 0 & 0 & 0 \\ 0 & 8{,}4 & 0 & 0 & 9{,}9 \end{pmatrix}$$

lze ve formátu COO uložit takto:

AA	6,8	7,3	2,7	1,1	3,2	9,9	5,6	8,4	4,5
JR	3	4	1	1	2	5	3	5	2
JC	5	2	4	1	1	5	3	2	3

To, že jsou tato pole obecně neseřazená, zajišťuje rychlé vkládání dalších nenulových prvků.

1.2.3 CSR

Formát komprimovaných řídkých řádků, anglicky *Compressed Sparse Row format*, je asi nejběžnějším formátem pro uložení obecných řídkých matic. Pro provádění maticových operací je většinou vhodnější než souřadnicový formát.

Potřebuje méně úložného místa než souřadnicový formát. Komprimuje totiž řádkové indexy. Tuto komprimaci z formátu COO si lze představit tak, že se seřadí pole čísel řádků JR a také se seřadí jednotlivé souvislé části pole JC příslušející stejnému řádku. Alternativně se seřadí pole JC a potom se stabilně seřadí pole JR, totiž bez prohození shodných prvků. Potom se pole JR nahradí ukazateli na začátky řádků v polích AA a JC a na konec se připojí ukazatel na první již neplatnou pozici za těmito poli. Pole JC se přejmenuje na JA.

Datová struktura tohoto formátu sestává z následujících tří polí:

- AA: pole obsahující všechny reálné *nenulové prvky* matice $\mathbb{A}$, uložené vzestupně po řádcích, prvky téhož řádku jsou uložené vzestupně po sloupcích. Velikost tohoto pole je NNZ.

- JA: pole přirozených čísel představujících *čísla sloupců* patřičných, od začátku pole stejně vzdálených, prvků pole AA. Tato čísla jsou pro každý řádek zvlášť seřazena vzestupně. Velikost tohoto pole je NNZ.

- IA: pole přirozených čísel představujících *indexy začátků řádků* v polích AA a JA. $i$tý prvek tohoto pole obsahuje index, na kterém v polích AA a JA začíná $i$tý řádek. $(i+1)$ní prvek pole IA je potom součtem $i$tého prvku pole IA a počtu nenulových prvků v $i$tém řádku matice $\mathbb{A}$, na této pozici v polích AA a JA tedy skončil $i$tý řádek a začíná zde $(i+1)$ní řádek. Poslední prvek pole IA je výjimkou, obsahuje totiž číslo $\mathrm{IA}[1] + \mathrm{NNZ}$, totiž první již neplatnou pozici v polích AA a JA. Velikost tohoto pole je počet řádků matice $\mathbb{A} + 1$.

Příklad. Matice

$$\mathbb{A} = \begin{pmatrix} 1{,}1 & 0 & 0 & 2{,}7 & 0 \\ 3{,}2 & 0 & 4{,}5 & 0 & 0 \\ 0 & 0 & 5{,}6 & 0 & 6{,}8 \\ 0 & 7{,}3 & 0 & 0 & 0 \\ 0 & 8{,}4 & 0 & 0 & 9{,}9 \end{pmatrix}$$

se ve formátu CSR uloží takto:

AA	1,1	2,7	3,2	4,5	5,6	6,8	7,3	8,4	9,9
JA	1	4	1	3	3	5	2	2	5
IA	1	3	5	7	8	10			

1.2.4 CSC

Formát komprimovaných řídkých sloupců, anglicky *Compressed Sparse Column format*, je v praxi také často používán. Je analogií formátu CSR, od kterého se liší tím, že pro každý prvek ukládá číslo jeho řádku, namísto sloupce, a pole indexů začátků řádků je nahrazeno polem indexů začátků sloupců.

1.3 Arnoldiho metoda

Tato sekce, společně se sekcí následující, využívá navíc pojmy vektorový prostor, podprostor, lineární kombinace, lineární nezávislost, lineární obal $\langle \ldots \rangle$, podobnost matic, skalární součin, ortogonalita a ortonormalita vektorů, euklidovská norma $\|\cdot\|_2$ ze studijního textu [1].

Arnoldiho metoda byla poprvé představena ve článku [5] jako prostředek redukce husté matice na Hessenbergovu matici. Už zde bylo zmíněno, že by tato metoda mohla poskytovat dobré aproximace některých vlastních čísel, pokud by se její běh zastavil v průběhu výpočtu, před dokončením. Později bylo zjištěno, že taková úprava je dobrým stavebním kamenem pro aproximování vlastních čísel velkých řídkých matic. Tuto metodu ukazuje algoritmus 1. Jeho zdrojem je kniha [6] a technická zpráva [7]. [2]

Výstupní matice $\mathbb{V}$, $\mathbb{H}$ a vektor $\mathbb{f}$ se mohou značit s dolním indexem m indikujícím počet sloupců matic $\mathbb{H}$ a $\mathbb{V}$, a tedy i počet iterací Arnoldiho algoritmu. [2] V této práci má dolní index přednost před horním indexem.

Sloupce matice $\mathbb{V}_m$ jsou ortonormální bází Krylovova [2] podprostoru $\mathcal{K}_m(\mathbb{A}, \mathbb{v}) = \langle \mathbb{v}, \mathbb{A}\mathbb{v}, \mathbb{A}^2\mathbb{v}, \ldots, \mathbb{A}^{m-1}\mathbb{v} \rangle$. [2]

Vztah

$$\mathbb{A}\mathbb{V}_m = \mathbb{V}_m\mathbb{H}_m + \mathbb{f}_m\mathbb{e}_m^T \tag{1.1}$$

se pak nazývá **Arnoldiho faktorizace matice $\mathbb{A}$ velikosti** m [6]. Jestliže se rovnice (1.1) vynásobí zleva maticí $\mathbb{V}_m^H$, pak je výsledkem vztah

$$\mathbb{V}_m^H\mathbb{A}\mathbb{V}_m = \mathbb{H}_m, \tag{1.2}$$

Algoritmus 1: `arnoldi`: Arnoldiho metoda

Input: $m, n \in \mathbb{N}, m \leq n$,
$\mathbb{A} \in \mathbb{C}^{n,n}$ vstupní matice,
$\mathbb{v} \in \mathbb{C}^n, \|\mathbb{v}\|_2 = 1$ počáteční vektor

Output: $\mathbb{V} \in \mathbb{C}^{n,m}$ matice s ortonormálními sloupci,
$\mathbb{H} \in \mathbb{C}^{m,m}$ Hessenbergova matice,
$\mathbb{f} \in \mathbb{C}^n$ vektor ortogonální vůči sloupcům matice $\mathbb{V}$,
tak, že platí $\mathbb{A}\mathbb{V} = \mathbb{V}\mathbb{H} + \mathbb{f}\mathbb{e}_m^T$,
kde $\mathbb{e}_m \in \mathbb{R}^m$ a $\mathbb{e}_m^T = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$

1 $\mathbb{V}_{:,1} \leftarrow \mathbb{v}$
2 $\mathbb{f} \leftarrow \mathbb{A}\mathbb{V}_{:,1}$
3 ortogonalizuj $\mathbb{f}$ vzhledem k matici $\mathbb{V}_{:,1}$ a koeficient ortogonalizace ulož do prvku $\mathbb{H}_{1,1}$
4 **for** $j = 1, 2, \ldots, m-1$ **do**
5 **if** $\|\mathbb{f}\|_2 = 0$ **then**
6 zastav
7 **end**
8 $\mathbb{H}_{j+1,j} \leftarrow \|\mathbb{f}\|_2$
9 $\mathbb{V}_{:,j+1} \leftarrow \mathbb{f}/\|\mathbb{f}\|_2$
10 $\mathbb{f} \leftarrow \mathbb{A}\mathbb{V}_{:,j+1}$
11 ortogonalizuj $\mathbb{f}$ vzhledem k matici $\mathbb{V}_{:,1:j+1}$ a koeficienty ortogonalizace ulož do sloupce $\mathbb{H}_{1:j+1,j+1}$
12 **end**

neboť matice $\mathbb{V}_m$ má ortonormální sloupce, a tedy $\mathbb{V}_m^H\mathbb{V}_m = \mathbb{E}$, kde $\mathbb{E} \in \mathbb{R}^{m,m}$ je jednotková matice, totiž neutrální prvek vůči násobení, a vektor $\mathbb{f}$ je ortogonální ke sloupcům matice $\mathbb{V}_m$, a proto se člen $\mathbb{f}_m\mathbb{e}_m^T$ vynuluje. [2]

V případě $m = n$ jsou $\mathbb{H}_n$ i $\mathbb{V}_n$ čtvercové matice stejného typu jako matice $\mathbb{A}$. Matice $\mathbb{V}_n$ je pak unitární, totiž $\mathbb{V}_n^{-1} = \mathbb{V}_n^H$. Tento fakt společně s platností rovnice (1.2) implikuje, že je matice $\mathbb{H}_n$ podobná matici $\mathbb{A}$, a tedy má stejná všechna vlastní čísla. [2]

Ve zbývajících případech, když $m < n$ a algoritmus proběhne standardním způsobem, tedy se nezastaví na řádku 6, poskytují vlastní čísla matice $\mathbb{H}_m$ pouhé aproximace vlastních čísel matice $\mathbb{A}$. [2]

Jestliže se však algoritmus po vyhodnocení podmínky na řádku 5 zastaví v průběhu iterace j, pak je všech j vlastních čísel matice $\mathbb{H}_j$ totožných s j některými vlastními čísly matice $\mathbb{A}$, totiž se jedná o přesné aproximace. V takovém případě platí rovnost $\mathbb{A}\mathbb{V}_j = \mathbb{V}_j\mathbb{H}_j$. [2]

V praxi se zde tento algoritmus téměř nikdy nezastaví, i kvůli nepřesnostem v aritmetice s plovoucí desetinnou čárkou s omezenou přesností. [2]

Je důležité dokázat rychle odhadnout přesnost vypočteného vlastního čísla,

a to především pro stanovení kritéria pro zastavení. Nechť ve faktorizaci (1.1) $\lambda \in \mathbb{C}$ je vlastní číslo matice $\mathbb{H}_m$ a $\mathbb{y} \in \mathbb{C}^n$ je vlastní vektor matice $\mathbb{H}_m$ příslušející vlastnímu číslu λ, totiž platí $\mathbb{H}_m\mathbb{y} = \lambda\mathbb{y}$. Pak λ je aproximací vlastního čísla matice $\mathbb{A}$ a $\mathbb{u} = \mathbb{V}_m\mathbb{y}$ je aproximovaný vlastní vektor matice $\mathbb{A}$ příslušející aproximovanému vlastnímu číslu λ. Po vynásobení faktorizace (1.1) vektorem $\mathbb{y}$ jsou pak ekvivalentní následující úpravy:

$$\begin{aligned}
\mathbb{A}\mathbb{V}_m\mathbb{y} &= \mathbb{V}_m\mathbb{H}_m\mathbb{y} + \mathbb{f}_m\mathbb{e}_m^T\mathbb{y} \\
\mathbb{A}\mathbb{V}_m\mathbb{y} &= \lambda\mathbb{V}_m\mathbb{y} + \mathbb{f}_m\mathbb{e}_m^T\mathbb{y} \\
\mathbb{A}\mathbb{V}_m\mathbb{y} - \lambda\mathbb{V}_m\mathbb{y} &= \mathbb{f}_m\mathbb{e}_m^T\mathbb{y} \\
\mathbb{A}\mathbb{u} - \lambda\mathbb{u} &= \mathbb{f}_m\mathbb{e}_m^T\mathbb{y} \\
(\mathbb{A} - \lambda\mathbb{E})\mathbb{u} &= \mathbb{f}_m\mathbb{e}_m^T\mathbb{y}.
\end{aligned}$$

A proto platí:

$$\begin{aligned}
\|(\mathbb{A} - \lambda\mathbb{E})\mathbb{u}\|_2 &= \|\mathbb{f}_m\mathbb{e}_m^T\mathbb{y}\|_2 \\
\|(\mathbb{A} - \lambda\mathbb{E})\mathbb{u}\|_2 &= \|\mathbb{f}_m\|_2|\mathbb{e}_m^T\mathbb{y}|
\end{aligned}$$

Pro odhad přesnosti vlastního čísla λ se proto používá absolutní hodnota posledního prvku vlastního vektoru matice $\mathbb{H}_m$ příslušejícího tomuto vlastnímu číslu, tedy absolutní hodnota posledního prvku vektoru $\mathbb{y}$, vynásobená číslem $\|\mathbb{f}_m\|_2$, totiž odhad $\|\mathbb{f}_m\|_2|\mathbb{e}_m^T\mathbb{y}|$. [2]

