VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra aplikované matematiky

Efektivní SVD a jeho využití při zpracování biometrických dat

Efficient implementation of SVD and its application to biometric data processing

2009 Petr Kotas

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 16. dubna 2009 .....

Chtěl bych poděkovat především Doc. Mgr. Vítu Vondrákovi, Ph.D. za jeho podnětné připomínky a rady, které mě vždy navedly na správnou cestu řešení problémů. A také za jeho trpělivou kontrolu a korekci mého rukopisu, který by bez jeho pomoci nebyl zcela bez chyb. Dále bych rád poděkoval Ing. Davidu Horákovi, Ph.D. a Ing. Filipu Staňkovi za jejich pomoc při realizaci paralelních algoritmů na školním clusteru. Také bych chtěl poděkovat pánům Larrymu Page and Sergey Brinovi za úžasný vyhledávač, bez kterého bych nenašel tolik materiálů důležitých pro tuto práci. A v neposlední řadě patří můj dík mé přítelkyni, která mě vždy morálně povzbudila a vydržela to se mnou až do zdárného konce.

Abstrakt

Tato práce se zabývá efektivním nalezení SVD pro velké problémy. Tyto problémy jsou známé svou výpočetní náročností, která může být snížena využitím paralelních počítačů. Právě využitím paralelních algoritmů bylo dosaženo nejen potřebného urychlení, ale i zvýšení dostupné dimenze problému. Algoritmus zde představený je rozdělen na tři logické části - bidiagonalizace, výpočet singulárních čísel a singulárních vektorů. V druhé části práce je ukázána aplikace SVD na zpracování biometrických dat, konkrétně duhovek. Tyto data jsou vhodnou aplikací, protože generují velké dimenze problému, které je třeba řešit s malou relativní chybou. Je představen způsob, jak duhovky předzpracovat a jak je pomocí SVD porovnávat.

Klíčová slova: SVD, LSI, paralelní, duhovka, Gabor wavelet

Abstract

This thesis deals with efficient parallel implementation of SVD. It describes methods for implementation of parallel code on clusters and how to minimize communication between parallel processes. In second part, there is a general description of SVD application to biometric data processing. It describes method for image preprocessing that is vital for successful data comparison using LSI.

Keywords: SVD, LSI, parallel, iris, Gabor wavelet

Seznam použitých zkratek a symbolů

$\mathbb{R}$	–	množina reálných čísel
$\mathbb{C}$	–	množina komplexních čísel
A	–	matice
A^H	–	hermitovsky sdružená matice
T	–	třídiagonální matice
B	–	bidiagonální matice vzniklá transformací z matice A
I	–	jednotková matice
$\mathbf{x}$	–	vektor
rank(A)	–	hodnost matice
diag(A)	–	diagonála z A
SVD	–	singulární rozklad
superdiagonála	–	diagonála nad hlavní diagonálou
subdiagonála	–	diagonála pod hlavní diagonálou
$f * g$	–	konvoluce funkcí f a g

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Singulární rozklad (*SVD*) nachází v poslední době celou řadu aplikací. Mezi ty klasické patří řešení soustav rovnic se singulární maticí v řadě aplikací numerické matematiky nebo analýza rozptylu ve statistice. Mezi nové aplikace patří například filtrování signálů, komprese dat nebo měření podobnosti.

Právě problematikou měření podobností se zabývá Latentní sémantická analýza (*LSI*), která využívá *SVD* jako její hlavní nástroj. *LSI* je metoda hledání podobných slov ve velkých dokumentech, a jako taková našla uplatnění v mnoha vyhledávacích systémech [35].

Vše má bohužel svoji cenu a pro *SVD* je to její výpočetní náročnost. Po dlouhou dobu byla složitost tohoto algoritmu brzdou bránící jeho plnému nasazení pro velké problémy. Tyto problémy byly řešeny stochaicky, nebo rozložením problému na menší podproblémy za cenu nížší přesnosti, což ovšem není vždy možné, případně byla použita upravená verze *SVD*. Tyto upravené verze byly „ušity na míru“ danému problému, díky čemuž bylo dosaženo rozumného urychlení.

V dnešní době chytá *SVD* „druhý dech“. Rozvoj paralelních počítačů umožnil nový pohled na problém a jeho novou implementaci.

Tato diplomová práce se věnuje urychlení výpočtu *SVD* pro zpracování obrazu. Zaměříme se na problém rychlého vyhledávání v databázi biometrických dat. Paralelizace byla zvolena, protože při zpracování biometrických dat nemůžeme dovolit zmenšit přesnost výsledku a zároveň data mohou být dosti velká.

V kapitole 2 vysvětlíme stručně *SVD* a definujeme *LSI* a jeho základní princip. Následující část představí algoritmy pro samotný výpočet *SVD* a možnosti jejich paralelizace. V kapitole 4 definujeme potřebný aparát pro zpracování biometrických dat a jejich normalizaci pro korektní srovnání. Kapitola 5 navazuje na předchozí text a doplňuje jej o možné paralelismy v procesu analýzy a zpracování vstupních obrázků.

2 SVD a LSI

Pro správné pochopení vyhledávání pomocí latentní sémantiky je potřeba znát důkladně vlastnosti singulárního rozkladu. Proto si v této části představíme *SVD* spolu s jeho vlastnostmi, díky čemuž budeme moci definovat *LSI* a ukázat souvislost s *SVD*.

2.1 SVD

V maticovém počtu existuje celá řada maticových rozkladů. Například jmenujme *LU*, Choleského rozklad, LDL^T. Ovšem nejvýznamnějším z nich je singulární rozklad, zkráceně *SVD*. Vlastnosti tohoto rozkladu lze najít v [3]. Abychom mohli *SVD* definovat, musíme si nejdříve říct co jsou unitární matice a připomenout co je ortogonální doplněk. Matice $Q \in \mathbb{R}^{m\times m}$ se nazývá unitární pokud platí $QQ^H = Q^HQ = I$ a její sloupce $Q = [q_1, \ldots, q_m]$ tvoří ortonormální bázi. Taková báze se vždy dá rozšířit na úplnou ortonormální bázi dle věty:

Věta 2.1 *Nechť $V_1 \in \mathbb{C}^{n\times r}$ má ortonormální sloupce, existuje $V_2 \in \mathbb{C}^{n\times(n-r)}$ tak, že platí*

$$V = [V_1, V_2]$$

je unitární. Dále platí $im(V_1)^\perp = im(V_2)$.

Důkaz. Věta vyplývá ze základních poznatků úvodu do lineární algebry a důkaz lze najít třeba v [2] nebo [3]. ∎

Věta 2.2 *Mějme pevně dané m a n a matici $A \in \mathbb{C}^{m\times n}$. Pak existují matice $U \in \mathbb{C}^{m\times m}$ a $V \in \mathbb{C}^{n\times n}$, které jsou unitární a diagonální matice $\Sigma \in \mathbb{R}^{m\times n}$, pro které platí*

$$A = U\Sigma V^H. \tag{1}$$

Pro diagonální prvky matice Σ navíc platí $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{min(m,n)} \geq 0$. Součin $U\Sigma V^T$ budeme nazývat singulárním rozkladem (SVD) matice A.

Důkaz.[1] Nechť $x \in \mathbb{C}^n$ a $y \in \mathbb{C}^n$ jsou jednotkové vektory v euklidovské normě pro které platí $Ax = \sigma y$, kde $\sigma = \|A\|_2$. Podle věty 2.1 víme, že existuje $V_2 \in \mathbb{C}^{n\times(n-1)}$ a $U_2 \in \mathbb{C}^{m\times(m-1)}$ tak, že $V = [x, V_2] \in \mathbb{C}^{n\times n}$ a $U = [y, U_2] \in \mathbb{C}^{m\times m}$ jsou ortogonální matice. Matice U^HAV má následující strukturu

$$U^HAV = \begin{bmatrix} \sigma & w^T \\ 0 & B \end{bmatrix} \equiv A_1.$$

Protože platí

$$\left\| A_1 \left(\begin{bmatrix} \sigma \\ w \end{bmatrix} \right) \right\|_2^2 \geq (\sigma^2 + w^Tw)^2,$$

[1] Důkaz přejat z [3].

dostaneme $\|A_1\|_2^2 \geq (\sigma^2 + w^T w)^2$. Ale protože $\sigma^2 = \|A\|_2^2 = \|A_1\|_2^2$, dostaneme $w = 0$. Odtud je zřejmé, že pro dokončení důkazu stačí postupovat indukcí pro zbylé B. c.b.d.

■

Čísla σ_i se nazývají singulární čísla matice A a sloupce matice U (resp. V) jsou tzv. levé (resp. pravé) singulární vektory. Matice Σ má tvar

$$\begin{bmatrix} \Sigma \\ 0 \end{bmatrix}, \; pro\, m \geq n \text{ nebo } \begin{bmatrix} \Sigma & 0 \end{bmatrix}, \; pro\, m < n,$$

kde Σ je diagonální matice.

Někdy můžeme chtít jenom několik nejvýznamnějších singulárních čísel a k nim patřící vektory. V takovém případě lze vypočítat částečné *SVD*, které je popsáno vztahem

$$A = U_k \Sigma_k V_k^T, k < min(m, n). \tag{2}$$

Pro matice platí $A \in \mathbb{C}^{m \times n}, U_k \in \mathbb{C}^{m \times r}, \Sigma_k \in \mathbb{C}^{r \times r}, V_k \in \mathbb{C}^{r \times n}$.

Pro ilustraci je na obrázku 1 ukázáno, co udělá částečný *SVD* s originálními daty. Jednotlivé obrazce vznikly jako zpětné přenásobení matic U, Σ, V z *SVD* rozkladu původního obrázku, který je vidět úplně v levé části. Při zpětném přenásobení matic byla postupně nulována nejmenší singulární čísla. Je vidět, že snižováním k (počet nenulových singulárních čísel) dosahujeme jistého druhu komprese (filtru) původní informace. Jinými slovy, zachovává se jen podstatná část informace a zbytek je vyfiltrován. Zajímavé je, že i při vynulování prvních tří singulárních čísel začne jako první mizet kříž uvnitř čtverce a zachovají se vodorovné a svislé informace. Až při vymazání 16 nejmenších singulárních čísel začnou mizet také vodorovné a svislé čáry.

Příklad 2.1
SVD Toeplitzovy matice o rozměrech 3×3 je

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 1 & 2 \\ 3 & 2 & 1 \end{bmatrix} = \begin{bmatrix} -0.61 & 0.71 & 0.36 \\ -0.52 & 0.00 & -0.86 \\ -0.61 & -0.71 & 0.36 \end{bmatrix} \cdot \begin{bmatrix} 5.70 & 0 & 0 \\ 0 & 2.0 & 0 \\ 0 & 0 & 0.70 \end{bmatrix} \cdot \begin{bmatrix} -0.61 & -0.71 & -0.36 \\ -0.52 & 0.00 & 0.86 \\ -0.61 & 0.71 & -0.31 \end{bmatrix}^T$$

■

Obrázek 1: Ukázka aproximace obrazu pomocí *SVD*. Zleva $k = 36$ (plná hodnost), $k = 20$ a $k = 10$.

S *SVD* také souvisí Frobéniova norma.

Věta 2.3 *Nechť $A \in \mathbb{C}^{m\times n}$, pak pro její Frobéniovu normu $\|A\|_F$ platí*

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a|_{ij}^2} = \sqrt{trace(A^H A)} = \sqrt{\sum_{i=1}^{min(m,n)}\sigma_i^2} \tag{3}$$

$trace(A)$ je takzvaná stopa matice A a je definovaná jako

$$trace(A) := \sum_{i=1}^{min(m,n)} a_{ii},$$

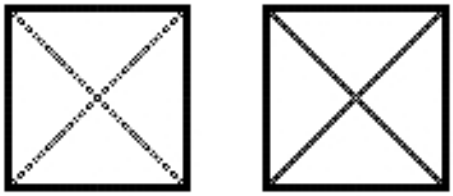
kde $\sigma_i, i = 1, \ldots, min(m,n)$, jsou singulární čísla matice A.

Důkaz. Důkaz lze najít v [3]. ∎

Nyní známe vše potřebné pro definici aproximace matice A pomocí *SVD*.

Definice 2.1 *Mějme matici $A \in \mathbb{C}^{m\times n}$. Matici $A_k \in \mathbb{C}^{m\times n}$ budeme nazývat nejlepší aproximací matice A ve smyslu Frobéniovy normy, jestliže pro k platí*

$$k = \arg\min_k(\|A - A_k\|_F), k < min(m,n). \tag{4}$$



Obrázek 2: Aproximace s minimalizací chyby. Zleva $k = 33$ (přesnost nastavena na 0.1), $k = 35$ (přesnost nastavena na 0.01).

Praktický výsledek takovéto aproximace je na obrázku 2. Obrázek je stejný jako na obrázku 1, ale pro aproximaci je použita rovnice 4.

2.2 LSI

„Latent semantic indexing“ je metoda vyhledávání ve velkém množství dokumentů podle určitých klíčových slov. Pro náš účel se omezíme na zjednodušenou variantu, ve které jsou vstupní data pouze ve formě jednoznačných výrazů. Úvod k této problematice lze najít v [33] a vysvětlení z pohledu pravděpodobnostního počtu v [34].

Jednotlivé dokumenty lze předzpracovat podle výskytu klíčových slov a sestavit z nich matici D, která má tvar

$$d_i^T \rightarrow \begin{matrix} \\ \\ \end{matrix} \begin{matrix} t_j \\ \downarrow \end{matrix} \begin{bmatrix} d_{1,1} & \dots & d_{1,n} \\ \vdots & \ddots & \vdots \\ d_{m,1} & \dots & d_{m,n} \end{bmatrix} = D.$$

Řádky $d_i^T = [d_{i,1}, \dots, d_{i,n}]$ představují jednotlivé dokumenty a jejich vazbu ke klíčovým slovům. Sloupce $t_j = [d_{1,j}, \dots, d_{m,j}]$ reprezentují vybraná klíčová slova a jejich výskyt v dokumentech[2].

