FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Algoritmy pro výpočet vlastních čísel matic
Student:	Lucie Procházková
Vedoucí:	Ing. Tomáš Kalvoda, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	do konce letního semestru 2021/2022

Pokyny pro vypracování

1) Seznamte se s problémem výpočtu vlastních čísel a vlastních vektorů matic. Zaměřte se na matematickou formulaci problému, jeho možné aplikace a historický vývoj této problematiky.
2) Proveďte rešerši algoritmů používaných pro výpočet vlastních čísel a jejich vlastností, zaměřte se nejen na 'obecné' algoritmy, ale i na algoritmy využívající strukturu dané matice (symetrické, tridiagonální, řídké, atp.).
3) Implementujte QR algoritmus s Givensovými rotacemi v prostředí Julia.
4) Experimentálně porovnejte výkonnost dostupných algoritmů (např. v Mathematica a LAPACK) a vaší implementace na vhodně zvolených třídách matic.

Elektronicky schválil/a doc. Ing. Jan Janoušek, Ph.D. dne 17. prosince 2020 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Algoritmy pro výpočet vlastních čísel

Lucie Procházková

Katedra teoretické informatiky
Vedoucí práce: Ing. Tomáš Kalvoda, Ph.D.

12. května 2021

Poděkování

Ráda bych na tomto místě poděkovala především svému vedoucímu Ing. Tomáši Kalvodovi, Ph.D. Děkuji za vedení, pravidelné konzultace, cenné připomínky a rady a veškerou pomoc při psaní práce. Dále pak chci poděkovat Antonínu Křížovi, Adamu Procháskovi, Jakubu Horákovi a zbytku týmu Student Hub za motivaci, morální podporu, korektury a trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla 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 12. května 2021 .....

Odkaz na tuto práci

Procházková, Lucie. *Algoritmy pro výpočet vlastních čísel*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Cílem práce je shrnout historii a aplikace výpočtu vlastních čísel. Práce analyzuje některé algoritmy používané pro výpočet vlastních čísel. V praktické části práce je implementován QR algoritmus s Givensovými rotacemi v jazyce Julia a tato implementace je porovnána s již dostupnými implementacemi tohoto algoritmu v prostředí Mathematica a knihovně LAPACK. Implementace dosahuje horších časů i přesnosti než srovnávané alternativy, na rozdíl od nich však poskytuje možnost ovlivňovat přesnost výsledku a poskytuje výpis algoritmu.

Klíčová slova vlastní čísla, vlastní vektory, problém vlastních čísel, QR algoritmus, Givensovy rotace, programovací jazyk Julia

Abstract

This thesis aims to summarize the history and applications of eigenvalue computation. The work analyzes some eigenvalue algorithms. In the practical part of the thesis, a QR algorithm with Givens rotations is implemented in the Julia language and compared with the already available implementations of this algorithm in the Mathematica environment and the LAPACK library. The implementation is slower and less precise than already developed implementations, it does however provides possibility of logging the run of algorithm and possibility of setting precision.

Keywords eigenvalues, eigenvectors, QR algorithm, Julia language, eigenvalue problem, Givens Rotation

Obsah

Seznam obrázků

Seznam výpisů kódu

Úvod

Vlastní čísla, vektory a jejich výpočet jsou jedním ze základních problémů lineární algebry, kterému se žádný student této oblasti matematiky nevyhne. Lineární algebra má své kořeny již ve starověku, kdy se řešily lineární rovnice týkající se například výsadby plodin. Samotná oblast vlastních čísel, vektorů a jejich výpočtu doznala většího rozmachu až v období novověku zejména pak v devatenáctém století. Algoritmy na výpočet vlastních čísel se pak nejvíce rozvinuly ve dvacátém století a to především díky využití počítačů. Tato problematika nachází širokých aplikací nejenom v matematice, ale i ve fyzice, v biologii, ve stavebnictví, v chemii nebo například v informatice při implementaci vyhledávacích softwarů.

Toto téma jsem si vybrala, jelikož při výzkumném projektu, kde jsme se zabývali spektrální analýzou Schrödingerova operátoru na rovinných oblastech, byl právě výpočet vlastních čísel na řídkých maticích nejnáročnější částí výpočtu. Proto jsem se rozhodla v rámci své bakalářské práce prozkoumat jakým způsobem se vlastní čísla počítají, jaké možnosti jejich výpočtu jsou, jaké algoritmy jsou efektivní pro jaké třídy matic a dále jaké jsou možnosti zefektivnění a zrychlení tohoto výpočtu.

V této práci si kladu za cíl zmapovat vývoj výpočtu vlastních čísel v průběhu dějin, aplikace této problematiky a analyzovat běžně používané algoritmy pro tento výpočet. V praktické části práce jsem se pak zaměřila na implementaci QR algoritmu s Givensovými rotacemi v jazyce Julia a experimentální porovnání výkonnosti této implementace s již dostupnými implementacemi v prostředí Mathematica a v knihovně LAPACK.

V první kapitole stručně shrnuji základní pojmy a definice související s problematikou vlastních čísel. Analýza této problematiky je obsahem dalších dvou kapitol, přičemž v rámci druhé kapitoly se věnuji historii vlastních čísel, vektorů a jejich výpočtu přes starověk, středověk a především pak rozebírám vývoj této problematiky v novověku a ve dvacátém století. V kapitole třetí pak následuje hrubá analýza možných aplikací problému, přičemž je kladen

důraz na aplikace na poli informatiky a matematiky. Čtvrtá kapitola se pak věnuje analýze vybraných algoritmů pro výpočet vlastních čísel, konkrétně mocninné metody, Lanczosovy metody a QR algoritmu.

V páté kapitole pak rozebírám vlastní implementaci QR algoritmu s Givensovými rotacemi v jazyce Julia a srovnávám implementaci s již dostupnými implementacemi v prostředí Mathematica a v knihovně LAPACK.

Kapitola **1**

Základní pojmy a definice

Aby bylo možné hovořit o vlastních číslech matic, je potřeba zavést několik zcela zásadních pojmů. V této kapitole se vychází z definic ze studijního textu předmětu Lineární algebra na FIT ČVUT [1]. Tento text lze také využít k hlubšímu pochopení zde definovaných pojmů i pro dohledání důkazů tvrzení z této kapitoly.

1.1 Základní pojmy

Prvním z pojmů je vektorový prostor, jeden ze základních pojmů celé lineární algebry.

Definice 1.1.1 (Vektorový prostor) *Ať* T *je libovolné komutativní těleso, jeho neutrální prvky vůči operacím sčítání resp. násobení se označí* 0*, resp.* 1*. Nechť je dále dána neprázdná množina* V *a dvě zobrazení* $: V \times V \to V$, $: \mathrm{T} \times V \to V$*. Řekneme, že* V *je vektorový prostor nad tělesem* T *s vektorovými operacemi* a *, právě když platí následující axiomy vektorového prostoru:*

1. $a, b \quad V : a \quad b = b \quad a,$
2. $a, b, c \quad V : (a \quad b) \quad c = a \quad (b \quad c),$
3. $\alpha, \beta \quad \mathrm{T}, \ a \quad V : \alpha \quad (\beta \quad a) = (\alpha\beta) \quad a,$
4. $\alpha \quad \mathrm{T}, \ a, b \quad V : \alpha \quad (a \quad b) = (\alpha \quad a) \quad (\alpha \quad b),$
5. $\alpha, \beta \quad \mathrm{T}, \ a \quad V : (\alpha + \beta) \quad a = (\alpha \quad a) \quad (\beta \quad a),$
6. $a \quad V : 1 \quad a = a,$
7. $\theta \quad V, \ a \quad V : 0 \quad a = \theta.$

Tato definice umožňuje vystavět celou lineární algebru, operace v tomto textu vždy splňují tyto definice. Typicky se v tomto textu uvažuje těleso reálných čísel R, případně těleso čísel komplexních C, operacemi resp. , je zde myšleno sčítání, resp. násobení. Vektorovým prostorem pak se typicky myslí R^n, případně C^n, což je vždy jasně řečeno.

Vektory je důležité umět porovnávat. K tomuto účelu se používají různé normy umožňující počítat vzdálenosti vektorů. Norma je funkce $\cdot : R^n \to R$, která splňuje následující podmínky:

$$\begin{aligned} x &\geq 0, & & x \quad R^n, \\ x &= 0, & &\text{pouze pro } x = \theta, \\ x+y &\leq x + y, & & x, y \quad R^n, \\ \alpha x &= |\alpha|\, x, & & \alpha \quad R,\ x \quad R^n. \end{aligned}$$

V tomto textu se používají dvě vektorové normy, konkrétně

$$\begin{aligned} x_{\ 1} &= |x_1| + |x_2| + \cdots + |x_n|, \\ x_{\ 2} &= \sqrt{|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2}, \end{aligned}$$

$x_{\ 1}$ budeme pro jednoduchost označovat i jako x . Norma $x_{\ 2}$ se nazývá Euklidovská a je rovna vzdálenosti x od počátku souřadnicové soustavy θ.

Aby bylo možné s vektorovými prostory pracovat, je důležité je umět popisovat. Zavede se proto několik pojmů, jež to umožní. Prvním z těchto pojmů je *lineární (ne)závislost* souboru resp. množiny vektorů. K tomu je třeba definovat pojem lineární kombinace vektorů. Souborem pak je myšlena konečná uspořádaná $n$tice vektorů.

Definice 1.1.2 *Nechť V je vektorový prostor nad* T*, $x \quad V$ a $(x_1, \ldots, x_n)$ je soubor vektorů z V. Říkáme, že vektor x je lineární kombinací souboru $(x_1, \ldots, x_n)$, právě když existují čísla $\alpha_1, \ldots, \alpha_n \quad T$ taková, že*

$$x = \sum_{i=1}^{n} \alpha_i x_i.$$

Čísla α_i, $i = 1, \ldots, n$, se nazývají koeficienty lineární kombinace. Jestliže $i = 1, \ldots, n$: $\alpha_i = 0$, nazývá se taková lineární kombinace triviální. V opačném případě jde o lineární kombinaci netriviální.

Definice 1.1.3 *Nechť $(x_1, \ldots, x_n)$ je soubor vektorů z V. Řekneme, že soubor $(x_1, \ldots, x_n)$ je lineárně nezávislý (LN) soubor, právě když pouze triviální lineární kombinace tohoto souboru je rovna nulovému vektoru θ. V opačném případě se soubor nazývá lineárně závislý (LZ).*

Definice 1.1.4 *Buď V vektorový prostor nad* T, $= M \quad V$. *Řekneme, že M je lineárně závislá (LZ) množina, právě když existují vektory $x_1, \ldots, x_n \quad M$ takové, že $x_i = x_j$ pro $i = j$, kde $i, j = 1, \ldots, n$, a soubor $(x_1, \ldots, x_n)$ je LZ. V opačném případě je množina M lineárně nezávislá (LN).*

V tuto chvíli je třeba definovat pojem *báze lineárního prostoru*, díky němuž je možno využít poměrně málo vektorů k popisu celého vektorového prostoru. K tomu je ještě nutno definovat pojem lineární obal souboru.

Definice 1.1.5 *Nechť $(x_1, \ldots, x_n)$ je soubor vektorů z V, pak množina všech lineárních kombinací souboru $(x_1, \ldots, x_n)$ se nazývá lineární obal souboru $(x_1, \ldots, x_n)$, značí se $x_1, \ldots, x_n$.*

Buď $= M \quad V$. Množina všech lineárních kombinací všech souborů vektorů z množiny M se nazve lineárním obalem množiny M a značí se M .

Definice 1.1.6 (Báze) *Pokud existuje ve vektorovém prostoru V uspořádaná množina vektorů B taková, že*

1. *B je lineárně nezávislá,*
2. *$B = V$,*

pak se tato uspořádaná množina B nazývá báze vektorového prostoru V.

Jelikož každý vektor z V je lineární kombinací vektorů báze a báze je lineárně nezávislá množina, lze jej proto jednoznačně popsat pomocí koeficientů této kombinace a zavést pojem souřadnic.

Definice 1.1.7 *Nechť $X = (x_1, \ldots, x_n)$ je báze V a vektor $z \quad V_n$ splňuje*

$$z = \sum_{i=1}^{n} \alpha_i x_i.$$

Souřadnicemi vektoru $z \quad V$ v bázi X se nazve uspořádaná ntice (sloupcový vektor)

$$(z)_X := \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} \quad \mathrm{T}^{n,1}$$

Definice 1.1.8 *Buď V vektorový prostor nad* T. *Řekneme, že dimenze vektorového prostoru V je rovna*

- 0, *pokud ve V neexistuje LN soubor délky 1, tedy $V = \{\theta\}$.*
- $n \quad$ N *pokud ve V existuje LN soubor délky n, ale každý soubor délky $n + 1$ už je nutně LZ.*

- ∞, *pokud ve* V *existuje LN soubor libovolné délky.*

Dimenze vektorového prostoru V *se označuje symbolem* $\dim V$.

Dalším neméně důležitým pojme je pak lineární zobrazení. Připomeňme si definici zobrazení, která je zde vystavena na definici kartézského součinu dvou množin.

Definice 1.1.9 (Kartézský součin) *Kartézským součinem množin* A *a* B, *ozn.* $A \times B$, *je množina všech uspořádaných dvojic* (a, b), *kde* $a \quad A$ *a* $b \quad B$.

Definice 1.1.10 (Zobrazení) *Zobrazením z* A *do* B, $f : A \to B$ *se rozumí podmnožina kartézského součinu* $f \quad A \times B$, *kde pro každé* $a \quad A$ *existuje nejvýše jedno* $b \quad B$, *takové, že* $(x, y) \quad f$. *Skutečnost* $(x, y) \quad f$ *se standardně zapisuje* $y = f(x)$

V rámci lineární algebry se pak zkoumají zobrazení lineární.

Definice 1.1.11 *Buďte* P *a* Q *dva vektorové prostory nad stejným tělesem* T, *nechť* $A : P \to Q$. *Zobrazení* A *se nazývá lineární, právě když současně platí:*

1. *(aditivita):* $x, y \quad P : A(x + y) = A(x) + A(y)$,
2. *(homogenita):* $\alpha \quad \mathrm{T}$, $\alpha \quad P : A(\alpha x) = \alpha A(x)$.