1.4 Restart Arnoldiho metody

Ukázalo se, že pro výpočet několika největších vlastních čísel předchozí algoritmus 1 není příliš vhodný. Ortogonalizace nových vektorů je stále těžší a časová složitost výpočtu vlastních čísel Hessenbergovy matice je řádově $\mathcal{O}(m^3)$, kde m je velikost této matice. I paměťová složitost roste rychle, protože je potřeba ukládat všech m vektorů ortonormální báze Krylovova podprostoru. Nadto není možné zjistit kolik iterací tohoto algoritmu bude nutné provést pro dostatečně přesnou aproximaci požadovaných vlastních čísel. Z těchto důvodů se začaly vyvíjet techniky restartování této metody. [6]

Restartování Arnoldiho metody typicky spočívá v tom, že se sestaví Arnoldiho faktorizace velikosti m a pak začne nový výpočet stejně velké faktorizace se snahou efektivního využití vypočtených informací z předchozí faktorizace. Tento proces se opakuje, dokud není dostatečně přesně aproximováno požadované množství vlastních čísel. [7]

1.4.1 Implicitně restartovaná Arnoldiho metoda

Implicitní restartování Arnoldiho metody poskytuje snad nejefektivnější způsob využití zajímavých informací z předchozí Arnoldiho faktorizace. Restart spočívá v tom, že se vypočtená Arnoldiho faktorizace zkomprimuje do

faktorizace velikosti k, kde $k \in \mathbb{N}$ je fixní a představuje požadovaný počet aproximací vlastních čísel. Při této komprimaci se zachovají pouze zajímavé informace týkající se požadovaných vlastních čísel a nezajímavé se odstraní, čehož se docílí implicitním použitím QR algoritmu s posuny. [6]

Na matici $\mathbb{H}_m$ z faktorizace (1.1) se aplikuje $m-k$ kroků QR algoritmu, každý z těchto kroků s rozdílným posunem, který je jedním z $m-k$ zvolených posunů, a nechť $\mathbb{Q} \in \mathbb{C}^{m,m}$ je těmto aplikacím příslušející unitární matice. Tato matice se aplikuje implicitně i na zbytek faktorizace. Tím se obdrží následující vztah:

$$\mathbb{A}\mathbb{V}_m^+ = \mathbb{V}_m^+\mathbb{H}_m^+ + \mathbb{f}_m\mathbb{e}_m^T\mathbb{Q},$$

kde $\mathbb{V}_m^+ = \mathbb{V}_m\mathbb{Q}$ a $\mathbb{H}_m^+ = \mathbb{Q}^H\mathbb{H}_m\mathbb{Q}$. Matice $\mathbb{V}_m^+$ si zachovala ortonormalitu sloupců, neboť byla vynásobena unitární maticí, a matice $\mathbb{H}_m^+$ je stále Hessenbergovou maticí podobnou matici $\mathbb{H}_m$. Prvních $k-1$ prvků vektoru $\mathbb{e}_m^T\mathbb{Q}$ jsou nuly. Proto platí tato Arnoldiho faktorizace matice $\mathbb{A}$ velikosti k:

$$\mathbb{A}\mathbb{V}_k^+ = \mathbb{V}_k^+\mathbb{H}_k^+ + \mathbb{f}_k^+\mathbb{e}_k^T, \tag{1.3}$$

kde $\mathbb{V}_k^+ = \mathbb{V}_m^+\mathbb{Q}_{:,1:k}$, $\mathbb{H}_k^+ = (\mathbb{H}_m^+)_{1:k,1:k}$ a $\mathbb{f}_k^+ = (\mathbb{H}_m^+)_{k+1,k}\mathbb{v}_{k+1} + \mathbb{Q}_{m,k}\mathbb{f}_m$. [6]

Faktorizace (1.3) je identická s faktorizací, která by byla výsledkem k iterací standardní Arnoldiho metody 1 s novým počátečním vektorem $\mathbb{v}^+ = \mathbb{w}/\|\mathbb{w}\|_2$, kde $\mathbb{w} = (\prod_{i=1}^{m-k}(\mathbb{A} - \mu_i\mathbb{E}))\mathbb{v}$, zde potom $\mathbb{E} \in \mathbb{R}^{n,n}$ je jednotková matice, $\mathbb{v}$ je původní počáteční vektor a $\mu_1, \mu_2, \ldots, \mu_{m-k}$ jsou zvolené posuny. Tím došlo k *implicitní* aplikaci polynomiálního filtrování stupně $m-k$ na počáteční vektor Arnoldiho faktorizace velikosti k, a to bez provedení žádného dalšího součinu matice $\mathbb{A}$ s vektorem. [2]

V praxi se osvědčila volba těchto posunů jako $m-k$ vlastních čísel matice $\mathbb{H}_m$, které jsou nezajímavé, a totiž se jedná o přesné posuny. Kdyby ve výpočtech nedocházelo k žádným nepřesnostem, potom by důsledkem této volby bylo, že by spektrum matice $\mathbb{H}_k^+$ obsahovalo jen a právě k vlastních čísel matice $\mathbb{H}_m$, které jsou zajímavé, a nový počáteční vektor by byl určitou lineární kombinací k aktuálně nejlepších aproximací vlastních vektorů příslušejících těmto zajímavým vlastním číslům. [6]

Volba posunů pro QR algoritmus je motivována faktem, že když je počáteční vektor Arnoldiho metody roven lineární kombinaci vlastních vektorů příslušejících požadovaným k vlastním číslům matice $\mathbb{A}$, kde žádný z koeficientů této lineární kombinace není roven nule, potom je po k iteracích Arnoldiho metody vektor $\mathbb{f}_k$ nulovým vektorem, a tedy se tento algoritmus výjimečně zastaví v průběhu, a tedy spektrum matice $\mathbb{H}_k$ poskytuje přesné aproximace požadovaných vlastních čísel. [6]

Zkomprimovaná faktorizace (1.3) se potom může opět rozšířit na Arnoldiho faktorizaci velikosti m upravenou Arnoldiho metodou, která je totožná s Arnoldiho metodou 1, pouze počítá s již existující vstupní Arnoldiho faktorizací (1.3) a tuto faktorizaci rozšíří pomocí zbytkového vektoru $\mathbb{f}_k^+$ z této

faktorizace. Tato metoda je zapsána jako algoritmus 3 v příloze, jehož zdrojem je kniha [6] a technická zpráva [7].

Implicitně restartovaná Arnoldiho metoda je popsána v algoritmu 2, jeho zdrojem je kniha [6].

Algoritmus 2: Implicitně restartovaná Arnoldiho metoda

Input: $m, n \in \mathbb{N}, m < n$,
$k \in \mathbb{N}$ požadovaný počet aproximací vlastních čísel,
$\mathbb{A} \in \mathbb{C}^{n,n}$ vstupní matice,
$\mathbb{v} \in \mathbb{C}^n, \|\mathbb{v}\|_2 = 1$ počáteční vektor

Output: $p \in \mathbb{N}, p \geq k$,
$\lambda_{aprox} \in \mathbb{C}^p$ vektor aproximovaných vlastních čísel,
$\mathbb{V} \in \mathbb{C}^{n,m}$ matice s ortonormálními sloupci,
$\mathbb{H} \in \mathbb{C}^{m,m}$ Hessenbergova matice,
$\mathbb{f} \in \mathbb{C}^n$,
tak, že platí $\mathbb{A}\mathbb{V} = \mathbb{V}\mathbb{H} + \mathbb{f}\mathbb{e}_m^T$,
kde $\mathbb{e}_m \in \mathbb{R}^m$ a $\mathbb{e}_m^T = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$

```
[V_m, H_m, f_m] ← arnoldi(m, n, A, v)
while true do
    spočti σ(H_m)
    if počet dobrých aproximací vlastních čísel ≥ k then
        zastav
    end
    zvol (m − k)tici posunů (μ_1, μ_2, ..., μ_{m−k}) na základě σ(H_m),
     popřípadě na základě nějakých jiných informací
    Q ← E_m, kde E_m ∈ R^{m,m} je jednotková matice
    for j = 1, 2, ..., m − k do
        [Q_j, R_j] ← qr(H_m − μ_j E_m)
        H_m ← Q_j^H H_m Q_j
        Q ← Q Q_j
    end
    f_k ← (H_m)_{k+1,k} v_{k+1} + Q_{m,k} f_m
    V_k ← V_m Q_{:,1:k}
    H_k ← (H_m)_{1:k,1:k}
    [V_m, H_m, f_m] ← modifiedArnoldi(m, n, k, A, V_k, H_k, f_k)
end
```

Opakovaná aktualizace počátečního vektoru $\mathbb{v}$ implicitním restartem je navržena tak, aby byl nový počáteční vektor více nakloněn ve směru vlastních vektorů příslušejících požadovaným vlastním číslům a odkloněn od směru ostatních vektorů. [6]

1.4.2 Další způsoby restartu

Explicitně restartovaná Arnoldiho metoda
Explicitně restartovaná Arnoldiho metoda provede restartování spojené s kompresí faktorizace explicitně, a to tak, že se pro nový běh Arnoldiho metody zvolí za počáteční vektor aproximace vlastního vektoru příslušejícího prvnímu ještě nedostatečně přesně aproximovanému vlastnímu číslu. Tím se v novém běhu docílí rychlé konvergence tohoto vlastního čísla, neboť zvolený počáteční vektor již má směr podobný směru příslušného vlastního vektoru. Po zkonvergování daného vlastního čísla se toto číslo a k němu příslušející vlastní vektor uzamkne a algoritmus se zaměří na výpočet aproximace dalšího vlastního čísla, přičemž musí nové vektory vznikající při expanzi Arnoldiho faktorizace ortogonalizovat i vzhledem k již uzamknutým vektorům. Tento přístup však není ideální, když je zapotřebí vypočítat více vlastních čísel, protože ostatní vlastní vektory nejsou v novém počátečním vektoru zahrnuty. [7]

Restart Arnoldiho metody polynomiálním filtrováním
Nechť $d, k, n \in \mathbb{N}$, dále buď $\mathbb{A} \in \mathbb{C}^{n,n}$ vstupní matice, p polynom stupně d, $z_0 \in \mathbb{C}^n$ lineární kombinace aproximací vlastních vektorů, $v_{k+1}, z \in \mathbb{C}^n$ vektory. Restart Arnoldiho metody polynomiálním filtrováním pak spočívá v tom, že se spočte $z = p(\mathbb{A})z_0$ a nový počáteční vektor se zvolí jako $v_{k+1} = z/\|z\|_2$. Polynom p musí být definován tak, aby byl nový počáteční vektor odkloněn od směru nezajímavých vlastních vektorů a nakloněn ve směru zajímavých vlastních vektorů. Toho lze i pro nehermitovské vstupní matice docílit použitím Čebyševových polynomů. [7]

Je však obtížné zvolit lineární kombinaci z_0 aproximací vlastních vektorů tak, aby byla dosažena stejně rychlá konvergence všech požadovaných vlastních čísel, protože není snadné reprezentovat celý podprostor jediným vektorem. Řešením tohoto problému může být aplikování tohoto filtrování ve všech krocích Arnoldiho metody, nikoli pouze v rámci restartu, to jest, nahradit matici $\mathbb{A}$ maticí $p(\mathbb{A})$. Tato strategie se nejčastěji nazývá polynomiální předurčování, anglicky *polynomial preconditioning*. Nevýhodou restartu Arnoldiho metody polynomiálním filtrováním je také silný vliv volby parametrů použitého polynomu na efektivitu tohoto algoritmu. [7]

Již popsaný implicitní restart je implicitní aplikace polynomiálního filtrování stupně $m - k$, kde $m \in \mathbb{N}$ je maximální velikost Arnoldiho faktorizace a $k \in \mathbb{N}$ je požadovaný počet vlastních čísel. [7]

Kapitola 2

Knihovny pro práci s maticemi

V této kapitole jsou popsány knihovny, které implementují maticové operace zejména nad hustými maticemi, a které je tak možné použít jako základ implementované kalkulačky pro výpočet několika vlastních čísel řídkých matic. Podpora operací s řídkými maticemi je vítaným doplňkem, avšak není nutná.