Závislost mezi jednotlivými řádky (resp. sloupci) můžeme popsat pomocí korelace, tedy

$$c_{kl}^r = d_k^T d_l \text{ (resp. } c_{kl}^c = t_k t_l^T).$$

V maticovém zápisu vyjádříme vztah mezi všemi řádky jako DD^T a mezi všemi sloupci jako $D^T D$. Z předchozí části víme, že *SVD* existuje pro každou matici, tedy i pro matice výskytů

$$\begin{aligned} D &= U\Sigma V^T, \\ DD^T &= U\Sigma^2 U^T, \\ D^T D &= V\Sigma^2 V^T. \end{aligned}$$

Vztah těchto rovnic a *LSI* lze najít v [32]. Vazbu mezi tímto rozkladem a dokumenty si ukážeme v následující rovnosti

$$\begin{matrix} \\ d_i^T \rightarrow \end{matrix} \begin{matrix} t_j \\ \downarrow \end{matrix} \begin{bmatrix} d_{1,1} & \dots & d_{1,n} \\ \vdots & \ddots & \vdots \\ d_{m,1} & \dots & d_{m,n} \end{bmatrix} = \begin{matrix} \\ \hat{d}_i^T \rightarrow \end{matrix} \begin{bmatrix} u_{1,1} & \dots & u_{1,n} \\ \vdots & \vdots & \vdots \\ u_{m,1} & \dots & u_{m,n} \end{bmatrix} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_m \end{bmatrix} \begin{matrix} \\ \end{matrix} \begin{matrix} \hat{t}_j \\ \downarrow \end{matrix} \begin{bmatrix} v_{1,1} & \dots & v_{1,n} \\ \vdots & \dots & \vdots \\ v_{m,1} & \dots & v_{m,n} \end{bmatrix}.$$

$$D \qquad\qquad U \qquad\qquad \Sigma \qquad\qquad V$$

Vektor $\hat{d}_i^T$ ukazuje, že z celé matice U přispívá do původního vektoru dokumentu d_i^T pouze *i*-tý řádek. Pro vektor klíčových slov t_j je situace podobná, jedná se však o *j*-tý sloupec matice V, označme jej $\hat{t}_j$. Je zřejmé, že se nejedná o singulární vektory, ale o vektory závislé na nich.

Budeme-li chtít zjistit podobnost vektorů $\hat{t}_j$ a $\hat{t}_q$ (resp. $\hat{d}_i^T$ a $\hat{d}_q^T$) v prostoru generovaném singulárním rozkladem, vypočítáme mezi nimi úhel (5). Přirozeně, je-li úhel malý, je malý i rozdíl v informaci nesené těmito daty.

$$\theta = \arccos\left(\frac{\hat{t}_j \cdot \hat{t}_q}{|\hat{t}_j| \cdot |\hat{t}_q|}\right) \tag{5}$$

[2]Vektory jsou zapsány do řádku kvůli úspoře místa. Vektory bez transpozice jsou myšleny sloupcové.

Víme tedy jakým způsobem zjistit podobnost dvou vektorů ve vypočteném vektorovém prostoru tvořeném bází U (resp. V). Otázkou je, jak porovnat vektor q_d, který představuje „mini dokument“ pro nalezení. Je zřejmé, že nejdříve jej budeme muset promítnout do prostoru s bází V. Toto lze provést ortogonální projekcí (7) a pomocí (5) lze nalézt podobný dokument v bázi V.

$$q_d = U\Sigma\hat{q_d}, \tag{6}$$
$$\hat{q_d} = \Sigma^{-1}U^T q_d. \tag{7}$$

Budeme-li hledat vektor q_t obsahující klíčová slova, platí analogický vztah

$$q_t^T = \hat{q_t}^T\Sigma V^T, \tag{8}$$
$$\hat{q_t}^T = q_t^T V\Sigma^{-1}, \tag{9}$$
$$\hat{q_t} = \Sigma^{-1}V^T q_t. \tag{10}$$

Výše uvedené vztahy jsou ovšem výpočetně náročné a jejich přímé použití by vedlo k nemalým finančním nárokům na hardware. Z toho důvodu se při praktické implementaci využívá aproximace matice pomocí definice 2.1. Vztahy (6) a (7) přejdou na

$$q_d = U_k\Sigma_k\hat{q_d}, \tag{11}$$
$$\hat{q_d} = \Sigma_k^{-1}U_k^T q_d, \tag{12}$$

kde Σ_k a U_k jsou matice z definice 2.1.

3 Paralelizace numerického výpočtu

V této části se seznámíme s možnostmi urychlení výpočtu *SVD*. Tohoto se pokusíme dosáhnout modifikací stávajících algoritmů do podoby vhodné pro řešení na paralelních počítačích. Postupně si rozebereme jednotlivé fáze výpočtu, ukážeme jakým způsobem se dají paralelizovat a ukážeme jakého urychlení jsme dosáhli. Výsledné algoritmy jsou optimalizovány pro cluster výkonných počítačů. Tato část do značné míry čerpá z bakalářské práce [1] a dále rozvíjí metody v ní uvedené.

3.1 Bidiagonalizace

Většina algoritmů počítajících *SVD* začíná redukcí plné matice na matici bidiagonální. Tímto krokem se zrychlí celková konvergence algoritmu a sníží celková výpočetní náročnost. Připomeňme, že bidiagonální matice má pouze diagonálu a superdiagonálu (resp. subdiagonálu).

Bidiagonální matici můžeme získat pomocí Householderova zrcadlení.

Definice 3.1 *Nechť $v \in \mathbb{C}^n$, $v \neq 0$ a $\|v\| = 1$. Pak*

$$H = I - 2vv^*. \tag{13}$$

Matice H je matice Householderova zrcadlení a v Householderův vektor, který je defnován vztahem

$$v = x + \|x\|e, \tag{14}$$

kde $x \in \mathbb{C}^n$ je vektor, kterému chceme vynulovat všechny prvky až na první a e je bázový vektor příslušné délky.

Věta 3.1 *Pro matici H platí:*

$$\begin{aligned} H &= H^H, \\ H^{-1} &= H^H, \\ H^2 &= I. \end{aligned}$$

Důkaz. Po rozepsání vztahů dle definice 3.1 je důkaz zřejmý. ∎

Matici A můžeme zredukovat na matici B pomocí Householderových transformací tak, aby platilo

$$B = H_l A H_r, \tag{15}$$

kde H_l je levá Householderova transformace a H_r je pravá Householderova transformace. Pro tyto matice platí

$$\begin{aligned} H_l &= \prod_i H_i, i = 1, 3, 5, \ldots \\ H_r &= \prod_i H_i, i = 2, 4, 6, \ldots \end{aligned}$$

Postup bidiagonalizace nejlépe ilustrujeme na příkladu 3.1. Algoritmus implementovaný pro Matlab je na výpisu 1.

Příklad 3.1
Transformace $H_i = I - 2 \cdot \frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{v}_i}$ vytvářejí z $A \in \mathbb{R}^{m \times n}$ bidiagonálu následovně:

$$H_1 A = \begin{bmatrix} x & x & x & x & x \\ * & x & x & x & x \\ * & x & x & x & x \\ * & x & x & x & x \\ * & x & x & x & x \end{bmatrix}, \quad H_1 A H_2 = \begin{bmatrix} x & x & * & * & * \\ & x & x & x & x \\ & x & x & x & x \\ & x & x & x & x \end{bmatrix},$$

$$H_3 H_1 A H_2 = \begin{bmatrix} x & x & & & \\ & x & x & x & x \\ & * & x & x & x \\ & * & x & x & x \\ & * & x & x & x \end{bmatrix}, \quad H_3 H_1 A H_2 H_4 = \begin{bmatrix} x & x & & & \\ & x & x & * & * \\ & & x & x & x \\ & & x & x & x \end{bmatrix},$$

$$H_5 H_3 H_1 A H_2 H_4 = \begin{bmatrix} x & x & & & \\ & x & x & & \\ & & x & x & x \\ & & * & x & x \\ & & * & x & x \end{bmatrix}, \quad H_5 H_3 H_1 A H_2 H_4 H_6 = \begin{bmatrix} x & x & & & \\ & x & x & & \\ & & x & x & * \\ & & & x & x \end{bmatrix},$$

$$H_7 H_5 H_3 H_1 A H_2 H_4 H_6 = \begin{bmatrix} x & x & & & \\ & x & x & & \\ & & x & x & \\ & & & x & x \\ & & & * & x \end{bmatrix}, \quad B = \begin{bmatrix} x & x & & & \\ & x & x & & \\ & & x & x & \\ & & & x & x \\ & & & & x \end{bmatrix},$$

kde symbol $*$ reprezentuje vynulovaný prvek a x představuje modifikovaný prvek. Příklad převzat z [1]. ■

Příklad 3.2
Ukázka bidiagonální matice vzniklá Householderovými transformacemi:

$$B = \begin{bmatrix} b_{1,1} & b_{1,2} & & & \\ & b_{2,2} & b_{2,3} & & \\ & & b_{3,3} & b_{3,4} & \\ & & & b_{4,4} & b_{4,5} \\ & & & & b_{5,5} \end{bmatrix}.$$

■

```
1  % m,n dimenze matice
2  for j = 1:min(m,n)
3      % nulujeme sloupce zleva
4      x = A(j:m,j);
5      v = I*sign(x(1))*norm(x) + x;
6      nv=norm(v);
7      P = I - 2*v*v';
8      A(j:m,j:n) = P*A(j:m,j:n);
9      % nulujeme radky zprava
10     if j <= n-2
11         x  = A(j, j+1:n)';
12         v  = I*sign(x(1))*norm(x) + x;
13         nv = norm(v);
14         P  = I - 2*v*v';
15         A(j:m,j+1:n) = A(j:m,j+1:n)*P;
16     end
17 end
```

Výpis 1: Bidiagonalizace

Pro velké matice je tento algoritmus díky své řádové složitosti $O(n^3)$ značně pomalý. Z tohoto důvodu se jej pokusíme paralelizovat[3]. Pro paralelní algoritmus je vždy problémem komunikace mezi jednotlivými uzly. Tato generuje zpoždění, které zpomaluje celkový běh výsledného algoritmu. Z tohoto důvodu se vždy snažíme tuto komunikaci minimalizovat a zrychlit tak celkový běh programu.

Prohlédněme si nyní algoritmus 1 z pohledu paralelizace. Není těžké si povšimnout, že násobení matic na řádcích 8 a 15 bude vyžadovat značné množství komunikace napříč všemi uzly[4]. Tato komunikace by výslednou implementaci značně brzdila, proto je nutné ji omezit. Tohoto lze docílit pomocí přeuspořádání operací a správného rozložení matice mezi uzly.

[3] Veškerý kód v této práci je paralelizován pro počítače s distibuovanou pamětí. Z tohoto hlediska je předpokládána základní znalost paralelního programování.

[4] Matice bude rozdělena mezi jednotlivé uzly, které spolu budou muset komunikovat.

```
1  % vstupuje plná matice A,
2  % výstup je horní trojuhelníková matice v A
3  % a dolní trojuhelníková matice s koefienty L
4  for k = 1:(n-1)
5      % výpocet koeficientu
6      for i = (k+1):n
7          L(i,k) = A(i,k)/A(k,k)
8      end
9      % eliminace rádku
10     for j = (k+1):n
11         for i = (k+1):n
12             A(i,j) = A(i,j) - L(i,k)*A(k,j)
13         end
14     end
15 end
```

Výpis 2: Gaussova eliminace

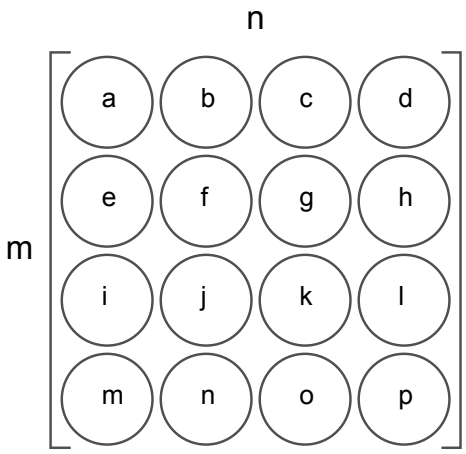
Jako pomůcka nám poslouží kód pro trojúhelníkový rozklad matice[5], jehož matlabovská implementace je na výpisu 2. Není těžké si všimnout, že oba algoritmy, jak bidiagonalizace tak trojúhelníkový rozklad, mají stejnou myšlenku. Zkusme ji tedy podrobněji popsat.

Algorithm 1 Bidiagonalizace

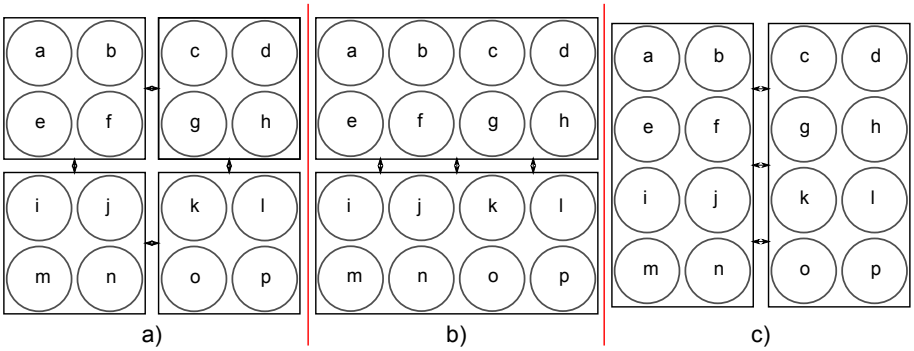
for $i = 1$ **to** $min(m, n)$ **do**
    provádí se nulování sloupců
    $\alpha = -sign(a_{ii})\sqrt{\sum_{k=i}^{m} a_{ki}^2}$ ▷ výpočet koeficientů
    $v_i = [0 \dots 0 a_{ii} \dots a_{mi}]^T - \alpha e_i$
    $\beta = v_i^T v_i$
    if $\beta = 0$ **then**
        continue to next i
    end if
    for $j = i$ **to** n **do** ▷ eliminace
        $\gamma = v_i^T a_j$ ▷ násobí se j-tý sloupec A
        $a_j = a_j - (2\gamma/\beta)v_i$ ▷ nuluje se j-tý sloupec A
    end for
    if $i < (n-2)$ **then**
        provádí se nulování řádku, operace jsou zcela symetrické k nulování sloupců
    end if
end for

Oba algoritmy eliminují prvky pod diagonálou. Rozdíl mezi nimi je v tom, že trojúhelníkový rozklad používá řádkové operace a bidiagonalizace používá ortogonální transformace pro postoupnou eliminaci sloupců. Při bližším zkoumání pak zjistíme, že lze

[5]Trojúhelníkový rozklad je definován později v textu na straně 24 případně v [3].