Množina všech lineárních zobrazení z P *do* Q *se značí* $L(P, Q)$.

Pro lineární zobrazení se často skutečnost $(x, y) \quad f$ zapisuje jako $y = fx$. Spolu s lineárním zobrazením lze rovnou definovat i pojem lineární operátor.

Definice 1.1.12 (Lineární operátor) *Lineární zobrazení prostoru* V *do* V *se nazývá lineární operátor (transformace) na* V.

Poslední definicí z oblasti zobrazení je definice pojmu jádro zobrazení.

Definice 1.1.13 *Nechť* $A \quad L(P, Q)$. *Jádro zobrazení* A *se definuje jako množina*

$$\ker A := \{x \quad P \mid Ax = \theta_Q\}.$$

1.2 Matice

Následně je třeba definovat i pojmy matice, identická matice a operace s nimi, jelikož jsou to v algoritmech popisovaných v tomto textu velice časté pojmy.

Definice 1.2.1 (Matice) *Nechť $m, n \in \mathbb{N}$. Uspořádaný soubor mn čísel zapsaný do tabulky o m řádcích a n sloupcích se nazývá matice typu $m \times n$. Matice obvykle značíme takto:*

$$\mathsf{A} = \begin{pmatrix} a_{11}, & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ & & \vdots & \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

kde a_{ij} jsou prvky matice (někdy se značí taky jako A_{ij}). Číslu i se říká řádkový a číslu j sloupcový index. Množina všech matic typu $m \times n$ (s reálnými prvky) se značí $\mathbb{R}^{m,n}$. Jako $\mathsf{A}_{:j} \in \mathbb{R}^{m,1}$ se značí $j$tý sloupec matice A:

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

Podobně $\mathsf{A}_{i:} \in \mathbb{R}^{1,n}$ značí $i$tý řádek matice A:

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

Nyní je možné oba pojmy spojit a definovat matici zobrazení.

Definice 1.2.2 *Nechť $\mathsf{A} \in L(P_m, Q_n)$, $X = (x_1, \dots, x_n)$ a $Y = (y_1, \dots, y_n)$ báze P_m, respektive Q_n. Matice ${}^{X}\mathsf{A}^{Y} \in \mathbb{T}^{n,m}$, jež se definuje po sloupcích předpisem*

$$j \in 1, \dots, m : \left({}^{X}\mathsf{A}^{Y}\right)_{:j} := (\mathsf{A}x_j)_Y,$$

se nazve maticí zobrazení A v bázích X, Y (nebo „z báze X do báze Y“). Matici lineárního operátoru ${}^{X}\mathsf{A}^{X}$ zkráceně značíme ${}^{X}\mathsf{A} := {}^{X}\mathsf{A}^{X}$

Zjednodušeně řečeno, A ve svých sloupcích obsahuje souřadnice obrazů $\mathsf{A}x_j$ vektorů z báze X v bázi Y.

Speciálně se pak definuje matice přechodu.

Definice 1.2.3 *Nechť $X = (x_1, \dots, x_n)$ a $Y = (y_1, \dots, y_n)$ jsou báze V_n. Matice identického operátoru E na V_n, tj. ${}^{X}\mathsf{E}^{Y} \in \mathbb{T}^{n,n}$, se nazývá maticí přechodu od báze X k bázi Y.*

Z praktických důvodů se definuje i několik speciálních druhů matic.

Definice 1.2.4 (Jednotková matice) *Matice* $A \in R^{n,n}$*, která má nulové všechny prvky, kromě prvků* a_{ij}*, kde* $i = j$*, jež jsou rovny* 1 *se nazývá jednotková matice nebo identická matice. Značí se* E_n*, případně lze v literatuře najít i označení* I_n*.*

Příkladem jednotkové matice je

$$E_5 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Dále je třeba definovat obecnou diagonální matici

Definice 1.2.5 (Diagonální matice) *Diagonální matice je libovolná čtvercová matice* $D \in T^{n,n}$ *splňující*

$$i, j = 1, \ldots, n : \ i = j \ = \ d_{ij} = 0.$$

Příkladem diagonální matice z $R^{5,5}$ je

$$D_5 = \begin{pmatrix} 1.2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 4.5 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0.2 \end{pmatrix}$$

Dále lze definovat tridiagonální matici.

Definice 1.2.6 (Tridiagonální matice) *Tridiagonální matice je libovolná čtvercová matice* $T \in T^{n,n}$ *splňující*

$$i, j = 1, \ldots, n : \ |i - j| > 1 \ = \ t_{ij} = 0.$$

Příkladem tridiagonální matice z $R^{5,5}$ je

$$T_5 = \begin{pmatrix} 1.2 & 2 & 0 & 0 & 0 \\ 1 & 2 & 8.9 & 0 & 0 \\ 0 & 0 & 4.5 & 5 & 0 \\ 0 & 0 & 4.2 & 6 & 7 \\ 0 & 0 & 0 & 3 & 0.2 \end{pmatrix}$$

Jedná se tedy o matici, která je mimo diagonálu a pás čísel nad ní a pod ní nulová.

Dalším speciálním druhem je Hessenbergova matice.

Definice 1.2.7 (Hessenbergova matice) *Hessenbergova matice je libovolná čtvercová matice* $H \quad T^{n,n}$ *splňující*

$$i, j = 1, \ldots, n : \; i + 1 > j \;\; = \;\; h_{ij} = 0.$$

Znamená to tedy, že prvky hned pod diagonálou mohou ještě být nenulové, prvky, které jsou pod diagonálou a přímo s ní nesousedí, už nulové být musí.

Příkladem Hessenbergovy matice z $R^{5,5}$ je

$$H_5 = \begin{pmatrix} 1.2 & 2 & 6 & 1 & 2 \\ 1 & 2 & 8.9 & 4 & 3 \\ 0 & 0 & 4.5 & 5 & 5 \\ 0 & 0 & 4.2 & 6 & 7 \\ 0 & 0 & 0 & 3 & 0.2 \end{pmatrix}$$

Dle struktury pak lze rozlišovat i horní a dolní trojúhelníkové matice.

Definice 1.2.8 (Horní trojúhelníková matice) *Horní trojúhelníková matice je libovolná čtvercová matice* $U \quad T^{n,n}$ *splňující*

$$i, j = 1, \ldots, n : \; i > j \;\; = \;\; u_{ij} = 0.$$

Příkladem horní trojúhelníkové matice z $R^{5,5}$ je

$$U_5 = \begin{pmatrix} 1.2 & 2 & 6 & 1 & 2 \\ 0 & 2 & 8.9 & 4 & 3 \\ 0 & 0 & 4.5 & 5 & 5 \\ 0 & 0 & 0 & 6 & 7 \\ 0 & 0 & 0 & 0 & 0.2 \end{pmatrix}$$

Definice 1.2.9 (Dolní trojúhelníková matice) *Dolní trojúhelníková matice je libovolná čtvercová matice* $L \quad T^{n,n}$ *splňující*

$$i, j = 1, \ldots, n : \; i < j \;\; = \;\; l_{ij} = 0.$$

Příkladem dolní trojúhelníkové matice z $R^{5,5}$ je

$$L_5 = \begin{pmatrix} 1.2 & 0 & 0 & 0 & 0 \\ 2.1 & 2 & 0 & 0 & 0 \\ 1 & 3 & 4.5 & 0 & 0 \\ 9 & 5.4 & 8 & 6 & 0 \\ 1 & 2 & 8 & 9 & 0.2 \end{pmatrix}$$

Velice důležitou třídou matic jsou matice symetrické.

Definice 1.2.10 (Symetrická matice) *Symetrická matice je taková matice, pro kterou platí,*

$$i, j = 1, \ldots, n : \; a_{i,j} = a_{j,i}.$$

Pro tuto matici tedy platí $A^T = A$. Příkladem symetrické matice z $R^{5,5}$ je

$$S_5 = \begin{pmatrix} 1.2 & 7 & 2 & 0 & 1 \\ 7 & 2 & 0 & 8 & 0 \\ 2 & 0 & 4.5 & 3 & 5 \\ 0 & 8 & 3 & 6 & 0 \\ 1 & 0 & 5 & 0 & 0.2 \end{pmatrix}$$

Poslední speciální maticí, která je použita v algoritmech v tomto textu je pak matice ortogonální, tedy ta, která vynásobená svoji transpozicí je rovna identické matici.

Definice 1.2.11 (Ortogonální matice) *Ortogonální matice je libovolná čtvercová matice* $Q \in R^{n,n}$ *splňující* $Q^T \cdot Q = E$.

Výše uvedené typy matic jsou jasně definovány z hlediska tvaru, struktury či vlastností. O maticích v informatice se často hovoří i z hlediska jejich uložení – klasická hustá matice o rozměrech $m \times n$ se ukládá jako 2D pole, ale například, je-li o některé matici známo, že je diagonální, stačí uložit její diagonálu, tedy z $O(mn)$ prostoru v případě matice $A \in R^{m,n}$ stačí $O(n)$. Podobně lze nakládat i s tridiagonální maticí, kde se uloží pouze diagonála a pás kolem ní. V souvislosti s tímto se pak často hovoří o řídkých maticích, jež nemají přesnou definici, respektive mají, ale je jich vícero. Řídkou maticí se rozumí matice, která má většinu prvků nulových. Jak přesně je většina určena, záleží na implementaci – řídké matice se dají ukládat jako prvky a jeho indexy, kdy je toto uložení ale vhodné, je pak vždy třeba zvažovat.

Zcela zásadní operací, již je nutné definovat, je pak násobení matic.

Definice 1.2.12 (Násobení matic) *Buďte* $m, n, p \in \mathbb{N}$, $A \in R^{m,n}$ *matice s prvky* a_{ij} *a* $B \in R^{n,p}$ *matice s prvky* b_{ij}. *Součinem matic* A *a* B *je matice* $D \in R^{m,p}$ *s prvky* d_{ij}, *pro jejíž prvky platí*

$$d_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

značí se $D = AB$.

Pro násobení matic pak v implementacích lze použít následující algoritmus 1. Naivní implementace pomocí toho definičního vzorečku má pro husté matice složitost n^3.

Lze jednoduše vidět, že algoritmus 1 má složitost $O(n^3)$, což se podařilo optimalizovat až na $O(n^{2.3728596})$ [2].

Dalším pojmem, který se přímo využívá k výpočtu vlastních čísel je pak pojem determinantu matice.

Algoritmus 1: Násobení matic

```
Input: A ∈ R^{m,n}, B ∈ R^{n,p}
Output: C ∈ R^{m,p}
for i = 1:m do
    for j = 1:p do
        C[i,j] = 0;
        for k = 1:n do
            C[i,j] += A[i,k]B[k,j];
        end
    end
end
```

Definice 1.2.13 *Buď* A *čtvercová matice z* $\mathrm{T}^{n,n}$. *Determinant matice* A *je číslo definované vztahem*

$$\det A = \sum_{\pi \in S_n} \operatorname{sgn}(\pi) \prod_{i=1}^{n} a_{i,\pi(i)}.$$

V sumě se sčítá přes *všechny permutace množiny* $\{1, \ldots, n\}$, počet sčítanců je proto roven $n!$. Činitelé součinu jsou pak prvky matice v i-tém řádku a $\pi(i)$-tém sloupci. $\operatorname{sgn}(\pi)$ označuje znaménko permutace π.

1.3 Vlastní čísla a jejich vlastnosti

Za pomoci těchto definic je možno definovat i pojem vlastní číslo a vlastní vektor.

Definice 1.3.1 (Vlastní číslo a vlastní vektor) *Nechť* $A : V \to V$ *je lineárním operátorem ve* V *nad* C. *Číslo* $\lambda \in \mathrm{C}$ *se nazývá vlastním číslem operátoru* A, *pokud existuje vektor* $x \in V$, $x = \theta$ *takový, že* $Ax = \lambda x$. *Vektor* x, *který splňuje uvedenou rovnost, se nazývá vlastní vektor operátoru* A *příslušný vlastnímu číslu* λ. *Množina všech vlastních čísel* A *se nazývá spektrem operátoru* A, *značí se symbolem* $\sigma(A)$.

Vlastní čísla operátorů na konečně-dimenzionálním prostoru lze najít jako kořeny tzv. *charakteristického polynomu*.

Definice 1.3.2 *Nechť* $A \in L(V_n)$. *Pak polynom*

$$p_A(\lambda) := \det(A - \lambda E), \quad \lambda \in \mathrm{C},$$

se nazve charakteristickým polynomem operátoru A.

Polynom p_A je polynomem stupně n a nezávisí na volbě báze X. Násobnost čísla λ jako kořene charakteristického polynomu se nazývá algebraická násobnost, značí se $\nu_a(\lambda)$.

Vlastní čísla mají dále násobnost geometrickou, $\nu_g(\lambda)$, která označuje dimenzi podprostoru vlastních vektorů příslušných danému vlastnímu číslu.

Definice 1.3.3 *Nechť* $\lambda \in \mathbb{C}$ *je vlastní číslo operátoru* $A \in L(V_n)$. *Číslo* $d(A - \lambda E) = \dim \ker(A - \lambda E)$ *se nazve geometrická násobnost vlastního čísla* λ *a značí se* $\nu_g(\lambda)$.

Vlastní čísla je možno definovat i pro matice.

Definice 1.3.4 *Komplexní číslo* λ *se nazve vlastním číslem matice* $\mathbf{A} \in \mathbb{C}^{n,n}$, *právě když existuje nenulový vektor* $x \in \mathbb{C}^n$ *splňující*

$$\mathbf{A} \cdot x = \lambda x,$$

Takovýto vektor x *se pak nazývá vlastní vektor matice* $\mathbf{A}$ *příslušející vlastnímu číslu* λ. *Spektrem matice* $\mathbf{A}$ *(ozn.* $\sigma(\mathbf{A})$*) je množina všech vlastních čísel matice* $\mathbf{A}$. *Charakteristický polynom matice* $\mathbf{A}$ *(ozn.* $p_{\mathbf{A}}$*) se definuje předpisem*

$$p_{\mathbf{A}}(\lambda) := \det(A - \lambda E).$$

Jak bylo zmíněno, vlastní čísla operátorů i matic lze hledat jako kořeny charakteristického polynomu. Ten však je stupně n, a proto je tento úkol poměrně nesnadný, stejně jako samotné určení tohoto polynomu. Ve výjimečných případech je možné kořeny uhodnout, což není implementovatelné a fakticky to není řešení tohoto problému, navíc tímto způsobem je řešitelná poměrně malá množina matic a proto se v praxi využívá jiných metod. Mnoho z nich využívá podobnosti matic a souvislosti se spektrem.