2.1 BLAS

BLAS [8], s plným názvem Basic Linear Algebra Subprograms, jsou funkce, které poskytují standardní stavební kameny vykonávající základní operace s vektory a s maticemi.

Tyto funkce jsou rozděleny na tři úrovně:

- Level 1: Operace typu skalár, vektor a vektor–vektor.
- Level 2: Operace typu matice–vektor.
- Level 3: Operace typu matice–matice.

Pro rozhraní těchto funkcí existuje mnoho dalších implementací, které jsou často optimalizovány pro konkrétní architekturu počítače.

Protože jsou tyto funkce výkonné, přenositelné a snadno dostupné, jsou často užívány ve vývoji vysoce kvalitních programů řešících složitější problémy lineární algebry.

2.2 LAPACK

LAPACK [9], neboli Linear Algebra PACKage, je sadou funkcí, které poskytují řešení problémů lineární algebry, jako je hledání řešení soustavy lineárních rovnic, hledání vlastních čísel, či konstrukce maticových rozkladů, a to například QR rozkladu, LU rozkladu, nebo Schurova rozkladu.

Pro výpočet na nižší úrovni je často použita funkcionalita BLAS.

Stejně jako pro BLAS, tak i pro LAPACK existuje mnoho specifických implementací. Jednou z nich je ScaLAPACK [10], jež je určená pro paralelní výpočet v systémech s distribuovanou pamětí a jejíž komunikační vrstvou je BLACS [11] a je volně dostupná pod novou BSD licencí.

2.3 Eigen

Eigen [12] je čistě hlavičková knihovna napsaná v jazyce C++, která poskytuje řešení mnoha problémů z lineární algebry a rychlé rozhraní pro práci s maticemi. Jak již název napovídá, lze tuto knihovnu využít pro hledání vlastních čísel. Je však možné spočítat pouze všechna, a nikoli jen některá, vlastní čísla.

Podporuje práci s hustými maticemi všech rozměrů, počínaje malými maticemi s pevně stanoveným počtem řádků a sloupců, až po libovolně velké husté matice. Implementuje také třídu `SparseMatrix`, jež zajišťuje pohodlné rozhraní pro práci s řídkou maticí, kterou ukládá v modifikovaném formátu Compressed Sparse Row, respektive Compressed Sparse Column, který mezi jednotlivými řádky, respektive sloupci, nechává pro rychlé vložení nového nenulového prvku volná místa, takže další prvek rozdílný od nuly lze vložit v čase asymptoticky lineárně rostoucím s počtem nenulových prvků v daném řádku, respektive sloupci, nebo dokonce až v asymptoticky konstantním čase, jestliže jsou do daného řádku, respektive sloupce, prvky vkládány postupně s rostoucími indexy sloupců, respektive řádků. Tuto matici lze ovšem snadno převést do standardního formátu Compressed Sparse Row, či Compressed Sparse Column.

Hierarchie tříd datového modelu, tedy matic a vektorů, je v této knihovně navržena tak, aby se vyhnulo nutnosti volat virtuální funkce v případech, kdy by to znamenalo značnou ztrátu výpočetní rychlosti. Namísto tradičního dědění, které využívá virtuální funkce, je zde pro dosažení polymorfismu implementován *Curiously Recurring Template Pattern*, se zkratkou CRTP. V tomto C++ programovacím vzoru je rodičovská třída, například `MatrixBase`, parametrizována šablonovým parametrem, a odvozená třída, například `Matrix`, dědí z této rodičovské třídy se šablonovým parametrem rovným právě této odvozené třídě, tedy v tomto příkladě `Matrix` dědí z `MatrixBase<Matrix>`. Metoda rodičovské třídy pak používá pro zavolání metody odvozené třídy statické přetypování své samotné instance, na které byla tato metoda zavolána, tedy například `static_cast<Derived*>(this)->implementation();` zavolá z rodičovské třídy metodu `Derived::implementation` na této instanci již s odvozeným typem `Derived`, kde `Derived` je jméno daného šablonového parametru rodičovské třídy. To umožňuje vyřešit polymorfní volání metod již v době kompilace. Tomuto polymorfismu se proto říká statický polymorfismus, na rozdíl od běžného dynamického polymorfismu, který řeší polymorfní volání až v době běhu programu.

Eigen se také, i kvůli výkonu, vyhýbá vícenásobnému dědění.

Knihovnu lze používat se všemi standardními datovými typy jazyka C++, včetně celých či komplexních čísel, a dokonce i včetně uživatelem definovaných čísel, neboť její implementace využívá C++ šablon pro poskytnutí této generiky nad skaláry.

V současné verzi lze Eigen nastavit tak, aby pro násobení a dekompozice hustých matic používal implementace jakýchkoli BLAS a LAPACK knihoven, které jsou kompatibilní s verzí F77 programovacího jazyka Fortran.

Obsahuje také ekosystém nepodporovaných modulů, který poskytuje několik specifických funkcí, jako je například načtení vstupní matice ze souboru ve formátu MatrixMarket.

Tato knihovna je dostupná volně pod licencí MPL2.

Kapitola 3

Přehled existujících řešení

V této kapitole jsou popsány některé z nejznámějších knihoven s implementací restartované Arnoldiho metody.

3.1 ARPACK

ARPACK [6], neboli ARnoldi PACKage, je matematický software napsaný v jazyce Fortran77. Je navržený pro hledání několika vlastních čísel velkých matic. Tato vlastní čísla jsou aproximacemi největších čísel ze spektra podle uživatelem zvoleného způsobu porovnávání. Takto je možné hledat vlastní čísla například s největší či nejmenší absolutní hodnotou a k nim příslušející vlastní vektory. Pomocí posunutí o σ a následné inverze vstupní velké matice lze počítat vlastní čísla nejblíže k posunu σ.

Tato knihovna požaduje po uživateli, aby on sám prováděl operace se vstupní velkou maticí. Pro standardní použití stačí často poskytnout pouze součin vstupní velké matice s daným vektorem. Knihovna tedy nezávisí na formátu uložení vstupní velké matice, protože sama s touto maticí nezachází.

Používá se zde implicitně restartovaná Arnoldiho metoda. Pro implicitní restart se používá QR algoritmus s exaktními posuny. Ve výchozím nastavení jsou tyto posuny počítány knihovnou, ale může je zvolit i uživatel.

Je specializována pro hermitovské a nehermitovské, reálné a komplexní, matice. Lze zadat požadovanou přesnost aproximací vlastních čísel.

Existuje i paralelní verze, a to P_ARPACK [13]. Podporované komunikační vrstvy jsou zde BLACS [11] a MPI [14].

Knihovna je dostupná volně pod novou BSD licencí.

3.2 Spectra

Knihovna se jménem Spectra [15], které je zkratkou pro Sparse Eigenvalue Computation Toolkit as a Redesigned ARPACK, je knihovnou pro

výpočet vlastních čísel velkých matic napsanou v jazyce C++. Jak již plný název napovídá, tato knihovna se inspirovala knihovnou ARPACK popsanou v sekci 3.1. Na této knihovně nijak nezávisí, pouze implementuje stěžejní algoritmy popsané v knize [6], zejména implicitně restartovanou Arnoldiho metodu, ovšem se zcela jiným rozhraním.

Jedná se o čistě hlavičkovou knihovnu, jejíž jediná závislost je na další čistě hlavičkové knihovně Eigen, popsané v sekci 2.3, a proto ji lze snadno používat v C++ projektech a kompilátor je schopen poskytnout dobré optimalizace z této knihovny používaných funkcionalit.

Pro výpočet několika vlastních čísel vstupní matice je potřeba poskytnout především operaci součinu vstupní matice s vektorem. Knihovna nabízí několik tříd implementujících tuto operaci, ale může ji definovat i samotný uživatel, čímž je zajištěna volnost výběru formátu uložení vstupní matice.

Tato knihovna není paralelizována. Ve většině případů je rychlejší než knihovna ARPACK popsaná v sekci 3.1.

Její kód je dostupný volně pod licencí MPL2, stejně jako touto knihovnou používaný kód knihovny Eigen.

3.3 SLEPc

SLEPc [16], s plným názvem Scalable Library for Eigenvalue Problem Computations, je matematický software napsaný v jazyce C. Je rozšířením knihovny PETSc [17], neboli Portable, Extensible Toolkit for Scientific Computation, o vše potřebné pro hledání vlastních čísel a k nim příslušejících vlastních vektorů a pro řešení několika dalších souvisejících problémů. Jejím největším zájmem je řešení takových problémů s velkými řídkými maticemi na paralelních počítačích. Podporovanou komunikační vrstvou je MPI [14].

Nabízí rozhraní nad jinými známými řešeními, například nad již zmíněnou knihovnou ARPACK i nad její paralelní verzí P_ARPACK.

Jednotlivé funkce mohou být volány přímo ze zdrojového kódu, nebo přes příkazovou řádku.

Tato knihovna implementuje explicitně restartovanou Arnoldiho metodu. Uživatel může zvolit, jakým způsobem bude probíhat reortogonalizace.

Ačkoli jazyk C nemá přímou podporu pro objektově orientované programování, tato knihovna využívá tří základních principů objektově orientovaného programování, tedy zapouzdření, polymorfismus a dědičnost. Například matice má abstraktní rozhraní, které je implementováno několika realizacemi s rozdílnými formáty uložení matice.

Knihovna SLEPc je, stejně jako knihovna PETSc, zveřejněna pod zjednodušenou BSD licencí.

Kapitola 4

Analýza požadavků

V této části jsou zachyceny požadavky na implementovanou aplikaci. Tyto požadavky jsou zaměřeny na co nejvěrnější splnění zadání této bakalářské práce. Jejich vlastníkem, též stakeholderem, totiž osobou, která danou funkcionalitu či vlastnost požaduje, je autor této práce.

4.1 Funkční požadavky

FP1 Zadání počtu vlastních čísel

- Uživatel zadá požadovaný počet aproximací největších vlastních čísel.

FP2 Volba srovnávacího kritéria

- Uživatel si bude moci vybrat srovnávací kritérium, podle kterého budou vypočtené aproximace vlastních čísel vzájemně porovnávána pro rozlišení zajímavých a nezajímavých aproximací. Bude si moci vybrat z těchto srovnávacích kritérií:
  - Největší absolutní hodnota.
  - Nejmenší absolutní hodnota.
  - Největší reálná část.
  - Nejmenší reálná část.

FP3 Zadání přesnosti aproximací

- Uživatel bude moci zadat požadovanou přesnost vypočítaných vlastních čísel.

FP4 Zadání maximální velikosti Arnoldiho faktorizace

- Uživatel bude moci zadat velikost Arnoldiho faktorizace, při které má dojít k restartu Arnoldiho metody a tím ke kompresi faktorizace.

FP5 Zadání maximálního počtu restartů

- Uživatel bude moci zadat maximální počet restartů Arnoldiho metody. Po jeho dosažení se výpočet zastaví. Bude možné získat již zkonvergovaná vlastní čísla.

FP6 Zadání maximálního počtu iterací iterativní ortogonalizace

- Kalkulačka umožní uživateli zadat maximální povolený počet iterací iterativní ortogonalizace.

FP7 Zadání éty iterativní ortogonalizace

- Uživatel bude moci zadat číslo $\eta \in \mathbb{R}$, kde $0 \leq \eta \leq 1$, rozhodující o tom, zda se má provést opětovná iterace algoritmu iterativní ortogonalizace, použitého pro ortogonalizaci nového vektoru v rámci generování ortonormální báze Krylovova podprostoru nad skaláry s omezenou přesností. Opětovná iterace algoritmu iterativní ortogonalizace se provede, jestliže $\eta \cdot \|\widetilde{\mathbb{f}}\|_2 > \|\underset{\sim}{\mathbb{f}}\|_2$, kde $n \in \mathbb{N}, \widetilde{\mathbb{f}}, \mathbb{f} \in \mathbb{R}^n$, $\mathbb{f}$ je vektor po poslední ortogonalizaci a $\widetilde{\mathbb{f}}$ je vektor před poslední ortogonalizací, a zároveň nebude splněna žádná další ukončující podmínka, jako je například dosažení maximálního povoleného počtu iterací. Tuto metodu autor převzal z technické zprávy [18].

FP8 Načtení vstupní matice

- Uživatel zadá název souboru, ze kterého se načte vstupní řídká reálná nesymetrická matice. Tato matice bude v tomto souboru uložena v souřadnicovém formátu MatrixMarket [19].