Obrázek 3: Rozložení matice.



Obrázek 4: Typy dělení fine-grain : (a) po blocích (b) po řádcích, (c) po sloupcích.

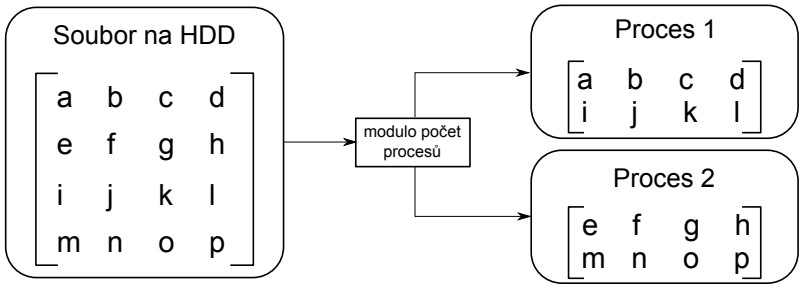
násobení ortogonální maticí chápat jako „řádkovou operaci“. Algoritmus rozdělíme na dvě fáze. První bude výpočet Householderova vektoru, který budeme brát jako výpočet eliminačních koeficentů. Druhá bude samotná eliminace sloupců pomocí ortogonální matice sestavené z vypočtených koeficientů. Na pseudokódu 3.1 je vidět, jakým způsobem se tyto úpravy provedou a není obtížné uvědomit si, že je lze využít ve prospěch paralelizace.

Pro efektivní paralelní algoritmus je důležité dobře rozdělit data na jednotlivé uzly. Na obrázku 3 je rozdělení typu fine-grain. Toto rozdělení předpokládá segmentaci na rovnoměrné bloky rozložené na jednotlivé uzly. Zbývá rozhodnout, jakým způsobem data rozdělíme. Existují tři možnosti, přičemž všechny jsou na obrázku 4. S ohledem na implementaci a náročnost na komunikaci byla v této práci zvolena distribuce po řádcích.

Popišme si nyní paralelní verzi algoritmu 3.1. Je patrné, že algoritmus 3.1 je velmi podobný své sekvenční verzi. Navíc byly přidány tři globální komunikace pro výměnu nezbytných dat. Podívejme se nyní na efektivnost této paralelní verze a na případná urychlení.

Algorithm 2 Bidiagonalizace

1: **for** $k = 1$ **to** $min(m, n)$ **do**
2: gatherAll(lokální kousky sloupců) $\rightarrow a$ ▷ sesbírání kousků vektorů
3: $a \rightarrow v$ ▷ sestavení Householderova vektoru
4: $beta = v^T v$
5: **for all** $j \in mycolumns$ **do**
6: $\gamma_j = v_i^T a_j$ ▷ násobí se j-tý sloupec A, ale pro lokální kousek
7: **end for**
8: reduceAll(γ_j) $\rightarrow \gamma_j$ ▷ sečtení všech γ výsledek na každý proces
9: **for** $j = k$ **to** n **do** ▷ eliminace
10: $a_j = a_j - (2\gamma_j/\beta)v_k$ ▷ nuluje se j-tý sloupec A
11: **end for**
12: **if** $i < (n - 2)$ **then**
13: broadcast(a) ▷ rozeslání aktuálního řádku
14: *provádí se nulování řádku, operace jsou zcela symetrické k nulování sloupců*
15: **end if**
16: **end for**



Obrázek 5: Distribuce řádků mezi procesy.

Počet procesů	Čas běhu (s)	Urychlení (%)
1	1365.3	0
2	1260.16	7
4	662.580	51.5
8	362.485	73.5
16	194.356	85.8
32	113.694	91.7
64	89.4507	93.4
128	128.537	90.6

Tabulka 1: Doba běhu algoritmu pro různé počty procesorů (úloha běžela na matici 4096×4096).

Algoritmus 3.1 pracuje s řádkovou distribucí dat a tedy je lehké si povšimnout, že pro sekvenční rozdělení řádků (viz obrázek 5) bude výkon na jednotlivé procesy rozložen nerovnoměrně. Z tohoto důvodu rozložíme řádky cyklickým způsobem, jak je naznačeno na obrázku 5. Tímto docílíme rovnoměrného rozložení zátěže mezi zúčastněné procesy. Na obrázku 6 je diagram běhu algoritmu pro 4 procesy s vyznačenou komunikací.

Zbývá ukázat, jaká je výpočetní složitost paralelní verze oproti sekvenční verzi. Sekvenční verze algoritmu má řádovou složitost $O(n^3)$, přičemž počet násobení při výpočtu Householderova vektoru je přibližně $2 \cdot n^2$ a počet operací násobení a sčítání při eliminační fázi je přibližně $4 \cdot n^3$. Paralelní verze algoritmu má celkovou časovou náročnost

$$T_p = t_c \frac{n^3}{3p} + 2t_s n + t_w \frac{n^2}{\sqrt{p}}, \tag{16}$$

kde t_c, t_s a t_w jsou postupně časy pro výpočet, odeslání a čekání na zprávu. n je dimenze matice a p je počet procesů.

$$E(n,p) = 1 + \frac{t_s}{t_c}\frac{6}{n^2} + \frac{t_w}{t_c}\frac{3}{\sqrt{p}} \tag{17}$$

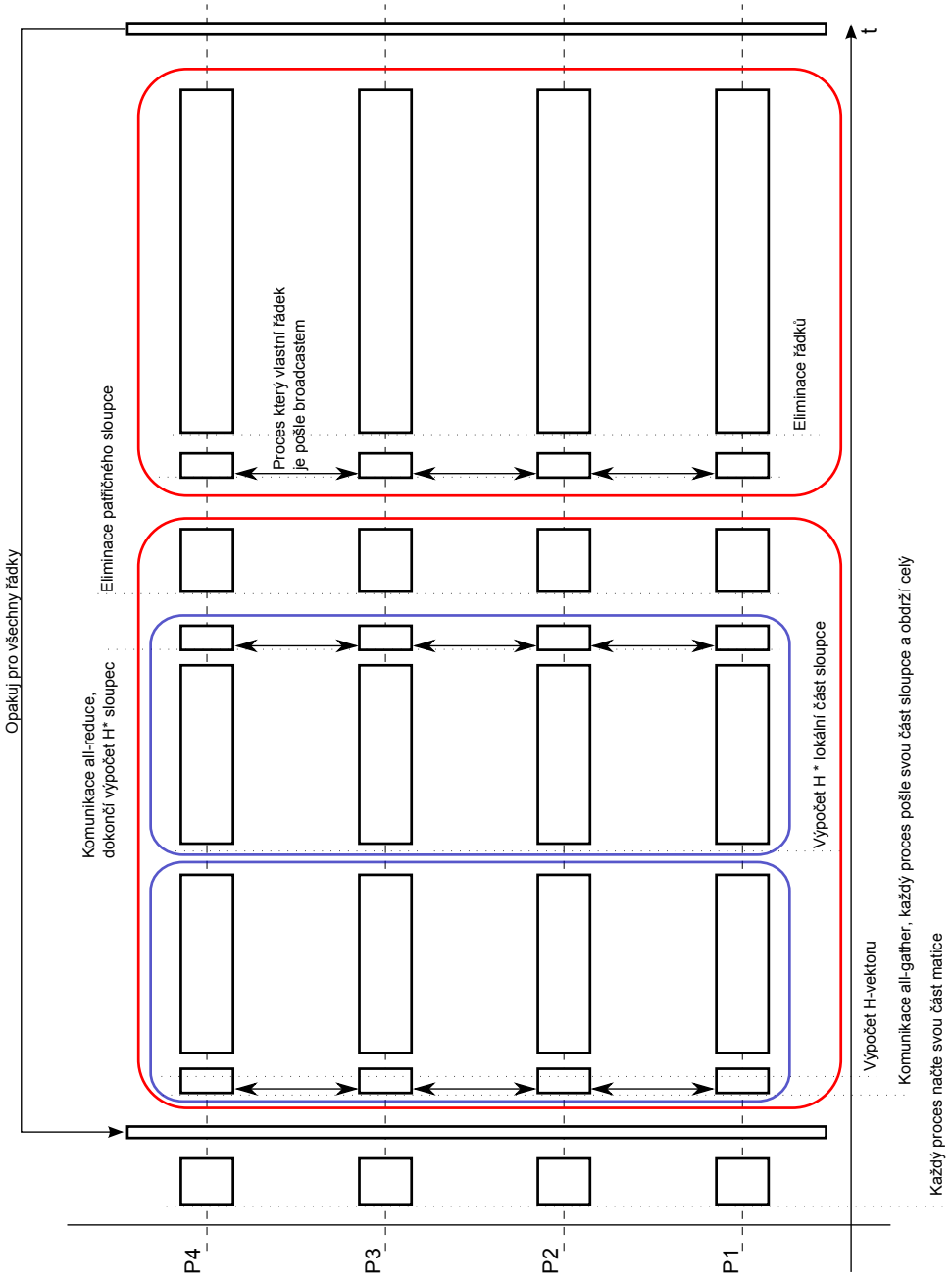
3.2 Výpočet singulárních čísel

Máme-li k dispozici bidiagonální matici můžeme přistoupit k výpočtu singulárních čísel původní matice A.

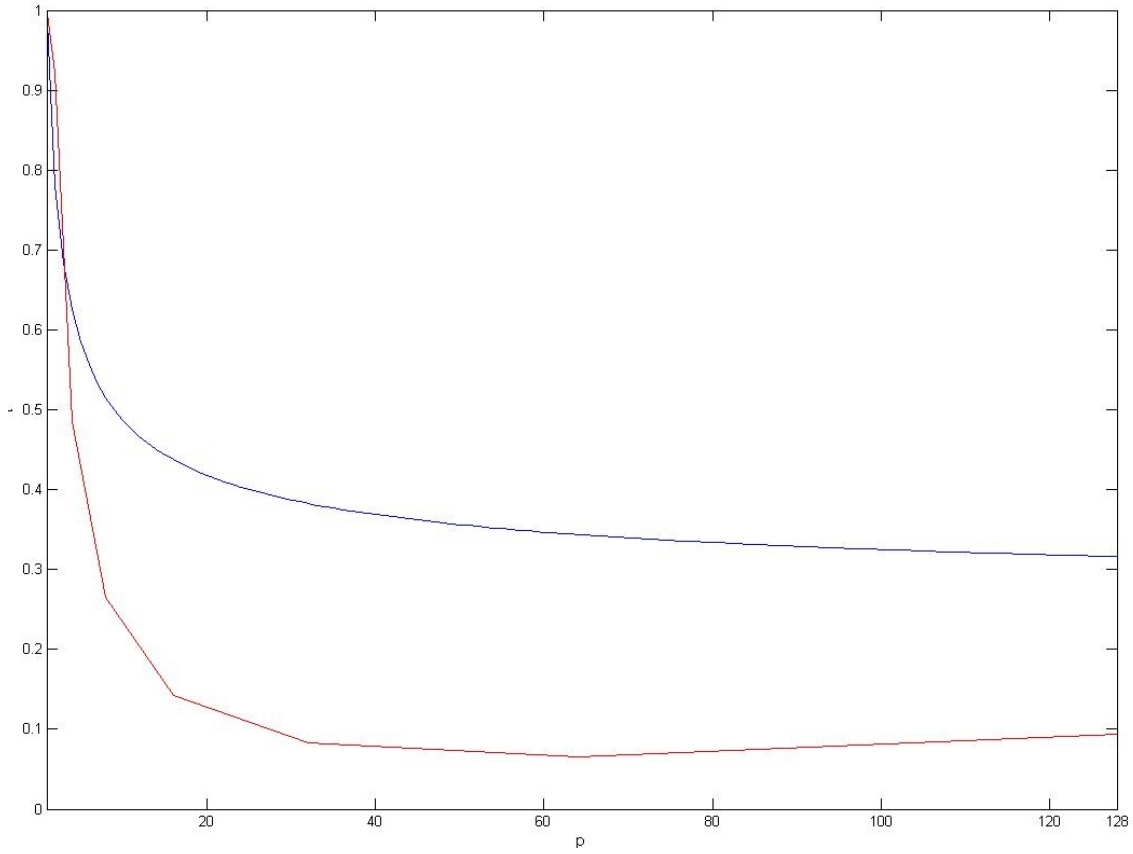
Věta 3.2 *Nechť bidiagonální matice B vznikla ortogonální transformací z matice A, tj. $B = QAQ^T$. Pak matice B má stejná singulární čísla jako matice A.*

Důkaz. Z věty 2.2 víme, že existuje singulární rozklad $A = U_A \Sigma_A V_A^H, A \in \mathbb{C}^{m \times n}$ a rozklad $B = U_B \Sigma_B V_B^H, B \in \mathbb{C}^{r \times r}, r = \min(m,n)$. Dále platí $B = Q_U A Q_V^H$, kde Q_U a Q_V jsou unitární. Pro $A^H A \in \mathbb{C}^{m \times m}$, platí

$$A^H A = (Q_V^H B^H Q_U)(Q_U^H B Q_V) = Q_V^H B^H B Q_V, \tag{18}$$



Obrázek 6: Diagram pro komunikaci mezi jednotlivými procesy.