Definice 1.3.5 *Matice* $\mathbf{A}, \mathbf{B} \in \mathbb{C}^{n,n}$ *se nazývají podobné, právě když existuje regulární matice* $\mathbf{P} \in \mathbb{C}^{n,n}$ *taková, že platí*

$$\mathbf{A} = \mathbf{P}^{-1} \cdot \mathbf{B} \cdot \mathbf{P}.$$

Pokud jsou dvě matice podobné, mají tyto matice shodné charakteristické polynomy a spektra obou matic jsou stejná.

Pro některé operátory je možné najít bázi, v níž je matice operátoru diagonální. Těmto operátorům a jejich maticím se říká diagonalizovatelné.

Definice 1.3.6 *Operátor* $A \in L(V_n)$ *se nazývá diagonalizovatelný, jestliže existuje báze* X *prostoru* V_n *taková, že matice* ${}^{X}A$ *je diagonální. Matice* $A \in \mathbb{C}^{n,n}$ *je diagonalizovatelná, jestliže je podobná diagonální matici.*

Operátor je diagonalizovatelný, právě když každé jeho vlastní číslo má stejnou algebraickou a geometrickou násobnost. Pak lze sestavit bázi z vlastních vektorů matice a matici A tohoto operátoru je možné zapsat následovně:

$$\mathsf{A} = \mathsf{P}^{-1} \cdot \mathsf{B} \cdot \mathsf{P},$$

kde P bude matice přechodu ze standardní báze do báze tvořené vlastními vektory.

Matice A pak vzhledem k bázi

$$X = (x_{1,1}, \ldots, x_{1,\ _1}, x_{2,1}, \ldots, x_{2,\ _2}, \ldots, x_{k,1}, \ldots, x_{k,\ _k}),$$

kde $_i$ je násobnost vlastního čísla λ_1, $x_{i,j}$ je j-tý vlastní vektor $i$tého vlastního čísla, bude mít tvar

$$^{X}\mathsf{A} = \begin{pmatrix} \lambda_1\mathsf{E}_{\ 1} & \Theta & \Theta & \cdots & \Theta \\ \Theta & \lambda_2\mathsf{E}_{\ 2} & \Theta & \cdots & \Theta \\ \Theta & \Theta & \lambda_3\mathsf{E}_{\ 3} & \cdots & \Theta \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \Theta & \Theta & \Theta & \cdots & \lambda_k\mathsf{E}_{\ k} \end{pmatrix}$$

Matice B pak bude diagonální a na diagonále bude mít vlastní čísla matice A. Matice přechodu ze standardní báze do báze X pak bude mít ve sloupcích zapsány vlastní vektory matice příslušné k vlastním číslům v odpovídajících sloupcích matice B.

Dále pak užitečnou vlastností je i to, že vlastní čísla horní ale i dolní trojúhelníkové matice se nachází na diagonále, neboť charakteristický polynom takovýchto matic B má tvar:

$$p_{\mathbf{B}}(\lambda) = (a_{1,1} - \lambda)(a_{2,2} - \lambda)\ldots(a_{n,n} - \lambda),$$

zbytek členů charakteristického polynomu je díky nulám pod, respektive nad, diagonálou roven nule.

Právě těchto vlastností lze využít při hledání vlastních čísel. Je možno najít matici B, podobnou zadané matici a tím pádem i matici přechodu – vlastní vektory zadané matice? A jak? To je otázka, která trápila matematiky po několik staletí.

Kapitola **2**

Historie problematiky vlastních čísel a vektorů

2.1 Nejstarší historie

První zmínky o maticích lze najít už v dílech ze starověké Číny [3]. Dochoval se nám například text s názvem Devět kapitol matematického umění, který byl pravděpodobně sepsán někdy kolem roku 200 př. n. l [4].

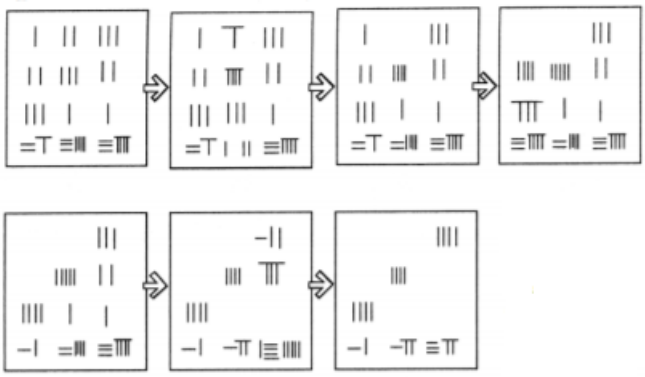
Tento text patří mezi nejstarší matematické texty na světě. Dle [5] se v knize řeší soustavy až šesti rovnic o až šesti neznámých. Pro řešení těchto soustav vyžívá výpočetní tabulku s názvem fāng chéng[1], čínsky 方程, kterou bychom dnes nazývali maticí. Rovnice se zapsaly do sloupců, a pak byla soustava redukována do horního stupňovitého tvaru pomocí metody, které se dnes říká Gausova eliminace. Na obrázku je pak ilustrace, jak taková výpočetní tabulka vypadala, obrázek je převzat z [4].

Podle [3] lze zmínky o soustavách lineárních rovnic z podobné doby, i když zdaleka ne tak detailní, najít i v pramenech ze starověkého Babylonu.

Problematika diagonalizace matic a další s tím spojené fenomény se pak v záznamech objevují až v sedmnáctém století. Nizozemský matematik Jan de Witt ve své publikaci *Elementa curvarum linearum*[2], která byla publikována jako komentář k Descartově dílu *La Géométrie*[3], ukázal, že lze diagonalizovat symetrickou matici, bohužel v té době se ještě pojem matice nepoužíval a tak toto zůstalo bez dalšího zkoumání, jak se lze dočíst v [3].

Další rozvoj problematiky matic lze najít v Japonsku. Matematik Takakazu Seki (v některých pramenech taky Kowa Seki) ve svém díle *Kaifukudai no Hō (*解伏之法*)*, [4] z roku 1683 objevuje mimo Bernoulliho čísel a diskri-

[1] Výpočetní tabulka
[2] Elementy křivek
[3] Geometrie
[4] Způsoby řešení disimulovaných problémů



Obrázek 2.1: Fāng chéng

minantu pojem determinant, dokonce pak představuje i obecnou formuli pro jeho výpočet, známou jako Laplaceův rozvoj determinantu [3].

O deset let později, v roce 1693 lze pak nalézt první zmínky o determinantu a jeho výpočtu z evropské scény. V roce 1693 píše Leibniz l'Hospitalovi dopis, ve kterém vysvětluje, že systém rovnic

$$10 + 11x + 12y = 0,$$
$$20 + 21x + 22y = 0,$$
$$30 + 31x + 32y = 0,$$

má netriviální řešení, protože

$$10 \cdot 21 \cdot 32 + 11 \cdot 22 \cdot 30 + 12 \cdot 20 \cdot 31 = 10 \cdot 22 \cdot 31 + 11 \cdot 20 \cdot 32 + 12 \cdot 21 \cdot 30$$

což odpovídá formuli pro determinant [3].

Další formule pro determinant pak zkoumali kromě Leibnize také Gabriel Cramer, Colin Maclaurin, Étienne Bézout a Pierre-Simon Laplace. V pramenech z té doby, tedy osmnáctého století, se vyskytuje pro determinant výraz *resultant*. Pojem *determinant* zavádí v roce 1801 Johann Carl Friedrich Gauss v díle *Disquisitiones arithmeticae*, nicméně ne ve stejném kontextu, jako se determinant používá dnes. Prvním, kdo užil pojmu determinant v jeho dnešním významu, byl Augustin Louis Cauchy v roce 1812, který také dokázal některá významná tvrzení týkající se determinantu. [3]

2.2 Vývoj pojmenování

Byl to také Cauchy, kdo jako první užil pojem vlastní čísla a to v roce 1829 v díle *Sur l'équation à l'aide de laquelle on détermine les inégalités séculaires des mouvements des planétes*[5]. Tento text, který byl součástí jeho *Exercises*

[5] O rovnici, která pomáhá určit sekulární nerovnosti v pohybu planet

$$(43)\qquad \begin{cases} (A_{xx}-s)x + A_{xy}y + A_{xz}z = 0, \\ A_{xy}x + (A_{yy}-s)y + A_{yz}z = 0, \\ A_{xz}x + A_{yz}y + (A_{zz}-s)z = 0; \end{cases}$$

Obrázek 2.2: Cauchy: *Sur l'équation à l'aide de laquelle on détermine les inégalités séculaires des mouvements des planétes* [6]

et

$$(53)\qquad \begin{cases} (A_{xx}-s)(A_{yy}-s)(A_{zz}-s) \\ \quad -A_{yz}^2(A_{xx}-s) - A_{xz}^2(A_{yy}-s) - A_{xy}^2(A_{zz}-s) + 2A_{xy}A_{xz}A_{yz}. \end{cases}$$

Obrázek 2.3: Cauchy: *Sur l'équation à l'aide de laquelle on détermine les inégalités séculaires des mouvements des planétes* [6]

de mathématiques[6] bychom dnes popsali jako důkaz, že symetrická matice má reálná vlastní čísla.

Jak je možné vidět na obrázcích 2.2 a 2.3, Cauchy popisuje v textu hledání vlastních čísel pomocí charakteristického polynomu. Cauchy nazýval vlastní čísla jako *valeurs propres*[7], což je dodnes používaný termín ve francouzštině.

Zhruba dvanáct let po Cauchym přišel James Joseph Sylvester s článkem ve *Philosophical Magazine*[8] [7] o využití matic pro výpočet kořenů kvadratických polynomů. V článku nepoužil žádné terminologie pro tyto pojmy, o to se pokusil až o dvacet let později, kde prosazoval slovo *latent* jako vlastní číslo. [8]

Nebyl jediný, kdo se snažil pojmenovat vlastní čísla a vektory. Další, kdo se o vlastní čísla a funkce zajímal, byl Henri Poincaré. Ten roku 1894 [8] napsal spis *Sur les Équations de la Physique Mathématique*, kde pro tyto pojmy zvolil název *fonction harmonique* a *nombre caractéristique* [9]. Ještě i dnes se používá v některých oblastech *characteristic values (of a matrix)*.

Dalším matematikem, který se snažil vlastní čísla pojmenovat byl Émil Picard. Ten používal pojmu singulární body. Název singulární hodnoty se používá i dnes, nicméně se používá pro zobecnění vlastních čísel na i nečtvercové matice. Tato partie má také velké aplikace (viz *Singular Value Decomposition*).

S germanistickým názvem *eigenvalues*, vlastní hodnoty, tedy termínem, který se v anglicky mluvících zemích používá dodnes, přišel David Hilbert a to

[6] Cvičení z matematiky
[7] správné hodnoty
[8] Filosofický magazín

konkrétně ve svém díle z roku 1904 *Grundzüge einer allgemeinen Theorie der linearen Integralgleichungen*[9] [8].

2.3 Metody hledání a nejnovější vývoj

Při odhlédnutí od historie pojmenování, nelze opomenout práci Carla Gustava Jacoba Jacobiho. Ten v roce 1846 přišel v článku *Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen*[10] [10] se svou iterační metodou, která byla po něm pojmenována. Její nevýhodou je především její pomalost.

V roce 1929 pak přišli R. von Mises a H. Pollaczek-Geiringer s takzvanou mocninnou metodou, což je poměrně jednoduchý algoritmus, který však může konvergovat poměrně pomalu a hledá pouze jedno tzv. dominantní vlastní číslo. Tato metoda je nicméně poměrně využívaná a to zejména v případech, kdy není potřeba více vlastních čísel anebo když je matice řídká [11]. Její podrobnější analýza následuje v kapitole 4.

Jistou obměnou mocninné metody je Rayleighova poměrná iterace, *Rayleigh quotient iteration*, jejíž kořeny lze vystopovat do devatenáctého století k Johnu Williamu Srtuttovi, třetímu baronu Rayleigh [12]. Praktické využití této metody přinesly až počítače a práce Alexandra Markoviče Ostrovského z let 1958 až 1959 [11].

Na konci padesátých let, konkrétně v roce 1958, pak přišel Heinz Rutishauser s myšlenkou LR algoritmu. Tento algoritmus využíval rozkladu matice A na součin matic L a R, kde L je dolní trojúhelníková matice s jedničkami na diagonále a R je horní trojúhelníková matice. Rutishauser si všiml, že pokud máme rozklad $A = LR$, pak matice RL je podobná matici A, protože

$$L^{-1}AL = L^{-1}LRL = ERL = RL.$$

Rutishauser navrhnul rozložit součin RL znovu a znovu, pak R konverguje k horní trojúhelníkové matici a L k matici jednotkové [11].

Na základě tohoto poznatku pak stojí QR algoritmus, se kterým přišel mezi roky 1961 a 1962 John G. F. Francis. Tento algoritmus je dodnes nejpoužívanějším algoritmem k výpočtu vlastních čísel a vektorů. Dnes se používá s jistými variacemi z devadesátých let, jako jsou implicitní posuny [11]. Také analýzu tohoto algoritmu lze nalézt v kapitole 4.

Kromě iterativních algoritmů se v průběhu dvacátého století objevily i algoritmy redukční. Jedním z nich je Lánczosova metoda, kterou v roce 1950 představil maďarský matematik Kornél Lánczos. Metoda se používá pro výpočet vlastních čísel symetrických matic. Princip této metody je vysvětlen v kapitole 4. Tato metoda byla v padesátých letech považována za nestabilní a nespolehlivou [11].

[9] Základy všeobecné teorie lineárních integrálních rovnic

[10] O snadném postupu k řešení rovnic, vyskytujících se v teorii sekulárních poruch

O rok později, v roce 1951, představil Walter Edwin Arnoldi svůj přístup k metodě, kterou navrhl Lánczos. Arnoldiho metodu, narozdíl od Lánczosovy, lze aplikovat i na nesymetrické matice, jedná se pouze o drobnou modifikaci [13].