FP9 Libovolné načtení počátečního vektoru

- Kalkulačka vygeneruje počáteční vektor s využitím libovolného algoritmu. Uživatel tedy nebude povinen zvolit počáteční vektor sám.

FP10 Výpočet vlastních čísel

- Kalkulačka vypočítá vlastní čísla podle zvolených parametrů a požadavků.

FP11 Zobrazení vypočtených vlastních čísel

- Kalkulačka po dokončení výpočtu zobrazí vypočtená vlastní čísla.

FP12 Zobrazení statistik

- Uživatel si bude moci po skončení výpočtu zobrazit jeho statistiky, a to délku výpočtu a počet restartů, totiž počet iterací restartované Arnoldiho metody.

4.2 Nefunkční požadavky

NP1 Rozhraní přes příkazovou řádku

- Kalkulačku bude možné používat přes příkazovou řádku.
- **Typ**: Použitelnost.

NP2 Jednoduché desktopové rozhraní

- Kalkulačku bude možné používat přes jednoduché desktopové rozhraní.
- **Typ**: Použitelnost.

NP3 Platforma

- Kalkulačka bude dostupná pro platformu Linux.
- **Typ**: Použitelnost.

NP4 Rychlost výpočetní části

- Parametry bude možné zvolit tak, aby doba výpočetního běhu implementované kalkulačky nebyla pro sekvenční výpočet 5 vlastních čísel s největší absolutní hodnotou s relativní přesností 10^{-10} více než $10\times$ větší než doba výpočetního běhu knihovny Spectra se stejnými, nebo podobnými, parametry, na tomtéž, libovolně zvoleném, testovacím hardware, pro některých 5 různých libovolně vybraných reálných nesymetrických matic typu alespoň $10^5 \times 10^5$, jejichž průměrný počet nenulových prvků na řádku je nejméně 1,5. Pojem relativní přesnost zde označuje absolutní hodnotu z rozdílu vypočtené aproximace vlastního čísla a vzorového vlastního čísla, vydělenou absolutní hodnotou vzorového vlastního čísla. Do doby výpočetního běhu se nepočítá doba načítání matice.
- **Typ**: Výkon.

Kapitola 5

Návrh

5.1 Volba algoritmů

Pro výpočet několika největších vlastních čísel vstupní reálné, řídké a nesymetrické matice je zvolena implicitně restartovaná Arnoldiho metoda popsaná v sekci 1.4.1.

Pro realizaci implicitního restartu je použit QR algoritmus s posuny volící za posuny vlastní čísla aktuální Hessenbergovy matice, která jsou vzhledem ke zvolenému porovnávacímu kritériu vyhodnocena jako nezajímavá.

Pro ortogonalizaci vektorů v rámci generování ortonormální báze Krylovova podprostoru Arnoldiho metodou je zvolen algoritmus iterativní Gram-Schmidtovy ortogonalizace.

5.2 Technologie

5.2.1 Programovací jazyk

5.2.1.1 Výpočetní část

Je požadována rychlost výpočetní části požadavkem NP4. Je tedy nutné vybrat některou výpočetně nenáročnou knihovnu pro maticové operace, která poskytuje rozhraní pro efektivní práci s velkými vektory a maticemi. Nebo je zapotřebí výpočetní část implementovat v programovacím jazyce, který je dostatečně časově efektivní.

Kdyby byla zvolena možnost využít některé knihovny pro maticové operace, programovací jazyk by mohl být téměř libovolný, do jaké míry by byla vybraná knihovna v těchto jazycích přístupná. Velmi by se snížila náročnost implementace základní verze a bylo by samozřejmostí soustředit se na rozšiřitelnost a celkovou kvalitu kódu. Jestliže by se však ukázalo, že výsledná implementace není dostatečně efektivní, bylo by těžší pokusit se to napravit. Proto je výpočetní část napsána v jazyce C++, který je dostatečně

výkonný a zároveň umožňuje objektově orientované programování. Byla zvolena jeho aktuálně nejnovější verze, totiž C++20.

5.2.1.2 Uživatelská rozhraní

V kapitole 4 nejsou kladeny žádné nároky na výkon uživatelských rozhraní. Od obou rozhraní je požadováno pouze malé množství funkcionalit. Kdyby byl použit pro libovolné z těchto dvou rozhraní jiný programovací jazyk než pro výpočetní část, a přitom by ono rozhraní přímo používalo tuto implementovanou knihovnu, tak by to samo o sobě přispělo ke větší složitosti implementace tohoto rozhraní.

Rozhraní přes příkazovou řádku
Z výše uvedených důvodů je rozhraní přes příkazovou řádku napsáno také v jazyce C++.

Grafické uživatelské rozhraní
Grafické uživatelské rozhraní by alternativně mohlo být napsáno v jiném jazyce než předchozí dvě části, bez přímého použití výpočetní části, a komunikovat s rozhraním přes příkazovou řádku, přičemž by se data přenášela v nějaké předem specifikované podobě, například ve formátu JSON. Protože však po desktopovém rozhraní není požadováno mnoho funkcionalit a byl nalezen kvalitní C++ framework, jak je popsáno v následující sekci, je i toto rozhraní pro jednotnost napsáno v jazyce C++.

5.2.2 Framework grafického uživatelského rozhraní

Při výběru frameworku grafického uživatelského rozhraní byly brány v potaz následující technologie.

5.2.2.1 Qt

Qt [20] je multiplatformní framework pro tvorbu aplikací pro desktop, vestavěných systémů a pro mobil. Je napsaný v jazyce C++. Mezi podporované platformy patří mimo jiné i Windows, Linux a OS X, Android a iOS. Nejedná se o samostatný programovací jazyk. Preprocesor, zvaný Meta-Object Compiler, se zkratkou MOC, je použit pro rozšíření jazyka C++, mimo jiné o podporu signálů a jim naslouchajícím slotům pro třídy, jejichž deklarace obsahuje makro `Q_OBJECT`. MOC ze zdrojových souborů používajících Qt vygeneruje před kompilací zdrojové soubory, které jsou v souladu se standardem C++. Proto lze aplikace využívající framework Qt zkompilovat tradičními kompilátory jako například Clang, GCC a MinGW. Lze jej používat i v programovacím jazyku Python.

Qt projekty lze sestavit s pomocí nástroje CMake [21], ale tento framework ke svému sestavení poskytuje i svůj vlastní nástroj, a to *qmake*.

Qt nabízí také svůj vlastní Integrated Development Environment, neboli IDE, zvaný *Qt Creator*.

Grafická uživatelská rozhraní lze s frameworkem Qt psát přímo v C++ za použití modulu Widgets. Interaktivní grafický nástroj *Qt Designer* pak slouží pro generování kódu těchto rozhraní. Může být použit samostatně, ale je integrovaný i v nástroji *Qt Creator*.

K této technologii existuje kvalitní dokumentace často podpořená příklady použití.

Framework Qt je možné používat primárně pod komerční licencí, ale je dostupný i pod GPL a LGPL open-source licencemi.

5.2.2.2 wxWidgets

wxWidgets [22] je multiplatformní knihovna pro tvorbu desktopových aplikací napsaná v jazyce C++. Jsou podporovány standardní platformy jako Windows, Linux a OS X. Tuto technologii je možné využívat v jazycích C++, C#, Python a v mnoha dalších.

Jelikož se jedná o knihovnu, a nikoli o framework, lze projekty používající tuto knihovnu sestavit jednoduše zkompilováním daného projektu, například pravidly v souboru Makefile. Oficiálním systémem pro sestavování wxWidgets projektů je však *Bakefile*, který slouží pro vygenerování nativního souboru Makefile pro příslušnou platformu a pro příslušný kompilátor.

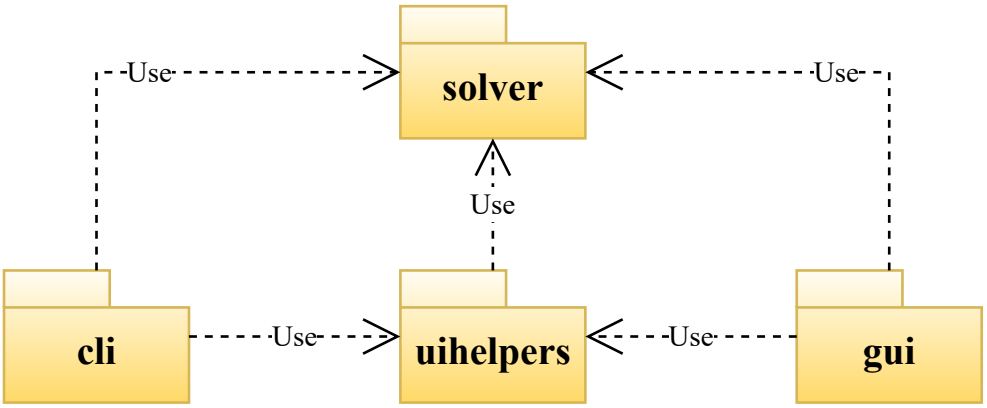
Existuje více open-source Rapid Application Development, neboli RAD, interaktivních grafických nástrojů pro tvorbu kódů grafických uživatelských rozhraní používajících knihovnu *wxWidgets*, například designer wxFormBuilder, nebo designer *wxSmith*, který je integrován do open-source IDE s názvem Code::Blocks.

Tato knihovna má rozměrnou dokumentaci a k ní zhruba 80 příkladů použití.

Tato knihovna je distribuovaná pod modifikovanou LGPL licencí, která vývojářům umožňuje nalinkovat jejich aplikaci buď staticky, nebo dynamicky, na wxWidgets, bez povinnosti zpřístupnit zdrojový kód jejich aplikace. Tím je umožněno používat tuto knihovnu jak v open-source, tak i v komerčních, projektech.

5.2.2.3 Vybraný framework

Autor se rozhodl pro implementaci grafického uživatelského rozhraní využít multiplatformní C++ framework Qt, protože se jedná o rozsáhlý C++ framework se širokou uživatelskou komunitou, ke kterému je dostupná obsáhlá, a přesto snadno pochopitelná, dokumentace s mnoha příklady použití. Autorovi se zdá dokumentace technologie Qt velmi kvalitní, kvalitnější než dokumentace ostatních technologií. Kvalita dokumentace znamenala pro autora při výběru frameworku mnoho, neboť s tvorbou grafických uživatelských rozhraní neměl



Obrázek 5.1: Diagram balíčků

velké zkušenosti. Konkrétně je použita verze Qt 5, která je dostupná i pod open source licencí LGPL.

5.3 Architektura

5.3.1 Rozdělení do balíčků

Je požadováno rozhraní přes příkazovou řádku společně s grafickým uživatelským rozhraním. Výpočetní část kalkulačky by neměla být závislá na uživatelském rozhraní. Jedná se tedy jistě o samostatný modul, který sám může být dále dělený. Výpočetní část je proto samostatnou knihovnou.

Rozhraní přes příkazovou řádku i grafické uživatelské rozhraní pak využívá výpočetní část. Některé funkcionality, nepřítomné ve výpočetní části, jako například validaci vstupních parametrů, mají tato rozhraní společné. Proto je zde ještě jeden balíček s takovými funkcionalitami.

Tato architektura je zobrazena na obrázku 5.1. Balíček `solver` představuje výpočetní část, balíček `uihelpers` představuje pomocný balíček, balíček `cli` poskytuje rozhraní přes příkazovou řádku, balíček `gui` poskytuje grafické uživatelské rozhraní.

5.3.2 Rozhraní přes příkazovou řádku

Rozhraní přes příkazovou řádku je navrženo podle vzoru MVC, což je zkratkou pro *Model–View–Controller*. Cílem tohoto vzoru je oddělit logiku aplikace od uživatelského rozhraní. [23]

View se stará o zobrazení dat uživateli. Obsahuje tak třídy starající se o renderování zpráv, v podobě tak zvaných stránek, uživateli. Použití těchto pomocných tříd usnadňuje otestování vrstvy *Controller*. [23]

Pro tuto práci jsou potřebné stránky s vypočítanými vlastními čísly a se statistikou výpočtu. Pro zpříjemnění uživatelské zkušenosti jsou zde navíc tyto stránky:

- Stránka s instrukcemi pro použití aplikace.
- Stránka shrnující zadané parametry.
- Stránka s informací o začátku výpočtu.
- Stránka s informací o ukončení výpočtu.

Všechny tyto stránky implementují abstraktní třídu `CTextPage`, která obsahuje jedinou metodu, a to metodu `Render`, která vrací řetězec znaků obsahující příslušné informace.