Obrázek 7: Graf paralelní škálovatelnosti paralelní bidiagonalizace.

našli jsme tedy zobrazení transformující matici $A^H A$ na matici $B^H B$. Připomeňme, že matice X a Y, pro které platí

$$X = Q^{-1} Y Q,$$

se nazývají podobné a je o nich známo, že mají stejná vlastní čísla. Z (18) platí $B^H B = Q_V A^H A Q_V^H$ a protože Q_V je unitární platí $Q_V^{-1} = Q_V^H$, z toho vyplývá, že matice $A^H A$ a $B^H B$ jsou podobné a tudíž mají stejná vlastní čísla. Zbývá dokázat souvislost mezi vlastními a singulárními čísly. Tato je přímým důsledkem vztahu

$$A^H A = (U_A \Sigma_A V_A^H)^H (U_A \Sigma_A V_A^H) = V_A \Sigma_A^H U_A^H U_A \Sigma_A V_A^H = V_A \Sigma_A^H \Sigma_A V_A^H = V_A \Lambda_A V_A^H.$$

Analogicky platí pro B

$$B^H B = (U_B \Sigma_B V_B^H)^H (U_B \Sigma_B V_B^H) = V_B \Sigma_B^H U_B^H U_B \Sigma_B V_B^H = V_B \Sigma_B^H \Sigma_B V_B^H = V_B \Lambda_B V_B^H.$$

Z poznatku první části důkazu vyplývá

$$\Lambda_A = \Lambda_B \Rightarrow \Sigma_A = \Sigma_B.$$

c.b.d. ∎

Díky tomuto faktu můžeme nalézt singulární čísla matice B a zároveň tak vypočíst singulární čísla matice A. Budeme se tedy snažit redukovat B do diagonální formy. V této práci byla zvolena diagonalizace pomocí implicitního *QR* bez posunu spektra.

3.2.1 Implicitní *QR*

Tato metoda umožnuje aplikovat ortogonální transformace přímo na matici B bez nutnosti sestavovat kompletní *QR* viz. [1]. Mějme singulární rozklad matice $B = U \Sigma V^H$. Protože matice U a V jsou unitární, můžeme provést zpětnou transformaci

$$\Sigma = U^H B V. \tag{19}$$

Díky tomuto můžeme použít podobnou metodu jako u bidiagonalizace. Rozdíl ovšem je v druhu transformace, Householderovo zrcadlení nahradíme Givensovou rotací. Givensova rotace je rovinná transformace, která provádí otočení kolem jedné z hlavních os. Pro dvojrozměrný případ je Givensova rotace definována následovně.

Definice 3.2 *Nechť φ je úhel, o který chceme rotovat vektor $\mathbf{x} \in \mathbb{C}^2$ kolem počátku souřadné soustavy. Pak matici*

$$G(\varphi) = \begin{bmatrix} cos(\varphi) & sin(\varphi) \\ -sin(\varphi) & cos(\varphi) \end{bmatrix}, \tag{20}$$

nazýváme Givensovou maticí rovinné rotace.

Prvky matice $G(\varphi)$ můžeme explicitně vyjádřit. Pro implicitní QR požadujeme, aby platila rovnost

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

musí platit podmínky $-sa + cb = 0,\ c^2 + s^2 = 1$. Z tohoto pak přímo vyjádříme

$$c = \frac{a}{\sqrt{a^2 + b^2}}, \tag{21}$$

$$s = \frac{b}{\sqrt{a^2 + b^2}}. \tag{22}$$

Obecná matice Givensovy rotace, která nuluje prvek vektoru na pozici j pomocí hodnoty prvku na pozici $i,\ i < j$, vypadá následovně

$$G_{ij}(\varphi) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & [\cos(\varphi)]_{i,i} & \dots & [\sin(\varphi)]_{i,j} & & \\ & & \vdots & \ddots & \vdots & & \\ & & [-\sin(\varphi)]_{j,i} & \dots & [\cos(\varphi)]_{j,j} & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{bmatrix}. \tag{23}$$

Budeme tedy chtít pomocí (23) postupně vynulovat všechny prvky mimo hlavní diagonálu matice B jak je naznačeno na následující ilustraci, kde $G_1 = G_{12}(\varphi_1)$, $G_2 = G_{12}(\varphi_2)$, $G_3 = G_{22}(\varphi_3)$, $G_4 = G_{23}(\varphi_4)$, $G_5 = G_{34}(\varphi_5)$ a $G_6 = G_{34}(\varphi_6)$.

$$BG_1 = \begin{bmatrix} x & 0 & & \\ + & x & x & \\ & & x & x \\ & & & x \end{bmatrix}, \qquad G_2BG_1 = \begin{bmatrix} x & x & + & \\ 0 & x & x & \\ & & x & x \\ & & & x \end{bmatrix},$$

$$G_2BG_1G_3 = \begin{bmatrix} x & x & 0 & \\ & x & x & \\ & + & x & x \\ & & & x \end{bmatrix}, \qquad G_4G_2BG_1G_3 = \begin{bmatrix} x & x & & \\ & x & x & + \\ & 0 & x & x \\ & & & x \end{bmatrix},$$

$$G_4G_2BG_1G_3G_5 = \begin{bmatrix} x & x & & \\ & x & x & 0 \\ & & x & x \\ & & + & x \end{bmatrix}, \quad G_6G_4G_2BG_1G_3G_5 = \begin{bmatrix} x & x & & \\ & x & x & \\ & & x & x \\ & & 0 & x \end{bmatrix}. \tag{24}$$

Zbývá tedy rozhodnout o úhlu φ_1 tak, aby vnesl nulu do první rotace $G_{12}(\varphi_1)$, tohoto docílíme volbou

$$\tan\varphi_1 = \frac{(B^TB)_{12}}{\sigma^2 - (B^TB)_{11}}, \tag{25}$$

kde σ je nejmenší vlastní číslo submatice o rozměrech 2×2, umístěné na konci matice B^TB („Wilkinsonovo posunutí“).

Jak bylo řečeno výše, v této práci byl zvolen nulový posun spektra, vypustíme tedy σ ze vztahu (25) a dostaneme

$$\tan\varphi_1 = \frac{-b_{12}}{b_{11}}.$$

Potom aplikace první rotace na původní matici B vypadá následovně

$$B^{(1)} = BG_1 = \begin{bmatrix} b_{11}^{(1)} & 0 & & \\ b_{21}^{(1)} & b_{22}^{(1)} & b_{23} & \\ & & b_{33} & b_{34} \\ & & & b_{44} \end{bmatrix},$$

kde horní index u B označuje, že na ni byla aplikována matice G_1. Porovnáme-li tento výsledek s první rotací při posunu spektra, vidíme, že se na superdiagonále objevila nula navíc. Tato nula pak prochází celým procesem a je klíčem k jeho efektivitě. Po aplikaci G_2 vypadá B takto

$$B^{(2)} = G_2BG_1 = \begin{bmatrix} b_{11}^{(2)} & b_{12}^{(2)} & b_{13}^{(2)} & \\ 0 & b_{22}^{(2)} & b_{23}^{(2)} & \\ & & b_{33} & b_{34} \\ & & & b_{44} \end{bmatrix}.$$

Je dobré si uvědomit, že

$$\begin{bmatrix} b_{12}^{(2)} & b_{13}^{(2)} \\ b_{22}^{(2)} & b_{23}^{(2)} \end{bmatrix} = \begin{bmatrix} \sin(\varphi_2)b_{22}^{(1)} & \sin(\varphi_2)b_{23} \\ \cos(\varphi_2)b_{22}^{(1)} & \cos(\varphi_2)b_{23} \end{bmatrix},$$

protože tato matice má řádkovou hodnost rovnu 1, přenásobení maticí G_3 zprava proto vynuluje nejen prvek $(1,3)$, ale i prvek $(2,3)$

$$B^{(3)} = G_2BG_1G_3 = \begin{bmatrix} b_{11}^{(2)} & b_{12}^{(3)} & 0 & \\ 0 & b_{22}^{(3)} & 0 & \\ & b_{32}^{(3)} & b_{33}^{(3)} & b_{34} \\ & & & b_{44} \end{bmatrix}.$$

Při porovnání s druhou rotací z (24), zjistíme, že na superdiagonále je jedna nula navíc. Při rotaci G_4 se opakuje stejná situace jako při rotaci G_2. Implementace pro Matlab je na výpisu 3.

```
1  % B -> bidiagonalni matice
2  % s <- singularni cisla matice B
3  while norm(e,'inf') >= tol*ref
4      % aplikace algoritmy "QR step"
5      cs = 1; sn = 0;
6      oldcs = 1; oldsn = 1;
7      for i = 1 : n-1
8          % givensova rotace zprava
9          ta = s(i) * cs;
10         [cs, sn, r] = givens( s(i) * cs, e(i) );
11         if i ~= 1
12           e(i-1) = oldsn * r;
13         end
14         % givensova rotace zleva
15         ta = oldcs * r; tb = s(i+1) * sn;
16         [oldcs, oldsn, s(i)] = givens( oldcs * r, s(i+1) * sn );
17     end
18     h = s(n) * cs;
19     e(n-1) = h * oldsn;
20     s(n) = h * oldcs;
21 end
```

Výpis 3: Implicitní QR s nulovým posunem spektra

Pro správnou funkčnost algoritmu 3 je třeba zvolit konvergenční kritéria, která urychlí samotný proces nulování mimodiagonálních prvků. Základní myšlenka je vynulovat všechny dostatečně malé prvky. Zbývá tedy rozhodnout, co jsou dostatečně malé prvky, které můžeme zanedbat a které ne. Tohoto bylo dosaženo volbou následujících podmínek

$$s_i + s_{i+1} + e_i = s_i + s_{i+1}, \quad i = 1, 2, \ldots, n \tag{26}$$

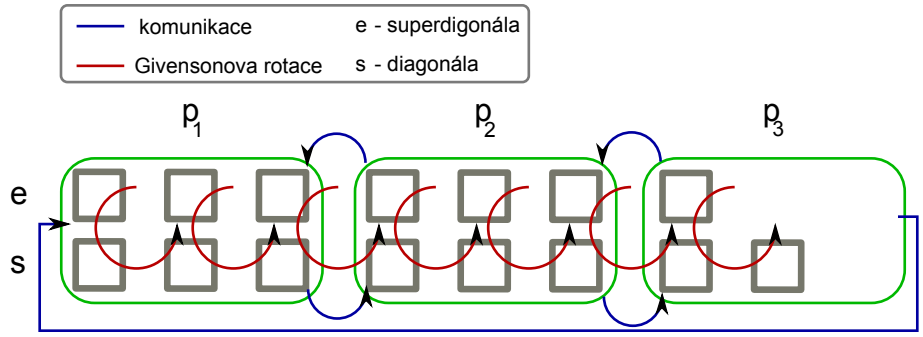
$$e_i + e_{i+1} + s_i = e_i + e_{i+1}, \quad i = 1, 2, \ldots, n-1 \tag{27}$$

kde $s_i, \forall i$ (resp.$e_i, \forall i$) je diagonála (resp. superdiagonála) matice B. Je-li splněna podmínka (26), můžeme prohlásit prvek e_i za dostatečně blízký epsilon[6] a tedy jej vynulovat. Podmínka (27) funguje stejně, ale slouží pro kontrolu s_i.

3.2.2 Paralelizace impicitního QR

Nyní, když známe implicitní QR můžeme diskutovat možnosti jeho paralelizace. Diagonalizace pomocí QR je velmi obtížně paralelizovatelná úloha, protože není možné zvolit takovou dekompozici dat, která by minimalizovala komunikaci. Pro názornost si na obrázku 8 ilustrujeme množství potřebné komunikace. Na obrázku je ukázáno rozdělení na bloky. Každý proces dostane část matice, na které bude provádět diagonalizaci. Jak je vidět, každý proces potřebuje pro svou práci výsledek předchozího procesu. Stejně tak potřebuje pro poslední krok Givensovy rotace data z následujícího procesu. Je zřejmé, že pomocí paralelizace by k urychlení nedošlo, proto se paralelizací implicitního QR nebudeme zabývat.

[6] epsilon - počítačová nula, tedy nejmenší možné číslo, které je počítač schopný rozlišit.



Obrázek 8: Komunikaci při paralelním *QR* algoritmu.

3.3 Výpočet singulárních vektorů

Pouze znalost singulárních čísel není dostatečná. Pro některé aplikace potřebujeme znát jak singulární čísla, tak singulární vektory. Ukažme nyní metodu, jak vypočítat všechny singulární vektory, známe-li všechna singulární čísla. Abychom mohli následující větu dokázat, musíme říct, co je spektrální rozklad matice.

Věta 3.3 *Nechť $A \in \mathbb{C}^{n\times n}$ je symetrická matice. Pak existují právě jedna unitární matice $U \in \mathbb{C}^{n\times n}$ a právě jedna diagonální matice $\Lambda \in \mathbb{R}^{n\times n}$ tak, že platí*

$$A = U\Lambda U^H. \tag{28}$$

Platí $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$ kde $\lambda_i, i = 1, \ldots, n$ jsou vlastní čísla matice A. Navíc platí

$$AU_i^r = \lambda_i U_i^r \text{ pro } i = 1, \ldots, n,$$

kde U_i^r je i-tý řádek matice U.

Důkaz. Důkaz lze najít v [3]. ∎

Věta 3.4 *Nechť $T_U \in \mathbb{C}^{n\times n}$ (resp. $T_V \in \mathbb{C}^{n\times n}$) je komplexní třídiagonální matice pro kterou platí $T_U = BB^H$ (resp. $T_V = B^HB$). $B \in \mathbb{C}^{n\times n}$ je bidiagonální matice získaná jako $B = U_B^H A V_B$, kde $A \in \mathbb{C}^{m\times n}$ a $U_B \in \mathbb{C}^{m\times m}$, $V_B \in \mathbb{C}^{n\times n}$ jsou unitární matice. Dále ať $\Lambda = \Sigma^2 \in \mathbb{R}^n$ jsou vlastní čísla matice T. Pak pro singulární vektory $u_i, v_i,\ i = 1 \ldots n$, matice B platí*

$$(T_U - I\lambda_i)u_i = 0, \tag{29}$$
$$(T_V - I\lambda_i)v_i = 0, \tag{30}$$

kde $i = 1 \ldots n$.