V roce 1975 navrhl chemik Ernest R. Davidson iterativní metodu, která fungovala podobně jako Arnoldiho iterace, nicméně promítala na jiný podprostor. Tato metoda však vyžadovala poměrně dobrou volbu počátečního vektoru, aby dávala uspokojivé výsledky [11].

Na konci dvacátého století pak Gerard L. G. Sleijpen a Henk A. Van der Vorst představili Jacobi-Davidsonovu metodu, pro iterativní výpočet několika v absolutní hodnotě největších vlastních čísel a jejich příslušných vlastních vektorů. V práci kombinují Jacobiovu metodu s prací E. R. Davidsona a díky tomu představují metodu, která je efektivní pro velké řídké matice, pro něž je přístup s posuny a inverzemi příliš drahý. Volba počátečních podmínek pro další třídy matic pak zůstává otevřeným problémem této metody [11, 14].

Mezi nejnovější algoritmy se pak řadí Cuppenův rekurzivní přístup a metodu Rozděl a panuj, kterou představil v roce 1981. Metoda je aplikovatelná na symetrické tridiagonální matice a na těchto je pak řádově rychlejší než QR algoritmus, který ale pro bloky menšího řádu než 25 dnes využívá [15, 11].

Kapitola 3

Aplikace vlastních čísel a vektorů

Vlastní čísla a vlastní vektory nacházejí celou řadu uplatnění. V této kapitole se rozebírají především aplikace v oboru informatiky, a to Markovského řetězce, Page Rank, jež je základním prvkem moderních vyhledávačů, analýza hlavních komponent a hledání kořenů polynomů. Další aplikace jsou pak zmíněny v poslední podkapitole.

3.1 Markovské řetězce

Markovské (nebo Markovovy) řetězce jsou pojmenovány po profesoru Andreji Andrejeviči Markovovi, ruském matematikovi, který žil a působil v druhé polovině devatenáctého století a prvních dvou dekádách století dvacátého. Působil především na univerzitě v Petrohradu a byl členem Ruské Akademie věd. Jeho největším objevem jsou bezpochyby právě Markovské řetězce, model posloupnosti stavů nějakého systému takových, že stav v určitém čase závisí pouze na stavu v čase předchozím. [16] Markovský řetězec si lze představit jako konečný automat, kde je pro přechod mezi stavy definována pravděpodobnost.

Tyto pravděpodobnosti lze zapsat do matice, kde pak indexy sloupců a řádků představují jednotlivé stavy a prvky představují pravděpodobnosti přechodu ze stavu i do stavu j, kde i, resp. j, je index řádku, resp. sloupce. Takové matici se říká matice pravděpodobnostního přechodu. Tato matice bude mít řádkový součet – tj. součet hodnot v řádku – roven jedné. Matici, jež toto splňuje, se říká *(řádkově) stochastická*. Počáteční stav nám určuje řádkový vektor v, jeho přesná hodnota závisí na konkrétním řetězci. Stav po n opakováních experimentu lze určit jako $v \cdot P^n$ [17].

Markovské řetězce lze využít v mnoha vědách: sociologii, biologii a dalších. Ukázka využití z „matematické biologie“ je například monitorování pohybu zvěře: Je známo, že x lišek žije v oblasti I a y lišek v oblasti J, s tím, že

liška přejde z oblasti I do oblasti J s pravděpodobností 30 %, ale z oblasti J do oblasti I pouze s 20% pravděpodobností. Existuje nějaký stav, který bude rovnovážný? Kde bude pak kolik lišek? Obdobný případ jde najít i pro pohyb obyvatelstva – tedy například pro stěhování lidí mezi městem a vesnicí.

Pro rovnovážný stav bude platit, že k němu po dostatečném množství kroků začne stav konvergovat, nechť se tak stane po n krocích, tedy

$$V \approx v \cdot \mathrm{P}^n$$

Po každém dalším kroku bude řešení vypadat následovně

$$v \cdot \mathrm{P}^n \cdot \mathrm{P} \approx V \cdot \mathrm{P}$$

$$v \cdot \mathrm{P}^{n+1} \approx V \cdot \mathrm{P}$$

a protože je rovnovážný, pak také pro $n + 1$, $v \cdot \mathrm{P}^{n+1} \approx V$, tedy $v \cdot \mathrm{P}^{n+1} \approx V \cdot \mathrm{P}$ a $v \cdot \mathrm{P}^{n+1} \approx V$, z čehož vyplývá

$$V\mathrm{P} = V$$

Pokud je V nenulové, je levým vlastním vektorem matice P odpovídající vlastnímu číslu 1. Toto řešení se nazývá ekvilibrium, rovnovážné řešení nebo stacionární řešení.

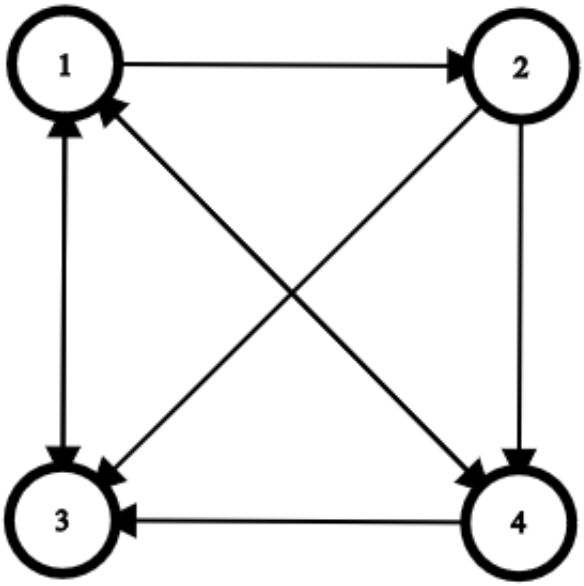
3.2 Google Page Rank

Jednou z nejvýznamnějších aplikací vlastních čísel z oblasti informačních technologií je vyhledávací engine od společnosti Google. Princip, který představili Sergey Brin a Lawrence Page v roce 1998 ve svém článku *The Anatomy of a Large-Scale Hypertextual Web Search Engine* [18][11] stojí na výpočtu jistého vlastního čísla označovaného jako Page Rank. Tento algoritmus byl poměrně revoluční, protože nejen že už na přelomu tisíciletí stačilo pro dvacet šest milionů stránek několik hodin na průměrné pracovní stanici na výpočet [18], ale také dával velice přesné výsledky.

Cílem vyhledávače je nabídnout uživateli nejrelevantnější výsledky vyhledávání jako první. Otázkou ovšem je, jak relevantní stránky poznat. Metoda Page Ranku stojí na předpokladu, že na důležité a relevantní stránky vede více odkazů než na ty méně relevantní. Pokud by jako skóre stránky byl použit čistě počet odkazů vedoucích na stránku, dává to velkou váhu „hlasu“ stránek, ze kterých vede hodně odkazů na stránky jiné. Nejjednodušším řešením tohoto problému je normalizovat „hlas“ každé stránky, tedy vydělit příspěvek každé stránky počtem odkazů z ní odcházejících.

Princip systému vysvětluje [19]: Nechť web obsahuje n stránek $A_1 \ldots A_n$. Z každé z těchto stránek vede n_j odkazů, množinu indexů těchto stránek,

[11] Anatomie rozsáhlého hypertextového webového vyhledávače



Obrázek 3.1: Síť o čtyřech stránkách

odkazujících se na tuto stránku označíme L_j. Pokud stránka A_j odkazuje na stránku A_k, přispívá k jejímu skóre $\frac{x_j}{n_j}$, kde x_j je skóre stránky A_j. Tedy výsledkem je n rovnic ve tvaru:

$$x_k = \sum_{j \ L_k} \frac{x_j}{n_j},$$

Odkazy sama na sebe se ignorují.

Pokud se tyto rovnice zanesou do matice H, hledaná hodnocení stránek budou prvky vlastního vektoru příslušejícího k vlastnímu číslu 1. Matice H má tedy prvky definovány takto:

$$h_{i,j} = \begin{cases} \frac{1}{n_j} & \text{pokud stránka } A_j \text{ odkazuje na stránku } A_i \\ 0 & \text{jinak} \end{cases}$$

Jako příklad lze použít síť z obrázku 3.1. Má čtyři stránky, šipka znázorňuje odkaz. Matice H bude vypadat tedy takto:

$$\mathsf{H} = \begin{bmatrix} 0 & 0 & 1 & \frac{1}{2} \\ \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{3} & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

Vlastním číslem této matice je (mimo jiné) číslo $\lambda = 1$ a k němu příslušející vlastní vektor je například $v = (12, 4, 9, 6)$. Toto skóre lze normalizovat, vydělit 31 tak, aby součet prvků vektoru byl roven 1. Tedy výsledkem je pak vektor

$$v = \begin{bmatrix} \frac{12}{31} & \frac{4}{31} & \frac{9}{31} & \frac{6}{31} \end{bmatrix} \approx \begin{bmatrix} 0.387 & 0.129 & 0.290 & 0.194 \end{bmatrix}.$$

Lze si povšimnout, že oproti obyčejnému počtu odkazů, podle kterého by byla nejrelevantnější stránkou stránka č. 3, je dle této metodiky nejvýznamnější stránkou stránka č. 1, a to především díky odkazu ze stránky č. 3.

Matice má 1 jako vlastní číslo, pokud síť neobsahuje tzv. visící stránky (anglicky *dangling nodes*), tedy stránky bez odchozích odkazů a pokud má matice H hodnost rovnou n [19].

První problém vzniká, když je vstupem nesouvislá síť, která postrádá rámec, v němž by bylo možné stránky porovnat. Jak rozhodnout, která stránka je relevantnější, když spolu nesouvisejí? Druhý problém vzniká u obrázků a dalších stránek, které nemají odchozí odkazy, jelikož pak 1 nebude vlastním číslem matice H.

Druhý problém má poměrně jednoduché řešení, nulový sloupec v matici se nahradí vektorem, jehož složky budou rovny $1/n$, kde n je počet stránek [20], tedy se sečte matice H s maticí H, kde $i$tý sloupec je roven $\frac{1}{n}$, pokud je $i$tá stránka visící.

$$\mathsf{S} = \mathsf{H} + \mathsf{A}$$

Matici S lze interpretovat jako matice Markovského řetězce, který představuje osobu surfující po internetu, vstupy matice jsou pak pravděpodobnosti, že klikne na nějaký odkaz [21]. Nahrazení nul $\frac{1}{n}$ pak je možno interpretovat jako náhodně zvolená další stránka.

První problém lze vyřešit následující modifikací: přidáním parametru α, který představuje pravděpodobnost, že si uživatel zvolí následující stránku dle matice S, $(1 - \alpha)$ pak představuje pravděpodobnost, že se uživatel dostane náhodně jinam. Matici pak lze upravit následovně:

$$\mathsf{G} = \alpha S + (1 - \alpha)\frac{1}{n}$$

Tato matice se nazývá Google matice [21]. Parametr alfa je pak, dle článku [18], pro vyhledávač Google zvolen jako 0.85.

K výpočtu vlastního vektoru je pak použita mocninná metoda [20], která bude popsána v kapitole 4. Dnes je Page Rank pouze částí algoritmu pro doporučování stránek, nicméně stále poměrně významnou [22]. Systém Page Ranku je využíván i v dalších podobných systémech, například je částí doporučování koho sledovat na sociální síti Twitter [23].

3.3 PCA – Analýza hlavních komponent

Analýza hlavních komponent je metoda používaná ve zpracování dat k redukci dimenzionality, tedy zmenšení počtu sloupců v datasetu. Cílem metody je extrahovat co nejvíce informací z dat, komprimovat velikost datasetu za minimální ztráty informace, zjednodušit strukturu dat a analyzovat strukturu proměnných [24]. Analýza hlavních komponent se využívá i při rozpoznávání obličejů [25].

K výpočtu se užívá výběrové varianční matice, která má jako ijtou složku výběrovou kovarianci $i$tého a $j$tého příznaku, statistickou mírou lineární závislosti těchto příznaků. Tato matice je nutně symetrická a na diagonále má výběrové rozptyly. Ze symetričnosti matice plyne nutně její diagonalizovatelnost. Jelikož kovariance i rozptyly jsou nutně kladné, je matice i pozitivně semi-definitní a tím pádem jsou její vlastní čísla kladná.

Metoda stojí na hypotéze, že příznak s odpovídajícím největším vlastním číslem nese nejvíce informací a tedy je ideální jej zachovat [26].

Transformace datasetu se provede přenásobením středovaného datasetu transformační maticí V, jež má q sloupců a tyto sloupce odpovídají vlastním vektorům kovarianční matice.

Algebraicky řečeno, hlavní komponenty jsou q konkrétních lineárních kombinací původních proměnných, geometricky tyto kombinace představují nový systém souřadnic, který vznikl rotací toho původního. Dalo by se říci, že se volí jiná báze prostoru. Nový systém reprezentuje „směr“ největšího rozptylu dat a poskytuje jednodušší a šetrnější popis kovarianční struktury [27].

3.4 Hledání kořenů polynomů

V první kapitole byl popsán charakteristický polynom a jak se pomocí něj dají hledat vlastní čísla. Dají se ale i naopak počítat komplexní kořeny komplexního či reálného polynomu pomocí hledání vlastních čísel. Nechť je

$$p(\lambda) = \sum_{k=0}^{n} a_k \lambda^k$$

polynom stupně n jehož kořeny je třeba najít. Bez újmy na obecnosti lze předpokládat, že $a_n = 1$.

Skutečně, z předpokladu o stupni n nutně plyne $a_n = 0$, a pak lze místo $p(\lambda) = 0$ řešit $\frac{1}{a_n}p(\lambda) = 0$. Nyní je možno sestrojit matici

$$\mathrm{A}_p = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -a_{n-2} \\ 0 & 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix}$$

Charakteristický polynom této matice je pak polynom p, a tím pádem vlastní čísla matice A jsou právě kořeny polynomu p a lze je tedy napočítat pomocí známých algoritmů. Matici A se říká *matice společnice*, anglicky *companion matrix*.