View se také stará o čtení uživatelského vstupu, o jeho validaci a o jeho převedení do konkrétních datových typů. Úkolem této vrstvy je též následně zavolat náležitou funkcionalitu vrstvy *Controller*. [23]

Controller potom reaguje na uživatelský vstup a podle zadaných požadavků náležitě modifikuje *Model*. [23]

Model obsahuje jak datový model, tak i veškerou podnikovou logiku aplikace. [23]

Touto logikou je v případě této práce především výpočet několika největších vlastních čísel vstupní matice, o jejíž načtení se stará opět tato vrstva. V této práci je tak vrstvou *Model* celá výpočetní část kalkulačky.

5.3.3 Grafické uživatelské rozhraní

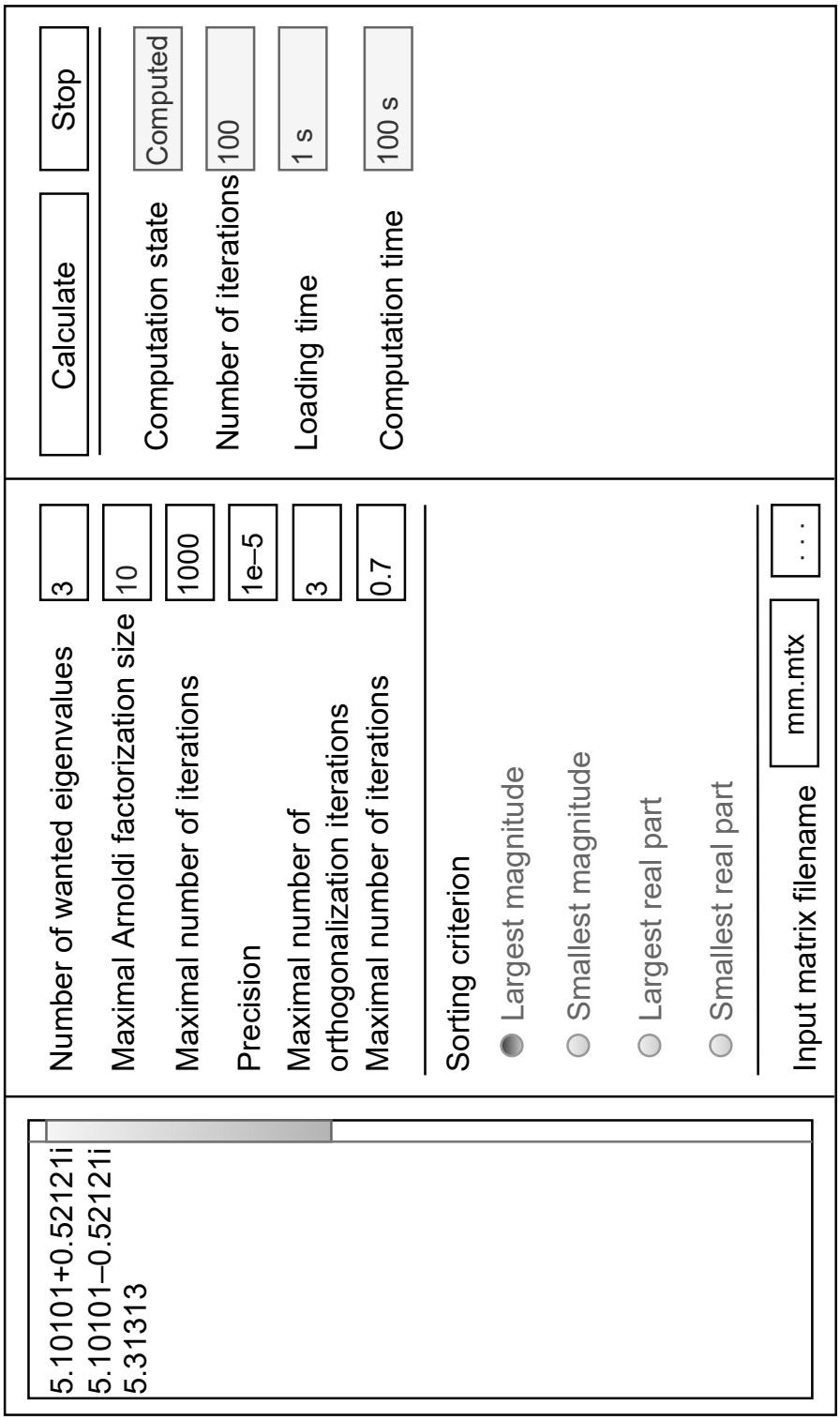
Grafické uživatelské rozhraní taktéž dodržuje architekturu MVC.

5.4 Vzhled grafického uživatelského rozhraní

Vzhled grafického uživatelského rozhraní vychází z požadavků analyzovaných v kapitole 4. Návrh v podobě wireframe je na obrázku 5.2.

Okno kalkulačky je horizontálně rozděleno na tři sloupce:

- První sloupec je tvořen pouze z textového pole, které obsahuje vypočtená vlastní čísla a je možné jej posouvat dolů či nahoru, pokud je čísel tolik, že je nelze všechny naráz zobrazit v tomto textovém poli.
- Druhý sloupec je určen pro zadání požadavků na výpočet vlastních čísel. Lze takto zadat počet požadovaných vlastních čísel, následně maximální velikost Arnoldiho faktorizace, totiž velikost Arnoldiho faktorizace, při které má dojít k restartu a tím ke kompresi faktorizace, dále maximální počet iterací restartované Arnoldiho metody, požadovanou přesnost vlastních čísel, maximální počet iterací iterativní ortogonalizace



Obrázek 5.2: Wireframe grafického uživatelského rozhraní

společně s étou iterativní ortogonalizace. V neposlední řadě je možné vybrat srovnávací kritérium pro porovnávání vlastních čísel. Poslední řádek tohoto sloupce je určen pro zadání vstupní matice. Název souboru lze vložit ručně do příslušného textového pole, nebo s využitím sousedního tlačítka, jehož stisk otevře prohlížeč souborů v souborovém systému a uživatel pak může s jeho pomocí vybrat konkrétní soubor se vstupní maticí.

- Poslední sloupec nejprve obsahuje tlačítko pro spuštění výpočtu a vedle něj tlačítko pro ukončení výpočtu. Poté je zde zobrazena statistika výpočtu, a to aktuální stav výpočtu, počet již provedených iterací, doba načítání a doba běhu výpočtu.

Kapitola 6

Realizace

V této kapitole je diskutována implementace výpočetní části a grafického uživatelského rozhraní společně s rozhraním přes příkazovou řádku.

6.1 Dokumentace

Pro generování dokumentace je použit nástroj Doxygen [24]. Tento nástroj používá pro vygenerování dokumentace objektu formátovaný C++ komentář umístěný v kódu před tímto objektem. Takto lze zdokumentovat například třídy, jejich metody a atributy.

6.2 Struktura projektu

Adresářová struktura je důležitá pro vývojáře, protože mu usnadňuje orientaci v kódu aplikace. Proto se autor rozhodl implementaci rozčlenit do hluboké adresářové struktury, s důrazem na logické a intuitivní rozdělení tříd do jednotlivých adresářů.

Část adresářové struktury tohoto projektu lze vidět na stromové struktuře C.1 v příloze.

6.3 Výpočetní část

Výpočetní část je realizována formou čistě hlavičkové knihovny v programovacím jazyce C++ ve verzi C++20.

6.3.1 Použití knihoven

Pro výpočet vlastních čísel Hessenbergovy matice $\mathbb{H} \in \mathbb{R}^{n,n}$, kde $n \in \mathbb{N}$, je použit algoritmus implementovaný třídou `Eigen::RealSchur` z knihovny Eigen, konkrétně ve verzi Eigen 3.3.9, který nalezne reálný Schurův rozklad

$\mathbb{T} = \mathbb{Q}^T\mathbb{H}\mathbb{Q}$, kde $\mathbb{Q} \in \mathbb{R}^{n,n}$ je ortogonální matice a $\mathbb{T} \in \mathbb{R}^{n,n}$ je kvazi-trojúhelníková matice podobná matici $\mathbb{H}$. Matice $\mathbb{T}$ se pak použije pro zjištění spektra matice $\mathbb{H}$ a z matice $\mathbb{Q}$ se spočtou odhady přesnosti příslušných vlastních čísel.

Z knihovny Spectra je použita třída `Spectra::UpperHessenbergQR`, respektive `Spectra::DoubleShiftQR`, pro provedení kroku QR algoritmu s reálným, respektive komplexním, posunem, a to v průběhu QR algoritmu s posuny, který je klíčovou částí implicitního restartu Arnoldiho metody. Ve výchozím nastavení jsou tyto posuny zvoleny exaktně, tedy jako spočtená vlastní čísla Hessenbergovy matice, která byla na základě zvoleného srovnávacího kritéria vyhodnocena jako nezajímavá.

6.3.2 Generika nad skaláry

Datový model poskytuje generiku nad skaláry s využitím C++ šablon pouze pro vektory, a nikoli pro matice, a tedy ani algoritmus restartované Arnoldiho metody neposkytuje tuto generiku. Za výchozí skalár byl zvolen `double`, tedy celá výpočetní část pracuje implicitně s čísly s plovoucí desetinnou čárkou s dvojitou přesností. Autor plánoval poskytnout generiku nad skaláry, ovšem až poté, co by se mu podařilo naimplementovat všechny algoritmy, jejichž implementace je aktuálně delegována na externí knihovny.

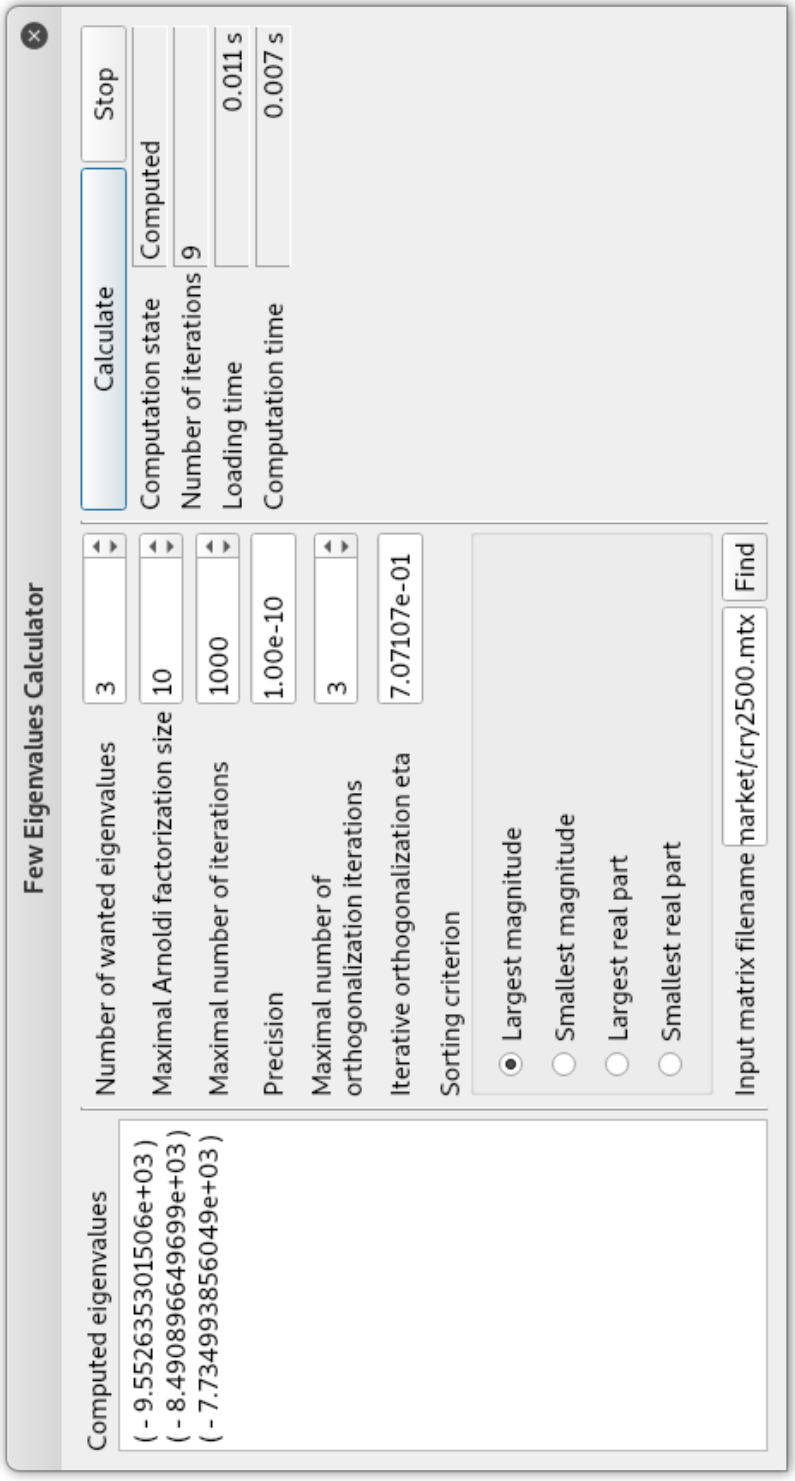
6.4 Rozhraní přes příkazovou řádku

Pro začátek výpočtu se spustí kalkulačka se všemi potřebnými parametry. S kalkulačkou po jejím spuštění již nelze dále komunikovat, vše totiž běží pouze na jednom vlákně.