Důkaz. Větu dokážeme pro $(T_U - I\lambda_i)u_i = 0$, pro druhý případ je důkaz zcela analogický. Na základě věty 2.2 víme, že existuje singulární rozklad $B = U_B\Sigma_B V_B^H$ a podle věty 3.3 existuje spektrální rozklad $T_U = U_T\Lambda_T U_T^H$. Matice T_U je definována jako

$$T_U = BB^H = (U_B\Sigma_B V_B^H)(U_B\Sigma_B V_B^H)^H = U_B\Sigma_B V_B^H V_B\Sigma_B^H U_B^H = U_B\Sigma_B\Sigma_B^H U_B^H = U_B\Lambda_B U_B^H.$$

Platí tedy

$$T_U = U_T \Lambda_T U_T^H = U_B \Lambda_B U_B^H.$$

Dle věty 3.3 exituje právě jediný spektrální rozklad, musí tedy platit také rovnost $U_T = U_B$. c.b.d. ∎

Díky větě 3.4 můžeme spočítat pravé i levé singulární vektory. Problém ovšem nastává při počítačové implementaci, při které nikdy nedosáhneme přesné nuly. Jinými slovy soustava rovnic (29) (resp. (30)) je v přesné aritmetice singulární, ale z důvodu zaokrouhlovací chyby na počítači se tato soustava může stát nesingulární, důsledkem čehož nebudou výsledné vektory ortogonální.

Z tohoto faktu vyplývá, že pro počítačovou implementaci musíme vymyslet způsob, který bude numericky stabilnější.

K tomu nám poslouží tzv. *LU* rozklad.

Věta 3.5 *Nechť $A \in \mathbb{C}^{n\times n}$ je regulární. Pak existuje dolní trojúhelníková matice L, horní trojúhelníková matice U a permutační matice P tak, že platí*

$$AP = LU. \tag{31}$$

Důkaz. Důkaz lze najít v [2]. ∎

Příklad 3.3
LU rozklad matice $A \in \mathbb{R}^{3\times 3}$ je

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & a_{33} \end{bmatrix}.$$

∎

O *LU* rozkladu je obecně známo, že jej lze získat jak pro matice regulární tak pro matice singulární. Otázkou zůstává jednoznačnost. Ukazuje se, že pokud budeme volit *LU* faktorizaci jako na příkladu 3.3, bude existovat pouze jeden unikátní rozklad matice A. Podrobné vysvětlení a důkaz lze najít třeba v [3].

Budeme-li znát *LU* rozklad matice A reprezentující soustavu rovnic (32), můžeme pomocí tohoto rozkladu danou soustavu efektivně vyřešit. Tomuto procesu se říká dopředná a zpětná substituce a provádí se takto

$$\begin{aligned} Ax &= b, && (32)\\ (LU)x &= b, \rightarrow L(Ux) = b, && (33)\\ Ly &= b, Ux = y. && (34) \end{aligned}$$

Matici soustavy rozložíme na součin matic L a U a přeuspořádáme pořadí násobení jak je vidět na rovnici (33). Nyní můžeme Ux nahradit y a vyřešit soustavu $Ly = b$

(dopředná substituce). Vypočtené y pak dosadíme do rovnice $Ux = y$ (zpětná substituce) – viz rovnice (34). Tímto jsme tedy získali řešení x původní soustavy (32). Bližší vysvětlení lze najít v [3].

Vzpomeňme, že matice soustavy (29) (resp. (30)) je singulární a třídiagonální. O *LU* víme, že ho lze najít i pro tyto matice. Popišme tedy jakým způsobem lze tento rozklad vypočítat a využít k stabilizaci a řešení těchto soustav.

Nalézt *LU* dekompozici třídiagonální matice je výpočetně jednoduchý proces. Stačí si uvědomit, jakým způsobem funguje algoritmus na výpisu 2 a využít struktury třídiagonální matice, jejíž *LU* rozklad je naznačen na rovnici (35).

$$\begin{bmatrix} T_{1,1} & T_{1,2} & 0 & 0 \\ T_{1,2} & T_{2,2} & T_{2,3} & 0 \\ 0 & T_{2,3} & T_{3,3} & T_{3,4} \\ 0 & 0 & T_{3,4} & T_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ L_{2,1} & 1 & 0 & 0 \\ 0 & L_{3,2} & 1 & 0 \\ 0 & 0 & L_{4,3} & 1 \end{bmatrix} \cdot \begin{bmatrix} U_{1,1} & U_{1,2} & 0 & 0 \\ 0 & U_{2,2} & U_{2,3} & 0 \\ 0 & 0 & U_{3,3} & U_{3,4} \\ 0 & 0 & 0 & U_{4,4} \end{bmatrix}. \tag{35}$$

Je vidět, že dekompozice zachovává strukturu a tudíž výsledkem je horní a dolní bidiagonála. Faktorizace třídiagonální matice je „algoritmicky levná záležitost“, protože nemusíme počítat prvky pod subdiagonálou a nad superdiagonálou. Tyto prvky jsou nulové a díky zachování struktury nulové zůstanou. Můžeme je tedy při výpočtu vynechat a zmenšit tak potřebný počet operací.

Zůstává tedy problém se singularitou v matici soustavy. Existence *LU* pro singulární matice je diskutována v [7]. Zde si kvůli stručnosti pouze ukážeme praktické řešení a implementaci.

Singularita v matici T se při faktorizaci projeví na diagonále matice U jako číslo blízké ϵ. V takovém případě řešíme rovnici

$$tx = r,$$

pro parametry $t \in \mathbb{R}$ a $r \in \mathbb{R}$, který zvolíme roven 1. Tím docílíme eliminace problematického řádku z matice a umožníme dokončení rozkladu.

Výsledný algoritmus je na výpisu 4. Ten už nemá složitost $O(n^3)$, ale pouze $O(n)$.

```
1  for k = 1:(n−1)
2      if T(k,k) < eps
3          T(k,k) = 1
4      end
5      % výpocet koeficientu
6      T(k+1,k) = T(k,k+1)/T(k,k)
7      % eliminace rádku
8      T(k+1,k+1) = T(k+1,k+1) − L(k+1,k)∗T(k,k+1)
9  end
```

Výpis 4: LU rozklad třídiagonální matice

Protože tento algoritmus potřebuje na vstupu třídiagonální matici a nevypočítá posunutí spektra, budeme jej dále modifikovat.

Pro názornost rozepíšeme do maticové podoby soustavy rovnic (29) a (30). Z nichž snadno vysledujeme pravidla, podle nichž jsou tyto matice vytvořeny.

Příklad 3.4
Matice soustavy rovnic (29).

$$\begin{bmatrix} b_{1,1}^2 - \lambda_i & b_{1,1} \cdot b_{1,2} & & & & \\ b_{1,1} \cdot b_{1,2} & (b_{2,2}^2 + b_{1,2}^2) - \lambda_i & b_{2,2} \cdot b_{2,3} & & & \\ & b_{2,2} \cdot b_{2,3} & (b_{3,3}^2 + b_{2,3}^2) - \lambda_i & b_{3,3} \cdot b_{3,4} & & \\ & & b_{3,3} \cdot b_{3,4} & (b_{4,4}^2 + b_{3,4}^2) - \lambda_i & b_{4,4} \cdot b_{4,5} & \\ & & & b_{4,4} \cdot b_{4,5} & (b_{5,5}^2 + b_{4,5}^2) - \lambda_i \end{bmatrix} \tag{36}$$

Matice soustavy rovnic (30).

$$\begin{bmatrix} (b_{1,1}^2 + b_{1,2}^2) - \lambda_i & b_{1,2} \cdot b_{2,2} & & & & \\ b_{1,2} \cdot b_{2,2} & (b_{2,2}^2 + b_{2,3}^2) - \lambda_i & b_{2,3} \cdot b_{3,3} & & & \\ & b_{2,3} \cdot b_{3,3} & (b_{3,3}^2 + b_{3,4}^2) - \lambda_i & b_{3,4} \cdot b_{4,4} & & \\ & & b_{3,4} \cdot b_{4,4} & (b_{4,4}^2 + b_{4,5}^2) - \lambda_i & b_{4,5} \cdot b_{5,5} & \\ & & & b_{4,5} \cdot b_{5,5} & b_{5,5}^2 - \lambda_i \end{bmatrix} \tag{37}$$

■

Při bližším zkoumání těchto matic si lze povšimnout, že jejich diagonální a mimodiagonální prvky lze vyjádřit předpisem. Díky tomuto poznatku můžeme upravit algoritmus 4 tak, že nebude potřebovat na vstupu třídiagonální matici a zároveň vypočítá posunutí spektra.

Vztahy, které platí pro diagonální prvky jsou v následujících rovnicích.

$$T_{1,1}^U = b_{1,1}^2 - \lambda_i \tag{38}$$

$$T_{j,j}^U = b_{j,j}^2 + b_{j-1,j}^2 - \lambda_i, \quad j = 2, 3, \ldots, n \tag{39}$$

$$T_{j,j}^V = b_{j,j}^2 + b_{j,j+1}^2 - \lambda_i, \quad j = 1, 2, \ldots, n-1 \tag{40}$$

$$T_{n,n}^V = b_{n,n}^2 - \lambda_i \tag{41}$$

Superdiagonála a subdiagonála jsou stejné. Pro algoritmus tedy stačí vyjádřit superdiagonálu.

$$T_{j,j+1}^U = b_{j,j} \cdot b_{j,j+1}, \quad j = 1, 2, \ldots, n \tag{42}$$

$$T_{j,j+1}^V = b_{j,j+1} \cdot b_{j+1,j+1}, \quad j = 1, 2, \ldots, n \tag{43}$$

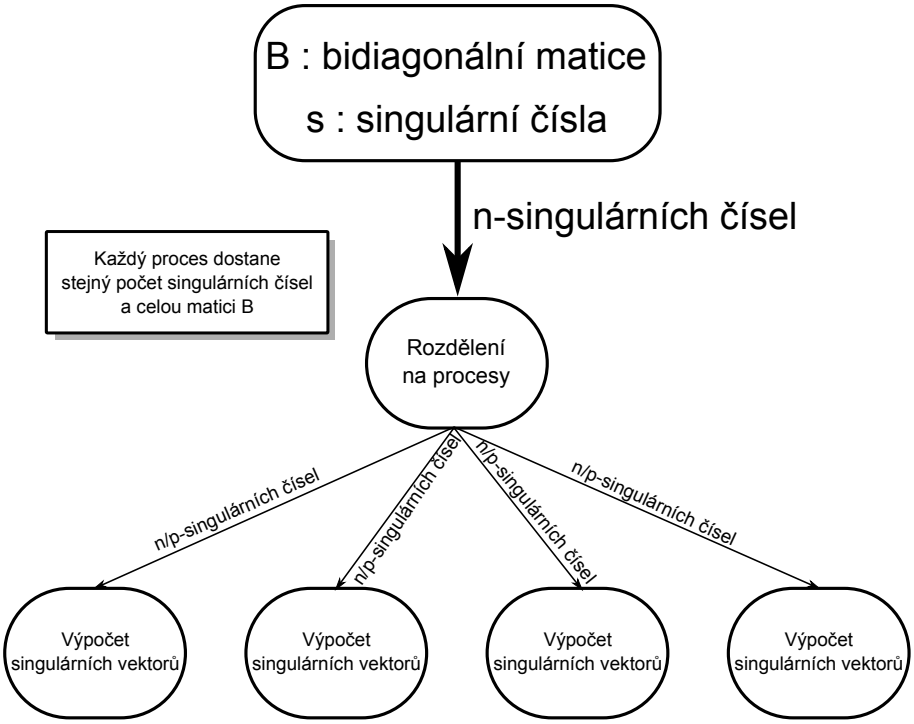
Dosadíme-li předchozí rovnice do algoritmu 4, dostaneme algoritmus 5. Výsledkem jsou *LU* rozklady matic (36) a (37), ze kterých dále pomocí zpětné a dopředné substituce získame řešení soustav (29) a (30), tedy příslušné singulární vektory.

```
1  Up(1,1) = a(1)^2−s;
2  Up(1,2) = a(1)∗b(1);
3
4  Um(1,1) = a(1)^2 + b(1)^2 −s;
5  Um(1,2) = a(2)∗b(1);
6
7  for k = 1:(n−2)
8      if abs(Up(k,k)) < myEps
9          Up(k,k) = 1.0;
10     end
11     if abs(Um(k,k)) < myEps
12         Um(k,k) = 1.0;
13     end
14     Lp(k+1,k) = (a(k)∗b(k))/Up(k,k);
15     Lm(k+1,k) = (a(k+1)∗b(k))/Um(k,k);
16
17     Up(k+1,k+1) = (a(k+1)^2 + b(k)^2 − s) − Lp(k+1,k)∗Up(k,k+1);
18     Um(k+1,k+1) = (a(k+1)^2 + b(k+1)^2 − s) − Lm(k+1,k)∗Um(k,k+1);
19
20     Up(k+1,k+2) = a(k+1)∗b(k+1);
21     Um(k+1,k+2) = a(k+2)∗b(k+1);
22 end
23
24 if abs(Up(n−1,n−1)) < myEps
25     Up(n,n) = 1.0;
26 end
27 if abs(Um(n−1,n−1)) < myEps
28     Um(n,n) = 1.0;
29 end
30
31 Lp(n,n−1) = (a(n−1)∗b(n−1))/Up(n−1,n−1);
32 Lm(n,n−1) = (a(n)∗b(n−1))/Um(n−1,n−1);
33 Up(n,n) = (a(n)^2 + b(n−1)^2 − s) − Lp(n,n−1)∗Up(n−1,n);
34 Um(n,n) = (a(n)^2 − s) − Lm(n,n−1)∗Um(n−1,n);
35
36 if abs(Up(n,n)) < myEps
37     Up(n,n) = 1.0;
38 end
39 if abs(Um(n,n)) < myEps
40     Um(n,n) = 1.0;
41 end
```

Výpis 5: LU s posunutím spektra na bidiagonální matici

3.3.1 Paralelizace výpočtu singulárních vektorů

Postup uvedený v předchozí části patří mezi dobře paralelizovatelné úlohy. To znamená, že pro výpočet nepotřebuje kromě inicializace žádnou komunikaci napříč procesy, a tudíž je dobře škálovatelný. Paralelní verze algoritmu je nejlépe ilustrována na obrázku 9.



Obrázek 9: Distribuce a paralelní výpočet levých a pravých singulárních vektorů.

Jak obrázek 9 naznačuje, každý účastnící se proces dostane svoji část singulárních čísel, pro která vypočítá příslušné singulární vektory. Protože paralelní algoritmus je prakticky totožný s algoritmem popsaným dříve, nebudeme jej zde uvádět.