3.5 Další aplikace

Vlastní čísla se dají využít například i v medicíně. Příkladem může být difuzní tenzorové zobrazování (DTI) [28], což je technika magnetické rezonance (MRI) používaná především ve zdravotnictví k zobrazení vnitřních orgánů lidského těla. Umožňuje diagnostikovat roztroušenou sklerózu, Alzheimerovu chorobu, fokální kortikální dysplasii, používá se při předoperačních vyšetřeních před neurologickými operacemi a pro brzké zjištění patologických změn v mozku [29]. Vlastní čísla a vektory se zde používají pro určení hlavních směrů difuze a příslušných difuzivit [30].

V rámci medicíny, respektive epidemiologie, lze dle [31] využít i Lotka–Volterrovy rovnice, jejichž stabilita závisí právě na vlastních číslech [32]. Tyto rovnice, známé jako model predátor–kořist, či lovec–kořist, z anglického *predator–prey*, popisují například populace predátora a jeho kořisti v závislosti na čase. Epidemiologickou interpretací tohoto fenoménu je pak Kermack–McKendrickův model, který byl představen v roce 1927 v článku *A contribution to the mathematical theory of epidemics*[12] [33]. Jedná se o model diferenciálních rovnic, popisující vývoj epidemiologické situace v čase. Další možnou interpretací jsou některé chemické reakce, aplikaci však můžeme nalézt i v ekonomice [32].

Analýza vlastních čísel najde využití i ve stavebnictví, kde se pomocí ní počítají statické zátěže a pomáhají určit limity stavby. Příkladem může být i zkoumání toho, co se stalo s mostem Tacoma Narrows [34]. Most Tacoma Narrows se nacházel ve státě Washington v USA. Kvůli nekvalitnímu návrhu a snaze ušetřit na jeho stavbě se zřítil ani ne půl roku po svém uvedení do provozu v roce 1940. Z celého incidentu, při kterém díky včasném uzavření mostu

[12] Příspěvek k matematické teorii epidemie

nikdo nezemřel, se dochoval i videozáznam [35]. Vysvětlení tohoto incidentu přináší právě analýza vlastních čísel a jejich vliv na statiku stavby [34].

Využití naleznou vlastní čísla i v chemii a fyzice, respektive kvantové mechanice, vždyť výraz spektrum, používaný pro množinu vlastních čísel má původ právě v těchto oborech. Hledání vlastních čísel a vektorů najde své uplatnění třeba při řešení diferenciálních rovnic, například pomocí metody konečné diskretizace. Tato metoda spočívá v nahrazení derivací diferencemi a tím pádem diskretizaci spojitého analytického problému. Ten se pak dá řešit pomocí numerických metod. Využití diferenciálních rovnic je pak celá řada, od akustiky, přes výpočty obvodů až po využití v kvantových počítačích či při zkoumání tepelné vodivosti.

Kapitola 4

Analýza vybraných algoritmů

V této kapitole se rozebírají tři algoritmy pro výpočet vlastních čísel a vektorů matice a to: mocninná metoda, Lanczosova metoda a QR algoritmus, jejich výhody, nevýhody, složitost a použití. Popisy algoritmů a jejich vlastností vycházejí z knihy [36].

4.1 Mocninná metoda

Mocninná metoda je metodou k nalezení *dominantního* neboli v absolutní hodnotě největšího vlastního čísla a k němu příslušného vlastního vektoru. Metoda spočívá v násobení matice vektorem, proto je vhodná především pro matice řídké, takže s většinou nulových prvků. Předpokladem k použití této metody je, že naše $n \times n$ matice má nějaké dominantní vlastní číslo λ_1 a že má n lineárně nezávislých vlastních vektorů, tedy je diagonalizovatelná.

Nechť je na vstupu matice $A \quad C^{n,n}$ a vektor x. Z předpokladu o lineární nezávislosti vlastních vektorů je zřejmé, že x lze zapsat následovně:

$$x = \sum_{i=1}^{n} \alpha_i v_i$$

kde $(v_1, \ldots, v_n)$ jsou vlastní vektory matice A příslušné vlastním číslům $(\lambda_1, \ldots, \lambda_n)$.

Pokud se toto vyjádření bude postupně násobit maticí A zleva, dojde se k následujícím rovnostem:

$$Ax = \sum_{i=1}^{n} \alpha_i A v_i = \sum_{i=1}^{n} \alpha_i \lambda_i v_i,$$
$$A^2 x = \sum_{i=1}^{n} \alpha_i A^2 v_i = \sum_{i=1}^{n} \alpha_i \lambda_i^2 v_i.$$

Po k krocích dojdeme k:

$$A^k x = \sum_{i=1}^{n} \alpha_i A^k v_i = \sum_{i=1}^{n} \alpha_i \lambda_i^k v_i.$$

Vytkne-li se nyní dominantní vlastní číslo λ_1, dostaneme

$$A^k x = \lambda_1^k \sum_{i=1}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^k v_i.$$

Jelikož platí, že $|\lambda_1| > |\lambda_i|$ pro $j = 2 \ldots n$, je zřejmé, že

$$\lim_{k \to \infty} \left(\frac{\lambda_j}{\lambda_1}\right)^k = 0$$

a proto lze říci, že

$$A^k x \approx \lambda_1^k \alpha_1 v_1.$$

Postup konverguje k nule, jestliže $|\lambda_1| < 1$ a diverguje, pokud $|\lambda_1| \geq 1$ za předpokladu, že $\alpha_1 = 0$.

Tento postup je zároveň důkaz následující věty:

Věta 4.1.1 *Má-li matice* A *n lineárně nezávislých vektorů a je-li vlastní číslo* λ_1 *dominantní a pro vektor* $x_0 \in C^n$ *platí, že* x_0 *není kolmý na* v_1. *Pak*

$$\lim_{k \to \infty} \frac{A^k x_0}{\lambda_1^k} = \alpha_1 v_1.$$

Z tohoto pak plyne, že je-li y libovolný vektor, jež není kolmý na v_1, platí

$$\lambda_1 = \lim_{k \to \infty} \frac{y^T x_{k+1}}{y^T x_k},$$

kde $x_{k+1} = A x_k = A^k x_0$.

Nevýhodou této metody je především to, že v praxi nejsme schopni ověřit vstupní podmínky, tedy existenci dominantního vlastního čísla a neortogonalitu vektoru x_0 na příslušný vlastní vektor, takže není zajištěno, ani že metoda bude konvergovat, ani že vypočte dominantní vlastní číslo. Dalším problémem je problematický odhad chyby. Metoda je zapsána jako algoritmus 2.

Jedna iterace tohoto algoritmu má časovou složitost $O(n^2)$ operací v pohyblivé desetinné čárce. Rychlost konvergence a tedy celkový počet iterací závisí na volbě x_0 a poměru $\frac{\lambda_2}{\lambda_1}$. Jediné, co je potřeba implementovat, je násobení matice s vektorem, ukládat je třeba vektor x a vlastní hodnotu. Jelikož matici A pro tento algoritmus není nutné ukládat jako $n \times n$ pole, metoda je obzvláště zajímavá pro řídké matice. Algoritmus se zastaví ve chvíli, kdy je odhad $Ax - \lambda x$ menší než nějaké . Tato metoda ovšem nepředpokládá žádnou zvláštní strukturu matice, což je její velkou výhodou. Další výhodou je i možnost využití na komplexní matice.

Algoritmus 2: Mocninná metoda

Input: A $C^{n,n}$, x_0 C^n, přesnost *tol*
Output: Dominantní vlastní číslo λ_1

```
1  x = x_0;
2  e = x · A · x;
3  while True do
4      Ax = A · x;
5      x = x/ Ax|_2);
6      e_n = x ·A· x;
7      if |e − e_n| < tol then
8          break;
9      end
10     e = e_n;
11 end
12 return e;
```

4.2 Lanczosova metoda

Lanczosova metoda je metodou určenou k řešení problému hledání několika největších a nejmenších vlastních čísel. Jedná se o metodu určenou pro symetrické, respektive hermitovské matice. Zobecněním této metody je Arnoldiova metoda.

Metoda stojí na využití Krylovových podprostorů a jejich vlastností. Krylovy podprostory jsou podprostory generované vektory, které vznikají při iteracích mocninné metody, tedy pro matici A, vektor q a parametr m N klademe

$$K_m(A, q) = q, Aq, A^2q, \ldots, A^{m-1}q$$

Posloupnosti $q, Aq, A^2q, \ldots, A^nq$ se říká Krylovova posloupnost.

Tyto Krylovovy podprostory mají několik zajímavých vlastností, které umožňují hermitovskou či symetrickou matici tridiagonalizovat, což umožní jednodušší výpočet vlastních čísel.

Cílem metody je najít regulární matici Q $R^{n,m}$ takovou, že platí

$$T = Q^{-1}AQ$$

a zároveň je matice T $R^{m,m}$ tridiagonální. Toho se dá dosáhnout pomocí Householderových transformací či Givensových rotací, jež budou rozebrány v další části. Ty vyžadují v každém kroku úpravu matice A, což je nevýhodné, je-li matice řídká, a to obzvláště v případě Householderových transformací, které řídkost nezachovávají. Matici $T = Q^{-1}AQ$ lze však zkonstruovat i přímo, postupnou konstrukcí posloupnosti vektorů – sloupců matice Q. Jsou-li sloupce Q po sloupcích rovny vektorům Krylovovy posloupnosti

$q_1, \ldots, q_m$ a je-li

$$T = \begin{bmatrix} \alpha_1 & \beta_1 & & \ldots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{j-1} \\ 0 & \ldots & & \beta_{j-1} & \alpha_j \end{bmatrix},$$

pak při úpravě rovnice $AQ = QT$, je možno získat následující rekurenci

$$q_{j+1} = Aq_j - \alpha_j q_j - \beta_{k-1} q_{k-1},$$

Tím provedeme ortonormalizaci báze Krylovova podprostoru. Ortonormalita vektorů q_i pak naznačuje, že $\alpha_j = q_j^T A q_j$. Navíc, pokud $r_j = (A - \alpha_j E) q_j - \beta_{j-1} q_{j-1}$ je nenulové, pak platí $q_{j+1} = \frac{r_j}{\beta_j}$. Koeficient β_j se spočte jako norma $\pm \|r_j\|_2$. Pokud je r_j rovno nule, iterace se rozbije, nicméně v praxi toto příliš často nenastává [36] a pokud ano, znamená to, že $q_1, \ldots, q_k$ je vlastní podprostor matice A.

Výsledkem algoritmu je tedy tridiagonální matice $T \in R^{m,m}$, jejíž spektrum je podmnožinou spektra matice A. Matice T může mít přitom mnohem menší rozměry.

Výsledkem těchto poznatků je algoritmus 3

Algoritmus 3: Lanczosova symetrická metoda

Input: Symetrická matice $A \in R^{n,n}$ počáteční vektor $q \in R^n$

Output: Ortonormální vektory $q_1, \ldots, q_m$, takové, že

$$\langle q_1, \ldots, q_m \rangle = \langle q_1, Aq_1, \ldots, A^{m-1} q \rangle$$

```
1  q_1 = q/||q||_2
2  for k=1:m-1 do
3      q_{k+1} = A q_k;
4      α_k = q_k^T q_{k+1};
5      q_{k+1} = q_{k+1} - α_k q_k;
6      β_k = ||q_{k+1}||_2;
7      if β_k == 0 then
8          set flag invariant;
9          break;
10     end
11     q_{k+1} = q_{k+1}/β_k
12 end
```

Vektory $q_1, \ldots, q_m$ jsou nazývány Lanczosovy vektory.

Vlastní čísla matice T jsou pak poměrně dobrou aproximací vlastních hodnot matice A. Matice T ale může mít menší rozměry, což je výhodné, pokud

nás zajímá pouze několik málo hodnot. Pro jejich samotný výpočet je pak možno použít další metody, například metodu půlení intervalů, která je detailně popsána například v [36]. Tato metoda dokonce konverguje lineárně, stále probíhá výzkum optimalizací této metody, viz například [37]. Lze využít i další metody, ať už mocninnou, nebo například QR algoritmus.

Časová složitost jedné iterace Lanczosova algoritmu je $O(n^2)$, nejnáročnější je zde výpočet $\mathbf{A}q_k$. Celý algoritmus má tedy složitost $O(n^3)$, jeho výsledkem nicméně nejsou vlastní hodnoty jako takové, nýbrž matice $\mathbf{T}$.

4.3 QR algoritmus

V praxi nejrozšířenějším a nejpoužívanějším algoritmem pro výpočet vlastních hodnot je QR algoritmus. Používá se především pro husté matice v případě, že potřebujeme znát celé spektrum matice. Algoritmus umožňuje také jednoduše spočítat všechny příslušné vlastní vektory.

Tento algoritmus využívá vlastností QR rozkladu matice. QR rozklad je rozklad zadané matice $\mathbf{A}$ na ortogonální matici $\mathbf{Q}$ a horní trojúhelníkovou matici $\mathbf{R}$. Tedy $\mathbf{A} = \mathbf{QR}$.

QR rozklad lze implementovat několika různými způsoby, v praxi nejčastěji používané jsou Householderovy reflexe a Givensovy rotace.

Givensovy rotace, pojmenované po svém objeviteli, kterým byl americký matematik James Wallace Givens Jr., jsou transformace, jež umožňují „vyrobit“ nuly na specifických místech v matici. Využívá se zde následujících matic:

$$\mathbf{G}(i,k,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix},$$

kde $c = \cos(\theta)$ a $s = \sin(\theta)$ pro nějaké θ a $g_{i,i} = g_{k,k} = c$, $g_{k,i} = s$ a $g_{i,k} = -s$. Givensovy matice jsou pak očividně ortogonální [36]. Násobením vektoru zleva maticí $\mathbf{G}(i,k,\theta)^T$ se dostane rotace proti směru hodinových ručiček o θ radiánů v rovině dané $i$tou a $k$tou souřadnicí. Například pokud $x \in \mathbb{R}^n$ a $y = G(i,k,\theta)^T x$,

$$y_j = \begin{cases} cx_i - sx_k, & \text{pokud } j = i, \\ sx_i + cx_k, & \text{pokud } j = k, \\ x_j, & \text{jinak} \end{cases}$$

Díky tomu je možné při vhodné volbě c a s dostat do výsledného vektoru na $k$tou. Takové c a s lze najít pomocí následující rovnice:

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}} \quad \text{a} \quad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}.$$

Při výpočtu rozkladu se proto prochází postupně po sloupcích prvky pod diagonálou a nulují se. Tyto změny se násobením zprava Givensovou maticí ukládají do matice Q. V matici R naopak násobením zleva maticí G^T postupně vzniká horní trojúhelníková matice. Pseudokód algoritmu je popsán v algoritmu č. 4, jeho složitost je je $6n^3$ floating point operací [36].