6.5 Grafické uživatelské rozhraní

Výslednou podobu grafického uživatelského rozhraní lze vidět na obrázku 6.1.

Jako je výsledné okno vizuálně rozčleněno na tři sloupce spolu souvisejících informací, totiž první sloupec pro zobrazení vypočtených vlastních čísel, druhý sloupec pro zadání vstupních informací s požadavky na vlastní čísla a se specifikací vstupní matice a třetí, poslední, sloupec pro spuštění a zastavení výpočtu a pro zobrazení statistik, tak je implementované rozhraní rozčleněno i v kódu, a to postupně na třídy `CComputedEigenvaluesLayout`, `CCalculationInputParametersLayout` a `CCalculationExecutionLayout`, které všechny dědí z `QVBoxLayout`. Třída `QVBoxLayout` je třídou knihovny Qt 5, která se stará o svislé rozestavení jednotlivých zobrazitelných zařízení. Tyto třídy pak využívá třída `CRestartedArnoldiDialogView`, představující hlavní okno aplikace.



Obrázek 6.1: Grafické uživatelské rozhraní

Stisk tlačítka Calculate pak spustí v GUI vlákně metodu `CRestartedArnoldiDialogView::ReadInputParametersAndCalculate`, která pro načtení vstupních parametrů zavolá metodu `CCalculationInputParametersLayout::ReadInputParameters`, a poté s přečtenými parametry ve struktuře `CRestartedArnoldiInputArguments` spustí ve výpočetním vlákně metodu `CRestartedArnoldiController::ComputeFewEigenvalues`, jenž spočte požadované aproximace vlastních čísel. Výpočetní vlákno komunikuje průběžně s GUI vláknem zasíláním signálů. Takto je GUI vlákno informováno například o spuštění samotného výpočtu, nebo o změně stavu výpočtu, či o zkonvergování požadovaného počtu vlastních čísel.

Kapitola 7

Testování

7.1 Testovací data

Zhotovená kalkulačka načítá vstupní matice ve formátu MatrixMarket [19]. Pro její otestování zvolil autor tyto čtvercové, reálné a nesymetrické matice:

- Z repositáře Matrix Market [19]: tols90, tols4000, cry2500, olm5000 a af23560.
- Z kolekce řídkých matic University of Florida [25]: vas_stokes_4M, vas_stokes_2M, vas_stokes_1M, Freescale1 a ML_Geer.

7.2 Referenční řešení

Pro výpočet referenčních vlastních čísel byla využita knihovna Spectra, popsaná v kapitole 3.2. Výpočet byl spuštěn nad datovým typem `long double`. Testy, které vyžadovaly několik referenčních vlastních čísel matice $\mathbb{A} \in \mathbb{R}^{n,n}$, kde $n \in \mathbb{N}$, byly spouštěny s požadovanou přesností vždy větší než $\|\mathbb{A}\mathbb{v}_a - \lambda_a \mathbb{v}_a\|_2 / |\lambda_a|$ pro každou referenční vypočtenou aproximaci $\mathbb{v}_a \in \mathbb{C}^n$ vlastního vektoru matice $\mathbb{A}$ příslušejícího aproximaci vlastního čísla $\lambda_a \in \mathbb{C}$ s jednotkovou euklidovskou normou.

7.3 Testovací hardware

Testování proběhlo na notebooku s následujícími parametry:

- **Procesor:** Intel® Core™ i7-8550U CPU @ 1.80GHz × 8
- **RAM:** 7.7 GiB
- **HDD:** SSD 128 GB
- **Operační systém:** Kali GNU/Linux Rolling

7.4 Testování splnění požadavků

V této sekci je otestováno splnění požadavků kladených na kalkulačku v kapitole 4.

Jestliže není uvedeno jinak, všechny záznamy testů popisují testy provedené na rozhraní přes příkazovou řádku. Na grafickém uživatelském rozhraní pak byly provedeny z každé sady testů pouze některé testy testující základní funkčnost rozhraní.

Zejména v hlavičkách testovacích tabulek jsou použity následující názvy vstupních proměnných:

- d: Vstupní matice.
- k: Počet požadovaných vlastních čísel.
- m: Maximální velikost Arnoldiho faktorizace.
- M: Maximální počet iterací restartované Arnoldiho metody.
- τ: Přesnost vlastních čísel.
- $M^{(o)}$: Maximální počet iterací iterativní ortogonalizace.
- η: Éta iterativní ortogonalizace.
- s: Srovnávací kritérium.

A následující názvy výstupních proměnných:

- c: Počet vypočítaných vlastních čísel.
- e: Maximální absolutní chyba, tedy maximum z absolutních hodnot rozdílů vypočtených vlastních čísel od příslušných referenčních vlastních čísel, zaokrouhlená na dvě platné číslice.
- $\widetilde{e}$: Maximální absolutní relativní chyba, tedy maximum z absolutních hodnot rozdílů vypočtených vlastních čísel od příslušných referenčních vlastních čísel, vydělená absolutní hodnotou příslušného vlastního čísla, zaokrouhlená na dvě platné číslice.
- r: Počet iterací restartované Arnoldiho metody.
- t: Délka výpočtu, bez délky konstrukce potřebných struktur.

Srovnávací kritéria jsou dále označena takto:

- LM: Největší absolutní hodnota.
- SM: Nejmenší absolutní hodnota.
- LR: Největší reálná část.
- SR: Nejmenší reálná část.

7.4.1 Výchozí parametry

Každý z následujících testů testujících splnění analyzovaných požadavků byl proveden, jestliže daný test neuvádí jinak, s následujícími parametry:

- k, počet požadovaných vlastních čísel: 3.
- m, maximální velikost Arnoldiho faktorizace: 10.
- M, maximální počet iterací restartované Arnoldiho metody: 10^6.
- τ, přesnost vlastních čísel: 10^{-10}.
- $M^{(o)}$, maximální počet iterací iterativní ortogonalizace: 3.
- η, éta iterativní ortogonalizace: $\frac{\sqrt{2}}{2}$.
- s, srovnávací kritérium: LM, totiž největší absolutní hodnota.

7.4.2 Testování funkčních požadavků

Rozhraní přes příkazovou řádku i grafické uživatelské rozhraní bylo pro následující testy zkompilováno s přepínačem *O3*.

Test FP1 Zadání počtu vlastních čísel

Na obou rozhraních je možné zadat počet požadovaných vlastních čísel. Rozhraní přes příkazovou řádku rozpozná nevalidní vstup a výpočet se spustí jen tehdy, když je číslo správně formátováno, tedy se jedná o přirozené číslo větší než nula. Grafické uživatelské rozhraní dokonce ani neumožní vložit do vstupního pole nevalidní číslo.

Vstupy a výstupy testů jsou zaznamenány v tabulce 7.1.

Tabulka 7.1: Vstup a výstup testů FP1

i	d_i	k_i	m_i	c_i	e_i	$\widetilde{e}_i$
1	cry2500	1	5	1	$2{,}6 \times 10^{-8}$	$2{,}7 \times 10^{-12}$
2	olm5000	5	15	5	$2{,}0 \times 10^{-8}$	$7{,}8 \times 10^{-14}$
3	tols4000	15	40	16	$1{,}9 \times 10^{-6}$	$4{,}0 \times 10^{-10}$

Vždy bylo zobrazeno alespoň tolik vlastních čísel, kolik bylo požadováno. Maximální absolutní relativní chyba vypočítaných vlastních čísel je v prvních dvou testech menší než jejich požadovaná přesnost, totiž 10^{-10}, tedy je výstup těchto testů ve shodě s referenčním řešením, přesně podle vstupních požadavků. Maximální absolutní relativní chyba posledního testu je větší, než jak bylo požadováno, avšak autor tuto nesrovnalost hodnotí jako akceptovatelnou, protože se jedná o relativně malý rozdíl, totiž rozdíl mezi reálnou přesností $\widetilde{e}_3 = 4{,}0 \times 10^{-10}$ a výchozí požadovanou přesností 10^{-10}.

Závěr
Funkční požadavek je splněný.

Test FP2 Volba srovnávacího kritéria

Rozhraní přes příkazovou řádku umožňuje zvolit srovnávací kritérium zadáním příslušného řetězce znaků, které si vyhradilo pro identifikaci toho určitého srovnávacího kritéria. Nevalidní podoba identifikátoru je oznámena uživateli a výpočet se nespustí. Grafické uživatelské rozhraní pro zadání srovnávacího kritéria poskytuje sadu vzájemně výlučných přepínačů. Výchozí volbou je největší absolutní hodnota a z této sady je vždy vybrána právě jedna možnost. Obě rozhraní tak poskytují výběr z požadovaných čtyř srovnávacích kritérií.

Vstupy a výstupy testů tohoto funkčního požadavku lze vidět v tabulce 7.2.

Tabulka 7.2: Vstup a výstup testů FP2

i	d_i	m_i	s_i	c_i	e_i	$\widetilde{e}_i$
1	af23560	10	LM	4	$1{,}3 \times 10^{-9}$	$4{,}8 \times 10^{-12}$
2	tols90	30	SM	4	$1{,}8 \times 10^{-11}$	$1{,}5 \times 10^{-12}$
3	cry2500	10	LR	3	$6{,}1 \times 10^{-9}$	$2{,}1 \times 10^{-9}$
4	tols4000	10	SR	4	$6{,}3 \times 10^{-7}$	$1{,}3 \times 10^{-10}$

Pro každé srovnávací kritérium byl zobrazen nejméně požadovaný počet příslušných největších vlastních čísel, v postačující shodě s referenčním řešením.

Pro srovnávací kritérium SM, které za největší označí čísla s nejmenší absolutní hodnotou, byla nakonec zvolena matice typu pouze 90×90 a maximální velikost Arnoldiho faktorizace zde byla navýšena na 30, protože pro některé větší matice shledal autor úkol vypočítat několik vlastních čísel s nejmenší absolutní hodnotou jako příliš časově náročný. I pro zvolenou matici malých rozměrů trval výpočet déle, než jak by trval výpočet všech vlastních čísel této matice s využitím knihovny Eigen.

Výpočet několika vlastních čísel s nejmenší absolutní hodnotou shledává autor této bakalářské práce náročný i s použitím knihovny Spectra. Z toho autor usuzuje, že použitý algoritmus implicitně restartované Arnoldiho metody není tak výkonný pro výpočet vlastních čísel s nejmenší absolutní hodnotou, jako pro výpočet vlastních čísel s největší absolutní hodnotou, jak je to zmíněno i v knize [6], a odmítá odpovědnost za tuto potíž.

Závěr
Funkční požadavek je splněný.

Test FP3 Zadání přesnosti aproximací

Požadovanou přesnost aproximací vlastních čísel je možné zadat na obou rozhraních. Nevalidní vstup je opět ošetřen.

Splnění tohoto požadavku je otestováno testy, jejichž vstupy a výstupy jsou zobrazeny v tabulce 7.3.

Tabulka 7.3: Vstup a výstup testů FP3

i	d_i	τ_i	c_i	e_i	$\widetilde{e}_i$
1	olm5000	10^{-3}	3	$2{,}7 \times 10^{2}$	$1{,}1 \times 10^{-3}$
2	olm5000	10^{-5}	4	$5{,}3 \times 10^{0}$	$2{,}1 \times 10^{-5}$
3	olm5000	10^{-10}	3	$3{,}7 \times 10^{-8}$	$1{,}4 \times 10^{-13}$
4	olm5000	10^{-15}	3	$1{,}9 \times 10^{-7}$	$7{,}5 \times 10^{-13}$
5	af23560	10^{-15}	4	$1{,}5 \times 10^{-12}$	$5{,}6 \times 10^{-15}$

Výsledná přesnost vlastních čísel vypočítaných v testu 4, která je zaznamenána hodnotou $\widetilde{e}_4 = 7{,}5 \times 10^{-13}$, je nižší, než jak bylo požadováno, autor práce však stále tuto chybu hodnotí jako přijatelnou.