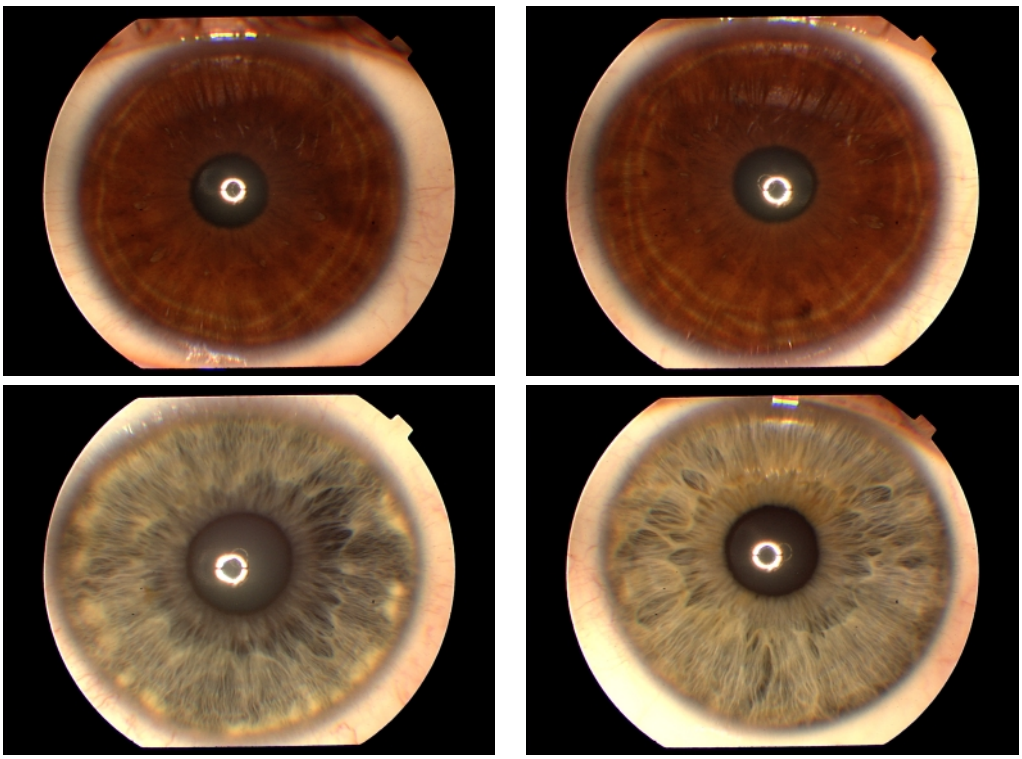
Řádová složitost sekvenční verze algoritmu je $O(n^2)$, počet operací je přibližne $6 \cdot n^2$. Paralelní verze má řádovou složitost $O(\frac{n^2}{p})$, kde p je počet procesů. Při úloze výpočtu singulárních vektorů bidiagonální matice o dimenzi 5000×5000 dokončil sekvenční algoritmus výpočet za 10.83 sekund. Paralelní verze na 8 procesorech výsledek vypočetla prakticky okamžitě.

4 Zpracování biometrických dat

V této části ukážeme, jakým způsobem aplikovat *SVD* na obrazová biometrická data, konkrétně duhovky. Protože není cílem této práce vytvořit funkční databázi duhovek, ale pouze aparát pro nalezení shody, nebudeme řešit konkrétní implementaci.

Celý proces analýzy a porovnání bude sestávat ze tří kroků. Lokalizace duhovky, kdy najdeme v daném snímku oblast, která nás zajímá a transformujeme ji do normalizovaného formátu. Předzpracování bude druhý krok a bude sloužit jak pro filtrování šumu a přebytečných informací, tak jako druhý stupeň normalizace dat. Poslední částí bude sestavení matice výskytu, se kterou jsme se seznámili v kapitole 2.2 a aplikace *SVD* na nalezení podobnosti mezi jednotlivými duhovkami.

Na obrázku 10 jsou vybrány 4 duhovky ze série využité v této práci.



Obrázek 10: Ukázka nasnímaných duhovek

4.1 Lokalizace duhovky

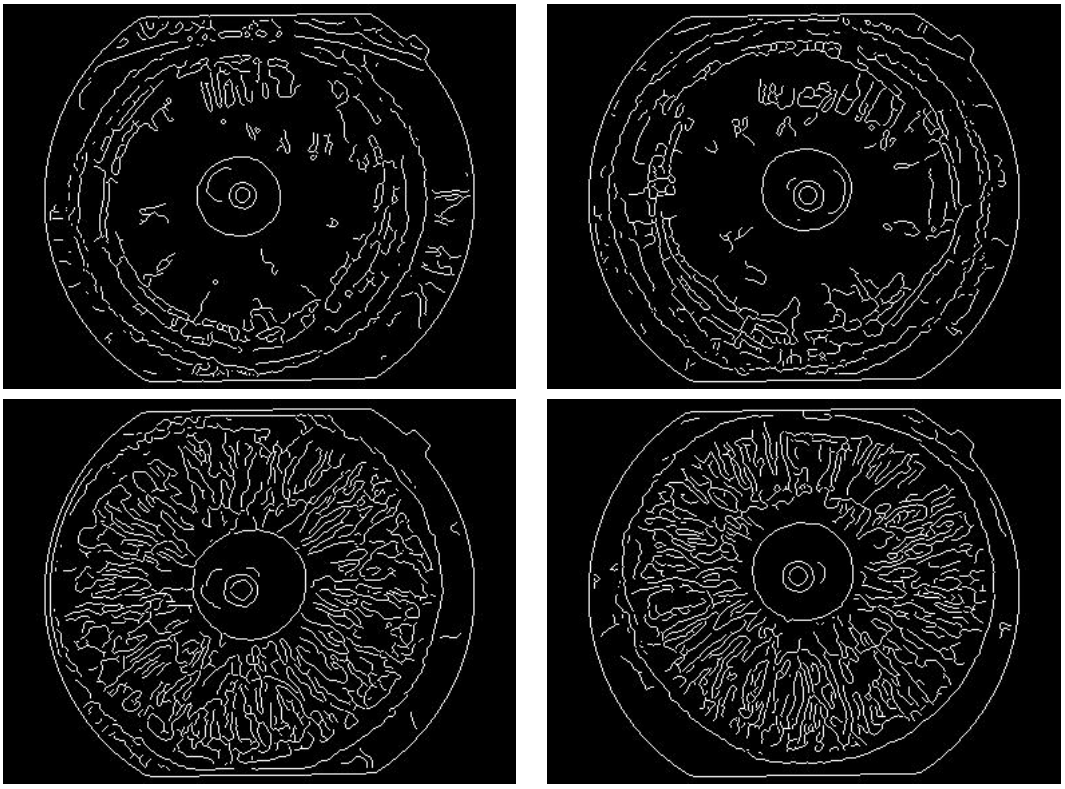
Lokalizace duhovky tvoří nejvíce problematickou část celého procesu. Toto je dáno faktem, že duhovka nemá na snímcích pevný rozměr ani polohu. Velký problém také představuje různá úroveň jasu, která ztěžuje korektní detekci hran[7].

[7] Hrana je označení ostré změny jasové funkce obrazu.

Jak již bylo naznačeno, prvním krokem lokalizace je detekce hran. Hrany se detekují pomocí konvolučních operátorů jako je třeba Prewittův operátor

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}.$$

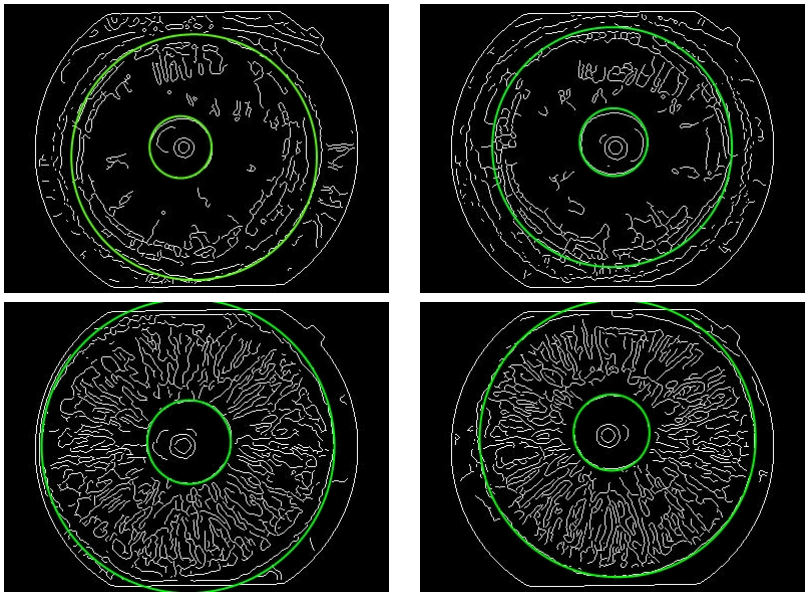
Podrobné vysvětlení jak funguje detekce hran lze najít v [15] nebo v [17]. V této práci byl zvolen, vzhledem k povaze dat, Cannyho detektor hran, který lze najít v [15]. Na obrázku 11 jsou nalezené hrany v duhovkách.



Obrázek 11: Nalezené hrany v duhovkách z obrázku 10

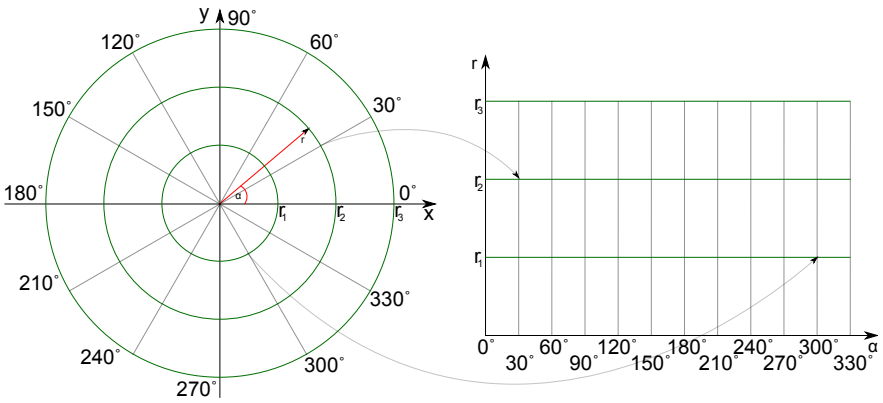
Druhým krokem je detekování kružnic v obraze a jejich vyfiltrování tak, že zůstanou pouze hranice duhovek. Toto je ilustrováno na obrázku 12, kde jsou ukázány již nalezené hranice duhovek.

Poté, co jsme nalezli hranice duhovky, je třeba provést preprocessing a normalizaci. Tento krok je důležitý, protože každá duhovka má zpravidla jinou velikost. Proto byla zvolena pružná polární transformace, která je ilustrována na obrázku 13. Duhovky jsou v kartézských souřadnicích a my je přepočítáme do polárních takovým způsobem, že pevně stanovíme rozměr obrázku v polárních souřadnicích a tento bude sloužit jako omezující faktor polární transformace. Tato transformace se také nazývá model gumového



Obrázek 12: Vnitřní a vnější okraje duhovek (pouze ilustrační obrázky).

pásu, a to proto, že vezmeme-li gumový pás spojený tak, že tvoří kruh a rozstřihneme jej, můžeme ho natáhnout do požadovaných rozměrů úplně stejně jako to dělá pružná polární transformace.

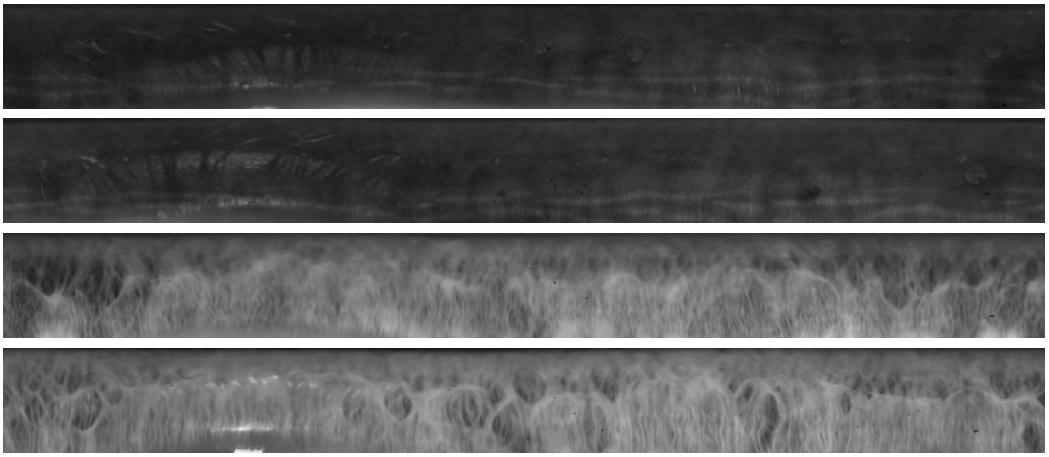


Obrázek 13: Převedení kartézských souřadnic do polárních souřadnic

Na obrázku 14 je ukázka transformovaných duhovek připravených k další fázi procesu.

4.2 Waveletový filtr

Pro přesnější porovnávání je třeba provést filtrování „rozbalené“ duhovky. Filtrem odstraníme přebytečný šum a vytvoříme otisku podobnou matrici duhovky, kterou můžeme



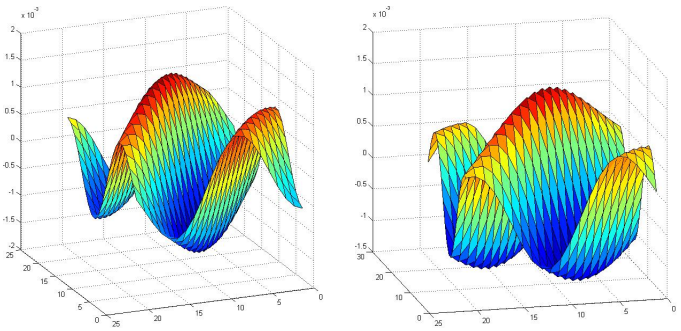
Obrázek 14: Transformované duhovky.

porovnávat vůči databázi. Rozsáhlý úvod do problematiky filtrů, waveletových bank a waveletové transformace lze najít třeba v [13], [14] nebo v [16].

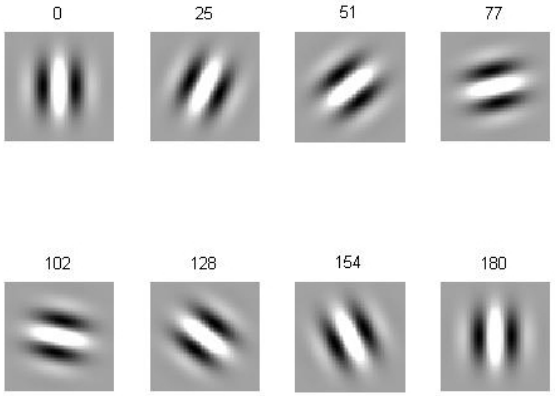
V této práci byl zvolen, pro své dobré vlastnosti při zpracování obrazových dat, Gaborův wavelet. Jedná se vlastně o okenní Fourierovu transformaci se speciálně zvoleným jádrem, které umožňuje škálovatelnost frekvencí i rozměru. Toto jádro je popsáno v následujícím vztahu,

$$G(x,y) = \frac{1}{2\pi\sigma\beta} e^{-\pi\left[\frac{(x-x_0)^2}{\sigma^2}\frac{(y-y_0)^2}{\beta^2}\right]} e^{i(\xi_0 x + \upsilon_0 y)}, \tag{44}$$

kde (x_0, y_0) je počátek soustavy, (ξ_0, υ_0) jsou optimální prostorové frekvence filtru ve frekvenční doméně, σ a β jsou výchylky eliptického Gausiánu podle x a y. Z (44) je vidět, že 2D Gaborův filtr je výsledkem součinu eliptického Gausiánu a rovinné vlny na komplexní množině. Na obrázku 15 je ukázka, jak vypadá reálná a imaginární část Gaborova waveletu.



Obrázek 15: Reálná a imaginární složka Gaborova filtru.



Obrázek 16: Banka Gaborových waveletů

Otázkou zůstává, jakým způsobem budeme filtr aplikovat pro nejlepší výsledek. Touto problematikou se zabývá řada prací pokrývajících široké spektrum aplikací, jako jsou otisky prstů, segmentace textur, případně detekce výrazů na tváři ([20], [21], [22], [23], [24]).

V této práci byl volen adaptivní filtr v kombinaci s waveletovou bankou. Tedy byla vytvořena banka (tato banka je zobrazena na obrázku 16) pro různé úhly otočení Gaborova waveletu.

Před filtrováním se transformovaná duhovka rozdělí na jednotlivé bloky, pro které se vypočítá gradient. Tento gradient pak bude určovat, který úhel Gaborova waveletu se zvolí tak, aby rozdíl byl minimální. Pro každý blok duhovky se pak provede konvoluce s waveletem

$$I_f = I * G_\phi, \tag{45}$$

kde I_f je filtrovaná část duhovky, I je vstupující blok duhovky a G_ϕ je Gaborův wavelet pro úhel ϕ.

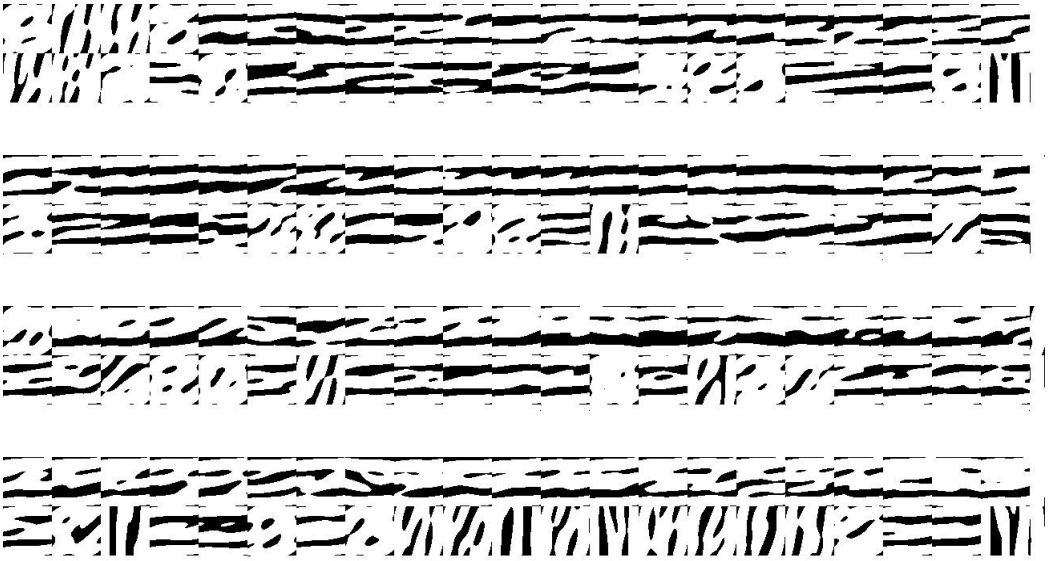
Duhovky po waveletové transformaci jsou na obrázku 17.

4.3 Vyhledávání

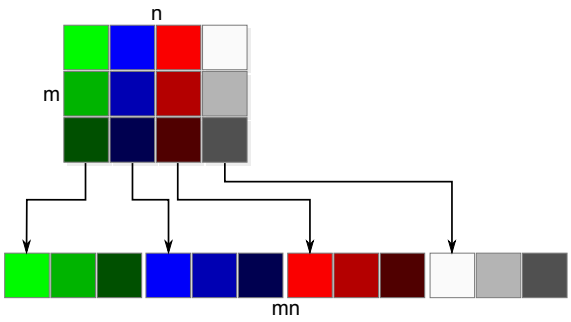
V této části si ukážeme, jak sestavit matici D (představenou v kapitole 2.2) z připravených obrázků. Dále tuto matici rozložíme na singulární rozklad, který použijeme k nalezení shody. Tato metoda je pro obrazová data představena v [36] a [37].

Na jednotlivé duhovky budeme nahlížet jako na samostatné dokumenty, které dosadíme jako řádky matice D. Tyto řádky pak budou tvořit bázi prostoru s referenčními duhovkami. Na obrázku 18 je ukázáno jakým způsobem jsou tyto řádky tvořeny. Tyto pak dosadíme do matice D a vypočítáme její singulární rozklad.

Máme-li nalezeno SVD matice D a vektor s dotazem (označme jej q), můžeme lehce zjistit, jestli se tento vektor nachází (nebo podobá vektoru) v původní bázi. Celý postup lze tedy shrnout do tří kroků :



Obrázek 17: Duhovky po aplikaci Gaborova filtru.



Obrázek 18: Sestavení jednoho řádku matice D z transformované duhovky.

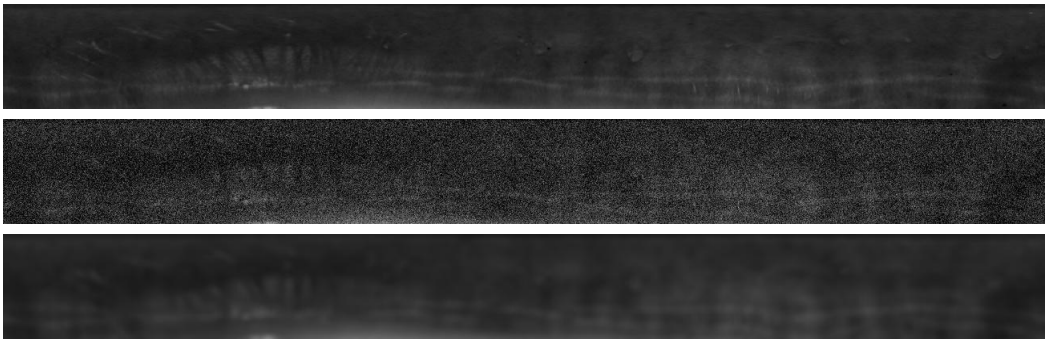
1. Nalezneme *SVD* matice D: $D = U\Sigma V^H$,
2. Transformujeme vektor q do báze generované sloupci matice V: $\hat{q} = \Sigma^{-1}U^H q$,
3. Projdeme všechny řádky matice V a porovnáme s vektorem $\hat{q}$: $\alpha_i = \frac{V_i^r \cdot \hat{q}^T}{\|V_i^r\| \cdot \|\hat{q}\|}, i = 1, \ldots, m.$

Přirozeně, pokud vektor q patří do báze, bude jeho skalární součin s některým z řádků matice V roven 1. Pokud se bude pouze podobat, bude výsledek číslo blízké 1.

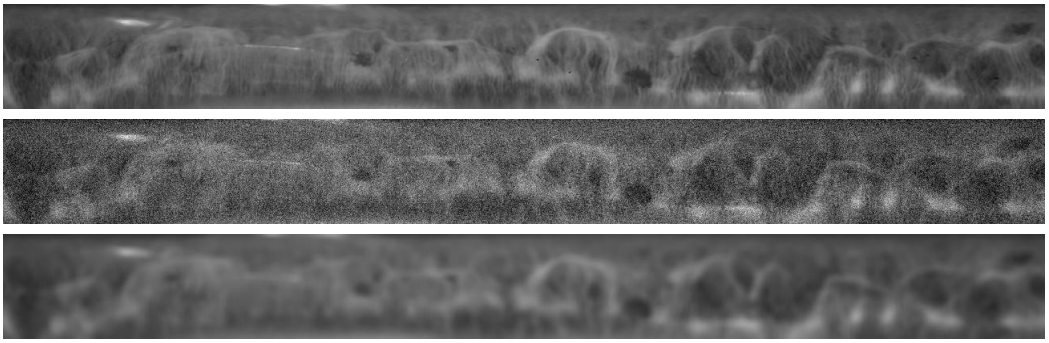
4.3.1 Numerické experimenty

Jako testovací dotaz jsme použili dvě duhovky, obě ve třech variantách. Originální - pouze transformovaná duhovka, zašuměná - duhovka byla transformována a byl do ní přidán Gaussův šum [15] s parametry $\mu = 0$ a $\sigma = 0.01$, rozmazaná - duhovka byla trans-

formována a byla následně rozmazána Gaussovým filtrem [15] s parametrem $\sigma = 10$ a velikost filtru byla zvolena 10. Ukázky těchto duhovek jsou na obrázcích 19 a 20.



Obrázek 19: Transformavé duhovky pro první testovací dotaz.



Obrázek 20: Transformavé duhovky pro druhý testovací dotaz.

Výsledky nalezneme v příloze A. Jedná se o prvních 10 nejlepších shod.

Z nalezených duhovek lze vidět, že metoda zde uvedená nalezla ve všech případech (až na jeden) na prvním místě správnou duhovku. Můžeme tedy usoudit, že postup zde uvedený je dobrým základem robustní metody pro porovnávání duhovek.

5 Závěr

V této práci jsem představil jednu z možností jak urychlit výpočet *SVD*. Zvolenou metodou bylo využití clusteru výkonných počítačů, čímž jsem výsledný algoritmus nejen urychlil, ale také umožnil řešit mnohem větší úlohy.

Nejdříve byla ukázána metoda, jak převést matici A do bidiagonální formy B pomocí Householderových transformací. Tento postup se ukázal jako rozumně paralelizovatelný při vhodně zvolené dekompozici dat na výpočetní uzly. Pro implementaci zde uvedenou byla zvolena dekompozice po řádcích, která umožnila pro 64 procesů (8 výpočetních uzlů, každý s 8 procesory) patnácti násobné urychlení výpočtu.

Když známe bidigonální matici B, můžeme vypočítat její singulární čísla. K tomu nám poslouží implicitní *QR* metoda, která byla představena v [1]. Tato metoda byla z [1] převzata pouze se změnou konvergenčních kritérií, které se ukázaly jako vhodnější pro efektivní implementaci. Protože se jedná o silně sekvenční algoritmus, jen obtížně paralelizovatelný bez velkého přínosu, zůstal i v této práci využit jako sekvenční kód.

Poslední fází *SVD* je výpočet levých a pravých singulárních vektorů. Při výpočtu singulárních vektorů je potřeba řešit soustavy rovnic, které jsou blízké singulárním ale z pohledu počítačové aritmetiky singulární nejsou. Tento problém byl vyřešen pomocí *LU* rozkladu, který umožnil lokalizaci těchto singularit. Vypočítaný *LU* rozklad byl také využit pro dopřednou a zpětnou substituci. Tyto umožnili efektivně vyřešit původní soustavu. Z následné analýzy tohoto kódu vyplynulo, že algoritmus je perfektně paralelně škálovatelný. Tedy jej lze rozložit na jednotlivé výpočetní uzly tak, že při výpočtu nebude probíhat žádná komunikace.

Zde představený algoritmus ukázal dobré výsledky jak z pohledu rychlosti, tak v oblasti přesnosti. Pro další práci se počítá s nalezením nové metody pro výpočet singulárních čísel, která by umožnila přímou paralelizaci.

V druhé části této diplomové práce bylo představeno latentní sémantické indexování (*LSI*), jako metoda porovnávání biometrických dat. *LSI* využívá jako hlavní nástroj právě *SVD* pro nalezení báze prostoru tvořeného biometrickými daty. Pro dostupnost byly jako biometrická data voleny duhovky.

Při zpracování duhovek bylo potřeba přijít s metodou robustní vůči rozmazání a vůči šumu, protože to jsou nejčastější defekty obrazových dat. Za tímto účelem byla navrhnuta podle [19] waveletová transformace. Tato je aplikována na transformovaná data a vytvoří matrici původních dat. Výsledná matrice již netrpí šumem ani rozmazáním a lze ji tedy přímo použít pro vytvoření báze prostoru.

Jak bylo zmíněno, data musí být nejdříve transformována, protože v původních snímcích nemá duhovka vždy stejnou pozici a rozměr. Proto byla představena metoda transformace původní duhovky do duhovky „rozbalené“. Tedy duhovky transformované z mezikruží na pás. Tato metoda je zatím návrh, tedy se předpokládá další zdokonalení.

Při numerických experimentech prokázala zde navržená metoda odolnost na šum a na rozmazání.

6 Reference

[1] Petr Kotas, *Efektivní implementace singulárního rozkladu matice*, VŠB-TU Ostrava

[2] Zdenek Dostál, *Lineární algebra,* VŠB - Technická univerzita Ostrava

[3] Gene H. Golub, Charles Van Loan, *Matrix Computation,* Library of Congress Cataloging - in - Publications

[4] Sonia Leach, *Singular Value Decomposition - A Primer,* Department of Computer Science, Brown University

[5] James Demmel, W. Kahan, *Accurate Singular Values of Bidiagonal Matrices,* Appeared in the SIAM J. Sci. Stat. Comput., v. 11, n. 5, pp. 873-912, 1990

[6] Demmel, James W., *Applied numerical linear algebra. [s.l.]* SIAM, 1997. 419 s.

[7] Gill P E, Murray W, Saunders M A, Wright M H, *Maintaining LU factors of a general sparse matrix*

[8] Benedikt Großer, Bruno Lang, *Efficient parallel reduction to bidiagonal form*, Bergische Universität

[9] K. Vince Fernando, Beresford N. Parlett, *Accurate singular values and differential QD qlgorithms*

[10] Feng Zhao, Wen Gao, *Singular Value Decomposition Based Image Matching*, Institute of Computing Technology, Chinese Academy of Sciences

[11] Napa Sae-Bae, Somkait Udomhunsakul, *Adaptive Block-Based Singular Value Decomposition Filtering*, Faculty of Engineering, Information Engineering Department

[12] Manolis G. Vozalis, Konstantinos G. Margaritis, *Applying SVD on Item-based Filtering*, Department of Applied Informatics, University of Macedonia

[13] Eugenio Hernández, Guido Weiss, *A first course on wavelets* CRC Press, 1996. 489 s

[14] Michael W. Frazier, *An introduction to wavelets through linear algebra*, Springer, 1999. 501 s

[15] Prat, K. William, *Digital image processing*, Willey, 2007. 807 s

[16] Gilbert Strang, Trough Nguyen, *Wavelets and filter banks*, Wellesley-Cambridge Press, 1996, 490 s

[17] Eduard Sojka, *Digitální zpracování a analýza obrazu*, VŠB - Technická univerzita Ostrava, 2000, 133s

[18] Javier R. Movellan, *Tutorial on Gabor Filters*

[19] John Daugman, *How Iris Recognition Works*, Cambridge

[20] Muhammad Umer Munir, Dr. Muhammad Younas Javed, *Fingerprint Matching using Gabor Filters*, National University of Sciences and Technology Pakistan

[21] In Ja Jeon, Mi Young Nam, Phill Kyu Rhee, *Adaptive Gabor Wavelet for Efficient Object Recognition*, Inha University

[22] Duan-Duan Yang, Lian-Wen Jin, Jun-Xun Yin, Li-Xin Zhen, Jian-Cheng Huang, *Facial Expression Recognition with Pyramid Gabor Features and Complete Kernel Fisher Linear Discriminant Analysis*, South China University of Technology, Motorola China Research Center

[23] Sanghoon Kim, Sun-Tae Chung, Souhwan Jung, Dusik Oh, Jaemin Kim, and Seongwon Cho, *Multi-Scale Gabor Feature Based Eye Localization*

[24] Dengsheng Zhang, Aylwin Wong, Maria Indrawan, Guojun Lu, *Content-based Image Retrieval Using Gabor Texture Features*, Monash University

[25] Tai Sing Lee, *Image Representation Using 2D Gabor Wavelets*

[26] In Ja Jeon, Mi Young Nam, and Phill Kyu Rhee, *Adaptive Gabor Wavelet for Efficient Object Recognition*, Inha University

[27] Tomaž Romih, Peter Planinšic, *Fast Image Segmentation Algorithm Using Wavelet Transform*, University of Maribor

[28] Emine Krichen, M. Anouar Mellakh, Sonia Garcia-Salicetti, Kamel Hamrouni, Nouredine Ellouze, Bernadette Dorizzi, *Iris Identification Using Wavelet Packet for Images in Visible Light Illumination*, Institut National des Télécommunications

[29] Sylvain Bernard, Nozha Boujemaa, David Vitale, Claude Bricot, *Fingerprint Segmentation using the Phase of Multiscale Gabor Wavelets*, INRIA Rocquencourt, THALES Identification

[30] Yeunggyu Park, Hoonju Yun, Myongseop Song, Jaihie Kim, *A Fast Circular Edge Detector for the Iris Region Segmentation*, Yonsei University

[31] M. Atiquzzaman, *Coarse-to-Fine Search Technique to Detect Circles in Images*, University of Dayton

[32] M.W. Berry, S.T. Dumais, G.W. O'Brien, *Using linear algebra for intelligent information retrieval*, 1994

[33] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman, *Indexing by Latent Semantic Analysis*, University of Chicago, Bell Communications Research, University of Western Ontario

[34] Thomas Hofman, *Probabilistic latent semnatic analysis*, Berkley, 1999

[35] Kurt Bryan, Tanya Leise, *The $25,000,000,000* eigenvector - The linear algebra behind google*

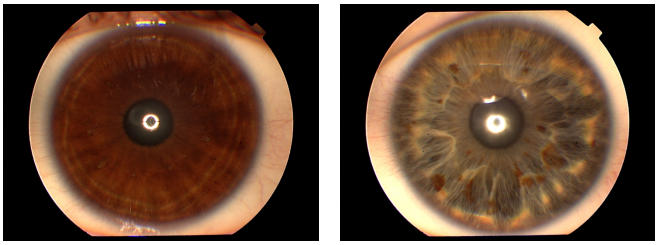
[36] Pavel Praks , Jiří Dvorský, Václav Snášel, *Latent Semantic Indexing for Image Retrieval Systems*, VŠB - Technická univerzita Ostrava

[37] Pavel Praks, Libor Machala, Václav Snášel, *On SVD-Free Latent Semantic Indexing for Iris Recognition of Large Databases*, VŠB - Technická univerzita Ostrava

[38] Pavel Praks, *O vztahu mezi smerodatnou odchylkou a singulárními čísly SVD rozkladu*, VŠB - Technická univerzita Ostrava

A Výsledky srovnání duhovek

Výsledky porovnávání duhovek z kapitoly 4.3.1. Pro vyhledávací dotazy byly voleny duhovky na obrázku A a jejich modifikace.



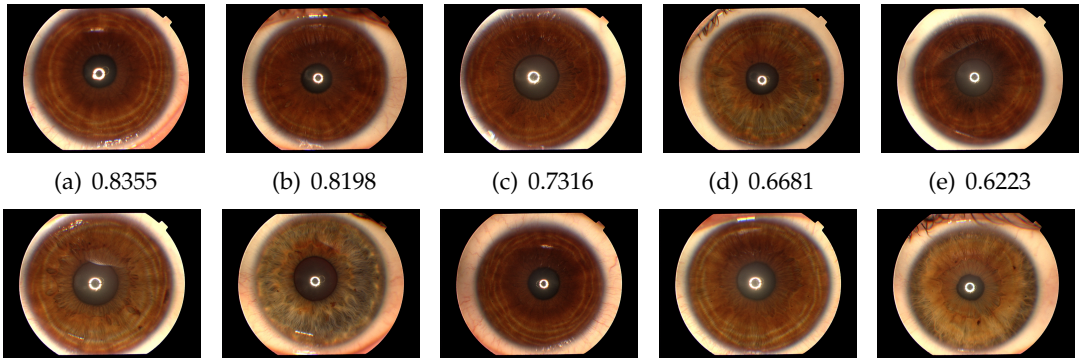
(a) 001L_1.jpg (b) 004L_3.jpg

Obrázek 21: Duhovky zvolené pro dotazy.

(a) 1.0000 (b) 0.6371 (c) 0.6220 (d) 0.6204 (e) 0.5445

(f) 0.5220 (g) 0.4836 (h) 0.4531 (i) 0.4499 (j) 0.4173

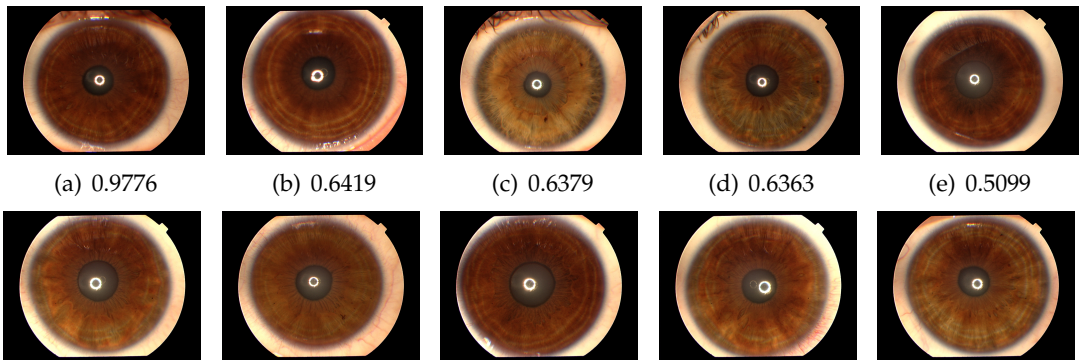
Obrázek 22: Výsledky porovnání pro první dotaz.



(a) 0.8355 (b) 0.8198 (c) 0.7316 (d) 0.6681 (e) 0.6223

(f) 0.6164 (g) 0.6088 (h) 0.5401 (i) 0.5239 (j) 0.5107

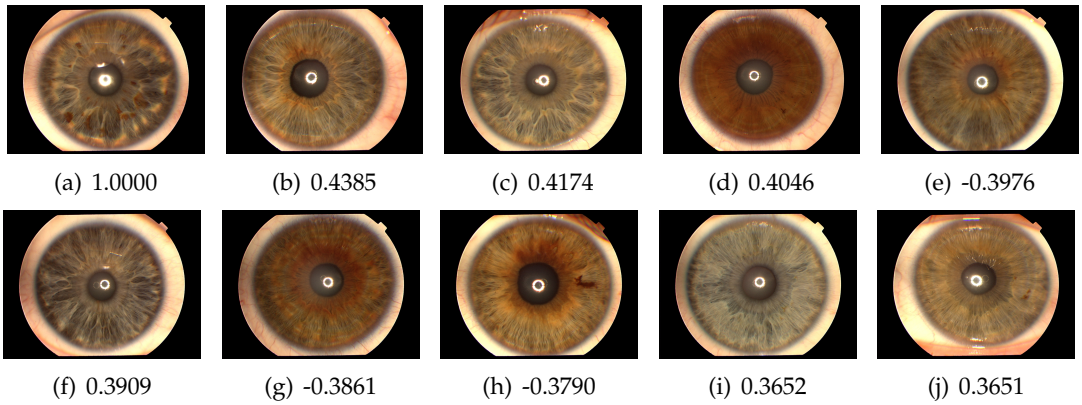
Obrázek 23: Výsledky porovnání pro druhý dotaz.



(a) 0.9776 (b) 0.6419 (c) 0.6379 (d) 0.6363 (e) 0.5099

(f) 0.4634 (g) 0.4615 (h) 0.4612 (i) 0.4532 (j) 0.4456

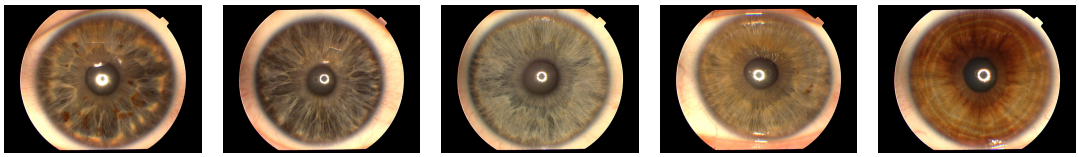
Obrázek 24: Výsledky porovnání pro třetí dotaz.



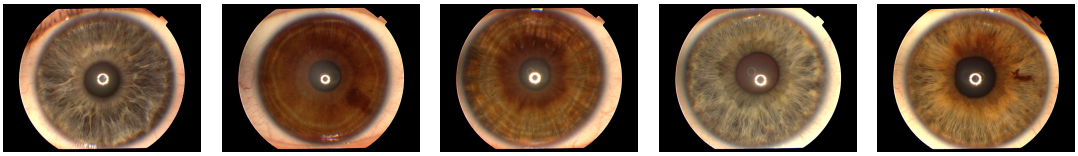
(a) 1.0000 (b) 0.4385 (c) 0.4174 (d) 0.4046 (e) -0.3976

(f) 0.3909 (g) -0.3861 (h) -0.3790 (i) 0.3652 (j) 0.3651

Obrázek 25: Výsledky porovnání pro čtvrtý dotaz.

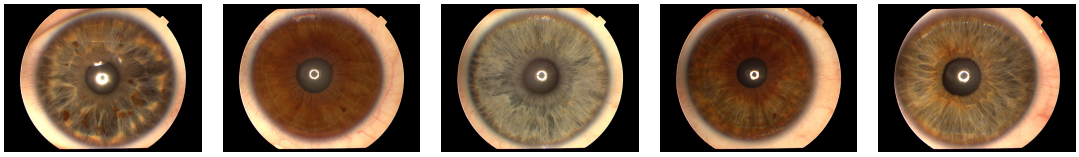


(a) 0.8417 (b) 0.4868 (c) 0.4552 (d) 0.4271 (e) -0.4163

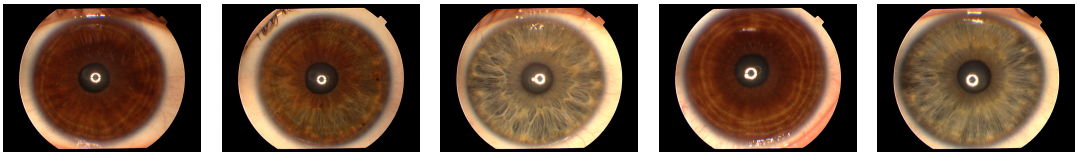


(f) 0.3884 (g) 0.3673 (h) -0.3662 (i) -0.3656 (j) -0.3471

Obrázek 26: Výsledky porovnání pro pátý dotaz.



(a) 0.9386 (b) 0.4521 (c) 0.4452 (d) 0.4253 (e) 0.4084



(f) 0.4054 (g) 0.3875 (h) 0.3779 (i) 0.3768 (j) -0.3754

Obrázek 27: Výsledky porovnání pro šestý dotaz.