Algoritmus 4: QR rozklad pomocí Givensových rotací

Input: Matice A
Output: Matice Q, R, takové, že Q je ortogonální a R je horní trojúhelníková matice a zároveň platí, že A = QR.

```
1  m = rows(A);
2  n = cols(A);
3  Q = E_n;
4  R = A;
5  for j = 1:n do
6      for i = m:-1:j+1 do
7          c,s = givens(r_{i-1,j}, r_{i,j});
8          R = G(i − 1, i, θ)^T R;
9          Q = QG(i − 1, i, θ);
10     end
11 end
```

Druhým způsobem výpočtu QR rozkladu jsou Householderovy reflexe, či transformace. Oproti Givensovým rotacím vyžaduje QR rozklad pomocí Householderových transformací polovinu operací [38]. Pomocí této transformace se nuluje celý sloupec až na jeho první prvek. Děje se tak za pomocí matice

$$\mathrm{P} = \mathrm{E} - \frac{2vv^T}{v^T v},$$

kde $v \quad \mathrm{R}^n$ a $v \quad = 1$. Matice H se nazývá Householderova matice a vektor v se nazývá Householderův vektor. Matice P je symetrická a ortogonální [36].

Pokud chceme, aby Px byl nulový až na první prvek, určí se v jako

$$v = x + \mathrm{sign}(x_1) \quad x \; {}_2 e_1,$$

kde $e_1 = \mathrm{E}_{:,1}$. V praxi se pak používá normalizovaný vektor v, tedy celý vektor se vydělí prvním prvkem. Algoritmus výpočtu Householderova vektoru ukazuje algoritmus 5.

Algoritmus 5: Výpočet Householderova vektoru, funkce house

Input: vektor $x \in R^n$, n
Output: Householderův vektor v

$\mu = |x|_2$;
v = x;
if $\mu = 0$ then
    $\beta = x_1 + sign(x_1)\mu$;
    $v_{2:n} = v_{2:n}/\beta$;
end
$v_1 = 1$;

Householderovy reflexe jsou zajímavé i z důvodu, že nulované části matice lze využít pro uložení vektoru v. QR rozklad pomocí Householderových transformací pak ukazuje algoritmus 6.

Algoritmus 6: QR rozklad za pomoci Householderových reflexí

Input: Matice $A \in R^{n,n}$
Output: Matice Q a R, takové, že A = QR

$R = A$
for *j=1:n* do
    $v_j = house(R_{j:n,j})$;
    $\beta = -2/v_j^T v_j$;
    $w = \beta R_{j:n,j:n} v_j$;
    $R_{j:n,j:n} = R_{j:n,j:n} + v_j w^T$;
    if $j < n$ then
        $A_{j+1:n,j} = v_{2:n}$;
    end
end
$Q = E_n$;
for *j = n:-1:1* do
    $\beta = -2/v_j^T v_j$;
    v $w = \beta Q_{j:n,j:n} v_j$;
    $Q_{j:n,j:n} = Q_{j:n,j:n} + v_j w^T$;
    $Q_{j:n,j:n} = Q_{j:n,j:n} + v_j w^T$;
end

Algoritmus 6 má pak složitost $8n^3/3$ floating point operací, tedy pro husté matice je výhodnější používat právě Householderovy transformace. Pro řídké matice nicméně může být výhodnější použít Givensovy rotace, jelikož nevadí ukládat několik řídkých matic a naopak Givensovy rotace zachovávají řídkost matic. Givensovy rotace je možné upravit tak, aby dosahovaly stejné rychlosti jako Householderovy reflexe [36].

Vlastní QR algoritmus pak využívá podobnosti matic. Základní vztah pro tento algoritmus je popsán následovně:

$$A_k = Q_k R_k,$$
$$A_{k+1} = R_k Q_k = E R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A_k Q_k$$

Rozložíme-li tedy matici A_k na součin matic Q a R, získáme matici A_{k+1} jako součin matic R a Q. Díky ortogonalitě matice Q jsou si pak matice A_k a A_{k+1} podobné.

Lze dokázat, že při QR rozkladu pomocí Givensových rotací, resp. Householderových transformacích posloupnost A_k konverguje k matici, jež má na diagonále právě vlastní čísla matice A_1, respektive v případě matic, jejichž vlastní čísla jsou komplexní, jejich reálné části. Důkaz je možné nalézt například v knize od J. H. Wilkinsona [39].

V praxi se ukazují způsoby, jak vylepšit konvergenci algoritmu. Jedním z nich je předem uvést matici do Hessenbergovského tvaru. Tento tvar je zachováván QR iteracemi a tedy při rozkladu matice stačí vždy projít pouze prvky přímo pod diagonálou a nikoli zbytek matice. Redukce na Hessenberovský tvar se provádí většinou přes Householderovy transformace [36]. Dále se pro zlepšení konvergence používá především strategie posunů, ilustrovaná algoritmem 7.

Algoritmus 7: QR aloritmu s posuny

```
   Input: Matice A   R^{n,n}
1  for k=1,2,.. do
2      urči skalár μ;
3      A-μE = QR;
4      A = RQ + μE;
5  end
```

Jedním ze způsobů určení skaláru μ je vzít prvek $A_{n,n}$. To se nazývá „Strategie implicitního posunu“ [36].

Konvergence algoritmu je ovlivňována poměry $\frac{\lambda_{k+1}}{\lambda_k}$ [40]. Při optimální strategii lze dosáhnout až složitosti $25n^3$ floating point operací při výpočtu jak vlastních hodnot tak i příslušných vlastních vektorů. Pokud jsou požadována pouze vlastní čísla, složitost je většinou $10n^3$, jak vychází dle [36] z empirického pozorování.

Kapitola 5

Implementace

Praktickým výstupem této práce je implementace QR algoritmu s Givensovými rotacemi. Implementace vychází z obecného QR algoritmu bez posunů.

Jako technologie byl zvolen programovací jazyk Julia [41]. Julia je dynamicky typovaný jazyk určený pro vědecké výpočty. Díky JIT – *Just In Time* – kompilaci implementované technologií LLVM dosahuje rychlosti skoro stejné jako jazyk C [42]. Mezi hlavní výhody jazyka patří jednoduchá syntaxe a paralelizace, ale i indexování pole od 1 nikoli od 0 [43], což je naopak částí odborné veřejnosti vnímáno jako nevýhoda. Jazyk byl publikován v roce 2012, nyní, v roce 2021, vyšla verze 1.6 [41, 42].

Výstup je koncipován jako projekt v jazyce Julia. Poskytuje několik funkcí a to: `qreigenall(A,atol,maxiteration,log)`, která vrací vlastní čísla i vektory zadané matice A, `qreigenval(A,atol,maxiteration,log)`, která vrací pouze vlastní hodnoty a `qreigvecs(A,atol,maxiteration,log)`, která vrací vlastní vektory. K modulu je dodána i uživatelská dokumentace.

Vnitřně je kód členěn do více funkcí, které jsou každá ve vlastním souboru, pojmenovaném jménem funkce. Implementace se dá rozdělit do několika pomyslných částí.

5.1 Givensovy rotace

První částí je implementace Givensových rotací, která je realizována pomocí několika funkcí. Naivní implementace využívá dvě funkce, a to funkci `givensrotations(A::Matrix{T}) where {T<:Number}` a dále pak funkci `givenscoefficients(a::Number, b::Number)`. `Number` je libovolný číselný typ jazyka Julia, notace s `T` pak umožňuje používat tento typ v rámci funkce – například při vytváření kompatibilní identické matice.

```
function givenscoeff(a::Number, b::Number)
  if b == 0
    cos = 1
    sin = 0
  elseif abs(b) > abs(a)
    r = a / b
    sin = 1 / sqrt(1 + r^2)
    cos = sin * r
  else
    r = b / a
    cos = 1 / sqrt(1 + r^2)
    sin = cos * r
  end

  return cos, sin
end
```

Výpis kódu 1: Funkce `givenscoeffinients(a,b)`

Funkce `givenscoefficients(a::Number, b::Number)` vrací koeficienty jež umožňují vynulovat prvek b – tedy spočítá $c = \cos(\theta)$ a $s = \sin(\theta)$ tak, že

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}.$$

Funkce provede 5 floating point operací a pouze jednou odmocní. Není třeba počítat přesnou hodnotu úhlu rotace θ, takže není nutné využívat trigonometrické funkce.

Funkce `givensrotations(A::Matrix{T}) where {T<:Number}` využívá klasického algoritmu na výpočet Givensovy rotace, kdy na základě koeficientů z funkce `givenscoefficients(a::Number, b::Number)` zkonstruuje celou Givensovu matici a násobením zleva, respektive zprava, počítá matici R, respektive Q.

Tento přístup je poměrně neefektivní, jelikož hlavní výhodou Givensových rotací je, že matice $G(i-1, i, \theta)$ je kromě čtyřech prvků ($g_{i-1,i-1}$, $g_{i-1,i}$, $g_{i,i-1}$, $g_{i,i}$) rovna matici E_n, tím pádem lze místo celých matic pracovat pouze s příslušnými řádky, resp. sloupci, $G_{i-1:}$ a $G_{i:}$ resp. $G_{:i-1}$ a $G_{:i}$.

Díky této optimalizaci má algoritmus 4 složitost $O(n^3)$, kde n je velikost matice. Bez optimalizací má samo násobení složitost $O(n^3)$, celý algoritmus tedy (n^5).

Tato optimalizace se poměrně zásadně projevila i při benchmarkovém testu spouštěném na stovce matic za použití datového typu `Float64` na maticích

o velikosti 75 × 75. Pro benchmark byl využit balíček BenchmarkTools.jl [44], výstupy z benchmarku jsou vidět na následujícím výpisu.

```
#Neoptimalizovaná verze
BenchmarkTools.Trial:
memory estimate: 6.92 GiB
allocs estimate: 1594059
--------------
minimum time: 5.898 s (19.85 % GC)
median time: 5.898 s (19.85 % GC)
mean time: 5.898 s (19.85 % GC)
maximum time: 5.898 s (19.85 % GC)
--------------
samples: 1
evals/sample: 1

#Optimalizovana verze
BenchmarkTools.Trial:
memory estimate: 8.15 MiB
allocs estimate: 1559
--------------
minimum time: 21.407 ms (0.00 % GC)
median time: 24.608 ms (0.00 % GC)
mean time: 25.100 ms (4.25 % GC)
maximum time: 53.806 ms (53.19 % GC)
--------------
samples: 200
evals/sample: 1
```

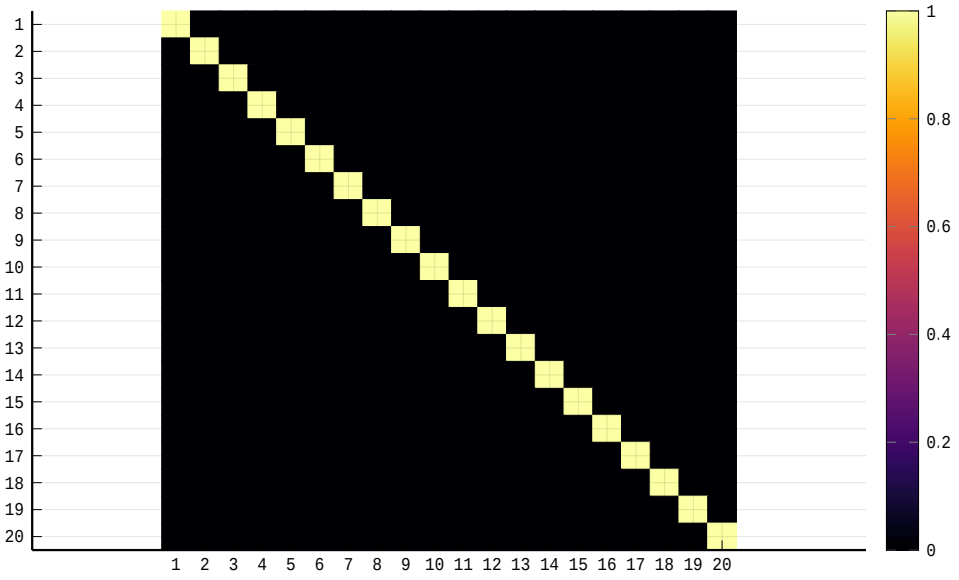
Test srovnával funkce `givensrotations(A)` a `givensrotationsopt(A)`. Finální verze funkce pro QR rozklad je `qrdecompositiongivens(A)`. Ta využívá optimalizace, jež počítá s tím, že matice na vstupu je Hessenbergovská, což funkce i kontroluje. Tím pádem není nutné provádět $n^2/2$ rotací, nýbrž pouze $n - 1$ rotací. Každá rotace má pak složitost $8n$ floating point operací. Celkem tedy dostáváme složitost $8n^2$ operací.

Funkce `givensrotationsopt(A)` a `qrdecompositiongivens(A)` využívají funkci `rotation(Q, R, cs, sn, i)`, která provádí samotnou rotaci.

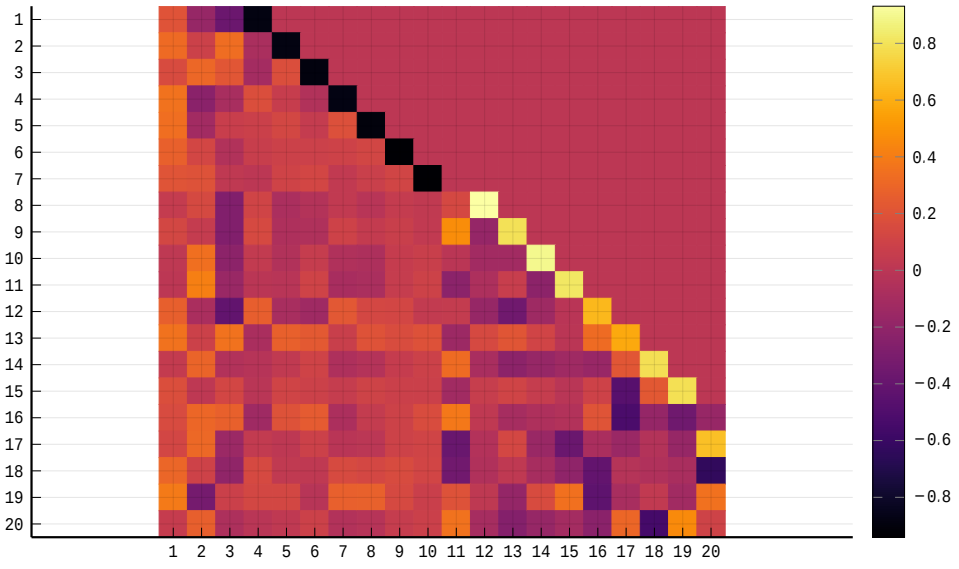
Průběh algoritmu jde poměrně hezky ilustrovat. Na obrázcích 5.2 a 5.1 lze pozorovat průběh algoritmu, kdy matice se Q postupně plní a matice R naopak vyprazdňuje.

5.2 Hessenbergovská redukce

Pro optimalizaci se využívá funkce `hessenbergstep(A)`. Tato funkce pomocí Householderových transformací spočítá matice H a U_0, tak, že platí A =



(a) Před začátkem rozkladu



(b) Zhruba třetina – stav po 63. iteraci ze 190

(c) Zhruba dve tretiny { stav po 126. iteraci ze 190

(d) Stav po konci algoritmu

Obrazek 5.1: Matice Q v prubehu QR rozkladu

UHU^T a zaroven plat,ze U je matice ortogonaln, H matice hessenbergovska.

Funkce je napsana ve dvou variantach, variante, ktera vrac jak matici H, tak matici U, hessenbergstep(A::Matrix{T}) where {T<:Number} a ve variante hessenbergstepvals(A::Matrix{T}) where {T<:Number}, ktera matici U neuklada a nevrac. Pokud jsou totiz pozadovany pouze vlastn hodnoty, nen matici U potreba a proto je mozne ji nepocitat a usetrit tak kapacity.

(a) Před začátkem rozkladu

(b) Zhrubá třetina – stav po 63. iteraci ze 190

(c) Zhruba dve tretiny { stav po 126. iteraci ze 190

(d) Stav po konci algoritmu

Obrazek 5.2: Matice R v prubehu QR rozkladu

5.3 Implementace QR algoritmu

Balcek poskytuje tri funkce, ktere vypocitaj konecny vysledek. Zakladn funkc je qralgorithm(A::Matrix f Tg, atol::Real = eps(T), maxiterations = MAXITERATIONS, showprogress = SHOWPROGRESS) where

f T<:Number}. Tato funkce vrac pro zadanou matici jej vlastn csla a vektory.

Funkce bere na vstupu matici A , jez muze byt jakehokoli cselneho typu, uzivatel muze pak zadat absolutn odchylku, se kterou jsou pak kontrolovany nuly pod diagonalou. Na cselnem typu Float64 je nejmens odchylka, pri ktere algoritmus konverguje 1.e-322 . Argument maxiteration urcuje alternativn podmnku zastaven, kdy se funkce zastav, probehne-li pocet iterac rovny parametru. Je-li tento parametr nastaven na 1, nema na zastaven funkce vliv. Poslednm argumentem je showprogress , ktery zapna a vypna log, jez do konzole vypisuje iterace a prvek na nemz konc kontrola popsana nze. Vychoz hodnoty argumentu se nachaz v souboru config.jl .

Pokud jsou vyzadovany pouze vlastn csla a nikoli vlastn vektory, poskytuje balcek funkci qreigenval(A::Matrix f Tg, atol::Real = eps(T), maxiterations = MAXITERATIONS, showprogress = SHOWPROGRESS) where f T<:Number}. U teto funkce funguj argumenty stejne jako u funkce qralgorithm . Funkce vyuzva optimalizovane hessenbergovske redukce a neuklada matice Q, coz vede k vyss casove efektivite.

Posledn funkc je qreigvecs(A::Matrix f Tg, atol::Real = eps(T), maxiterations = MAXITERATIONS, showprogress = SHOWPROGRESS) where f T<:Number}. Funkce je pouze wrapper pro funkci qralgorithm , ktery zahazuje spoctane vlastn hodnoty.

Vsechny tri poskytovane funkce vyuzvaj pro kontrolu zastaven funkci checkdeflation(A::Matrix f Tg, tol = eps(T)) where f T<:Number}. Implementace funkce je prilozena v ramci vypisu koduc. 2. Tato funkce prochaz prvky prmo pod diagonalou a kontroluje pomoc funkce isapprox , zda pro zadanou toleranci jsou rovny nule. Funkce vrac index sloupce a prvek, na nemz kontrola selhala. Pokud uzivatel nastavil parametr log na hodnotu true, prvek se mu zobraz ve vypisu a uzivatel muze upravit presnost.

5.4 Testy

Kod byl v prubehu vyvoje testovan. K testovan bylo vyuzito nekolik skriptu. Testy vyuzvaly modul Tests [45], ktery je soucast standardn knihovny jazyka Julia.

Funkce implementujc QR rozklad byly testovany pomoc skriptu givensrotationstest.jl , jez spoustel opakovane funkci givensrotation() a kontroloval rovnost matice A a soucinu vracenych matic Q a R .

Skript hessenbergtest.jl testoval podobnym zpusobem funkce pro vypocet Hessenbergovske redukce. Krome kontroly validnch vysledku pro nahodne generovane matice kontroloval validn vysledky pro diagonaln matici a dalsch nekolik malych matic. Skript kontroloval rovnost spekter matice H a puvodn matice A , tvar matice H , i rovnost $A = QHQ^T$.

Samotne funkce na vypocet vlastnch hodnot a vektoru se kontrolovaly pomoc skriptu qrtest.jl a eigvalstest.jl . Oba skripty obsahovaly podobne

```
function checkdeflation(A::Matrix{T}, tol = eps(T), backwards
= false) where {T<:Number}
    m,n = size(A)

    if !backwards
        for j = n - 1:-1:1
            if !isapprox(A[j + 1, j], zero(T), atol = tol)
                return j, A[j + 1, j]
            end
        end
    else
        for j = 1:n - 1
            if !isapprox(A[j + 1, j], zero(T), atol = tol)
                return j, A[j + 1, j]
            end
        end
    end

    return 0, T(0)
end
```

Vypis kodu 2: Ukazka funkce checkdeflation(A)

testy, testovala se diagonaln matice, male symetricke matice s vcenasobnymi vlastnmicsly a nahodne generovane symetricke matice. Testy dosahovaly asi 86% pokryt. Pokryt se merilo pomoc balcku Coverage.jl [46].

5.5 Srovnan implementac

Implementace byla porovnana s dostupnymi implementacemi v knihovne LAPACK a v prostred Mathematica.

LAPACK, tedy Linear Algebra PACKage, je knihovna napsana v jazyce Fortran 90 proresen soustav linearnch rovnic, problemu vlastnchcsel a hledan singularnho rozkladu [47]. Mathematica je komercn vypocetn software od spolecnosti Wolfram [48]. V obou prpadech se jedna o destky let vyvjene systemy.

Aby bylo mozne implementace porovnavat, bylo nagenerovano 97 matic od velikosti 3 3 po velikost 100 100. Tyto matice se ulozily do .csv sou-

```
@testset "Random symmetric matrices" begin
    testcount = 100
    testmin = 20
    testdimension = 20
    @info("Random tests begin")

    for i = 1:testcount
        @info("Test number ", i)
        m = rand(testmin:testdimension)
        A = rand(Float64 ,m,m)
        A = A*A

        my_evals, U = qralgorithm(A, 1.e-100, -1)
        @test isapprox(sort(my_evals), eigvals(A))
        @test isapprox(A * U, U*diagm(my_evals) )

    end
    @info("Random tests done")
end
```

Výpis kódu 3: Ukázka testů metody qralgorithm

bodů. V každém z testů byly pak matice načteny, spočtala se vlastní čísla. Výsledky testů se ukládaly k pozdější analyze, jsou dostupné na přiloženém médiu. Všechny testy byly provaděny na notebooku Dell G5 15 5587, s procesorem Intel Core i7 osmé generace. LAPACK byl volán přes Julii, jelikož je součást balíčku LinearAlgebra, skriptem `benchmark_lapack.jl`. Prostředí Mathematica bylo testováno pomocí skriptu `benchmark_mathematica.nb`.

Jako první varianta konfigurace pro implementaci v Julii byla nastavena přesnost `eps(Float64)`. Maximální počet iterací nebyl nastaven, v obrázku 5.3 označeno jako „Julia { `eps(Float64)`\. Dalšími testovanými variantami byla přesnost na 1.e-10 , přidaný maximální počet iterací { kvadratický s `n` při velikosti vstupu `n` `n` a poslední testovanou alternativou bylo nastavení maximálně třiceti iterací.

Rychlost implementace nebyla v porovnání s knihovnou LAPACK ani s prostředím Mathematica dobrá, jak je vidět na obrázcích 5.4 a 5.3. Nejlepší výsledky z testovaných variant ukazovalo nastavení s maximálně 1000 iteracemi, nicméně ani to se rychlostně nebližilo implementacím v prostředí Mathematica a knihovně LAPACK. To je dáno především použitím Given-

sowych rotac na huste matice a pouzitm jednoduss verze algoritmu. Meren mohlo byt lehce ovlivneno vytzenm poctace, vysvetluje to lehke rozdly ve srovnan jednotliwych kon gurac, kdy na nekterych vstupech vysla casove lne varianta s nastavenym maximalnm poctem iterac.

Na obrazku je wyrazne videt peak u nekterych matic, zvlaste u matice s rozmery 91 91. Algoritmus u techto vstupu konvergoval velmi pomalu, je to kwli degeneraci spektra, tyto matice maj nekolik vlastnch csel velmi blzko u sebe.

Dale byla zkoumana presnost algoritmu. Pro porovnan bylo zvoleno srovnan kumulativn sumy rezidua. Reziduum je rozdl soucinu Ax a λx, kde x je vlastn vektor a lambda k nemu prslusne vlastn cslo. Tyto dva wyrazy by si mely byt rovne, ale jelikoz wypocet nen presny, ve skutecnosti tomu tak nen. Proto se rozdl techto dvou wyrazu da wyuzt k meren chyby wypoctu. V techto testech se porovnava kumulativn chyba, tedy soucet rezidua { absolutnho rozdlu Ax pres wsechny vlastn vektory. Zapsano matematicky, de nujeme-li matici M jako rozdl matic AU a soucinu U s diagonaln matic, ktera ma na diagonale vlastn csla $\lambda_1, \ldots, \lambda_n$ { D .

$$M = AU \quad UD$$

. Porovnavame wyraz

$$\sum_{i=1}^{n} \sum_{j=1}^{n} m_{i,j} :$$

Chyba wypoctu se zwysuje v zawislosti na rozmerech vstupn matice. Je wyrazne videt, ze ve chwli, kdy algoritmus nedobehne, tedy je zastaven maximalnm poctem iterac, lze pozorovat wyss chybu, jak je videt na obrazku 5.5. Peaky odpowdaj vstupum, kde dosahowaly wyrazne wyss chcasu behy zastawovane bez ohledu na pocet iterac. Krwky, ktere se prekryvaj jsou pak detailne videt na obrazku 5.6.

Obrázek 5.3: Porovnání rychlosti implementací

Obrázek 5.4: Porovnání rychlosti implementací – prvních 50 vstupů

Obrázek 5.5: Porovnání přesnosti

```
for k=3:100
    display(k)
    M = CSV.File("samples/matrix_"*string(k)*".csv", header = false ) |> Tables.matrix
    vals,vecs = qralgorithm(M, eps( Float64 ),-1)
    res = sum(sum(broadcast(abs,M * vecs - vecs*diagm(vals)),dims=2),dims = 1)[1]

    dataframe = DataFrame(Technology = String[], Iteration = Int [], Time = Float64 [])
    push!(dataframe,("Julia - original",k,res))
    CSV.write("benchmarks/julia_error.csv", dataframe, writeheader = false , append = true )
end
```

Výpis kódu 4: Skript benchmark_julia.jl pro testování přesnosti

Na obrázku 5.6 lze také pozorovat, jak roste chyba s velikostí matice, jde o srovnání právě běhů, kdy nebyl algoritmus zastavován natvrdo s knihovnou LAPACK a prostředím Mathematica. Vzhledem k tomu, že jde o chybu kumulativní, tedy součet přes všechny prvky, je logické, že roste s rostoucí velikostí matice, nicméně můžeme vidět, že implementace je schopna poskytovat poměrně přesné výsledky bez ohledu na velikost vstupu.

Obrázek 5.6: Porovnání přesnosti a přesnějších metody

Závěr

Práce shrnuje dějiny problematiky vlastních čísel a to od nejstarších zmínek o maticích, přes vývoj pojmenování fenoménu vlastních čísel až po vývoj numerických metod ve dvacátém století. Popisuje některé aplikace výpočtu vlastních čísel. Podrobněji vysvětluje Markovské řetězce, algoritmus výpočtu Page Ranku, používaný například ve vyhledávači Google, analýzu hlavních komponent, což je jedna z metod zpracování dat, jako aplikace z oblasti informatiky. Dále je pak rozebírána souvislost hledání kořenů polynomů s vlastními čísly a další aplikace z mnoha různých oblastí, od stavebnictví, přes medicínu až po fyziku a chemii.

Práce analyzuje tři vybrané metody výpočtu vlastních čísel, potažmo vektorů. Rozebírá a analyzuje mocninnou metodu, což je jedna z nejstarších metod výpočtu, jež vypočítá jedno v absolutní hodnotě největší číslo, Lanczosovu metodu, a QR algoritmus, kde je rozebírán zejména rozdíl Householderových transformací a Givensových rotací.

Praktická část práce pak implementuje QR algoritmus s Givensovými rotacemi. Algoritmus byl implementován jako modul v jazyce Julia. Modul poskytuje několik metod, jež umožňují uživateli spočítat vlastní čísla, případně vektory. Je možné nastavovat přesnost i sledovat postup algoritmu pomocí logů.

Implementace byla porovnána s dalšími alternativami, konkrétně s knihovnou LAPACK a s prostředím Mathematica. Porovnávala se rychlost běhu a velikost chyby. Alternativy vyšly jako rychlejší a numericky stabilnější. Porovnávaly se i různé možnosti konfigurace metod poskytovaných v rámci této práce. Součástí práce je i uživatelská dokumentace

Cíle práce, tedy zmapovat historický vývoj problematiky, popsat možné aplikace výpočtu vlastních čísel, analyzovat metody výpočtu vlastních čísel, implementovat QR algoritmus s Givensovými rotacemi a tuto implementaci porovnat s alternativami - prostředím Mathematica a knihovnou LAPACK, se podařilo naplnit.

Prace by mohla byt rozsrena o implementaci dalsch metod, jako je napr klad mocninn metoda. Bylo by mozne optimalizovat algoritmus pro leps rychlost konvergence i pro vyss numerickou stabilitu metody.

Bibliografie

1. DOMBEK, Daniel; KALVODA, Tomáš; LUDÍK, Kleprík; KLOUDA, Karel-. Lineární algebra Studijní text [online]. FIT ČVUT, 2020 [cit. 2021-02-21]. Dostupné z: https://kam.fit.cvut.cz/deploy/bilin//lin-text.pdf .
2. ALMAN, Josh; WILLIAMS, Virginia Vassilevska. A Refined Laser Method and Faster Matrix Multiplication [online]. 2020 [cit. 2021-04-21]. Dostupné z arXiv: 2010.05846 [cs.DS] .
3. O'CONNOR, J. J.; ROBERTSON, E. F. Matrices and determinants [online]. School of Mathematics a Statistics University of St Andrews, Scotland, 2003 [cit. 2021-04-21]. Dostupné z: https://mathshistory.st-andrews.ac.uk/HistTopics/Matrices_and_determinants/ .
4. SCHWARTZ, Randy. A Classic from China: The Nine Chapters. The Right Angle [online]. 2008, roč. 16, s. 8–12 [cit. 2021-04-21]. Dostupné z: https://www.researchgate.net/publication/270904198_A_Classic_from_China_The_Nine_Chapters .
5. O'CONNOR, J. J.; ROBERTSON, E. F. Nine chapters [online]. School of Mathematics a Statistics University of St Andrews, Scotland, 2003 [cit. 2021-04-21]. Dostupné z: https://mathshistory.st-andrews.ac.uk/HistTopics/Nine_chapters/ .
6. CAUCHY, Augustin-Louis. Sur l'equation a l'aide de laquelle on determine les inegalites seculaires des mouvements des planetes. Oeuvres completes [online]. 1829, s. 174–195 [cit. 2021-04-27]. Dostupné z: doi: 10.1017/cbo9780511702686.009.
7. SYLVESTER, James Joseph. A new and more general theory of multiple roots. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 1841, roč. 18, č. 117, s. 249–254. Dostupné z: doi: 10.1080/14786444108650290.

8. TOU, Erik R. Math Origins: Eigenvectors and Eigenvalues[online]. University of Washington Tacoma, 2018 [cit. 2021-04-21]. Dostupné z: https://www.maa.org/press/periodicals/convergence/math-origins-eigenvectors-and-eigenvalues.
9. POINCARE, H. Sur les Equations aux Derivees Partielles de la Physique Mathematique. American Journal of Mathematics [online]. 1890, roč. 12, č. 3, s. 211 [cit. 2021-04-21]. Dostupné z doi: 10.2307/2369620.
10. JACOBI, Carl Gustav Jacob. "Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen"). Journal für die reine und angewandte Mathematik (Crelles Journal) [online]. 1846, roč. 1846, č. 30, s. 51–94 [cit. 2021-04-21]. Dostupné z doi: 10.1515/crll.1846.30.51.
11. GOLUB, Gene H.; VORST, Henk A. van der. Eigenvalue computation in the 20th century. Numerical Analysis: Historical Developments in the 20th Century [online]. 2001, s. 209–239 [cit. 2021-04-21]. Dostupné z doi: 10.1016/b978-0-444-50617-7.50010-0.
12. PARLETT, B. N. The Rayleigh quotient iteration and some generalizations for nonnormal matrices. Mathematics of Computation [online]. 1974, roč. 28, č. 127, s. 679–679 [cit. 2021-04-21]. Dostupné z doi: 10.1090/s0025-5718-1974-0405823-3.
13. ARNOLDI, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. Quarterly of Applied Mathematics [online]. 1951, roč. 9, č. 1, s. 17–29 [cit. 2021-04-21]. Dostupné z doi: 10.1090/qam/42792.
14. SLEIJPEN, Gerard L.; VAN DER VORST, Henk A. A Jacobi–Davidson Iteration Method for Linear Eigenvalue Problems. SIAM Review [online]. 2000, roč. 42, č. 2, s. 267–293 [cit. 2021-04-21]. Dostupné z doi: 10.1137/s0036144599363084.
15. CUPPEN, J. J. M. A divide and conquer method for the symmetric tridiagonal eigenproblem. Numerische Mathematik [online]. 1981, roč. 36, č. 2, s. 177–195 [cit. 2021-04-21]. Dostupné z doi: 10.1007/bf01396757.
16. O'CONNOR, J. J.; ROBERTSON, E. F. Andrei Andreyevich Markov - Biography [online]. 2006 [cit. 2021-04-21]. Dostupné z: https://mathshistory.st-andrews.ac.uk/Biographies/Markov/.
17. KEMENY, John G; SNELL, J Laurie. Markov chains [online]. Springer-Verlag, New York, 1976 [cit. 2021-04-21]. Dostupné z: https://www.academia.edu/download/50221101/markov.pdf.
18. BRIN, Sergey; PAGE, Lawrence. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems [online]. 1998, roč. 30, č. 1-7, s. 107–117 [cit. 2021-04-21]. Dostupné z doi: 10.1016/s0169-7552(98)00110-x.

19. BRYAN, Kurt; LEISE, Tanya. The $25,000,000,000 Eigenvector: The Linear Algebra behind Google. SIAM Review [online]. 2006, roč. 48, č. 3, s. 569–581 [cit. 2021-04-21]. Dostupné z: doi: 10.1137/050623280.

20. FERNANDEZ, Pablo. Google's pagerank and beyond: The science of search engine rankings. The Mathematical Intelligencer [online]. 2008, roč. 30, č. 1, s. 68–69 [cit. 2021-04-21]. Dostupné z: doi: 10.1007/bf02985759.

21. AUSTIN, David. Feature Column from the AMS: Pagerank [online]. 2006 [cit. 2021-04-21]. Dostupné z: http://www.ams.org/publicoutreach/feature-column/fcarc-pagerank.

22. Jak funguje algoritmy Vyhledávání [online]. Google, [n.d.]. Dostupné také z: https://www.google.com/search/howsearchworks/algorithms/.

23. GUPTA, Pankaj; GOEL, Ashish; LIN, Jimmy; SHARMA, Aneesh; WANG, Dong; ZADEH, Reza. Wtf: The who to follow service at twitter. In: Proceedings of the 22nd international conference on World Wide Web [online]. 2013, s. 505–514 [cit. 2021-04-21]. Dostupné z: doi: 10.1145/2488388.2488433.

24. ABDI, Herve; WILLIAMS, Lynne J. Principal component analysis. Wiley interdisciplinary reviews: computational statistics [online]. 2010, roč. 2, č. 4, s. 433–459 [cit. 2021-04-21]. Dostupné z: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wics.101.

25. TURK, Matthew; PENTLAND, Alex. Eigenfaces for Recognition. Journal of Cognitive Neuroscience [online]. 1991, roč. 3, č. 1, s. 71–86 [cit. 2021-04-21]. Dostupné z: doi: 10.1162/jocn.1991.3.1.71.

26. WOLD, Svante; ESBENSEN, Kim; GELADI, Paul. Principal component analysis. Chemometrics and Intelligent Laboratory Systems [online]. 1987, roč. 2, č. 1, s. 37–52 [cit. 2021-04-21]. ISSN 0169-7439. Dostupné z: doi: https://doi.org/10.1016/0169-7439(87)80084-9. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.

27. JOHNSON, Richard Arnold; WICHERN, Dean W et al. Applied multivariate statistical analysis [online]. Prentice hall Upper Saddle River, NJ, 2002 [cit. 2021-04-21]. Č. 8. Dostupné z: https://www.academia.edu/download/61022787/Applied_Multivariate_Statistical_Analysis_by_Johnson_and_Wichern20191025-53970-1j83v2h.pdf.

28. ROTH, Bradley J. The Eigenvalue Problem [online]. 2016 [cit. 2021-04-21]. Dostupné z: http://hobbieroth.blogspot.com/2016/04/the-eigenvalue-problem.html.

29. ABDRABOU, Ahmed; BELL, Daniel J. Di usion tensor imaging and ber tractography [online]. 2021 [cit. 2021-04-21]. Dostupn e z:https: //radiopaedia.org/articles/diffusion-tensor-imaging-and-fibre-tractography?lang=us .

30. LE BIHAN, Denis; MANGIN, Jean-Francois; POUPON, Cyril; CLARK, Chris A.; PAPPATA, Sabina; MOLKO, Nicolas; CHABRIAT, Hughes. Di usion tensor imaging: Concepts and applications.Journal of Magnetic Resonance Imaging[online]. 2021, roc. 13,c. 4, s. 534{546 [cit. 2021-04-21]. Dostupn e z doi : https://doi.org/10.1002/jmri.1076 .

31. HOPPENSTEADT, F. Predator-prey model. Scholarpedia [online]. 2006, roc. 1, c. 10, s. 1563 [cit. 2021-04-21]. Dostupn e zdoi : 10.4249/scholarpedia.1563 . revision #91667.

32. MACARTHUR, Robert. Species packing and competitive equilibrium for many species.Theoretical Population Biology [online]. 1970, roc. 1, c. 1, s. 1{11 [cit. 2021-04-21].issn 0040-5809. Dostupn e zdoi : https: //doi.org/10.1016/0040-5809(70)90039-0 .

33. KERMACK, William Ogilvy; MCKENDRICK, A. G.; WALKER, Gilbert Thomas. A contribution to the mathematical theory of epidemics. Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character[online]. 1927, roc. 115,c. 772, s. 700{721 [cit. 2021-04-21]. Dostupn e zdoi : 10.1098/rspa.1927.0118 .

34. ARIOLI, Gianni; GAZZOLA, Filippo. A new mathematical explanation of what triggered the catastrophic torsional mode of the Tacoma Narrows Bridge. Applied Mathematical Modelling [online]. 2015, roc. 39, c. 2, s. 901{912 [cit. 2021-04-21]issn 0307-904X. Dostupn e z doi : https: //doi.org/10.1016/j.apm.2014.06.022 .

35. BOROVY, Jir. Neco z historie ... Zricen Tacomskeho mostu v r.1940: Mosty [online]. 2006 [cit. 2021-04-21]. Dostupn e z:https : //www.izdoprava.cz/neco-z-historie-zriceni-tacomskeho-mostu-v-r-1940/ .

36. GOLUB, Gene H.; VAN LOAN, Charles F. Matrix Computations. Baltimore, USA: Johns Hopkins University Press, 2013.isbn 978-1-4214-0794-4.

37. OLIVEIRA, I. F. D.; TAKAHASHI, R. H. C. An Enhancement of the Bisection Method Average Performance Preserving Minmax Optimality. ACM Trans. Math. Softw. [Online]. 2020, roc. 47,c. 1 [cit. 2021-04-21]. issn 0098-3500. Dostupn e zdoi : 10.1145/3423597.

38. RALSTON, Anthony; RABINOWITZ, Philip. A rst course in numerical analysis Courier Corporation, 2001. isbn 978-0486414546.

39. WILKINSON, J.H. The algebraic eigenvalue problemOxford: Clarendon Press, 1972.isbn 9780198534181.

40. WATKINS, David. Fundamentals of matrix computations New York: Wiley-Interscience, 2002.isbn 0-471-21394-2.

41. BEZANSON, Je ; EDELMAN, Alan; KARPINSKI, Stefan; SHAH, Viral B. Julia: A fresh approach to numerical computing. SIAM Review [online]. 2017, roc. 59, c. 1, s. 65{98 [cit. 2021-04-24]. Dostupne zdoi: 10.1137/141000671.

42. Julia 1.6 Documentation [online]. 2021 [cit. 2021-04-24]. Dostupne z: https://docs.julialang.org/en/v1.6/ .

43. RAO, Vicky Singh. Julia Programming Language - A True Python Alternative [online]. 2018 [cit. 2021-04-24]. Dostupne z:https://www.technotification.com/2018/08/julia-programming-language.html.

44. JULIACI. BenchmarkTools.jl [online]. [N.d.] [cit. 2021-04-21]. Dostupne z: https://github.com/JuliaCI/BenchmarkTools.jl .

45. Unit Testing [online]. 2021 [cit. 2021-05-06]. Dostupne zhttps://docs.julialang.org/en/v1/stdlib/Test/ .

46. JULIACI. Coverage.jl [online]. GitHub, 2020 [cit. 2021-05-06]. Dostupne z: https://github.com/JuliaCI/Coverage.jl .

47. ANDERSON, E.; BAI, Z.; BISCHOF, C.; BLACKFORD, S.; DEMMEL, J.; DONGARRA, J.; DU CROZ, J.; GREENBAUM, A.; HAMMARLING, S.; MCKENNEY, A.; SORENSEN, D. LAPACK Users' Guide [online]. Third. Philadelphia, PA: Society for Industrial a Applied Mathematics, 1999 [cit. 2021-04-30]isbn 0-89871-447-8 (paperback). Dostupne z: http://www.netlib.org/lapack .

48. WOLFRAM RESEARCH, Inc. Mathematica, Version 12.1 [online]. [N.d.] [cit. 2021-04-30]. Dostupne z: https://www.wolfram.com/mathematica/. Champaign, IL, 2020.

Příloha **A**

Obsah přiloženého média

```
readme.txt ..... stručný popis obsahu a instrukce k instalac
code ..... zdrojové kódy implementace
thesis ..... text práce
    thesis.pdf ..... text práce ve formátu PDF
    src ..... zdrojové kódy práce
```