Závěr
Funkční požadavek je splněný.

Test FP4 Zadání maximální velikosti Arnoldiho faktorizace

Obě rozhraní umožňují zadat maximální velikost Arnoldiho faktorizace a detekují nevalidní vstup. To, že zadané číslo je pak skutečně adekvátně použito jako velikost faktorizace, při které dochází k restartu, bylo ověřeno pomocí ladicího programu.

Vstupy a výstupy testů v tabulce 7.4 ukazují závislost počtu iterací a délky běhu výpočtu na volbě maximální velikosti Arnoldiho faktorizace. Každé měření proběhlo pouze jednou.

Závěr
Funkční požadavek je splněný.

Test FP5 Zadání maximálního počtu restartů

Na obou rozhraních lze zadat maximální počet restartů Arnoldiho metody. Nevalidní vstup je opět ošetřen.

Vstupy a výstupy testů jsou zaznamenány v tabulce 7.5.

Testy 3 a 4 ukazují, že jsou po dosažení maximálního povoleného počtu restartů zobrazena již na požadovanou přesnost zkonvergovaná vlastní čísla, i když je jejich počet menší než požadovaný počet vlastních čísel.

Tabulka 7.4: Vstup a výstup testů FP4

i	d_i	m_i	c_i	e_i	$\widetilde{e}_i$	r_i	t_i (s)
1	olm5000	7	3	$3{,}8 \times 10^{-8}$	$1{,}5 \times 10^{-13}$	38693	28,2
2	olm5000	10	3	$2{,}8 \times 10^{-8}$	$1{,}1 \times 10^{-13}$	17764	25,2
3	olm5000	13	3	$6{,}0 \times 10^{-8}$	$2{,}4 \times 10^{-13}$	10758	25,3
4	olm5000	17	3	$1{,}9 \times 10^{-8}$	$7{,}4 \times 10^{-14}$	3295	10,7
5	olm5000	20	3	$4{,}0 \times 10^{-8}$	$1{,}6 \times 10^{-13}$	3843	18,3
6	olm5000	30	3	$5{,}3 \times 10^{-9}$	$2{,}1 \times 10^{-14}$	1309	12,2
7	olm5000	40	3	$3{,}9 \times 10^{-9}$	$1{,}5 \times 10^{-14}$	643	11,0
8	olm5000	50	3	$2{,}7 \times 10^{-9}$	$1{,}1 \times 10^{-14}$	501	13,0
9	olm5000	100	3	$5{,}1 \times 10^{-9}$	$2{,}0 \times 10^{-14}$	156	16,2

Tabulka 7.5: Vstup a výstup testů FP5

i	d_i	M_i	c_i	e_i	$\widetilde{e}_i$	r_i
1	olm5000	1	0	-	-	1
2	olm5000	17750	0	-	-	17750
3	olm5000	17760	1	$9{,}1 \times 10^{-9}$	$3{,}6 \times 10^{-14}$	17760
4	olm5000	17763	2	$9{,}5 \times 10^{-9}$	$3{,}7 \times 10^{-14}$	17763
5	olm5000	17764	3	$3{,}7 \times 10^{-8}$	$1{,}4 \times 10^{-13}$	17764
6	olm5000	100000	3	$3{,}7 \times 10^{-8}$	$1{,}4 \times 10^{-13}$	17764

Závěr
Funkční požadavek je splněný.

Test FP6 Zadání maximálního počtu iterací iterativní ortogonalizace

Maximální počet iterací iterativní ortogonalizace lze zadat na obou rozhraních, přičemž je požadována správnost formátu vstupního čísla.

V tabulce 7.6 lze vidět vstupy a výstupy testů.

Tabulka 7.6: Vstup a výstup testů FP6

i	d_i	m_i	$M_i^{(o)}$	c_i	e_i	$\widetilde{e}_i$	r_i	t_i (s)
1	olm5000	30	1	4	$3{,}7 \times 10^{6}$	$1{,}5 \times 10^{1}$	16	0,1
2	olm5000	30	2	3	$5{,}3 \times 10^{-9}$	$2{,}1 \times 10^{-14}$	1309	12,6
3	olm5000	30	3	3	$5{,}3 \times 10^{-9}$	$2{,}1 \times 10^{-14}$	1309	12,7
4	olm5000	30	4	3	$5{,}3 \times 10^{-9}$	$2{,}1 \times 10^{-14}$	1309	12,2

Z testu 1, kde $M_1^{(o)} = 1$, lze vypozorovat, že pouze jedna ortogonalizace nemusí stačit pro zajištění dostatečné ortogonality vektorů představujících ortonormální bázi Krylovova podprostoru, což může vést až ke zcela špatnému

řešení, jak je to popsáno i v knize [6]. V testu 1 jsou nalezená vlastní čísla zcela chybná. S využitím ladicího programu a vynásobením transponované matice ortonormálních vektorů touto maticí zprava se autor přesvědčil o tom, že je v tomto testu skutečně postupně ztracena ortogonalita těchto vektorů. Dokonce není zachována ani jednotková norma prvních $k_1 = 3$ vektorů, kde $k_1 \in \mathbb{N}$ je počet požadovaných vlastních čísel, které jsou implicitně upravovány při každém restartu.

Další testy s maximálním počtem iterací iterativní ortogonalizace však již neselhaly, neboť bylo vypočteno správné řešení.

Závěr
Funkční požadavek je splněný.

Test FP7 Zadání éty iterativní ortogonalizace

Étu iterativní ortogonalizace je možné zadat na obou rozhraních. Výpočet není s nevalidním číslem spuštěn.

Splnění tohoto funkčního požadavku je otestováno sadou testů, jejichž vstupy a výstupy lze vidět v tabulce 7.7.

Tabulka 7.7: Vstup a výstup testů FP7

i	d_i	m_i	η_i	c_i	e_i	$\widetilde{e}_i$	r_i	t_i (s)
1	olm5000	30	0,0	4	$3{,}7 \times 10^{6}$	$1{,}5 \times 10^{1}$	16	0,1
2	olm5000	30	0,1	4	$3{,}7 \times 10^{6}$	$1{,}5 \times 10^{1}$	16	0,1
3	olm5000	30	0,3	4	$3{,}1 \times 10^{6}$	$1{,}2 \times 10^{1}$	17	0,1
4	olm5000	30	0,4	4	$1{,}5 \times 10^{6}$	$6{,}0 \times 10^{0}$	12	0,1
5	olm5000	30	0,5	3	$5{,}7 \times 10^{-9}$	$2{,}2 \times 10^{-14}$	1309	13,2
6	olm5000	30	0,7	3	$5{,}3 \times 10^{-9}$	$2{,}1 \times 10^{-14}$	1309	12,4
7	olm5000	30	0,9	3	$5{,}6 \times 10^{-9}$	$2{,}2 \times 10^{-14}$	1309	12,8
8	olm5000	30	1,0	3	$6{,}4 \times 10^{-9}$	$2{,}5 \times 10^{-14}$	1309	13,9

Z těchto testů lze vidět že příliš malá éta iterativní ortogonalizace může zapříčinit ztrátu ortogonality a to pak zcela špatné řešení, jak je to zmíněno již v testu předchozího funkčního požadavku.

Závěr
Funkční požadavek je splněný.

Test FP8 Načtení vstupní matice

Načtení vstupní reálné řídké a nesymetrické matice ve validním souřadnicovém formátu MatrixMarket je již otestováno předchozími funkčními testy.

Obě rozhraní pro načtení vstupní matice požadují název souboru, ze kterého se má vstupní matice uložená v souřadnicovém formátu MatrixMarket

pro obecné nesymetrické matice nad reálným tělesem načíst. Nevalidní vstup je ošetřen:

- Nepodporované specifikace v MatrixMarket hlavičce jsou detekovány.
- Matice, jejíž počet řádků není roven jejímu počtu sloupců, není povolena.
- Načítání selže, jestliže je v souboru jiný počet nenulových prvků, než jak bylo na začátku oznámeno, nebo jestliže je některé číslo špatně formátováno, anebo je index nenulového prvku větší, než jaké jsou rozměry vstupní matice.

Závěr
Funkční požadavek je splněný.

Test FP9 Libovolné načtení počátečního vektoru

Rozhraní neumožňují zadat počáteční vektor, tedy jej kalkulačka zvolí sama.

Závěr
Funkční požadavek je splněný.

Test FP10 Výpočet vlastních čísel

Splnění tohoto požadavku je již otestováno a ověřeno v předchozích testech.

Závěr
Funkční požadavek je splněný.

Test FP11 Zobrazení vypočtených vlastních čísel

Předchozí testy využívaly tuto funkcionalitu a ověřily ji.

Rozhraní přes příkazovou řádku vypíše vypočtená vlastní čísla přímo do konzole a desktopové rozhraní vypočítaná vlastní čísla zobrazí v textovém poli na levé straně okna aplikace.

Závěr
Funkční požadavek je splněný.

Test FP12 Zobrazení statistik

V obou rozhraních jsou zobrazeny požadované statistiky, totiž délka výpočtu a počet iterací restartované Arnoldiho metody, jak již zdárně otestovaly předchozí testy.

Závěr
Funkční požadavek je splněný.

7.4.3 Testování nefunkčních požadavků

Test NP1 Rozhraní přes příkazovou řádku

Všechny testy funkčních požadavků byly provedeny primárně na rozhraní přes příkazovou řádku.

Závěr
Nefunkční požadavek je splněný.

Test NP2 Jednoduché desktopové rozhraní

Funkčnost byla pomocí výběru testů otestována i na desktopovém rozhraní.

Závěr
Nefunkční požadavek je splněný.

Test NP3 Platforma

Testování splnění funkčních požadavků bylo provedeno na platformě Linux, jak je uvedeno již v kapitole 7.3.

Závěr
Nefunkční požadavek je splněný.

Test NP4 Rychlost výpočetní části

Pro tento test byly obě srovnávané řešení zkompilovány s přepínačem *Ofast*.

Autor vybral pro tento test matice vas_stokes_4M, vas_stokes_2M, vas_stokes_1M, Freescale1 a ML_Geer, které splňují specifikace požadavku NP4.

Po implementovaném řešení s názvem fewEigenvaluesCalculator a po knihovně Spectra bylo požadováno aproximovat 5 vlastních čísel s přesností 10^{-10}, což jsou hodnoty vyplývající přímo z popisu požadavku NP4. Za maximální velikost Arnoldiho faktorizace byla zvolena hodnota 15. Pro implementované řešení pak byl dále zadán maximální počet iterací iterativní ortogonalizace roven 3 a éta iterativní ortogonalizace rovna $\frac{\sqrt{2}}{2}$.

Proměnné $\widetilde{e}_i^s$, r_i^s a t_i^s mají stejný význam jako proměnné bez horního indexu, pouze označují hodnoty vztahující se ke knihovně Spectra.

Výsledky testů NP4 lze vidět v tabulce 7.8.

Tabulka 7.8: Vstup a výstup testů NP4

		fewEigenvaluesCalculator			Spectra		
i	d_i	$\widetilde{e}_i$	r_i	t_i (s)	$\widetilde{e}_i^s$	r_i^s	t_i^s (s)
1	vas_stokes_4M	$4{,}3 \times 10^{-14}$	101	407	$5{,}6 \times 10^{-14}$	123	546
2	vas_stokes_2M	$2{,}5 \times 10^{-14}$	85	170	$3{,}6 \times 10^{-14}$	80	172
3	vas_stokes_1M	$2{,}0 \times 10^{-14}$	64	55,8	$2{,}4 \times 10^{-14}$	74	66,5
4	Freescale1	$4{,}7 \times 10^{-15}$	9	16,2	$2{,}4 \times 10^{-15}$	8	19,0
5	ML_Geer	$1{,}5 \times 10^{-14}$	20	45,5	$2{,}3 \times 10^{-14}$	21	43,2

Pro žádnou ze zvolených vstupních matic není doba výpočetního běhu implementované kalkulačky více než 10× větší než doba výpočetního běhu knihovny Spectra.

Závěr
Nefunkční požadavek je splněný.

Závěr

Hlavním cílem práce bylo vypracovat návrh kalkulačky několika vlastních čísel reálných nesymetrických matic a podle návrhu tuto kalkulačku implementovat, společně s rozhraním přes příkazovou řádku a s jednoduchým desktopovým rozhraním.

Nejdříve byly zavedeny základní teoretické pojmy. Také byly zmíněny a popsány jedny z nejčastějších formátů uložení řídkých matic. S pomocí těchto pojmů byla uvedena Arnoldiho metoda a její restart. Poté byly popsány vybrané knihovny pro maticové operace a některé knihovny s implementací restartované Arnoldiho metody. Následně, v poslední kapitole teoretické části, byly analyzovány požadavky na výsledný produkt.

Následně byla kalkulačka navržena, a to tak, aby byly splněny všechny požadavky, které na ni byly kladeny. Pro výpočet několika největších vlastních čísel vstupní matice byla zvolena implicitně restartovaná Arnoldiho metoda. Pro implementaci výpočetní části, rozhraní přes příkazovou řádku a grafického uživatelského rozhraní byl zvolen programovací jazyk C++. Pro realizaci desktopového rozhraní byl po diskuzi vybrán framework Qt. Výpočetní část byla navržena jako knihovna, kterou pak využívá rozhraní přes příkazovou řádku společně s grafickým uživatelským rozhraním. V neposlední řadě byl navržen i vzhled grafického uživatelského rozhraní.

Podle návrhu byl produkt implementován, včetně rozhraní přes příkazovou řádku a grafického uživatelského rozhraní. Dokumentaci produktu lze vygenerovat z jeho zdrojového kódu nástrojem Doxygen. Testování ukázalo, že byly splněny všechny analyzované požadavky.

V budoucnosti by bylo možné ve výpočetní části doimplementovat především krok QR algoritmu s reálným a s komplexním posunem, či algoritmus pro výpočet všech vlastních čísel reálné Hessenbergovy matice. Bylo by vhodné zavést generiku nad skaláry i u matic, či přidat podporu pro komplexní vstupní matice. Autor by za zajímavé považoval rozšíření aplikace také o více způsobů restartu Arnoldiho metody, o více způsobů ortogonalizace, o možnost aproximace vlastních čísel jen z určité části spektra, totiž o možnost výpočtu

takových vlastních čísel vstupní matice, které jsou nejblíže například číslu 3,3. Dalším rozšířením by mohlo být přidání další metody aproximací několika vlastních čísel vstupní řídké matice, třeba i specializované pro symetrické matice. Dále se nabízí výpočetní část paralelizovat. Je zde možnost přidat více formátů vnitřního či vnějšího uložení řídkých matic, či přidat více způsobů vizualizace výsledků.

Literatura

[1] Dombek, D.; Kalvoda, T.; Kleprlík, L.; aj.: Lineární algebra. [online], 2020. Dostupné z: `https://kam.fit.cvut.cz/deploy/bi-lin//lin-text.pdf`

[2] Saad, Y.: *Numerical Methods for Large Eigenvalue Problems*. Society for Industrial and Applied Mathematics, 2011, doi:10.1137/1.9781611970739.

[3] Golub, G. H.; Van Loan, C. F.: *Matrix computations*. Baltimore, USA: Johns Hopkins University Press, 2013, ISBN 978-1-4214-0794-4.

[4] Wilkinson, J. H.; Reinsch, C.: *Handbook for Automatic Computation: Volume II: Linear Algebra*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1971, doi:10.1007/978-3-642-86940-2.

[5] Arnoldi, W. E.: The principle of minimized iterations in the solution of the matrix eigenvalue problem. *Quarterly of applied mathematics*, ročník 9, č. 1, 1951: s. 17–29.

[6] Lehoucq, R. B.; Sorensen, D. C.; Yang, C.: *ARPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1998, doi:10.1137/1.9780898719628.

[7] Hernández, V.; Román, J. E.; Tomás, A.; aj.: Arnoldi Methods in SLEPc. Technická Zpráva STR-4, Universitat Politècnica de València, 2006, [cit. 2022-4-11]. Dostupné z: `https://slepc.upv.es`

[8] Blackford, L. S.; Petitet, A.; Pozo, R.; aj.: An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, ročník 28, č. 2, 2002: s. 135–151.

[9] Anderson, E.; Bai, Z.; Bischof, C.; aj.: *LAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, třetí vydání, 1999.

[10] Blackford, L. S.; Choi, J.; Cleary, A.; aj.: *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[11] BLACS. [online], [cit. 2022-4-11]. Dostupné z: `http://www.netlib.org/blacs/`

[12] Guennebaud, G.; Jacob, B.; aj.: Eigen v3. [online], [cit. 2022-4-11]. Dostupné z: `https://eigen.tuxfamily.org`

[13] ARPACK. [online], [cit. 2022-4-11]. Dostupné z: `https://www.caam.rice.edu/software/ARPACK/`

[14] MPI Forum. [online], [cit. 2022-4-11]. Dostupné z: `https://www.mpi-forum.org/`

[15] Spectra. [online], [cit. 2022-4-11]. Dostupné z: `https://spectralib.org`

[16] Hernández, V.; Román, J.; Vidal, V.: SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. *ACM Transactions on Mathematical Software (TOMS)*, ročník 31, č. 3, 2005: s. 351–362.

[17] Balay, S.; Abhyankar, S.; Adams, M. F.; aj.: PETSc/TAO Users Manual. Technická Zpráva ANL-21/39 - Revision 3.16, Argonne National Laboratory, 2021.

[18] Hernández, V.; Román, J. E.; Tomás, A.; aj.: Orthogonalization Routines in SLEPc. Technická Zpráva STR-1, Universitat Politècnica de València, 2007, [cit. 2022-4-11].

[19] Matrix Market. [online], [cit. 2022-4-11]. Dostupné z: `https://math.nist.gov/MatrixMarket`

[20] Qt. [online], [cit. 2022-4-11]. Dostupné z: `https://www.qt.io`

[21] CMake. [online], [cit. 2022-4-11]. Dostupné z: `https://cmake.org`

[22] wxWidgets. [online], [cit. 2022-4-11]. Dostupné z: `https://www.wxwidgets.org`

[23] Drozdík, M.: *UI Layer*. [přednáška]. Praha: Fakulta informačních technologií ČVUT v Praze, 27. října 2021.

[24] van Heesch, D.: Doxygen. [online], [cit. 2022-4-11]. Dostupné z: `https://www.doxygen.nl`

[25] Davis, T. A.; Hu, Y.: The University of Florida Sparse Matrix Collection. *ACM Trans. Math. Softw.*, ročník 38, č. 1, dec 2011, ISSN 0098-3500, doi:10.1145/2049662.2049663. Dostupné z: `https://doi.org/10.1145/2049662.2049663`

Příloha A

Seznam použitých zkratek

CLI Command Line Interface

COO Coordinate format

CRTP Curiously Recurring Template Pattern

CSC Compressed Sparse Column format

CSR Compressed Sparse Row format

FP Funkční požadavek

GUI Graphical User Interface

LM Largest Magnitude

LR Largest Real part

MVC Model–View–Controller

NNZ Number of Nonzeros

NP Nefunkční požadavek

SM Smallest Magnitude

SR Smallest Real part

Příloha **B**

Upravená Arnoldiho metoda

Algoritmus 3: `modifiedArnoldi`: Upravená Arnoldiho metoda

Input: $m, n, k \in \mathbb{N}$, $k < m \leq n$,
$\mathbb{A} \in \mathbb{C}^{n,n}$ vstupní matice,
$\mathbb{V} \in \mathbb{C}^{n,k}$ matice s ortonormálními sloupci,
$\mathbb{H} \in \mathbb{C}^{k,k}$ Hessenbergova matice,
$\mathbb{f} \in \mathbb{C}^{n}$ vektor ortogonální vzhledem k matici $\mathbb{V}$,
tak, že platí $\mathbb{A}\mathbb{V} = \mathbb{V}\mathbb{H} + \mathbb{f}\mathbb{e}_k^T$,
kde $\mathbb{e}_k \in \mathbb{R}^k$ a $\mathbb{e}_k^T = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$

Output: $\mathbb{V} \in \mathbb{C}^{n,m}$ matice s ortonormálními sloupci,
$\mathbb{H} \in \mathbb{C}^{m,m}$ Hessenbergova matice,
$\mathbb{f} \in \mathbb{C}^{n}$ vektor ortogonální vůči sloupcům matice $\mathbb{V}$,
tak, že platí $\mathbb{A}\mathbb{V} = \mathbb{V}\mathbb{H} + \mathbb{f}\mathbb{e}_m^T$,
kde $\mathbb{e}_m \in \mathbb{R}^m$ a $\mathbb{e}_m^T = \begin{pmatrix} 0 & \cdots & 0 & 1 \end{pmatrix}$

for $j = k, k+1, \ldots, m-1$ **do**
  if $\|\mathbb{f}\|_2 = 0$ **then**
    zastav
  end
  $\mathbb{H}_{j+1,j} \leftarrow \|\mathbb{f}\|_2$
  $\mathbb{V}_{:,j+1} \leftarrow \mathbb{f}/\|\mathbb{f}\|_2$
  $\mathbb{f} \leftarrow \mathbb{A}\mathbb{V}_{:,j+1}$
  ortogonalizuj $\mathbb{f}$ vzhledem k matici $\mathbb{V}_{:,1:j+1}$ a koeficienty ortogonalizace ulož do sloupce $\mathbb{H}_{1:j+1,j+1}$
end

Zdrojem algoritmu 3 je kniha [6] a technická zpráva [7].

Příloha C

Struktura projektu

```
impl
├── Doxyfile ..... konfigurační soubor pro vygenerování dokumentace
├── Makefile ..... příkazy pro sestavení projektu
├── README.md ..... stručný popis projektu, návod na sestavení a použití
└── src ..... zdrojové kódy kalkulačky
    ├── debug ..... funkce pro ladění
    ├── test ..... adresář s testy
    ├── solver ..... výpočetní část
    │   ├── arnoldi ..... třídy pro Arnoldiho metodu
    │   │   ├── basic ..... třídy pro základní Arnoldiho metodu
    │   │   └── restarted ..... třídy pro restartovanou Arnoldiho metodu
    │   ├── converter ..... třídy pro převod něčeho na něco jiného
    │   ├── decomposition ..... maticové rozklady
    │   ├── eigenlibhelpers . pomocné funkce pro práci s knihovnou Eigen
    │   ├── exporter ..... třídy exportující něco někam
    │   ├── loader ..... třídy, které něco načítají
    │   ├── model ..... datové modely
    │   │   ├── matrix ..... datové modely matic
    │   │   ├── vector ..... datové modely vektorů
    │   │   └── view ..... třídy usnadňující práci s alokovanou pamětí
    │   ├── observer ..... posluchači
    │   ├── operator ..... třídy s vybranými maticovými operacemi
    │   ├── orthogonalization ..... ortogonalizace
    │   │   ├── basic ..... třídy pro základní ortogonalizaci
    │   │   └── refined ..... třídy pro pokročilejší ortogonalizaci užívající
    │   │       základní ortogonalizaci
    │   └── sorting ..... třídy pro řazení
    └── ui ..... uživatelská rozhraní
        ├── cli ..... rozhraní přes příkazovou řádku
        │   ├── CMakeLists.txt ..... instrukce pro sestavení CLI
        │   ├── controller ..... vrstva Controller
        │   ├── main.cpp ..... soubor s funkcí main pro CLI
        │   └── view ..... vrstva View
        │       ├── option ..... rozpoznání a převedení vstupních parametrů
        │       └── page ..... stránky
        ├── gui ..... grafické uživatelské rozhraní
        │   ├── CMakeLists.txt ..... instrukce pro sestavení GUI
        │   ├── controller ..... vrstva Controller
        │   ├── exporter ..... třídy exportující něco někam
        │   ├── main.cpp ..... soubor s funkcí main pro GUI
        │   └── view ..... vrstva View
        │       ├── dialogpart ..... sloupce hlavního dialogového okna
        │       ├── observer .. posluchači změn hodnot proměnných výpočtu
        │       └── page ..... stránky
        └── uihelpers ..... funkcionality společné pro více rozhraní
```

Obrázek C.1: Struktura projektu

Příloha D

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `src`
  - `impl` ..... zdrojové kódy implementace
    - `README.md` ... stručný popis projektu, návod na sestavení a použití
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF