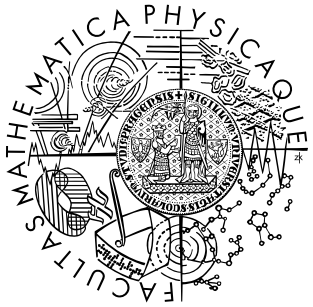
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jan Tejkal

QR-algoritmus

Katedra numerické matematiky

Vedoucí bakalářské práce: Doc. RNDr. Jan Zítko, CSc.

Studijní program: obecná matematika

2007

Děkuji vedoucímu bakalářské práce za věnovaný čas a za cenné připomínky a podněty.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 23. května 2007 Jan Tejkal

Obsah

Název práce: *QR-algoritmus*
Autor: *Jan Tejkal*
Katedra (ústav): *Katedra numerické matematiky*
Vedoucí bakalářské práce: *Doc. RNDr. Jan Zítko, CSc., KNM*
E-mail vedoucího: *zitko@karlin.mff.cuni.cz*
Abstrakt: *V práci se zabýváme výpočtem vlastních čísel matice pomocí QR-algoritmu. Studujeme jednotlivé QR-rozklady, na nichž je QR-algoritmus založen: Gram-Schmidtův, modifikovaný Gram-Schmidtův, Householderův a Givensův QR-rozklad. Uvádíme podrobně důkaz konvergence QR-algoritmu. Na základě asymptotického chování jsme zkusili aplikovat Aitkenův δ^2-proces. Ukázalo se, že tato extrapolace urychlila výpočetní postup. V případě nejmenšího vlastního čísla to dalo další návod, jak sestrojit posunutí, které se užívá v QR-algoritmu na urychlení konvergence. I když teoretické úvahy provádíme na plné matici, v praktických příkladech nejprve provedeme transformaci na horní Hessenbergův tvar. To je jednorázový proces, neboť QR-algoritmus zachovává strukturu Hessenbergovy matice.*

Všechny postupy testujeme na reálných maticích. Výše uvedené algoritmy na QR-rozklad jsme naprogramovali ve Fortranu, aplikovali v QR-algoritmu a sledovali vliv jednotlivých postupů na průběh QR-algoritmu. Na základě numerických výsledků jsme navrhli optimální algoritmus na výpočet vlastních čísel matice. Práce obsahuje algoritmické zápisy jednotlivých postupů.
Klíčová slova: *QR-algoritmus, QR-rozklad, Gramův-Schmidtův ortogonalizační proces, Givensův QR-rozklad, Householderův QR-rozklad*

Title: *The QR-algorithm*
Author: *Jan Tejkal*
Department: *Institute of Numerical Mathematics*
Supervisor: *Doc. RNDr. Jan Zítko, CSc., KNM*
Supervisor's e-mail address: *zitko@karlin.mff.cuni.cz*
Abstract: *In this work we deal with computation of the eigenvalues of matrices by using QR-algorithm. The QR-algorithm is based on a QR-factorization. We study these QR-factorizations: Gram-Schmidt, Modified Gram-Schmidt, Householder and Givens QR-factorization. In detail we show the proof of the convergence of the QR-algorithm. Because of asymptotic behaviour we tried to apply Aitken δ^2-process. This extrapolation accelerates the computation. In the case of the smallest eigenvalue it could be direction, how to construct the shift, which is used to accelerate the convergence of QR-algorithm. The theoretical considerations we make on the full matrix, but the practical examples we solve by means of the transformation into upper Hessenberg form. This transformation is made only once, because the QR-algorithm maintains the structure of Hessenberg matrix.*

All procedures are tested on the real matrices. The algorithms for QR-factorization we programmed in Fortran, applied in QR-algorithm and observed the influence of the choice from these methods on the run of the QR-algorithm. On the base of the numerical results we suggested the optimal algorithm for the computation of eigenvalues. The work contains algorithmic notations of all applied procedures.
Keywords: *QR-algorithm, QR-factorization, Gram-Schmidt orthogonalization process, Givens QR, Householder QR*

Úvod

Jedním z hlavních úkolů při počítání s maticemi je výpočet všech nebo alespoň některých vlastních čísel čtvercové matice. Jeden z postupů, který se propaguje a užívá, je QR-algoritmus a jeho různé modifikace. To je také hlavním předmětem této práce.

QR-algoritmus [1],[2],[5] je založen na QR-rozkladu matice, což je rozklad matice na součin matice ortonormální a matice horní trojúhelníkové. Nejprve tedy budeme studovat jednotlivé postupy QR-rozkladů. V numerických experimentech se podíváme na to, zdali bude QR-algoritmus ovlivněn použitím různých QR-rozkladů. Mezi hlavní rozklady patří Gram-Schmidtův, modifikovaný Gram-Schmidtův, Givensův a Householderův QR-rozklad. V první části práce tyto postupy popíšeme a uvedeme jejich algoritmy.

Další část pojednává o QR-algoritmu. Jedná se o iterační metodu, kdy provádíme QR rozklad matice, pomocí jehož konstruujeme posloupnost matic, které konvergují k horní trojúhelníkové matici s vlastními čísly na diagonále, případně ke kvazi-trojúhelníkové matici v případě komplexních vlastních čísel.

Parlett se ve své knize [5] soustředí na analogický QL-algoritmus (str. 151-188) s použitím QL-transformace. Pracuje hlavně s třídiagonální symetrickou maticí. Dokazuje například, že QL-algoritmus s užitím Wilkinsonova posunutí vždy konverguje, přičemž se omezuje na konvergenci poddiagonálního prvku. Konvergence je velice rychlá, což je dokumentováno na numerickém příkladě. Kromě toho studuje celou řadu posunutí.

Nesymetrický případ není již tak průhledný. Při studiu QR-algoritmu jsme v tomto případě vzali za základ Wilkinsonův článek [3], který vyšetřuje konvergenci LR a QR-algoritmu. Již z tohoto článku je patrné, že QR-algoritmus bude mít širší použití, proto jsme se omezili pouze na něj. Analogicky jsme rozpracovali konvergenci QR-algoritmu, která je ovšem postavena na některých neověřitelných předpokladech. Pokud je přijmeme, přesto obdržíme řadu informací, které se nakonec ukázaly být užitečné při výpočtu.

V numerických testech jsme uvažovali i plné matice ze sbírky příkladů [10] a to proto, že jsou zde uvedeny výsledky, se kterými provádíme srovnání. Pro srovnání jsme rovněž použili program stažený z balíku Lapac. Jinak při praktickém počítání se matice převede nejprve na Hessenbergův tvar, se kterým se dále počítá, stejně tak jako Parlett transformuje symetrickou matici na třídiagonální symetrickou matici.

Vyšetřovali jsme speciální tvary matice (třídiagonální a Hessenbergův tvar) a chování QR-algoritmu pro tyto matice. Numerickými experimenty jsme dospěli k závěru, že je nejlépe symetrickou matici nejprve převést na třídiagonální tvar a potom aplikovat QR-algoritmus. Jelikož QR-algoritmus nemění tuto strukturu matice, jednorázový převod na třídiagonální tvar se vyplatí, neboť díky němu v algoritmu ušetříme velké množství kroků. Totéž platí u nesymetrické matice, kterou převádíme na tvar Hessenbergův.

Provádíme posun spektra matice (metody se shifty), čímž se konvergence urychluje. Mnohdy je tento krok dokonce nezbytným, aby QR-algortimus vůbec konvergoval.

Všemi postupy, výpočty a algoritmy jsme se zabývali nejenom teoreticky, ale také prakticky. Naprogramovali jsme QR-rozklady, QR-algoritmy a další metody ve Fortranu, abychom mohli jednotlivé algoritmy a postupy numericky testovat. Vyšetřovali jsme, jak jsou metody přesné a rychlé pro různé (špatně podmíněné) matice.

Ke konci práce se zabýváme přesností jednotlivých QR-rozkladů a QR-algoritmu. Zkoumáme kritérium skončení algoritmu. Narážíme na několik dalších problémů, ke kterým navrhujeme řešení.

Při studiu průběhu logaritmu chyby jsme zjistili, že se graf chová jako přímka, což nás dovedlo k myšlence zkusit při výpočtu alespoň některých vlastních čísel použít Aitkenova δ^2-procesu. Numerické výsledky ukázaly účinnost tohoto postupu.

Práci jsme rozčlenili do tří částí: *QR-rozklad*, *QR-algoritmus* a *Výpočty a numerická srovnání*.

V první kapitole rozebereme čtyři různé postupy při QR-rozkladech: Gram-Schmidtův, modifikovaný Gram-Schmidtův, Householderův a Givensův QR-rozklad. Ukážeme jak tyto rozklady konstruovat a také u každého QR-rozkladu napíšeme algoritmy jeho konstrukce.

Druhá kapitola je věnována QR-algoritmu. Stěžejním bodem této části bude věta o konvergenci QR-algoritmu.

QR-algoritmus se používá s různými modifikacemi. Ukážeme, jak převádět matice na Hessenbergův tvar a na třídiagonální tvar. Popíšeme také, jakým způsobem lze posunout spektrum matice a jak se to projeví při výpočtech.

Třetí kapitola bude obsahovat výsledky našeho praktického pozorování jednotlivých metod. Nejprve porovnáme přesnost všech čtyř QR-rozkladů. Potom budeme studovat správnost výpočtu vlastních čísel pomocí QR-algoritmu. Závěrem nabídneme několik možností vylepšení výpočtu a především popíšeme metodu extrapolace vlastních čísel pomocí Aitkenova δ^2-procesu.

Kapitola 1

QR-rozklad

V této kapitole probereme Gram-Schmidtův QR-rozklad, modifikovaný Gram-Schmidtův QR-rozklad, Householderův QR-rozklad a Givensův QR-rozklad.

U Gram-Schmidtova QR-rozkladu uvedeme větu o existenci a jednoznačnosti tohoto rozkladu a důkladně rozepíšeme její konstruktivní důkaz, podle něhož bude snadné Gram-Schmidtův QR-rozklad matice zkonstruovat.

Popíšeme také postup modifikovaného Gram-Schmidtova QR-rozkladu a ukážeme, jak se v literatuře uvádí chyba tohoto rozkladu a chyba ortogonality matice Q. Ukážeme zajímavou myšlenku tzv. dvojitého modifikovaného Gram-Schmidtova QR-rozkladu, který přináší podstatně menší chybu ortogonality matice Q.

V další podkapitole představíme Householderovu transformaci a ukážeme jak zkonstruovat Householderovu matici, která tuto transformaci určuje. Householderovu transformaci využijeme ke konstrukci Householderova QR-rozkladu.

Nakonec popíšeme Givensovu rotaci a její použití na stanovení QR-rozkladu. Takto provedený QR-rozklad se nazývá Givensův QR-rozklad. Ukážeme, jak lze tento rozklad zjednodušit, pokud je matice v Hessenbergově tvaru.

Jak již bylo řečeno, u všech QR-rozkladů jsou přehledně uvedeny algoritmy jejich konstrukce.

1.1 Gram-Schmidtův QR-rozklad

Jako první metodou QR-rozkladu matice $A \in \mathbb{R}^{n\times n}$, kterou se budeme zabývat, bude Gram-Schmidtův QR-rozklad. Vyslovíme větu o existenci a jednoznačnosti tohoto QR-rozkladu a v důkazu podrobně ukážeme, jak zkonstruovat ortogonální matici $Q \in \mathbb{R}^{n\times n}$ a horní trojúhelníkovou matici $R \in \mathbb{R}^{n\times n}$, pro které platí $A = QR$.

Věta 1.1 *Nechť $A \in \mathbb{R}^{n\times n}$ je regulární matice. Pak existuje ortonormální matice $Q \in \mathbb{R}^{n\times n}$ a horní trojúhelníková matice $R \in \mathbb{R}^{n\times n}$ taková, že*

$$A = QR \tag{1.1}$$

Pokud jsou diagonální prvky matice R kladné, je tento rozklad jednoznačný.

Důkaz. Provedeme konstruktivní důkaz.

Označme i-tý sloupec matice A resp. Q symbolem a_i resp. q_i a dále $R = [r_{ij}]_{i,j=1}^{n}$. Podle (1.1) musí být $a_1 = r_{11}q_1$. Položíme tedy $r_{11} = ||a_1||_2$, a $q_1 = a_1/r_{11}$. Zde lze volit r_{11} i s opačným znaménkem. Zvolíme r_{11} kladné.

Předpokládejme, že již máme určeno s ($s \geq 1$) sloupců $q_1, \ldots, q_s$ matice Q a s sloupců matice R, přičemž platí $span\{q_1, \ldots, q_s\} = span\{a_1, \ldots, a_s\}$. Chceme nyní určit další sloupec q_{s+1}. Podle (1.1) musí nutně být

$$a_{s+1} = r_{1,s+1}q_1 + r_{2,s+1}q_2 + \cdots + r_{s+1,s+1}q_{s+1} \tag{1.2}$$

Vynásobíme-li rovnost (1.2) postupně zleva vektory q_i^T ($i = 1, 2, \ldots, s$) obdržíme pro $i = 1, 2, \ldots, s$ rovnost $q_i^T a_{s+1} = r_{i,s+1}$. Z (1.2) plyne

$$r_{s+1,s+1}q_{s+1} = a_{s+1} - r_{1,s+1}q_1 - \cdots - r_{s,s+1}q_s = z_{s+1}.$$

Platí $z_{s+1} \neq 0$, neboť v opačném případě by $a_{s+1} \in span\{q_1, \ldots, q_s\} = span\{a_1, \ldots, a_s\}$, což by bylo ve sporu s tím, že matice A je regulární.

Můžeme tedy určit

$$r_{s+1,s+1} = ||z_{s+1}||_2$$

a

$$q_{s+1} = z_{s+1}/r_{s+1,s+1}$$

Po provedení celé konstrukce obdržíme ortonormální matici Q a horní trojúhelníkovou matici R a platí $A = QR$. Tím jsme dokázali existenci rozkladu.

Nyní ukážeme jednoznačnost rozkladu $A = QR$. Platí $Q^TQ = I$. Předpokládejme existenci dalšího rozkladu $A = Q_1R_1$, kde $Q_1^TQ_1 = I$ a R_1 má kladné diagonální prvky.

Platí $Q_1 = QRR_1^{-1} \Rightarrow Q_1^T = (QRR_1^{-1})^T = (RR_1^{-1})^TQ^T$, tudíž

$$I = Q_1^TQ_1 = (RR_1^{-1})^TQ^TQRR_1^{-1} = (RR_1^{-1})^TRR_1^{-1}$$

a odtud plyne

$$((RR_1^{-1})^T)^{-1} = RR_1^{-1} \tag{1.3}$$

Ve vztahu (1.3) je matice vlevo je dolní trojúhelníková, matice vpravo horní trojúhelníková, proto je matice RR_1^{-1} diagonální. Rovnost (1.3) můžeme tedy psát jako $(RR_1^{-1})^{-1} = RR_1^{-1}$ a vidíme, že RR_1^{-1} se rovná diagonální matici s diagonálními prvky rovnými v absolutní hodnotě 1. Uvažujeme matice R i R_1 s kladnými prvky na diagonále a tudíž $RR_1^{-1} = I$, tedy

$$R = R_1.$$

Z rovnosti $QR = Q_1R_1$ plyne, že

$$Q = Q_1.$$

□

Poznámka 1.1: Tímto způsobem lze provést QR-rozklad i obdélníkové matice s maximální hodností. Věta i s důkazem je uvedena v knize [2] jako věta 11.5 na straně 207.

Poznámka 1.2: QR-rozklad se provádí analogicky i pro matici A s komplexními prvky, s tím rozdílem, že všude místo transpozice (A^T) používáme hermitovské sdružení (A^H) a nehovoříme o ortonormální matici, ale o matici unitární, viz článek [3].

Nyní si uvedeme algoritmický zápis konstrukce matic Q a R pomocí Gram-Schmidtova QR-rozkladu.

Algoritmus 1.1: ***Gram-Schmidtův QR-rozklad***

Vstup: *matice $A \in \mathbb{R}^{n\times n}$ skládající se ze sloupců a_k, pro $k = 1, \ldots, n$*
Výstup: *ortonormální matice $Q \in \mathbb{R}^{n\times n}$ a horní trojúhelníková matice $R \in \mathbb{R}^{n\times n}$, jejíž sloupce označíme q_k resp. r_k, pro $k = 1, \ldots, n$*

for $k = 1$ ***to*** n
    for $j = k + 1$ ***to*** n
        $r_{jk} = 0$
    end
end
$r_{11} = ||a_1||_2$
$q_1 = a_1/r_{11}$
for $k = 2$ ***to*** n
    $z = a_k$
    for $j = 1$ ***to*** $k - 1$
        $r_{jk} = q_j^T a_k$
        $z = z - r_{jk}q_j$
    end
    $r_{kk} = ||z||_2$
    $q_k = z/r_{kk}$
end

1.2 Modifikovaný Gram-Schmidtův QR-rozklad

Modifikovaný Gram-Schmidtův QR-rozklad (dále jen MGS) je metoda QR-rozkladu, která se využívá kvůli lepším numerickým vlastnostem, než má klasický Gram-Schmidtův QR-rozklad.

V jednotlivých krocích získáváme sloupce matice Q a řádky matice R.
Metoda se vyznačuje tím, že v každém kroku dochází k přepočítávání matice A. Toto přepočítávání je dáno tím, že nejprve stanovíme první sloupec matice Q, který odvodíme z prvního sloupce matice A a potom přepočítáme zbylé sloupce matice A, aby všechny byly kolmé na první sloupec matice Q. Prostor tvořený druhým až n-tým sloupcem matice A je nyní dimenze $n-1$. Dále postupujeme stejným způsobem, avšak v každém následujícím kroku pracujeme s prostorem dimenze o 1 menší.

Tato metoda je matematicky identická s Gram-Schmidtovým QR-rozkladem. Pomocí MGS však dosahujeme lepších numerických výsledků.

1.2.1 Postup

Matici A označíme jako $A^{(0)}$, v k-tém kroku ji označíme jako $A^{(k)}$. Matice $A^{(k)}$ vypadá takto: $A^{(k)} = [q_1, q_2, \ldots, q_k, a_{k+1}^{(k)}, \ldots, a_n^{(k)}]$.

V prvním kroku určíme $r_{11} = ||a_1||_2, q_1 = a_1/r_{11}$. Pro $j = 2, \ldots, n$ spočítáme j-tý sloupec matice A tímto způsobem: $a_j^{(1)} = a_j^{(0)} - r_{1j}q_1$. Aby platilo $q_1^T a_j^{(1)} = 0$, položíme $r_{1j} = q_1^T a_j^{(0)}$.

V k-tém kroku postupujeme následovně.

Určíme diagonální prvek matice R:

$$r_{kk} = ||a_k^{(k-1)}||_2$$

a k-tý sloupec matice Q:

$$q_k = a_k^{(k-1)}/r_{kk}.$$

Dále dopočítáme k-tý řádek matice R: chceme, aby $q_k^T a_j^{(k)} = 0$, proto bude

$$r_{kj} = q_k^T a_j^{(k-1)}, \text{ kde } j = k+1, \ldots, n.$$

V k-tém kroku tedy vyjadřujeme j-tý sloupec matice A takto:

$$a_j^{(k)} = a_j^{(k-1)} - r_{kj}q_k, \text{ kde } j = k+1, \ldots n.$$

Platí:

$$q_k = a_k^{(k-1)}/r_{kk} = (a_k^{(k-2)} - r_{k-1,k}q_k - 1)/r_{kk} =$$
$$= (a_k^{(0)} - r_{1k}q_1 - r_{2k}q_2 - \cdots - r_{k-1,k}q_{k-1})/r_{kk}.$$

Takže

$$a_k = r_{1k} + r_{2k}q_2 + \cdots + r_{k-1,k}q_{k-1} + r_{kk}q_k = \sum_{j=1}^{k} r_{jk}q_k,$$

což není nic jiného než

$$A = QR.$$

Algoritmus 1.2: ***Modifikovaý Gram-Schmidtův QR-rozklad***

Vstup: *matice* $A \in \mathbb{R}^{n\times n}$ *se sloupci* a_j, *pro* $j = 1, \ldots, n$
Výstup: *ortonormální matice* $Q \in \mathbb{R}^{n\times n}$ *a horní trojúhelníková matice* $R \in \mathbb{R}^{n\times n}$, *které obsahují sloupce* q_j *resp.* r_j, *pro* $j = 1, \ldots, n$

for $k = 1$ ***to*** n
  for $j = 1$ ***to*** $k - 1$
  $r_{kj} = 0$
  end
  $r_{kk} = ||a_k||_2$
  $q_k = a_k / r_{kk}$
  for $j = k + 1$ ***to*** n
    $r_{kj} = q_k^T a_j$
    $a_j = a_j - r_{kj} q_k$
  end
end

1.2.2 Chyba rozkladu

Pro regulární matici $A \in \mathbb{R}^{n\times n}$ se singulárními vlastními čísly $\sigma_1 \geq \cdots \geq \sigma_n$ můžeme spočítat číslo podmíněnosti jako $\kappa_2(A) = \sigma_1/\sigma_n$.

Pro matice $\widehat{Q}$ a $\widehat{R}$, které získáme z MGS, podle [8], [9] platí:

$$||A - \widehat{Q}\widehat{R}||_2 \leq c_1 u ||A||_2.$$

Porušení ortonormality matice $\widehat{Q}$ můžeme omezit takto:

$$||I - \widehat{Q}^T\widehat{Q}||_2 \leq c_2 \kappa(A) u. \tag{1.4}$$

K matici $\widehat{R}$, kterou jsme získali z MGS, existuje ortonormální matice Q, že platí:

$$||A - Q\widehat{R}||_2 \leq cu||A||_2.$$

Konstanty c_1, c_2 a c jsou závislé pouze na řádu matice n a na aritmetice stroje, který provádí výpočet, konstanta $u = 2^{-t}$ je zaokrouhlovací jednotka. V [7] jsou konstanty c_1, c_2 a c odvozeny a spočítány, jejich hodnota je

$$c_1 = 1,853 \cdot n^{\frac{3}{2}}, c_2 = 31,6863 \cdot n^{\frac{3}{2}} \text{ a } c = 18,53 \cdot n^{\frac{3}{2}}.$$

V [7] je studována ortogonalita matice $\widehat{Q}$ pomocí jejího čísla podmíněnosti. Autoři uvádějí tuto větu:

Věta 1.2 *Nechť* $A \in \mathbb{R}^{n\times n}$ *je regulární matice,* $\kappa(A)$ *číslo podmíněnosti matice* A *a platí:*

$$2,12 \cdot (n + 1)u < 0,01$$

$$c \cdot u \cdot \kappa(A) \leq 0,1,$$

kde $c = 18,53 \cdot n^{\frac{3}{2}}$ *a* u *je zaokrouhlovací jednotka.*

Potom pro matici $\widehat{Q} \in \mathbb{R}^{n \times n}$ *spočítanou pomocí MGS platí*

$$\kappa(\widehat{Q}) \leq 1,3. \tag{1.5}$$

Důkaz. Důkaz je podrobně rozveden v [7]. □

Metoda MGS by se měla používat pouze tehdy, pokud jsou ortonormalizované vektory matice A "pěkně" nezávislé (když je $\kappa_2(A)$ malé).

1.2.3 Dvojitý MGS

V [7] se také autoři zabývají tzv. dvojitým MGS. Po provedení MGS na matici A získáme matice $\widehat{Q}$ a $\widehat{R}$. Přesnost ortogonality matice $\widehat{Q}$ závisí na číslu podmíněnosti matice A, kterou rozkládáme. Provedeme-li MGS ještě jednou na matici $\widehat{Q}$, získáme matice $\widehat{Q}_2$ a $\widehat{R}_2$, ovšem matice $\widehat{Q}_2$ bude podstatně "ortogonálnější".

Vyjádříme-li $c_2 = 1.71 \cdot c$ a dosadíme-li do (1.4), získáme

$$||I - \widehat{Q}_2^T \widehat{Q}_2||_2 \leq c_2 \kappa(\widehat{Q}) u = 1,71 \cdot c \cdot u \cdot \kappa(\widehat{Q}).$$

Využijeme nerovnosti (1.5) a dosadíme za c, získáme

$$||I - \widehat{Q}_2^T \widehat{Q}_2||_2 < 40,52 \cdot u n^{\frac{3}{2}}.$$

Poznamenejme, že jsme $\kappa(\widehat{Q})$ ve skutečnosti omezili trochu přesnějším odhadem $(1,27848\ldots)$, než uvádíme ve větě.

Vidíme tedy, že přesnost ortogonality matice $\widehat{Q}_2$ už závisí pouze na přesnosti stroje provádějícího výpočet a na řádu matice.

1.3 Householderův QR-rozklad

Householderův QR-rozklad využívá tzv. Householderovy matice zrcadlení (dále jen Householderova matice). Budeme definovat, jak vypadá Householderova matice. Tato matice je určena tzv. Householderovým vektorem. Ukážeme, jak tento vektor spočítat, abychom mohli transformaci danou Householderovou maticí využít ke QR-rozkladu.

1.3.1 Householderova matice

Definice 1.1 *Nechť* $w \in \mathbb{R}^n$, $||w||_2 = 1$. *Matice* $P \in \mathbb{R}^{n \times n}$ *ve tvaru*

$$P = I - 2ww^T$$

se nazývá ***Householderovou maticí****. Vektor* w *se nazývá* ***Householderovým vektorem****.*

Transformace daná Householderovou maticí "zrcadlí" daný vektor x na vektor y, který je symetrický k vektoru x podle nadroviny ortogonální k vektoru w.

Chceme určit Householderův vektor, aby transformace měla podobu

$$Px = \alpha e_1.$$

Takto bude vypadat Householderova transformace matice A, když za vektor x vezmeme první sloupec matice A, (transformaci použijeme na všechny sloupce matice):

$$A \to \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}.$$

Stejný postup provedeme na podmatici $[a_{ij}]_{i,j=2}^{n}$ a analogicky dále, až získáme matici

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix},$$

což bude hledaná horní trojúhelníková matice R.

Ukážeme si, jak nalézt velikost čísla $\alpha \in \mathbb{R}$ a jak k zadanému vektoru x nalézt matici Householderovy transformace.

Householderova matice má tyto vlastnosti:

- je symetrická

  Důkaz: plyne ze symetrie matice ww^T.

- je ortonormální

  Důkaz:

  $PP^T = (I - 2ww^T)(I - 2ww^T) = I - 4ww^T + 4w\underbrace{w^Tw}_{1}w^T = I$

Pro zajímavost uveďme, že Householderova matice má determinant roven -1.

1.3.2 Výpočet Householderova vektoru

Mějme vektor $x \in \mathbb{R}^n,\ x \neq 0$. Hledáme vektor $w \in \mathbb{R}^n$, který určuje Householderovu matici $P = I - 2ww^T$ tak, aby Px byl násobkem vektoru e_1. Ukážeme, že platí

$$P\frac{x}{||x||_2} = e_1, \text{ tj. } Px = ||x||_2 e_1.$$

Označení e_1 používáme pro kanonický vektor s prvkem 1 v první složce, x_1 a w_1 používáme pro označení první složky vektoru x resp. w.

Vyjádříme

$$Px = (I - 2ww^T)x = x - 2w^T xw. \tag{1.6}$$

Tato rovnost a $Px \in span\{e_1\}$ znamenají, že $w \in span\{x, e_1\}$. Zvolíme $w = x + \alpha e_1$ a dostaneme rovnost:

$$w^T x = x^T x + \alpha x_1 \tag{1.7}$$

a po dosazení $x = w - \alpha e_1$ platí

$$1 = w^T w = x^T x + 2\alpha x_1 + \alpha^2. \tag{1.8}$$

Dosadíme do (1.6) $w = x + \alpha e_1$ a obdržíme

$$Px = x - 2w^T xw = x - 2w^T x(x + \alpha e_1) = x - 2w^T xx - 2w^T x\alpha e_1$$

Do druhého členu dosadíme za $w^T x$ výraz z (1.7) a vytkneme x z prvních dvou členů. Vznikne tato rovnost:

$$Px = (1 - 2x^T x - 2\alpha x_1)x - 2\alpha w^T xe_1 \overset{(1.8)}{=} (1 - 2\frac{x^T x + \alpha x_1}{x^T x + 2\alpha x_1 + \alpha^2})x - 2\alpha w^T xe_1.$$

Aby $Px \in span\{e_1\}$, potřebujeme vynulovat koeficient u x. Toho dosáhneme volbou $\alpha = \pm||x||_2$.

Pro vektor w tedy platí $w = x \pm ||x||_2 e_1$.

Do vyjádření $w^T x$ můžeme za $x^T x$ dosadit α^2:

$$w^T x = \alpha^2 + \alpha x_1 = \alpha(\alpha + x_1).$$

Platí

$$1 = w^T w = \alpha^2 + 2\alpha x_1 + \alpha^2 = 2(\alpha^2 + \alpha x_1) = 2\alpha(\alpha + x_1).$$

Platí tedy:

$$Px = -2\alpha w^T xe_1 = -2\alpha\alpha(\alpha + x_1)e_1 = -\alpha e_1 = \mp||x||_2 e_1.$$

Vektor w můžeme zvolit buď jako $x + ||x||_2 e_1$ nebo $x - ||x||_2 e_1$. Volbou $w = x - ||x||_2 e_1$ docílíme toho, že Px bude kladným násobkem e_1. Pro první složku vektoru w bude platit $w_1 = x_1 - ||x||_2$. Problém může nastat, pokud bude x blízké kladnému násobku e_1. Může dojít k tomu, že $x_1 - ||x||_2$ se při omezené přesnosti stroje provádějícího výpočet bude rovnat 0.

Pro kladné x_1 si ale stačí rovnost rozšířit:

$$w_1 = x_1 - ||x||_2 = \frac{x_1^2 - ||x||_2^2}{x_1 + ||x||_2} = \frac{-(x_2^2 + \cdots + x_n^2)}{x_1 + ||x||_2}$$

a vidíme, že tento problém už nenastává, protože nebudeme od sebe odečítat čísla blízká 1, ale budeme místo toho sčítat malá čísla.

Je praktické, aby první složka Householderova vektoru byla rovna 1, proto vektor budeme normalizovat přenásobením $\frac{1}{w_1}$.

Tento postup nyní shrneme do algoritmického zápisu:

Algoritmus 1.3: ***Výpočet Householderova vektoru***

Vstup: *vektor* $x \in \mathbb{R}^n$
Výstup: *vektor* $v \in \mathbb{R}^n$ *a číslo* $\beta \in \mathbb{R}$

$\sigma = [x_2, \ldots, x_n]^T [x_2, \ldots, x_n]$
$v^T = [1, x_2, \ldots, x_n])$
if $\sigma = 0$
    $\beta = 0$
else
    $\mu = \sqrt{x_1^2 + \sigma}$
    if $x_1 \leq 0$
        $v_1 = x_1 - \mu$
    else
        $v_1 = -\sigma/(x_1 + \mu)$
    end
    $\beta = \frac{2v_1^2}{\sigma + v_1^2}$
    $v = \frac{1}{v_1} v$
end

Tento algoritmus (funkci) budeme označovat $[v, \beta] = \mathbf{house}(x)$.

Vektor v a číslo β definují Householderovu matici ve tvaru

$$P = I - \beta v v^T.$$

Číslo β vyjadřuje normování vektoru v. Je spočítáno jako zlomek $\frac{2}{v^T v}$, který je navíc násoben první složkou vektoru v_1^2, protože potom vektor násobíme $\frac{1}{v_1}$, abychom získali první složku vektoru v rovnou 1.

1.3.3 Householderův QR-rozklad

Při provádění Householderova QR-rozkladu postupujeme následovně: K vektoru, který je tvořen j-tou až n-tou složkou j-tého sloupce matice A, $j = 1, \ldots, n$, určíme Householderův vektor. Pro jednotlivé sloupce j, $j = 1 \ldots n$, matice A tedy získáváme Householderovy vektory v_j. Z vektorů v_j vytvoříme Householderovu matici P_j. V každém kroku matici A zleva přenásobíme touto maticí P_j.

V prvním kroku přenásobíme matici A Householderovou maticí P_1. První sloupec matice A se přetransformuje na násobek e_1. Po k krocích bude matice A přetrans-

formována na matici tvaru:

$$P_k \dots P_2 P_1 A = \begin{bmatrix} \times & \times & \dots & \dots & \dots & \times \\ & \times & \dots & \dots & \dots & \times \\ & & \ddots & & & \vdots \\ & & & \times & \dots & \times \\ & \mathbf{0} & & \vdots & & \vdots \\ & & & \times & \dots & \times \end{bmatrix} \begin{matrix} \\ \\ \\ k \\ \\ \\ \end{matrix}$$

$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad k$$

Householderovy matice nebývá potřeba explicitně vyjadřovat.
Označíme si $P_n P_{n-1} \dots P_1 = H$. Po n krocích získáme horní trojúhelníkovou matici $HA = R$. Platí $A = H^T R$, neboť H je ortonormální, a tak $H^{-1} = H^T$. Označíme $Q = P_1 \dots P_n$, platí $A = QR$.

Poddiagonální prvky matice A budeme v algoritmu při každém kroku nahrazovat druhou až n-j+1-tou složkou příslušného Householderova vektoru. Označíme j-tý Householderův vektor: $v^{(j)} = [\underbrace{0, \dots, 0}_{j-1}, 1, v_{j+1}^{(j)}, \dots, v_n^{(j)}]^T$ a výsledná matice A bude vypadat takto:

$$A = \begin{bmatrix} r_{11} & r_{12} & r_{13} & \dots & r_{1n} \\ v_2^{(1)} & r_{22} & r_{23} & \dots & r_{2n} \\ v_3^{(1)} & v_3^{(2)} & r_{33} & \dots & r_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ v_n^{(1)} & v_n^{(2)} & v_n^{(3)} & \dots & r_{nn} \end{bmatrix}.$$

Algoritmus 1.4: *Householderův QR-rozklad*

Vstup: *matice $A \in \mathbb{R}^{n \times n}$ se sloupci a_j, pro $j = 1, \dots, n$*
Výstup: *matice $A \in \mathbb{R}^{n \times n}$, horní trojúhelníková část matice A obsahuje složky matice R, poddiagonální část obsahuje druhou až n-j+1-tou složkou Householderových vektorů*

for $j = 1$ ***to*** n
  $[v, \beta] =$ ***house***$([(a_j)_j, \dots, (a_j)_n])$
  $[a_{ik}]_{ik=j}^n = (I_{n-j+1} - \beta v v^T)[a_{ik}]_{ik=j}^n$
  if $j < n$
    for $i = j + 1$ ***to*** n
    $(a_j)_i = v_{i-j+1}$
    end
  end
end

Explicitní vyjádření matice Q můžeme získat tímto způsobem: pro každý Householderův vektor, jehož složky jsou uloženy v matici A, určíme Householderovy matice $P_1, \ldots, P_n$. Tyto matice jsou řádu 2×2 až $n \times n$, jejich součin provedeme následujícím způsobem. Násobíme "odzadu", tzn. nejprve provedeme součin matic P_{n-1} (matice řádu 3×3) a P_n (matice řádu 2×2):

$$P_{n-1} \begin{bmatrix} 1 & \boxed{0} \\ \boxed{\begin{matrix} \\ \\ 0 \end{matrix}} & \boxed{\begin{matrix} \\ \\ P_n \end{matrix}} \end{bmatrix}$$

Výslednou matici opět stejným způsobem rozšíříme o první sloupec a první řádek a zleva vynásobíme maticí P_{n-2}. Takto postupujeme až po matici P_1.

Tento způsob je popsán v následujícím algoritmu.

Algoritmus 1.5: ***Určení matice Q***

Vstup: *matice $A \in \mathbb{R}^{n \times n}$ se složkami Householderových vektorů, matice A se skládá ze sloupců a_j, kde $j = 1, \ldots, n$*
Výstup: *matice $Q \in \mathbb{R}^{n \times n}$*

$Q = I_n$
for $j = n - 1$ ***downto*** 1
    $[v_j, \ldots, v_n]^T = [1, (a_j)_{j+1}, \ldots, (a_j)_n]$
    $[q_{ik}]_{i,k=j}^n = (I - \beta_j [v_j, \ldots, v_n][v_j, \ldots, v_n]^T)[q_{ik}]_{i,k=j}^n$
end

Poznámka 1.3: Pro matici $\widehat{Q}$, která vznikne z Householderova QR-rozkladu, podle [1] platí:

$$||I - \widehat{Q}^T \widehat{Q}||_2 \leq c_3 u,$$

kde u je zaokrouhlovací jednotka a c_3 je konstanta, která závisí pouze na rozměru matice A.

Podstatou tohoto vztahu je, že v něm nefiguruje číslo podmíněnosti matice A.

1.4 Givensův QR-rozklad

Givensův QR-rozklad využívá tzv. Givensovy rotace. Tato rotace transformuje dvousložkový vektor na vektor, který má druhou složku nulovou. Každá transformace aplikovaná na dvojici řádků matice A vytvoří v této matici tedy jednu nulu. Prováděním Givensových rotací na matici A získáme horní trojúhelníkovou matici, kterou označíme R. Ortonomální matici Q získáme jako součin všech matic Givensovy rotace.

1.4.1 Givensova matice rotace

Definice 1.2 ***Givensova rotace*** *je matice ve tvaru*

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

$$\qquad\qquad\qquad\qquad\qquad\quad i \qquad\quad k$$

kde $s^2 + c^2 = 1$. *Indexy* i *a* k *označují řádky a sloupce s prvky* c *a* s.

Je-li vektor $x \in \mathbb{R}^n$ a $y = G(i,k)^T x$, potom

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

Budeme potřebovat, aby k-tá složka vektoru y byla nulová. Musí tedy platit

$$c = \frac{x_i}{\sqrt{x_i^2 + x_k^2}}, \qquad s = \frac{-x_k}{\sqrt{x_i^2 + x_k^2}}.$$

Zajímají nás pouze řádky a sloupce i a k, proto budeme uvažovat prostor $\mathbb{R}^2$. Naším cílem je vynulovat druhou složku vektoru $y \in \mathbb{R}^2$. To budeme potřebovat proto, abychom transformovali matici A na matici R, která bude mít pod diagonálou nuly. Rozepíšeme-li uvažovaný vztah $y = G^T x$ po složkách, obdržíme:

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}. \tag{1.9}$$

Následující algoritmus počítá Givensovu rotaci pro zadaný vektor $x = (a,b) \in \mathbb{R}^2$. Slouží k řešení soustavy rovnic (1.9), ve které hledáme čísla c a s taková, abychom vynulovali druhou složku vektoru y.

Algoritmus 1.6: ***Výpočet Givensovy rotace***

Vstup: *složky vektoru* $x \in \mathbb{R}^2$*: reálná čísla* a *a* b
Výstup: *čísla* $c, s \in \mathbb{R}$

if $b = 0$
  $c = 1$

$s = 0$
else
if $|b| > |a|$
$\tau = -a/b$
$s = 1/\sqrt{1+\tau^2}$
$c = s\tau$
else
$\tau = -b/a$
$c = 1/\sqrt{1+\tau^2}$
$s = c\tau$
end
end

Tuto funkci budeme označovat $[c, s] = \mathbf{givens}(a, b)$.

1.4.2 Konstrukce Givensova QR-rozkladu

Dalším postupem výpočtu QR-rozkladu matice může být využití Givensovy rotace. K poddiagonálním prvkům zadané matice $A \in \mathbb{R}^{n\times n}$ počítáme čísla c, s, která určují Givensovu rotaci a příslušný blok matice A potom zleva přenásobíme maticí rotace. Tím budeme v matici A získávat po diagonálou nuly. Matice A se v algoritmu přepisuje maticí $R \in \mathbb{R}^{n\times n}$. Platí $Q^T A = R$ a R je horní trojúhelníková a Q je ortonormální, $Q = G_1 \dots G_t$, kde t značí počet rotací.

Ukažme si příklad Givensovy rotace. U každé matice jedna Givensova rotace změní řádky, které jsou uváděny v závorkách.

$$A = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{(1,2)} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix} = R$$

V našem příkladu jsme použili tři Givensovy rotace, každá je charakterizovaná indexy i a k, které jsou uvedeny v závorkách a hodnotami c a s, které pro každou transformaci určujeme podle algoritmu 1.6.

Algoritmus 1.7: ***Givensův QR-rozklad***

Vstup: *matice* $A \in \mathbb{R}^{n\times n}$
Výstup: *matice* $A \in \mathbb{R}^{n\times n}$ *obsahující prvky matice* R

for $j = 1$ ***to*** n
for $i = n$ ***downto*** $j + 1$
$[c, s] = \mathbf{givens}(a_{i-1,j}, a_{ij})$
$[a_{kl}]_{k=i-1,i;\ l=j,\dots,n} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T [a_{kl}]_{k=i-1,i;\ l=j,\dots,n}$

end
end

1.4.3 Givensův QR-rozklad horní Hessenbergovy matice

V této podkapitole budeme uvažovat matice v Hessenbergově tvaru. Tento případ pro nás bude zajímavý tím, že v QR-algoritmu se Hessenbergův tvar zachovává, tj. všechny matice A_k mají tento tvar.

O QR-algoritmu pojednává následující kapitola, nyní jen předešleme, jak vypadá posloupnost matic A_k z tohoto algoritmu: za matici A_0 uvažujeme matici A, jejíž vlastní čísla chceme spočítat, provádíme QR-rozklad matice $A_k = Q_k R_k$ a další matici této posloupnosti získáme jako $A_{k+1} = R_k Q_k$.

Definice 1.3 *O čtvercové matici $A \in \mathbb{R}^{n\times n}$ řekneme, že je* ***(p,q)-pásová****, jestliže p a q jsou šířky minimálního naddiagonálního resp. poddiagonálního pásu, v němž jsou obsaženy všechny nenulové nediagonální prvky matice A.*

Příklad 1.1 *Matice, která je (2,1)-pásová:*

$$\begin{bmatrix} \times & \times & \times & & & \\ \times & & & \ddots & \boldsymbol{0} & \\ & \ddots & & & \ddots & \\ & & \ddots & & & \times \\ & \boldsymbol{0} & & \ddots & & \times \\ & & & & \times & \times \end{bmatrix}.$$

Speciální případy pásových matic:

- pokud $q = 0$, matice má horní trojúhelníkový tvar,
- pokud $p = 0$, matice má dolní trojúhelníkový tvar,
- pokud $p = 0$ a $q = 0$, matice je diagonální,
- pokud $p = 1$ a $q = 1$, matice je třidiagonální,
- pokud $q = 1$, matice je **Hessenbergova**, tj. má tvar:

$$A = \begin{bmatrix} \times & \cdots & \cdots & \cdots & \cdots & \times \\ \times & & & & & \vdots \\ & \ddots & & & & \vdots \\ & & \ddots & & & \vdots \\ & \mathbf{0} & & \ddots & & \vdots \\ & & & & \times & \times \end{bmatrix}.$$

Při QR-rozkladu Hessenbergovy matice užijeme Givensův QR-rozklad.
Po dvou krocích algoritmu bude matice vypadat takto:

$$G(2,3)^T G(1,2)^T A = \begin{bmatrix} \times & \times & \times & \dots & \dots & \times \\ 0 & \times & \times & \dots & \dots & \times \\ 0 & 0 & \times & & & \vdots \\ 0 & 0 & \times & & & \vdots \\ \vdots & & \ddots & \ddots & & \vdots \\ 0 & 0 & \dots & 0 & \times & \times \end{bmatrix}.$$

Jelikož je matice A v horním Hessenbergově tvaru, algoritmus se nám výrazně zjednoduší:

Algoritmus 1.8: ***Givensův QR-rozklad matice v Hessenbergově tvaru***

Vstup: *matice $A \in \mathbb{R}^{n\times n}$ v Hessenbergově tvaru*
Výstup: *matice $A \in \mathbb{R}^{n\times n}$ obsahující prvky matice R*

for $j = 1$ ***to*** $n-1$
$[c, s] =$ ***givens***$(a_{jj}, a_{j+1,j})$
$[a_{kl}]_{k=j,j+1;\, l=j,\dots,n} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}^T [a_{kl}]_{k=j,j+1;\, l=j,\dots,n}$
end

Ortonormální matici Q můžeme explicitně získat jako součin Givensových matic G_i, kde $G_i = G(i, i+1)$:

$$Q = G_1 G_2 \dots G_{n-1}.$$

Matice A_k z QR-algoritmu zachovávají Hessenbergův tvar nebo tvar reálné symetrické třídiagonální matice. Ukazuje to následující věta, která uvažuje obecnější tvar než Hessenbergův, a to tvar matice (n-1,q)-pásové.

Věta 1.3

i. Je-li matice A (n-1,q)-pásová, je i každá matice A_k z QR-algoritmu (n-1,q)-pásová.

ii. Je-li matice A reálná symetrická třídiagonální, je i každá matice A_k z QR-algoritmu reálná symetrická třídiagonální.

Důkaz. Viz [2] - věty 14.6 a 14.7 na str. 244 a 245. □

Kapitola 2

QR-algoritmus

Tato kapitola popisuje konstrukci QR-algoritmu. Naším cílem bude především dokázat konvergenci tohoto algoritmu k horní trojúhelníkové matici s vlastními čísly na diagonále.

Pro jednoduchost nejprve rozeberme případ, kdy má matice v absolutní hodnotě různá vlastní čísla. Uvedeme větu s důkazem. Pokusíme se ukázat, jaké prvky obsahují jednotlivé matice v důkazu a jak se asymptoticky chovají.

Budeme se zabývat rychlostí konvergence jednotlivých vlastních čísel. Zjistíme, že tato konvergence je logaritmická a že závisí na poměru mezi po sobě jdoucími vlastními čísly. Logaritmická konvergence nás přivedla na myšlenku Aitkenova δ^2-procesu, pomocí něhož budeme vlastní čísla extrapolovat.

Pro urychlení QR-algoritmu provedeme zmenšením počtu iterací pomocí shiftu. Shift je reálné číslo, pomocí něhož posuneme spektrum matice, a tak pozměníme poměry mezi jednotlivými vlastními čísly, čímž se konvergence urychlí. Někdy je použití shiftu nezbytné. Konvergence QR-algoritmu bez použití shiftu může být velice pomalá a bez shiftu by nebylo možné se dopracovat k výsledku.

V každé iteraci QR-algoritmu provádíme QR-rozklad matice. Provádění tohoto rozkladu může být náročné na velký počet operací. Proto se nesymetrická matice předem převedou na Hessenbergův tvar nebo na třídiagonální tvar, pokud se jedná o matici symetrickou. Tato transformace se provádí jedinkrát, pouze před započetím QR-algoritmu a ušetří nám potom v každé iteraci množství kroků, proto je výhodné ji využít.

Zde jsme vycházeli z výhodné vlastnosti QR-algoritmu: třídiagonální matice si v QR-algoritmu zachovává svůj třídiagonální tvar. (Stejě tak je zachováván tvar Hessenbergův.) Uvedeme algoritmy, které popisují převod matice na Hessenbergův nebo třídiagonální tvar.

2.1 Úvod

Naším úkolem je spočítat vlastní čísla matice $A_1 \in \mathbb{R}^{n\times n}$. Budeme definovat posloupnost matic $A_1, A_2, A_3, \ldots$ pomocí QR-rozkladu tímto způsobem: matici A_k rozložíme na součin ortonormální a horní trojúhelníkové matice

$$A_k = Q_k R_k$$

a matici A_{k+1} definujeme následovně:

$$A_{k+1} = R_k Q_k.$$

Ukážeme, že matice $A_k, k = 1, 2, 3, \ldots$ jsou podobné matici A_1, mají tedy stejná vlastní čísla a že posloupnost těchto matic konverguje k horní trojúhelníkové matici s vlastními čísly na diagonále.

Poznámka 2.1: Pro větší názornost prozatím předpokládáme, že vlastní čísla matice A jsou reálná.

Algoritmus 2.1: ***QR-algoritmus***

Vstup: *matice* $A \in \mathbb{R}^{n\times n}$
Výstup: *matice* $A_{k+1} \in \mathbb{R}^{n\times n}$ *ve tvaru horní trojúhelníkové matice*

$A_1 = A$
for $k = 1, 2 \ldots$
  $A_k = Q_k R_k$
  $A_{k+1} = R_k Q_k$
  test na ukončení
end

Test na ukončení algoritmu se v literatuře standardně uvádí tento:

$$\sum_{i>j} |a_{ij}| < \varepsilon \quad \Rightarrow \quad \text{konec}$$

Kromě tohoto testu, který zaručuje, že je matice A_k potřebně blízko trojúhelníkové matici, ještě stanovíme maximální počet iterací. To je z toho důvodu, abychom se nedostali do nekonečného cyklu, pokud by posloupnost matic konvergovala velice pomalu.

Nyní se podíváme na konvergenci matice A_1 k horní trojúhelníkové matici při použití QR-algoritmu a tuto konvergenci dokážeme. Nejdříve budeme potřebovat dokázat následující dvě lemmata.

Lemma 2.1 *Posloupnost matic* $A_k \in \mathbb{R}^{n\times n}$ *v QR-algoritmu je tvořena navzájem podobnými maticemi.*

Důkaz.

$$A_k = R_{k-1}Q_{k-1} = Q_{k-1}^T A_{k-1}Q_{k-1} = Q_{k-1}^T \dots Q_2^T Q_1^T A_1 Q_1 Q_2 \dots Q_{k-1}$$

Tedy:

$$A_k = Q_{k-1}^T \dots Q_2^T Q_1^T A_1 Q_1 Q_2 \dots Q_{k-1} \tag{2.1}$$

Zavedeme následující označení:

$$\mathfrak{Q}_k = Q_1 Q_2 \dots Q_{k-1} Q_k \text{ a } \mathfrak{R}_k = R_k R_{k-1} \dots R_2 R_1.$$

Potom můžeme rovnost (2.1) zapsat jako

$$A_k = \mathfrak{Q}_{k-1}^T A_1 \mathfrak{Q}_{k-1}. \tag{2.2}$$

Tímto jsme ukázali, že matice A_k je podobná matici A_1 pro všechna k.
Bude se nám ještě hodit vyjádření k-té mocniny matice A_1. Platí:

$$\overbrace{Q_1 Q_2 \dots Q_{k-1}(Q_k}^{\mathfrak{Q}_k} \overbrace{R_k) R_{k-1} \dots R_2 R_1}^{\mathfrak{R}_k} = Q_1 Q_2 \dots Q_{k-1} A_k R_{k-1} \dots R_2 R_1 \overset{(2.1)}{=}$$

$$= A_1 Q_1 Q_2 \dots (Q_{k-1} R_{k-1}) \dots R_2 R_1.$$

Tímto způsobem ještě $(k-1)$-krát pokračujeme a využíváme při tom rovnost (2.1). Získáme

$$Q_1 Q_2 \dots Q_{k-1}(Q_k R_k) R_{k-1} \dots R_2 R_1 = A_1^k = \mathfrak{Q}_k \mathfrak{R}_k.$$

□

Lemma 2.2 *Mějme matici $G_k \in \mathbb{R}^{n\times n}$, $G_k \to \mathbf{0}$. Provedeme-li QR-rozklad matice $(I + G_k)$:*

$$I + G_k = Q_k R_k, \tag{2.3}$$

můžeme Q_k a R_k vyjádřit tímto způsobem:

$$Q_k = (I + Q_G^{(k)}),\ kde\ Q_G^{(k)} \to \mathbf{0},$$

$$R_k = (I + R_G^{(k)}),\ kde\ R_G^{(k)} \to \mathbf{0}.$$

Důkaz. Z rovnosti (2.3) dostáváme vynásobením zleva maticí Q_k^T vztah

$$Q_k^T + Q_k^T G_k = R_k.$$

Přenásobíme maticí R_k^T zleva a dostaneme

$$R_k^T Q_k^T + R_k^T Q_k^T G_k = R_k^T R_k.$$

Matice $G_k \to \mathbf{0}$, proto i $R_k^T Q_k^T G_k \to \mathbf{0}$. Platí tedy $I + G_k \to R_k^T R_k$, proto platí

$$R_k^T R_k - I \to \mathbf{0}.$$

Nechť

$$R_k = \begin{bmatrix} r_{11}^{(k)} & r_{12}^{(k)} & \dots & r_{1n}^{(k)} \\ 0 & r_{22}^{(k)} & \dots & r_{2n}^{(k)} \\ \vdots & & & \\ 0 & 0 & 0 & r_{nn}^{(k)} \end{bmatrix}.$$

Provedeme-li součin $R_k^T R_k$, obdržíme tuto matici:

$$(R_k^T R_k)_{ij} = \sum_{s=1}^{min\{i,j\}} r_{si}^{(k)} r_{sj}^{(k)}. \tag{2.4}$$

Platí $R_k^T R_k \to I$, tedy mimodiagonální prvky matice $R_k^T R_k$ kovergují k nule a diagonální prvky k jedničce.

Bez újmy na obecnosti ukážeme řešení soustavy pro $n = 3$. Ze vztahu (2.4) postupně pro prvky s indexy (i, j) obdržíme:

$$\begin{array}{lrcrcrcr}
(1,1) & r_{11}^{(k)} r_{11}^{(k)} & = & 1 + O(1) & \Rightarrow & r_{11}^{(k)} & = & 1 + O(1) \\
 & \text{(uvažujeme kladné diagonální prvky)} & & & & & & \\
(1,2) & r_{11}^{(k)} r_{12}^{(k)} & = & O(1) & \Rightarrow & r_{12}^{(k)} & = & O(1) \\
(1,3) & r_{11}^{(k)} r_{13}^{(k)} & = & O(1) & \Rightarrow & r_{13}^{(k)} & = & O(1) \\
(2,2) & r_{12}^{(k)} r_{12}^{(k)} + r_{22}^{(k)} r_{22}^{(k)} & = & 1 + O(1) & \Rightarrow & r_{22}^{(k)} & = & 1 + O(1) \\
(2,3) & r_{12}^{(k)} r_{13}^{(k)} + r_{22}^{(k)} r_{23}^{(k)} & = & O(1) & \Rightarrow & r_{23}^{(k)} & = & O(1) \\
(3,3) & r_{13}^{(k)} r_{13}^{(k)} + r_{23}^{(k)} r_{23}^{(k)} + r_{33}^{(k)} r_{33}^{(k)} & = & 1 + O(1) & \Rightarrow & r_{33}^{(k)} & = & 1 + O(1)
\end{array}$$

Obecně bude platit $r_{ij}^{(k)} \to 0$ pro $i < j$ a $r_{ij}^{(k)} \to 1$ pro $i = j$.

Matici R_k tedy můžeme vyjádřit jako $I + R_G^{(k)}$, kde $R_G^{(k)} \to \mathbf{0}$.

Z (2.3) máme $I + G_k = Q_k(I + R_G^{(k)})$. Z této rovnosti získáme vztah

$$Q_k - I = \underbrace{G_k - Q_k R_G^{(k)}}_{\to \mathbf{0}}.$$

Platí $Q_k \to I$, můžeme tedy Q_k vyjádřit jako $I + Q_G^{(k)}$, kde $Q_G^{(k)} \to \mathbf{0}$. □

2.2 Konvergence QR-algoritmu

Pro větší názornost a srozumitelnost rozebereme nejprve případ matice A, která má v absolutní hodnotě navzájem různá vlastní čísla.

2.2.1 Matice s různými vlastními čísly

Tato podkapitola obsahuje důkaz konvergence matice A_1 k horní trojúhelníkové matici. Nejdříve budeme tedy předpokládat, že má matice A_1 v absolutní hodnotě navzájem různá vlastní čísla. V podkapitole 2.2.3 tento předpoklad odstraníme.

Věta 2.1 *Předpokládejme, že matice $A_1 \in \mathbb{R}^{n\times n}$ je regulární a platí*

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|. \tag{2.5}$$

Předpokládejme dále, že matice $X^{-1} \in \mathbb{R}^{n\times n}$, která vznikne při rozkladu matice A_1 na Jordanův tvar $A_1 = XDX^{-1}$, má regulární všechny hlavní minory. Potom posloupnost matic A_k při použití QR-algoritmu konverguje k horní trojúhelníkové matici.

Důkaz. Matici A_1 převedeme na Jordanův tvar $A_1 = XDX^{-1}$. Protože jsou λ v absolutní hodnotě různá vlastní čísla, bude Jordanův tvar diagonální maticí $diag(\lambda_i)$, kterou označíme D. Matice X je samozřejmě regulární. Pro zjednodušení zápisu si X^{-1} označíme jako Y.

$$A_1 = X\mathrm{diag}(\lambda_i)X^{-1} = XDY.$$

Provedeme LR-rozklad matice Y. Rozklad existuje tehdy, když má matice regulární všechny hlavní minory.

$$A_1^k = XD^kY = XD^kL_YR_Y = X(D^kL_YD^{-k})(D^kR_Y)$$

Matice L_Y je dolní trojúhelníková, její diagonální prvky jsou jedničky. Označíme-li $D^kL_YD^{-k} = I + F_k$, matice F_k bude vypadat následovně:

$$(F_k)_{ij} = \begin{cases} (L_Y)_{ij}(\lambda_i/\lambda_j)^k & i > j \\ 0 & i \le j \end{cases} \tag{2.6}$$

Tedy pro $k \to \infty$ je $F_k \to \mathbf{0}$. Matice X je regulární, má proto QR-rozklad a můžeme psát:

$$A_1^k = Q_XR_X(I + F_k)D^kR_Y = Q_X(I + R_XF_kR_X^{-1})R_XD^kR_Y = Q_X(I + G_k)R_XD^kR_Y,$$

kde $I + R_XF_kR_X^{-1}$ jsme si označili jako $I + G_k$. Zřejmě tedy i pro G_k platí, že pro $k \to \infty$ je $G_k \to \mathbf{0}$.

Matice $I + G_k$ má QR-rozklad:

$$I + G_k = Q_kR_k. \tag{2.7}$$

Podle lemmatu (2.2) můžeme tento rozklad vyjádřit jako:

$$I + G_k = (I + Q_G^{(k)})(I + R_G^{(k)}).$$

Ukázali jsme, že pro $k \to \infty$ platí $Q_G^{(k)} \to \mathbf{0}$ a $R_G^{(k)} \to \mathbf{0}$.

Máme tedy

$$A_1^k = Q_X(I + Q_G^{(k)})(I + R_G^{(k)})R_X D^k R_Y = \mathfrak{Q}_k \mathfrak{R}_k,$$

označení $\mathfrak{Q}_k, \mathfrak{R}_k$ jsme použili stejné jako v Lemmatu 2.1, kde jsme označili $\mathfrak{Q}_k = Q_1 Q_2 \dots Q_k$ a $\mathfrak{R}_k = R_k R_{k-1} \dots R_1$.

$$\mathfrak{Q}_k = Q_X(I + Q_G^{(k)}) \to Q_X$$

$$A_{k+1} \overset{(2.2)}{=} \mathfrak{Q}_k^T X D X^{-1} \mathfrak{Q}_k \to Q_X^T X D X^{-1} Q_X = R_X D R_X^{-1}$$

A_k tedy s ohledem na předpoklad (2.5) konverguje k horní trojúhelníkové matici (s vlastními čísly na diagonále). □

Zajímá nás, jak vypadá asymptotický rozvoj matic $Q_G^{(k)}$ a $R_G^{(k)}$. Pro názornost a stručnost zápisu to ukážeme na matici řádu 2×2. I když řadu prvků jsme schopni popsat přesně, pro jednoduchost zápisu se budeme vyjadřovat pomocí symbolu $O()$. Obecný případ půjde z případu 2×2 odvodit.

Matice F_k bude vypadat takto:

$$F_k = \begin{bmatrix} 0 & 0 \\ l_{21}(\frac{\lambda_2}{\lambda_1})^k & 0 \end{bmatrix}.$$

Tuto matici násobíme maticemi R_X a R_X^{-1}, jejich prvky označíme tímto způsobem:

$$R_X = \begin{bmatrix} r_1 & r_2 \\ 0 & r_4 \end{bmatrix},$$

$$R_X^{-1} = \begin{bmatrix} z_1 & z_2 \\ 0 & z_4 \end{bmatrix}.$$

Součin $R_X F_k R_X^{-1}$ bude vypadat takto:

$$R_X F_k R_X^{-1} = \begin{bmatrix} r_2 z_1 l_{21}(\frac{\lambda_2}{\lambda_1})^k & r_2 z_2 l_{21}(\frac{\lambda_2}{\lambda_1})^k \\ r_1 z_1 l_{21}(\frac{\lambda_2}{\lambda_1})^k & r_1 z_2 l_{21}(\frac{\lambda_2}{\lambda_1})^k \end{bmatrix}.$$

Prvky této matice označíme takto:

$$R_X F_k R_X^{-1} = (\frac{\lambda_2}{\lambda_1})^k \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} = G_k.$$

Vyjádříme matici $I + G_k$:

$$I + G_k = (\frac{\lambda_2}{\lambda_1})^k \begin{bmatrix} (\frac{\lambda_1}{\lambda_2})^k & 0 \\ 0 & (\frac{\lambda_1}{\lambda_2})^k \end{bmatrix} + (\frac{\lambda_2}{\lambda_1})^k \begin{bmatrix} b_1 & b_2 \\ b_3 & b_4 \end{bmatrix} =$$

$$= (\frac{\lambda_2}{\lambda_1})^k \begin{bmatrix} (\frac{\lambda_1}{\lambda_2})^k + b_1 & b_2 \\ b_3 & (\frac{\lambda_1}{\lambda_2})^k + b_4 \end{bmatrix}.$$

Matici označíme symbolem H_k:

$$\begin{bmatrix} (\frac{\lambda_1}{\lambda_2})^k + b_1 & b_2 \\ b_3 & (\frac{\lambda_1}{\lambda_2})^k + b_4 \end{bmatrix} = H_k.$$

Chceme provést QR-rozklad této matice. Použijeme Givensův QR-rozklad. Najdeme tedy prvky c_k a s_k Givensovy matice Q_k:

$$c_k = \frac{(\frac{\lambda_1}{\lambda_2})^k + b_1}{\sqrt{((\frac{\lambda_1}{\lambda_2})^k + b_1)^2 + b_3^2}},\ s_k = \frac{b_3}{\sqrt{((\frac{\lambda_1}{\lambda_2})^k + b_1)^2 + b_3^2}}.$$

Vidíme, že $c_k \to 1$ a $s_k \to 0$. Vytkneme-li z čitatele i jmenovatele prvku c_k člen $(\frac{\lambda_1}{\lambda_2})^k$, získáme

$$c_k = \frac{1 + (\frac{\lambda_2}{\lambda_1})^k b_1}{\sqrt{1 + 2b_1(\frac{\lambda_2}{\lambda_1})^k + (b_1^2 + b_3^2)(\frac{\lambda_2}{\lambda_1})^{2k}}}.$$

Nyní vidíme, jak se prvky c_k a s_k chovají:

$$c_k = 1 + O(\frac{\lambda_2}{\lambda_1})^k,\ s_k = O(\frac{\lambda_2}{\lambda_1})^k.$$

Vynásobíme $Q_k H_k$ (provedení Givensovy rotace):

$$Q_k H_k = \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix} \begin{bmatrix} (\frac{\lambda_1}{\lambda_2})^k + b_1 & b_2 \\ b_3 & (\frac{\lambda_1}{\lambda_2})^k + b_4 \end{bmatrix} =$$

$$= \begin{bmatrix} c_k((\frac{\lambda_1}{\lambda_2})^k + b_1) + s_k b_3 & c_k b_2 + s_k b_4 (\frac{\lambda_1}{\lambda_2})^k \\ 0 & -s_k b_2 + c_k b_4 (\frac{\lambda_1}{\lambda_2})^k \end{bmatrix}.$$

Rozklad matice $I + G_k$ tedy bude vypadat takto:

$$Q_k^{-1}(\frac{\lambda_2}{\lambda_1})^k \begin{bmatrix} c_k((\frac{\lambda_1}{\lambda_2})^k + b_1) + s_k b_3 & c_k b_2 + s_k b_4 (\frac{\lambda_1}{\lambda_2})^k \\ 0 & -s_k b_2 + c_k b_4 (\frac{\lambda_1}{\lambda_2})^k \end{bmatrix} =$$

$$= Q_k^{-1} \begin{bmatrix} c_k + c_k b_1 (\frac{\lambda_2}{\lambda_1})^k + s_k b_3 (\frac{\lambda_2}{\lambda_1})^k & c_k b_2 (\frac{\lambda_2}{\lambda_1})^k + s_k b_4 \\ 0 & -s_k b_2 (\frac{\lambda_2}{\lambda_1})^k + c_k b_4 \end{bmatrix} =$$

$$= \begin{bmatrix} 1 + O(\frac{\lambda_2}{\lambda_1})^k & O(\frac{\lambda_2}{\lambda_1})^k \\ O(\frac{\lambda_2}{\lambda_1})^k & 1 + O(\frac{\lambda_2}{\lambda_1})^k \end{bmatrix} \begin{bmatrix} 1 + O(\frac{\lambda_2}{\lambda_1})^k & O(\frac{\lambda_2}{\lambda_1})^k \\ 0 & 1 + O(\frac{\lambda_2}{\lambda_1})^k \end{bmatrix}.$$

Z tohoto příkladu 2×2 zkusíme odvodit, jak by vypadal obecný případ. Součin matic $R_X F_k R_X^{-1}$ bude vypadat takto:

$$\begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & \dots & r_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & r_{nn} \end{bmatrix} \begin{bmatrix} 0 & & & \\ l_{21}(\frac{\lambda_2}{\lambda_1})^k & & \mathbf{0} & \\ \vdots & \ddots & & \\ l_{n1}(\frac{\lambda_n}{\lambda_1})^k & \dots & l_{n,n-1}(\frac{\lambda_n}{\lambda_{n-1}})^k & 0 \end{bmatrix} \begin{bmatrix} z_{11} & z_{12} & \dots & z_{1n} \\ & z_{22} & \dots & z_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & z_{nn} \end{bmatrix} =$$

$$= \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & \dots & r_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & r_{nn} \end{bmatrix} \begin{bmatrix} 0 & 0 & \dots \\ z_{11}l_{21}(\frac{\lambda_2}{\lambda_1})^k & z_{12}l_{21}(\frac{\lambda_2}{\lambda_1})^k & \dots \\ \vdots & \vdots & \\ z_{11}l_{n1}(\frac{\lambda_n}{\lambda_1})^k & z_{12}l_{n1}(\frac{\lambda_n}{\lambda_1})^k + z_{22}l_{n2}(\frac{\lambda_n}{\lambda_2})^k & \dots \end{bmatrix} =$$

$$= \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1n} \\ & r_{22} & \dots & r_{2n} \\ & & \ddots & \vdots \\ \mathbf{0} & & & r_{nn} \end{bmatrix} \begin{bmatrix} 0 & 0 & \dots & 0 \\ O(\frac{\lambda_2}{\lambda_1})^k & O(\frac{\lambda_2}{\lambda_1})^k & \dots & O(\frac{\lambda_2}{\lambda_1})^k \\ \vdots & \vdots & & \vdots \\ O(\frac{\lambda_n}{\lambda_1})^k & O(\frac{\lambda_n}{\lambda_2})^k & \dots & O(\frac{\lambda_n}{\lambda_{n-1}})^k \end{bmatrix}.$$

V obecném případě nelze porovnat hodnoty $\frac{\lambda_i}{\lambda_{i-1}}$ a $\frac{\lambda_{i+1}}{\lambda_i}$ pro $i = 2, \dots, n-1$, nám ale stačí větší z těchto hodnot. Označíme $q_1 = \frac{\lambda_2}{\lambda_1}$, $q_i = \max\{\frac{\lambda_i}{\lambda i-1}, \frac{\lambda_{i+1}}{\lambda i}\}$ pro $i = 2, \dots n-1$ a $q_n = \frac{\lambda_n}{\lambda_{n-1}}$. Podstatné je to, že $|q_i| < 1$.

$$R_X F_k R_X^{-1} = \begin{bmatrix} O(q_1)^k & O(q_2)^k & O(q_3)^k & \dots & O(q_n)^k \\ O(q_1)^k & O(q_2)^k & O(q_3)^k & \dots & O(q_n)^k \\ O(\frac{\lambda_3}{\lambda_1})^k & O(\frac{\lambda_3}{\lambda_2})^k & O(\frac{\lambda_3}{\lambda_2})^k & \dots & O(\frac{\lambda_3}{\lambda_2})^k \\ O(\frac{\lambda_4}{\lambda_1})^k & O(\frac{\lambda_4}{\lambda_2})^k & O(\frac{\lambda_4}{\lambda_3})^k & \dots & O(\frac{\lambda_4}{\lambda_3})^k \\ \vdots & & & & \\ O(\frac{\lambda_n}{\lambda_1})^k & O(\frac{\lambda_n}{\lambda_2})^k & O(\frac{\lambda_n}{\lambda_3})^k & \dots & O(\frac{\lambda_n}{\lambda_{n-1}})^k \end{bmatrix}.$$

Matici $R_X F_k R_X^{-1}$ jsme opět označili jako G_k. Vyjádříme matici $I + G_k$:

$$I + G_k = \begin{bmatrix} 1 + O(q_1)^k & O(q_2)^k & O(q_3)^k & \dots & O(q_n)^k \\ O(q_1)^k & 1 + O(q_2)^k & O(q_3)^k & \dots & O(q_n)^k \\ O(\frac{\lambda_3}{\lambda_1})^k & O(\frac{\lambda_3}{\lambda_2})^k & 1 + O(\frac{\lambda_3}{\lambda_2})^k & \dots & O(\frac{\lambda_3}{\lambda_2})^k \\ O(\frac{\lambda_4}{\lambda_1})^k & O(\frac{\lambda_4}{\lambda_2})^k & O(\frac{\lambda_4}{\lambda_3})^k & \dots & O(\frac{\lambda_4}{\lambda_3})^k \\ \vdots & & & & \\ O(\frac{\lambda_n}{\lambda_1})^k & O(\frac{\lambda_n}{\lambda_2})^k & O(\frac{\lambda_n}{\lambda_3})^k & \dots & 1 + O(\frac{\lambda_n}{\lambda_{n-1}})^k \end{bmatrix}.$$

Vyjádřit obecně QR-rozklad této matice je složité. Lze však usoudit, jak by vypadalo asymptotické chování této matice. Prvky vykazují logaritmické chování, proto lze předpokládat použitelnost Aitkenova δ^2-procesu.

2.2.2 Aitkenův δ^2-proces

Teoreticky jsme zjistili, že v QR-algoritmu konvergují diagonální prvky matice A k jejím vlastním číslům logaritmicky. Zkusíme si tuto zkutečnost ilustrovat na příkladu:

Příklad 2.1 *Ukažme, jak probíhá konvergence QR-algoritmu aplikovaného na matici A. Jak rychle konvergují jednotlivé diagonální prvky k vlastním číslům?*

$$A = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 \\ 1 & 90 & 0 & 0 & 0 \\ 0 & 1 & 63 & 0 & 0 \\ 0 & 0 & 1 & 21 & 0 \\ 0 & 0 & 0 & 1 & 2.1 \end{bmatrix}.$$

Vlastní čísla této matice jsou 100, 90, 63, 21, *a* $2,1$. *Pro rychlost konvergence je rozhodující poměr mezi těmito vlastními čísly. Tyto poměry jsou následující:*

$$\frac{\lambda_2}{\lambda_1} = 9:10,\ \frac{\lambda_3}{\lambda_2} = 7:10,\ \frac{\lambda_4}{\lambda_3} = 1:3,\ \frac{\lambda_5}{\lambda_4} = 1:10.$$

Čím je poměr mezi vlastními čísly větší, tím příslušná hodnota k přesnému vlastnímu číslu konverguje rychleji. Můžeme se o tom přesvědčit v tabulce, kde po jednotlivých iteracích vypisujeme rozdíly spočítaných a přesných vlastních čísel.

iterace	$\lvert\lambda_1 - \lambda_1^*\rvert$	$\lvert\lambda_2 - \lambda_2^*\rvert$	$\lvert\lambda_3 - \lambda_3^*\rvert$	$\lvert\lambda_4 - \lambda_4^*\rvert$	$\lvert\lambda_5 - \lambda_5^*\rvert$
1	*0,008999*	*0,001221*	*0,002487*	*0,000538*	*0,004752*
2	*0,015385*	*0,006130*	*0,006903*	*0,001829*	*0,000522*
3	*0,019745*	*0,011398*	*0,007497*	*0,000796*	*5,27E-05*
4	*0,022544*	*0,015786*	*0,006467*	*0,000285*	*5,27E-06*
5	*0,024151*	*0,018972*	*0,005082*	*9,71E-05*	*5,27E-07*
6	*0,024860*	*0,021014*	*0,003813*	*3,25E-05*	*5,27E-08*
7	*0,024900*	*0,022100*	*0,002789*	*1,08E-05*	*5,27E-09*
8	*0,024454*	*0,022441*	*0,002009*	*3,63E-06*	*5,27E-10*
9	*0,023661*	*0,022226*	*0,001433*	*1,21E-06*	*5,27E-11*
10	*0,022631*	*0,021614*	*0,001016*	*4,03E-07*	*5,28E-12*
11	*0,021449*	*0,020731*	*0,000718*	*1,34E-07*	*5,30E-13*
12	*0,020179*	*0,019673*	*0,000505*	*4,48E-08*	*5,00E-14*
13	*0,018869*	*0,018513*	*0,000355*	*1,49E-08*	
14	*0,017554*	*0,017304*	*0,000249*	*4,98E-09*	
15	*0,016262*	*0,016086*	*0,000175*	*1,66E-09*	
16	*0,015010*	*0,014887*	*0,000122*	*5,53E-10*	
17	*0,013812*	*0,013726*	*8,60E-05*	*1,84E-10*	
18	*0,012676*	*0,012616*	*6,02E-05*	*6,14E-11*	
19	*0,011607*	*0,011565*	*4,22E-05*	*2,04E-11*	
20	*0,010607*	*0,010578*	*2,95E-05*	*6,81E-12*	
21	*0,009677*	*0,009656*	*2,07E-05*	*2,27E-12*	
22	*0,008814*	*0,008800*	*1,44E-05*	*7,30E-13*	
23	*0,008018*	*0,008008*	*1,01E-05*	*2,20E-13*	
24	*0,007285*	*0,007278*	*7,10E-06*	*4,00E-14*	
25	*0,006613*	*0,006608*	*4,97E-06*		
26	*0,005997*	*0,005993*	*3,48E-06*		
27	*0,005433*	*0,005431*	*2,43E-06*		
28	*0,004920*	*0,004918*	*1,70E-06*		
29	*0,004452*	*0,004451*	*1,19E-06*		
30	*0,004026*	*0,004025*	*8,36E-07*		
31	*0,003639*	*0,003638*	*5,85E-07*		
32	*0,003288*	*0,003287*	*4,09E-07*		
33	*0,002969*	*0,002969*	*2,86E-07*		
34	*0,002681*	*0,002681*	*2,00E-07*		
35	*0,002419*	*0,002419*	*1,40E-07*		
36	*0,002183*	*0,002183*	*9,83E-08*		
37	*0,001969*	*0,001969*	*6,88E-08*		
38	*0,001776*	*0,001776*	*4,821E-08*		
39	*0,001601*	*0,001601*	*3,37E-08*		
40	*0,001443*	*0,001443*	*2,36E-08*		
41	*0,001301*	*0,001301*	*1,65E-08*		
42	*0,001172*	*0,001172*	*1,15E-08*		
43	*0,001056*	*0,001056*	*8,10E-09*		
44	*0,000951*	*0,000951*	*5,67E-09*		
45	*0,000857*	*0,000857*	*3,96E-09*		
46	*0,000772*	*0,000772*	*2,77E-09*		
47	*0,000695*	*0,000695*	*1,94E-09*		
48	*0,000626*	*0,000626*	*1,36E-09*		
49	*0,000564*	*0,000564*	*9,53E-10*		
50	*0,000508*	*0,000508*	*6,67E-10*		
51	*0,000457*	*0,000457*	*4,67E-10*		

Grafické znazornění logaritmu odchylek spočítaných vlastních čísel od skutečných hodnot:

Tím se nám prakticky potvrdila úvaha s logaritmickou konvergencí, zkusíme tedy použít Aitkenův δ^2-proces.

V uvedeném příkladu jsme počítali logaritmus odchylky spočítaného vlastního čísla (λ_k, kde písmenem k označíme $k-tou$ iteraci) od skutečného vlastního čísla (λ^*) a do grafu jsme kreslili hodnoty

$$ln|\lambda_k - \lambda^*|.$$

Zjistili jsme, že graf s těmito hodnotami je velice podobný přímce (z toho důvodu, že rozdíly $\lambda_k - \lambda^*$ klesají logaritmicky). Znamená to tedy, že můžeme uvažovat tuto rovnost:

$$ln|\lambda_k - \lambda^*| - ln|\lambda_{k-1} - \lambda^*| \doteq q, \text{ kde } q \text{ je konst.}$$

Můžeme napsat

$$ln\frac{|\lambda_k - \lambda^*|}{|\lambda_{k-1} - \lambda^*|} \doteq q$$

a uvažujeme-li rozdíly ve zlomku prozatím kladné (níže rozebereme znaménko těchto rozílů) a označíme-li konstantu e^q jako c, bude platit:

$$\frac{\lambda_k - \lambda^*}{\lambda_{k-1} - \lambda^*} \doteq c.$$

Vezmeme-li tento vzorec také pro iteraci $k+1$ a vynásobíme-li jmenovateli, dostaneme dvě rovnice:

$$\begin{array}{rcl} \lambda_k - \lambda^* & \doteq & c(\lambda_{k-1} - \lambda^*) \\ \lambda_{k+1} - \lambda^* & \doteq & c(\lambda_k - \lambda^*) \end{array}.$$

Když křížem vynásobíme tyto rovnice a vykrátíme c, získáme vztah:

$$(\lambda_k - \lambda^*)^2 \doteq (\lambda_{k+1} - \lambda^*)(\lambda_{k-1} - \lambda^*).$$

Z tohoto vztahu lze vyjádřit, jak vypadá přesná hodnota λ^*:

$$\lambda^* \doteq \frac{\lambda_{k+1}\lambda_{k-1} - \lambda_k^2}{\lambda_{k+1} + \lambda_{k-1} - 2\lambda_k}. \tag{2.8}$$

V poslední kapitole s numerickými výpočty ukážeme, jak lze pomocí této extrapolace urychlit výpočet vlastních čísel.

2.2.3 Matice s některými stejnými vlastními čísly

Věta 2.2 *Nechť platí předpoklady předchozí věty, kromě předpokladu*

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|,$$

který nahradíme předpokladem

$$\begin{gathered} |\lambda_1| > \cdots > |\lambda_r| \\ \lambda_r = \lambda_{r+1} = \cdots = \lambda_t \\ |\lambda_t| > |\lambda_{t+1}| > \cdots > |\lambda_n|. \end{gathered}$$

Posloupnost matic A_k bude opět konvergovat k horní trojúhelníkové matici.

Důkaz. Matice D bude vypadat takhle:

$$D = \begin{bmatrix} \lambda_1 & & & & & & \\ & \ddots & & & & \mathbf{0} & \\ & & \lambda_r & & & & \\ & & & \ddots & & & \\ & & & & \lambda_t & & \\ & \mathbf{0} & & & & \ddots & \\ & & & & & & \lambda_n \end{bmatrix}$$

Oproti předchozímu důkazu rozložíme $D^k L_Y D^{-k} = \widetilde{L} + \widetilde{F}_k$.
Matice $\widetilde{L}$ vypadá takto:

$$\widetilde{L} = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & \mathbf{0} & \\ & & 1 & & & & & \\ & & & \times & \dots & \times & & \\ & & & \vdots & & \vdots & & \\ & & & \times & \dots & \times & & \\ & & & & & & 1 & \\ & \mathbf{0} & & & & & & \ddots \\ & & & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ r-1 \\ r \\ \\ t \\ t+1 \\ \\ \end{matrix}$$

, kde $\times$ značí prvky maticeL_Y

a matice $\widetilde{F_k}$ má tento tvar:

$$\widetilde{F_k} = \begin{bmatrix} 0 & & & & & & & & \\ \times & \ddots & & & & & & \mathbf{0} & \\ \vdots & \ddots & 0 & & & & & & \\ \vdots & & \times & 0 & \dots & 0 & & & \\ \vdots & & \vdots & \vdots & & \vdots & & & \\ \vdots & & \times & 0 & \dots & 0 & & & \\ \vdots & & \times & \times & \dots & \times & 0 & & \\ \vdots & & & & & & \ddots & \ddots & \\ \times & \dots & \dots & \dots & \dots & \dots & \dots & \times & 0 \end{bmatrix}, \text{ kde } \times \text{ jsou prvky matice (2.6).}$$

Vyjádříme matici A_1^k:

$$A_1^k = X(\widetilde{L} + \widetilde{F}_k)D^k R_Y = X\widetilde{L}(I + \widetilde{L}^{-1}\widetilde{F}_k)D^k R_Y.$$

Předpokládáme, že existuje QR-rozklad matice $X\widetilde{L}$:

$$A_1^k = \widetilde{Q}_X\widetilde{R}_X(I + \widetilde{L}^{-1}\widetilde{F}_k)D^k R_Y = \widetilde{Q}_X(I + \widetilde{R}_X\widetilde{L}^{-1}\widetilde{F}_k\widetilde{R}_X^{-1})\widetilde{R}_X D^k R_Y,$$

využijeme toho, že $\widetilde{F}_k \to \mathbf{0}$ a stejným způsobem jako v předchozí větě vyjádříme QR-rozklad matice $I + \widetilde{R}_X\widetilde{L}^{-1}\widetilde{F}_k\widetilde{R}_X^{-1}$:

$$I + \widetilde{R}_X\widetilde{L}^{-1}\widetilde{F}_k\widetilde{R}_X^{-1} = (I + Q_G^{(k)})(I + R_G^{(k)})$$

a opět platí $\mathfrak{Q}_k = \widetilde{Q}_X(I + Q_G^{(k)})$.

Pro matici A_{k+1} tedy bude platit

$$A_{k+1} = \mathfrak{Q}_k^T X D X^{-1}\mathfrak{Q}_k = (I + Q_G^{(k)})^T\widetilde{Q}_X^T X D X^{-1}\widetilde{Q}_X(I + Q_G^{(k)}).$$

Matici $X\widetilde{L}$ jsme rozložili na $\widetilde{Q}_X\widetilde{R}_X$, můžeme tedy psát:

$$A_{k+1} = \underbrace{(I + Q_G^{(k)})^T}_{\to I}\widetilde{R}_X\underbrace{\widetilde{L}^{-1}D\widetilde{L}}\widetilde{R}_X^{-1}\underbrace{(I + Q_G^{(k)})}_{\to I}. \tag{2.9}$$

Ukážeme, že platí vztah $\widetilde{L}^{-1}D\widetilde{L} = D$.

Vyjádříme, jak vypadají matice $D\widetilde{L}$ a $\widetilde{L}D$:

$$D\widetilde{L} = \begin{bmatrix} \lambda_1 & & & & & & & & & \\ & \ddots & & & & & & & \mathbf{0} & \\ & & \lambda_{r-1} & & & & & & & \\ & & & \lambda_r \cdot \times & \lambda_r \cdot \times & \dots & \lambda_r \cdot \times & & & \\ & & & \lambda_{r+1} \cdot \times & \lambda_{r+1} \cdot \times & \dots & \lambda_{r+1} \cdot \times & & & \\ & & & \vdots & & & \vdots & & & \\ & & & \lambda_t \cdot \times & \lambda_t \cdot \times & \dots & \lambda_t \cdot \times & & & \\ & & & & & & & \lambda_t + 1 & & \\ & & \mathbf{0} & & & & & & \ddots & \\ & & & & & & & & & \lambda_n \end{bmatrix},$$

symboly $\times$ značí prvky matice $\widetilde{L}$.

$$\widetilde{L}D = \begin{bmatrix} \lambda_1 & & & & & & & & & \\ & \ddots & & & & & & & \mathbf{0} & \\ & & \lambda_{r-1} & & & & & & & \\ & & & \lambda_r \cdot \times & \lambda_{r+1} \cdot \times & \dots & \lambda_t \cdot \times & & & \\ & & & \lambda_r \cdot \times & \lambda_{r+1} \cdot \times & \dots & \lambda_t \cdot \times & & & \\ & & & \vdots & & & \vdots & & & \\ & & & \lambda_r \cdot \times & \lambda_{r+1} \cdot \times & \dots & \lambda_t \cdot \times & & & \\ & & & & & & & \lambda_t + 1 & & \\ & & \mathbf{0} & & & & & & \ddots & \\ & & & & & & & & & \lambda_n \end{bmatrix}.$$

Jelikož platí $\lambda_r = \lambda_{r+1} = \cdots = \lambda_t$, jsou tyto matice stejné. Platí tedy $D\widetilde{L} = \widetilde{L}D$, což není nic jiného než

$$\widetilde{L}^{-1} D \widetilde{L} = D.$$

Podle vztahu (2.9) tedy můžeme říct, že matice A_k opět konverguje k horní trojúhelníkové matici:

$$A_{k+1} \overset{k\to\infty}{\longrightarrow} \widetilde{R}_X D \widetilde{R}_X^{-1}.$$

□

Ukažme si příklad matice s hodně blízkými vlastními čísly:

Příklad 2.2 *Zajímají nás vlastní čísla matice A:*

$$A = \begin{bmatrix} -2,37794438 & -0,19200000 & 0,18812081 & -0,18432000 & 0,90297990 \\ -18,71605025 & -0,12064653 & 1,09800487 & -1,07582067 & 5,27042339 \\ -24,25652135 & -0,15136653 & 2,12810420 & -1,10531187 & 5,41490018 \\ -28,72696998 & -0,12126720 & 2,09861300 & -0,07641651 & 5,27334242 \\ -17,42651025 & -0,94975119 & 1,13056232 & -0,90772032 & 5,44690322 \end{bmatrix}.$$

Pomocí programu z balíku Lapack jsme získali tyto výsledky:

$$\begin{aligned}
\lambda_1 &= 1,015354576 + 0.0110281i \\
\lambda_2 &= 1,015354576 - 0.0110281i \\
\lambda_3 &= 0,994244953 + 0.0182075i \\
\lambda_4 &= 0,994244953 - 0.0182075i \\
\lambda_5 &= 0,980800942
\end{aligned}$$

Pomocí QR-algoritmu, který jsme naprogramovali ve Fortranu jsme se dopracovali k výsledkům těmto:

$$\begin{aligned}
\lambda_1 &= 1,015354571 + 0,0110281i \\
\lambda_2 &= 1,015354571 - 0,0110281i \\
\lambda_3 &= 0,994244954 + 0,0182074i \\
\lambda_4 &= 0,994244954 - 0,0182074i \\
\lambda_5 &= 0,980800947
\end{aligned}$$

2.3 QR-algoritmus se shiftem

Rychlost konvergence QR-algoritmu závisí na rozdílu velikostí vlastních čísel. Ukázali jsme, že poddiagonální prvky matice A konvergují k nule stejně rychle jako $|\frac{\lambda_{i+1}}{\lambda_i}|^k$. Posuneme-li vhodně hodnoty vlastních čísel matice o nějaké reálné číslo μ, kterému budeme říkat *shift*, zvětší se poměr vlastních čísel, a tak urychlíme konvergenci.

Hodnoty vlastních čísel matice T posuneme tím způsobem, že přičteme k diagonálním prvkům matice T shift μ. Budeme tedy dále pracovat s maticí $T - \mu I$.

Matice T má charakteristický polynom $P_T(\lambda)$ a vlastní čísla $\lambda_1, \lambda_2, \ldots, \lambda_n$. Charakteristický polynom matice $T - \mu I$ vypadá takto:

$$P_{T-\mu I}(\lambda) = |T - \mu I - \lambda I| = \begin{vmatrix} t_{11} - \mu - \lambda & t_{12} & \cdots \\ t_{21} & t_{22} - \mu - \lambda & \\ \vdots & & \ddots \end{vmatrix} =$$

$$= \begin{vmatrix} t_{11} - (\mu + \lambda) & t_{12} & \cdots \\ t_{21} & t_{22} - (\mu + \lambda) & \\ \vdots & & \ddots \end{vmatrix} = P_T(\lambda + \mu),$$

takže vlastní čísla matice $(T - \mu I)$ jsou $\lambda_1 - \mu, \lambda_2 - \mu, \ldots, \lambda_n - \mu$. Vlastní čísla jsou stejná jako u matice T, jenom posunutá o shift μ.

Nyní zkonstruujeme QR-algoritmus se shiftem μ.

Počítáme vlastní čísla matice A. Převedeme ji na matici ve třídiagonálním tvaru, kterou označíme T_1. Pro zrychlení konvergence posuneme její vlastní čísla a pracovat budeme s maticí $T_1 - \mu I$. Provedeme její QR-rozklad:

$$T_1 - \mu I = QR$$

a další matici QR-algoritmu získáme takto:

$$T_2 = RQ + \mu I.$$

Matice T_2 je také třídiagonální a skutečně platí, že $T_2 = Q^T T_1 Q$:

$$T_2 = RQ + \mu I = Q^T(QR)Q + \mu I = Q^T(T_1 - \mu I)Q + \mu I = Q^T T_1 Q - \mu Q^T I Q + \mu I =$$

$$= Q^T T_1 Q - \mu I + \mu I = Q^T T_1 Q.$$

Zapišme postup algoritmicky:

Algoritmus 2.2: ***QR-algoritmus se shiftem***

Vstup: *matice* $A \in \mathbb{R}^{n\times n}$
Výstup: *matice* $T \in \mathbb{R}^{n\times n}$ *ve tvaru horní trojúhelníkové matice*

$T_1 = U^T A U$
for $k = 1, 2 \ldots$
  určení shiftu μ
  $T_k - \mu I = Q_k R_k$
  $T_{k+1} = R_k Q_k + \mu I$
  test na ukončení
end

Potřebujeme ještě vhodně zvolit hodnotu shiftu μ. Uvěďme tři z několika možností, které se nabízejí:

- nejjednodušší možností je $\mu = t_{nn}$
- efektivnější je zvolit za shift hodnotu vlastního čísla matice $\begin{bmatrix} t_{n-1,n-1} & t_{n-1,n} \\ t_{n,n-1} & t_{nn} \end{bmatrix}$, které je bližší k hodnotě t_{nn}
- používaný je tzv. *Wilkinsonův shift*, který vypadá takto:

  $$\mu = t_{nn} + d - sign(d)\sqrt{d^2 + t_{n-1,n}},$$

  kde $d = (t_{n-1,n-1} - t_{nn})/2$

Příklad 2.3 *Chceme spočítat vlastní čísla matice*

$$\begin{bmatrix} 10 & -19 & 17 & -12 & 4 & 1 \\ 9 & -18 & 17 & -12 & 4 & 1 \\ 8 & -16 & 15 & -11 & 4 & 1 \\ 6 & -12 & 12 & -10 & 4 & 1 \\ 4 & -8 & 8 & -6 & 1 & 2 \\ 2 & -4 & 4 & -3 & 1 & 0 \end{bmatrix}.$$

Podle knížky [10] víme, že přesná vlastní čísla této matice mají hodnotu

$$1, -1, -1, -1, i, -i.$$

Pomocí QR-algoritmu bez shiftu by bylo nemožné se k tomuto výsledku dobrat. Matice by sice k horní trojúhelníkové matici konvergovala, ale tato konvergence by byla tak pomalá, že bychom se jí nedočkali ani při použití seberychlejšího počítače.

Použili jsme tedy shift o hodnotě 1. *Výsledek jsme dostali velice rychle - hned po první iteraci:*

$$\begin{aligned} 0.00000000000001 &+ 0.99999999999980i \\ 0.00000000000001 &- 0.99999999999980i \\ -0.99999999999994 & \\ -1.00000000000000 & \\ -1.00000000000016 & \\ 1.00000000000008 & \end{aligned}$$

2.4 Jednorázové úpravy matic

V této podkapitole ukážeme, jak se převádí nesymetrická matice na horní Hessenbergův tvar a symetrická matice na tvar třídiagonální.

2.4.1 Převod matice na Hessenbergův tvar

V předchozí kapitole jsme ukázali, jak se provádí QR-rozklad Hessenbergovy matice. Řekli jsme, že je tento tvar výhodný z toho důvodu, že ho posloupnost matic A_k v QR-algoritmu zachovává a že se při QR-rozkladu matice v Hessenbergově tvaru redukuje počet operací.

Pro urychlení QR-algoritmu budeme tedy nejprve nesymetrickou matici A převádět na Hessenbergův tvar (v případě symetrické matice budeme matici převádět na třídiagonální tvar - viz následující podkapitola). Na Hessenbergův tvar matici převedeme pomocí Householderových transformací. Provedeme celkem $n-2$ transformací. Schématicky znázorněno:

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_2}$$

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix} \xrightarrow{P_3} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix} = H$$

Provedli jsme tuto transformaci:

$$(P_1P_2P_3)^T A(P_1P_2P_3) = H.$$

Převod matice A na Hessenbergův tvar zapsaný algoritmicky:

Algoritmus 2.3: ***Převod matice na Hessenbergův tvar***

Vstup: *matice $A \in \mathbb{R}^{n\times n}$ skládající se ze sloupců a_k, pro $k = 1, \dots n$*
Výstup: *matice $A \in \mathbb{R}^{n\times n}$ v Hessenbergově tvaru*

for $k = 1$ ***to*** $n-2$
$[v, \beta] =$ ***house*** $([(a_k)_{k+1}, \dots, (a_k)_n])$
$[a_{ij}]_{i=k+1,\dots,n;\ j=k,\dots,n} = (I - \beta vv^T)[a_{ij}]_{i=k+1,\dots,n;\ j=k,\dots,n}$
$[a_{ij}]_{i=1,\dots,n;\ j=k+1,\dots,n} = [a_{ij}]_{i=1,\dots,n;\ j=k+1,\dots,n}(I - \beta vv^T)$
end

2.4.2 Převod matice na třídiagonální tvar

Symetrickou matici A převádíme na třídiagonální tvar, abychom redukovali počet kroků prováděných při QR-rozkladu a urychlili tak QR-algoritmus. Platí totiž, že když $T_1 = QR$ je rozklad symetrické třídiagonální matice $T_1 \in \mathbb{R}^{n\times n}$, potom matice $T_2 = RQ$ je také symetrická a třídiagonální.

Pro symetrickou matici A_0 budeme chtít nalézt ortonormální matici Q tak, aby platilo

$$Q^T A_0 Q = T,$$

kde T je třídiagonální matice.

Třídiagonální matici T uvažujeme v tomto tvaru:

$$T = \begin{bmatrix} a_1 & b_1 & & & & & \\ b_1 & a_2 & b_2 & & & \mathbf{0} & \\ & b_2 & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & b_{n-2} & \\ & \mathbf{0} & & & b_{n-2} & a_{n-1} & b_{n-1} \\ & & & & & b_{n-1} & a_n \end{bmatrix}.$$

Převod na třídiagonální tvar budeme provádět pomocí Householderových transformací. Převáděnou matici A_0 budeme přenásobovat Householderovými maticemi

P_k. Po přenásobení matice A_0 maticí P_1 získáme matici v tomto tvaru:

$$\begin{bmatrix} \times & \times & 0 & 0 & 0 \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix}.$$

Matice A_0 a P_1 detailněji znázorníme následovně:

$$A_0 = \begin{bmatrix} a_1 & c_1^T \\ c_1 & M_1 \end{bmatrix}, \qquad P_1 = \begin{bmatrix} 1 & 0^T \\ 0 & \overline{P}_1 \end{bmatrix}.$$

V prvním kroku tedy přenásobíme matici A_0 Householderovými maticemi P_1 a vznikou matici označíme jako A_1:

$$A_1 = P_1 A_0 P_1.$$

Po tomto prvním kroku bude matice A_1 vypadat takto:

$$A_1 = \begin{bmatrix} 1 & 0^T \\ 0 & \overline{P}_1 \end{bmatrix} \begin{bmatrix} a_1 & c_1^T \\ c_1 & M_1 \end{bmatrix} \begin{bmatrix} 1 & 0^T \\ 0 & \overline{P}_1 \end{bmatrix} =$$

$$= \begin{bmatrix} a_1 & c_1^T \overline{P_1} \\ \overline{P}_1^T c_1 & \overline{P}_1^T M_1 \overline{P}_1 \end{bmatrix}.$$

Pokud $\overline{P}_1^T c_1 = e_1 b_1$, potom bude první řádek a první sloupec odpovídat třídiagonálnímu tvaru. K tomu použijeme Householderovu transformaci, která vektor c_1 přetransformuje na násobek e_1. Matice $\overline{P}_k$ bude tedy Householderovou maticí a lze ji vyjádřit ve tvaru $\overline{P}_k = I - \beta v v^T$.

V prvním kroku jsme tedy získali prvek a_1 a také prvek b_1:

$$|b_1| = ||e_1 b_1||_2 = ||\overline{P}_1^T c_1||_2 = ||c_1||.$$

Blok $\overline{P}_1^T M_1 \overline{P}_1$ bude vypadat následovně:

$$\overline{P}_1^T M_1 \overline{P}_1 = (I-\beta vv^T)M_1(I-\beta vv^T) = M_1-\beta v(v^T M_1)-\beta(M_1 v)v^T+\beta^2 v(v^T M_1 v)v^T =$$

$$= M_1 - vp^T - pv^T + (\beta v^T p)vv^T,$$

kde jsme si zjednodušili výpočet označením

$$p = \beta M_1 v.$$

Platí tedy $p^T = \beta v^T M_1^T$ a $\beta v^T p = \beta^2 v^T M_1 v$.

Pomocí vyjádření

$$w = p - v(\beta v^T p)/2,$$

můžeme výpočet $\overline{P}_1^T M_1 \overline{P}_1$ ještě zjednodušit:

$$\overline{P}_1^T M_1 \overline{P}_1 = M_1 - vw^T - wv^T.$$

Stejným způsobem jako matici A_0 budeme transformovat matici $\overline{P}_1^T M_1 \overline{P}_1$, což je matice řádu $(n-1)\times(n-1)$. Takto provedeme $n-2$ kroků, v každém kroku přenásobíme matici A_{k-1} maticí P_k:

$$A_k = P_k A_{k-1} P_k$$

a získáme třídiagonální matici.

Shrňme uvedený postup do algoritmu popisujícího převod matice A na třídiagonální tvar:

Algoritmus 2.4: ***Householderova třídiagonalizace***

Vstup: *symetrická matice $A \in \mathbb{R}^{n\times n}$ skládající se ze sloupců a_k, pro $k = 1, \dots n$*
Výstup: *matice $A \in \mathbb{R}^{n\times n}$ v třídiagonálním tvaru*

for $k = 1$ ***to*** $n-2$
  $[v, \beta] =$ ***house***$([(a_k)_{k+1}, \dots (a_k)_n])$
  $p = \beta[a_{ij}]_{ij=k+1}^n v$
  $w = p - (\beta p^T v/2)v$
  $(a_k)_{k+1} = |||[(a_k)_{k+1}, \dots (a_k)_n]||_2$
  $(a_{k+1})_k = (a_k)_{k+1}$
  for $l = k+2$ ***to*** n
    $a_{kl} = 0$
    $a_{lk} = 0$
  end
  $[a_{ij}]_{ij=k+1}^n = [a_{ij}]_{ij=k+1}^n - vw^T - wv^T$
end

Kapitola 3

Výpočty a numerická srovnání

Budeme porovnávat jednotlivé metody QR-rozkladů a QR-algoritmů založených na různých QR-rozkladech. Budeme sledovat přesnost rozkladů a rychlost konvergence QR-algoritmu.

Metody budeme aplikovat na matice, které jsou špatně podmíněné. Na začátku kapitoly uvedeme tři matice, které budeme dále používat.

Ve Fortranu jsme naprogramovali jednotlivé programy a algoritmy: QR-rozklady, QR-algoritmus a jeho modifikace a všechny další algoritmy, které v této práci uvádíme, jako např. různé transformace (na třídiagonální tvar, na Hessenbergův tvar, aj.). Mohli jsme tedy prakticky zkoumat, jak algoritmy fungují pro různé matice.

V první části porovnáme jednotlivé QR-rozklady aplikované na jednotlivé matice. Na matici Cerfacs ukážeme účinnost tzv. dvojitého MGS.

Dále porovnáme QR-algoritmus použitý na výpočet vlastních čísel "špatné" matice $SMCE_{20}$. Zde bude zajímavé porovnání výsledků nikoliv mezi jednotlivými použitými QR-rozklady, ale porovnání výsledků mezi jednotlivými systémy, které umožňují výpočet vlasntích čísel, jako je Maple, Matlab nebo Lapack.

I když jsme viděli patrné rozdíly v přesnosti jednotlivých QR-rozkladů, nedokázali jsme porovnat účinnost jednotlivých QR-algoritmů založených na těchto různých QR-rozkladech.

Závěrem se budeme zabývat návrhy vylepšení a zefektivnění QR-algoritmu. Budeme se zabývat otázkou týkající se vhodnosti kritéria (suma poddiagonálnách prvků) pro konec QR-algoritmu. Při zkoumání grafu logaritmu rozdílu počítaných a skutečných vlastních čísel jsme zjistili, že se graf chová jako přímka. Ukážeme, jak lze pomocí Aitkenova δ^2 procesu QR-algoritmus urychlit.

Ukážeme také, jak nakládat s případy, kdy vlastními čísly jsou čísla komplexně sdružená.

V této kapitole budeme používat tyto matice:

- "Jedničková" matice

Vygenerovali jsme si matici, která obsahuje na diagonále 1, pod diagonálou -1 a nad diagonálou 0. Tuto matici označíme $JEDN_N$, kde N značí řád této matice. Například matice $JEDN_5$ vypadá takto:

$$JEDN_5 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 1 & 0 & 0 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & -1 & 1 \end{bmatrix}.$$

- Matice Cerfacs

  Tato matice řádu 3×3 pochází z [7]. Při jejím QR-rozkladu i při výpočtu vlastních čísel jsme získávali zajímavé numerické výsledky.

$$\begin{bmatrix} 0.12100300219993308 & 2.09408775152625060 & 1.26139640819301024 \\ -0.10439395064078592 & -1.80665016070527140 & -1.08825526624380808 \\ 0.21661355806776747 & 0.49451660567698374 & -0.84174336538575500 \end{bmatrix}.$$

- "Špatná" matice

  Matici označíme $SMCE_N$. Říkáme jí "špatná", protože dokázala potrápit i komerční a "profesionální" software sloužící k výpočtu vlastních čísel. Uveďme příklad této matice 6×6:

$$SMCE_6 = \begin{bmatrix} 6 & 5 & 0 & 0 & 0 & 0 \\ 5 & 5 & 4 & 0 & 0 & 0 \\ 4 & 4 & 4 & 3 & 0 & 0 \\ 3 & 3 & 3 & 3 & 2 & 0 \\ 2 & 2 & 2 & 2 & 2 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix},$$

  matici lze pochopitelným způsobem analogicky definovat pro jakýkoliv rozměr.

- Sedmidiagonální matice

  Tato matice je řádu 11×11, označíme ji $SEDMI$:

$$SEDMI = \begin{bmatrix} 5 & 2 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 6 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 6 & 3 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 3 & 6 & 3 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 3 & 6 & 3 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 3 & 6 & 3 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 3 & 6 & 3 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 3 & 6 & 3 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 6 & 3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 3 & 6 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 2 & 5 \end{bmatrix}.$$

3.1 Numerická porovnání QR-rozkladů

Nejprve porovnáme jednotlivé QR-rozklady. Matici A program rozloží na matice $\widehat{Q}$ a $\widehat{R}$, které nejsou přesné z důvodu omezené přestnosti stroje, který tyto výpočty provádí.

Kvalitu rozkladu zhodnotíme podle přesnosti rozkladu a podle ortonormality matice $\widehat{Q}$.

Přesnost rozkladu změříme pomocí normy matice

$$||A - \widehat{Q}\widehat{R}|| =: ||E_1||,$$

ortonormalitu matice $\widehat{Q}$ pomocí normy matice

$$||I - \widehat{Q}^T\widehat{Q}|| =: ||E_2||.$$

Za normu jsme zvolili Frobeniovu normu:

$$||A|| = \sqrt{\sum_i \sum_j |a_{ij}|^2}.$$

3.1.1 Rozklad jedničkové matice $JEDN_{50}$

Jednotlivé algoritmy na výpočet QR-rozkladu vykazovaly následující chyby:

- Gram-Schmidt
$$||E_1|| = 0.347715E - 28,$$
$$||E_2|| = 0.219042E - 01.$$
- MGS
$$||E_1|| = 0.294344E - 28,$$
$$||E_2|| = 0.835548E - 02.$$
- Givens
$$||E_1|| = 0.716292E - 27,$$
$$||E_2|| = 0.321697E - 28.$$
- Householder
$$||E_1|| = 0.373311E - 26,$$
$$||E_2|| = 0.630567E - 28.$$

Gram-Schmidtovy metody (GS a MGS) QR-rozkladu vykazují velice špatnou ortogonalitu matice $\widehat{Q}$. Vliv kvality ortogonaliy matice $\widehat{Q}$ na průběh QR-algoritmu se nám ale nepodařilo prokázat. QR-algoritmus fungoval dobře i s "méně ortogonální" maticí $\widehat{Q}$.

3.1.2 Rozklad matice Cerfacs

Matice $CERFACS$ je akademickým příkladem, který má poukazovat na chybu ortogonality matice $\widehat{Q}$ při použití Gram-Schmidtova nebo modifikovaného Gram-Schmidtova QR-rozkladu. Při QR-rozkladu této matice jsme získali tyto výsledky:

- Gram-Schmidt
$$||E_1|| = 0.254223E - 31,$$
$$||E_2|| = 0.189044E + 01.$$
- MGS
$$||E_1|| = 0.254223E - 31,$$
$$||E_2|| = 0.189044E + 01.$$
- Givens
$$||E_1|| = 0.127304E - 30,$$
$$||E_2|| = 0.566589E - 31.$$
- Householder
$$||E_1|| = 0.124222E - 30,$$
$$||E_2|| = 0.685498E - 32.$$

Matice $CERFACS$ je velice špatně podmíněná. Číslo podmíněnosti $\kappa(A)$ používáme v nerovnosti sloužící k odhadu přesnosti ortogonality matice $\widehat{Q}$:

$$||I - \widehat{Q}^T\widehat{Q}||_2 \leq c_2\kappa(A)u,$$

která je uvedena výše jako nerovnost (1.4).

Při použití MGS (a pochopitelně i v Gram-Schmidtova QR-rozkladu) jsme očekávali nepřesnost v ortogonalitě matice $\widehat{Q}$.

Na této matici jsme si ověřili úvahu z kapitoly 1.2.3, která pojednává o tzv. dvojitém MGS. Po druhém použití MGS na matici $\widehat{Q}$ získáme matici $\widehat{Q}_2$, jejíž ortogonalita už není "postižena" vysokým číslem podmíněnosti matice A. Chyba ortogonality matice $\widehat{Q}_2$ je již ovlivněna pouze řádem matice a zaokrouhlovacími vlastnostmi stroje provádějícího výpočet.

Po druhém použití MGS jsme dostali tyto výsledky:

$$||A - \widehat{Q}\widehat{R}|| = ||A - \widehat{Q}_2\widehat{R}_2\widehat{R}|| = 0.254223E - 31,$$

$$||I - \widehat{Q}_2^T\widehat{Q}_2|| = 0.200334E - 29.$$

3.1.3 Rozklad sedmidiagonální matice

S QR-rozkladem sedmidiagonální matice $SEDMI$ si dobře poradily všechny algoritmy:

- Gram-Schmidt
$$||E_1|| = 0.147006E - 29,$$
$$||E_2|| = 0.198687E - 28.$$
- MGS
$$||E_1|| = 0.480269E - 29,$$
$$||E_2|| = 0.883279E - 29.$$
- Givens
$$||E_1|| = 0.599206E - 28,$$
$$||E_2|| = 0.147402E - 29.$$
- Householder
$$||E_1|| = 0.611989E - 28,$$
$$||E_2|| = 0.132259E - 29.$$

3.2 Numerické porovnání QR-algoritmů

QR-algoritmy jsme porovnávali nejenom podle QR-rozkladů, které jsme v nich použili, ale zkoušeli jsme výsledky porovnávat i v konkurenci již hotových programů jako je např. Maple, Matlab nebo program QZ z balíku Lapack.

3.2.1 Matice $SMCE_{20}$

S touto maticí jsme získali velice zajímavé a překvapivé výsledky. Překvapivé (bohužel v negativním smyslu) z toho důvodu, že ani algoritmy na výpočet vlastních čísel komerčního software nám nebyly schopny dát spolehlivé výsledky.

Porovnejme nejprve výsledky z Maplu, Matlabu a Lapacku:

λ	**Maple**	**Matlab**	**Lapack**
1	60.03324383	60.03324324	60.03324324
2	44.36524374	44.36524402	44.36524403
3	33.09210797	33.09210797	33.09210798
4	24.37523529	24.37523516	24.37523516
5	17.49772822	17.49772818	17.49772819
6	12.08708262	12.08708254	12.08708255
7	7.918744163	7.918744101	7.918744102
8	4.839247954	4.839244379	4.839244379
9	2.719957858	2.720101685	2.720101686
10	1.360117687	1.412338597	1.412338639
11	1.176296	0.708035	0.708045
12	$0.732624 + 0.569045i$	$-0.085672 + 0.056047i$	0.367634
13	$0.732624 - 0.569045i$	$-0.085672 - 0.056047i$	0.209228
14	$0.194246 + 0.685859i$	$-0.004493 + 0.140320i$	$0.138573 + 0.039938i$
15	$0.194246 - 0.685859i$	$-0.004493 - 0.140320i$	$0.138573 - 0.039938i$
16	$-0.492046 + 0.219907i$	0.374338	$0.064222 - 0.071850i$
17	$-0.492046 - 0.219907i$	$0.124481 + 0.154636i$	$0.064222 + 0.071850i$
18	0.099024	$0.124481 - 0.154636i$	$-0.001782 + 0.052749i$
19	$-0.216838 + 0.539154i$	$0.253962 + 0.083776i$	$-0.001782 - 0.052749i$
20	$-0.216838 - 0.539154i$	$0.253962 - 0.083776i$	-0.028004

Na výsledky z druhé poloviny tabulky už se vůbec nemůžeme spolehnout.

Poznamenejme ještě, že program Maple použil k výpočtu vlastních čísel QR-algoritmus, kterému předcházel převod na Hessenbergův tvar.

Nyní uvedeme výsledky třech našich programů: QR-algoritmus založený na Gram-Schmidtově QR-rozkladu, na Modifikovaném Gram-Schmidtově QR-rozkladu a na Givensově QR-rozkladu, kde jsme navíc ještě převedli matici na Hessenbergův tvar.

Po zkušenostech s výsledky, které nám poskytl komerční software, jsme získali překvapivě navzájem podobné výsledky:

λ	**GS**	**MGS**	**Givens**
1	60.03324324292650	60.03324324292660	60.03324324292662
2	44.36524402581347	44.36524402581368	44.36524402581404
3	33.09210797898585	33.09210797898584	33.09210797898638
4	24.37523516347238	24.37523516347229	24.37523516347295
5	17.49772818677922	17.49772818677925	17.49772818677963
6	12.08708254988644	12.08708254988643	12.08708254988680
7	7.91874410161812	7.91874410161812	7.91874410161838
8	4.83924437933158	4.83924437933159	4.83924437933160
9	2.72010168550744	2.72010168550771	2.72010168550961
10	1.41233863863721	1.41233863862805	1.41233863933241
11	0.70804558589638	0.70804558256297	0.70804545140925
12	0.36763351265592	0.36763350907941	0.36762908066469
13	0.20501850388083	0.20503459078622	0.20688569449115
14	0.15013478059831	0.15008626493790	0.12812085849051
15	$0.09747372 + 0.05298971i$	$0.09754806 + 0.05297885i$	$0.09185849 + 0.02891761i$
16	$0.09747372 - 0.05298971i$	$0.09754806 - 0.05297885i$	$0.09185849 - 0.02891761i$
17	$0.02972013 + 0.05812580i$	$0.02976364 + 0.05823738i$	$0.03636622 + 0.04533971i$
18	$0.02972013 - 0.05812580i$	$0.02976364 - 0.05823738i$	$0.03636622 - 0.04533971i$
19	$-0.01314503 + 0.02421829i$	$-0.01324666 + 0.02429963i$	$-0.00410024 + 0.02052555i$
20	$-0.01314503 - 0.02421829i$	$-0.01324666 - 0.02429963i$	$-0.00410024 - 0.02052555i$

3.3 Pozorování a vylepšení QR-algoritmu

Při chodu QR-algoritmu nás zaráželo, jak je konvergence pomalá. Zkoušeli jsme proto navrhnout a vyzkoušet několik vylepšení:

3.3.1 "Překlápění matice"

Matice A by měla konvergovat k horní trojúhelníkové matici. Předpokládali jsme, že se algoritmus zrychlí, když bychom v případě, že by naddiagonální prvky matice A byly menší než poddiagonální prvky, matici nějak transformovali, abychom zaměnili naddiagonální prvky s poddiagonálními.

Testovali jsme v každé iteraci QR-algoritmu, zda nejsou naddiagonální prvky menší než poddiagonální prvky a pokud tomu tak bylo, tak jsme provedli transformaci matice A. Tato podobnostní transformace byla dána maticí s jedničkami na vedlejší diagonále. Matice pak vypadala, jakoby se "překlopila" podle hlavní a podle vedlejší diagonály:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}.$$

Urychlení konvergence QR-algoritmu jsme dosáhli bohužel pouze u triviálního případu, kdy matice A měla dolní trojúhelníkový trvar.

3.3.2 Suma pod diagonálou a přesnost vlastních čísel

Zamýšleli jsme se také, jestli souvislost přesnosti spočítaných vlastních čísel a velikosté sumy poddiagonálních prvků je tak úzká, jak se zdá být. Prováděli jsme testy na jednoduché matici řádu 2×2:

$$\begin{bmatrix} 1 & 0 \\ a & 5 \end{bmatrix},$$

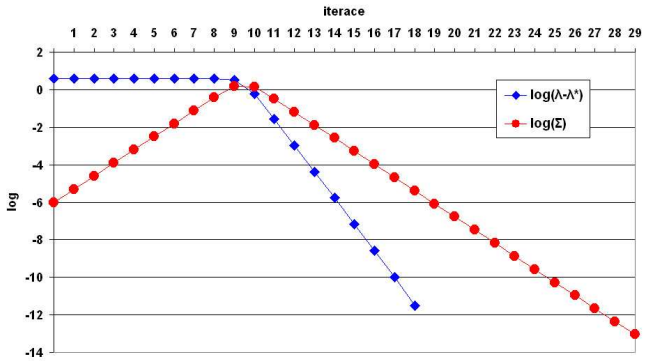
kde jsme za prametr $a \in \mathbb{R}$ volili různé hodnoty.

Zkoumali jsme, jaká je velikost sumy poddiagonálních prvků (v tomto případě velikost prvku a_{21}) a jaká je odchylka spočítaných vlastních čísel od skutečných vlastních čísel (1 a 5).

Nakreslili jsme si grafy znázorňující průběh konvergence. Na osu x jsme nanášeli iterace, osa y znázorňuje log_{10} příslušné hodnoty. Těmito hodnotami byl logaritmus sumy pod diagonálou a logaritmus rozdílu skutečného vlastního čísla od hodnoty spočítané QR-algoritmem. Kritérum pro skončení QR-algoritmu bylo takové, aby suma pod diagonálou byla menší než $0,1 \cdot 10^{-12}$.

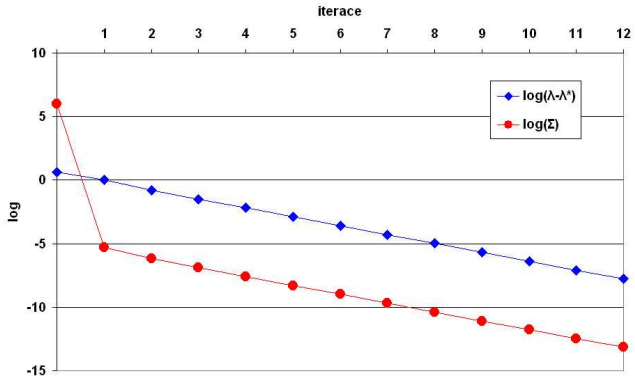
Ukážeme si dva případy: a) $a = 0,000001$, b) $a = 1000000$.

- $a = 0,000001$



Zde je zajímavostí, že do desáté iterace nám suma poddiagonálních prvků dokonce vzrůstala. Algoritmus na výpočet potřeboval 30 iterací. Podíváme-li se blíže na sledované hodnoty např. po 16. iteraci, vidíme, že suma pod diagonálou je veliká: $0,105 \cdot 10^{-3}$, přesto přesnost spočítáných vlastních čísel už je poměrně slušná: odchylka činí $0,272 \cdot 10^{-8}$. Pokud bychom se spokojili s touto přesností, stačilo by nám poloviční množství iterací.

- $a = 1000000$



V tomto případě je tomu naopak. Naše kritérium (velikost sumy poddiagonálních prvků) zde přináší poměrně malou přesnost spočítaných vlastních čísel. Např. po 6. iteraci už je suma docela malá: $0,102 \cdot 10^{-8}$, ale odchylka spočítaných vlastích čísel činí $0,256 \cdot 10^{-3}$. I po skončení QR-algoritmu (13 iterací) jsme nebyli spokojeni s přesností vlastních čísel, byť jsme dosáhli požadované velikosti sumy poddiagonálních prvků.

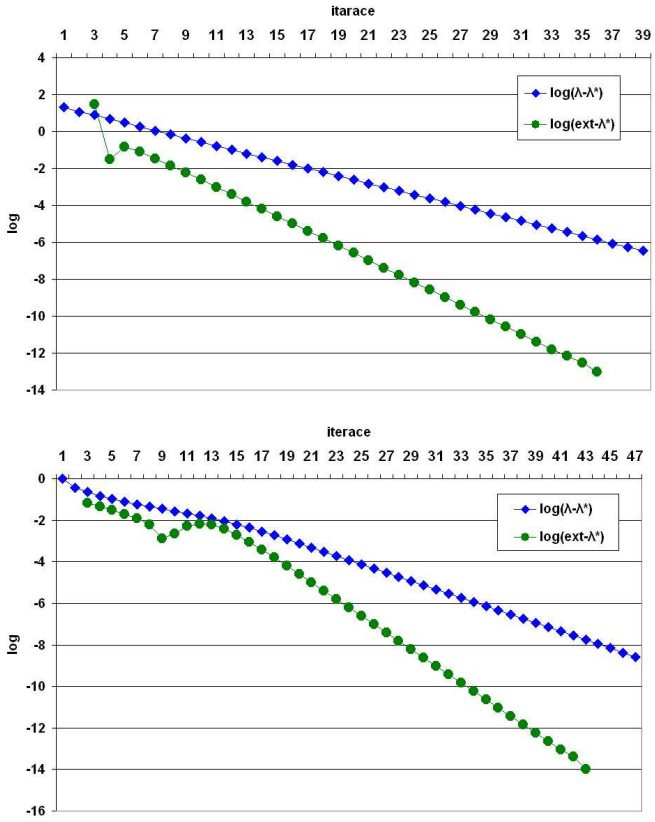
Vidíme zde, že velikost sumy poddiagonálních prvků nezaručuje přesnost spočítaných vlastních čísel. Nabízí se zde tedy otázka volby (a hlavně vymyšlení) vhodného kritéria pro přesnost QR-algoritmu.

Nicméně tento test a především grafy (přesněji jejich tvar přímky) nás přivedli na zajímavou myšlenku, že se zde jedná o logaritmickou konvergenci a že by se toho dalo využít.

3.3.3 Extrapolace vlastních čísel

Pomocí vzorce (2.8) jsme se snažili extrapolovat hodnotu vlastního čísla ze tří po sobě jdoucích spočítaných iteracích. Skutečně se tato extrapolace projevila jako výrazné urychlení konvergence algoritmu.

Počítáme-li vlastní čísla matice $SMCE_{12}$ (u které známe skutečné hodnoty vlastních čísel), můžeme se přesvědčit o přesnosti spočítaných vlastních čísel (modrá barva) a o přesnosti extrapolovaných vlastních čísel (zelená barva). Opět do grafu nanášíme logaritmus odchylky od skutečného vlastního čísla. První graf je pro největší vlastní číslo, druhý graf pro nejmenší vlastní číslo:



Poznamenejme, že při počítání vzorce (2.8) jsme uvažovali kladné rozdíly $\lambda_{k-1} - \lambda^*$, $\lambda_k - \lambda^*$, $\lambda_{k+1} - \lambda^*$ a s klidem jsme při výpočtu odstranili absolutní hodnotu. Kdyby tyto rozdíly nebyly všechny kladné, ale byly by všechny záporné nebo v tomto pořadí: kladný, záporný, kladný $(+-+)$ či záporný, kladný, záporný $(-+-)$, bude vzorec pro extrapolaci vypadat pořád stejně. Budou-li rozdíly vypadat $++-$, $-++$, $+--$ nebo $--+$, nepůjde λ^* tak jednoduše vyjádřit a v takových případech se extrapolaci vyhneme.

3.3.4 Komplexní vlastní čísla

Pokud má matice vlastní čísla komplexní, nekonverguje QR-algoritmus k horní trojúhelníkové matici, ale k tzv. kvazitrojúhelníkové. Pokud bude mít matice řádu 5×5 první dvě vlastní čísla komplexní a ostatní reálná, bude QR-algoritmus konvergovat k matici, která bude vypadat takto:

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times \end{bmatrix}.$$

Na diagonále na prvním a druhém řádku vznikne blok řádu 2×2, jehož vlastní čísla jsou i prvním a druhým vlastním číslem této celé matice.

Kritérium na ukončení QR-algoritmu jsme tedy nemohli použít $\sum_{i>j} |a_{ij}| < \varepsilon$, protože vznikl-li díky komplexním vlastním číslům blok na diagonále, vznikl tak i prvek po diagonálou, který nekonvergoval k nule. Bylo potřeba tedy identifikovat vznik tohoto bloku a trochu jinak s ním nakládat, tedy hlavně nezapočítávat prvek bloku pod diagonálou do této sumy.

Sumu jsme tedy začali počítat jiným způsobem: napřed jsme do sumy zařadili prvky pod druhou diagonálou a potom jsme prozkoumávali, jak vypadají prvky druhé diagonály. Pokud jsme objevili prvek, který byl obklopen nulami, označili jsme tento prvek, jako část bloku, který obsahuje komplexní vlastní čísla. Vznikne-li blok na diagonále, opravdu je poddiagonální prvek obklopen nulami, jak je vidět na příkladě, kdy matice řádu 6×6 měla 6 komplexních vlastních čísel:

$$\begin{bmatrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \\ 0 & 0 & 0 & 0 & \times & \times \end{bmatrix}. \tag{3.1}$$

Vidíme, že poddiagonální prvky náležící bloku 2×2 mají kolem sebe nuly (pokud je v druhém nebo posledním řádku, má nulu jenom z jedné strany pochopitelně).

Příklad 3.1 *Výpočet vlastních čísel matice A:*

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -7 \\ 0 & 0 & 1 & 0 & 0 & -5 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}.$$

Charakteristický polynom je

$$z^6 + 5z^3 + 7z^2 + 1.$$

Při užití QR-algoritmu konvergovala matice A ke kvazitrojúhelníkovému tvaru schématicky znázorněného jako (3.1). Vlastní čísla jednotlivých bloků na diagonále a tedy i vlastní čísla celé matice jsou:

$$\begin{aligned} 1.19470640451938 &\pm 1.56210679941984i \\ -1.23939907019597 &\pm 0.62708344210567i \\ 0.04469266567659 &\pm 0.36334499639425i \end{aligned}$$

3.3.5 Kritérium konce QR-algoritmu

V kapitole (3.3.2) jsme viděli, že kritérium konce QR-algoritmu $\sum_{i>j} |a_{ij}| < \varepsilon$ (v případě reálných vlastních čísel) není nejvhodnějším. I když je tato suma hodně malá, mohou se spočítaná vlastní čísla ještě hodně lišit od skutěčných hodnot. Rozhodli jsme se proto končit QR-algoritmus tehdy, až se vlastní čísla po dvou po sobě jdoucích iteracích od sebe lišila méně než o zadané ε. Zde jsme také využívali extrapolace (2.8) počítaných vlastních čísel.

Postupovali jsme takto: pokud jsme identifikovali na diagonále blok, spočítali jsme komplexní vlastní čísla. Z posledních tří iterací daného vlastního čísla jsme vypočítali extrapolovanou hodnotu. V další iteraci jsme provedli totéž a porovnali jsme spočítané hodnoty s předchozími. Pokud se žádná hodnota od hodnoty z minulé iterace nelišila více než o ε, QR-algoritmus jsme skončili. Pokud jsme porovnávali komplexní vlastní čísla, porovnávali jsme navzájem reálné a imaginální části.

Pokud nás zajímala všechna vlastní čísla, tak nám kritérium porovnávání dvou po sobě jdoucích iterací moc nepomohlo, neboť se vždycky našlo nějaké vlastní číslo, které se iteraci po iteraci lišilo více než o ε. Některá vlastní čísla ale byla spočítána prakticky hned a pokud nás zajímalo jenom toto vlastní číslo, nemuseli jsme čekat, až suma pod diagonálou bude dostatečně malá. Porovnali jsme dvě po sobě jdoucí extrapolace tohoto vlastního čísla a pokud se už moc neměnily (jejich rozdíl byl menší než ε), algoritmus jsme skončili.

Příklad 3.2 *Zajímají nás vlastní čísla matice $SMCE_{12}$. Mějme přesnost $\varepsilon = 10^{-6}$. Kolik je potřeba iterací, aby*

1. *suma poddiagonálních prvků byla menší než ε?*

   *Po **35** iteracích QR-algoritmu měla suma poddiagonálních prvků této matice hodnotu $0.97 \cdot 10^{-6}$. Skutečné hodnoty pro největší a nejměnší vlastní číslo činí $\lambda_1^* = 32.22889150157214$ a $\lambda_{12}^* = 0.03102806088596$ [1]. Vypočtená hodnota λ_1 činila 32.22889288746892, lišila se tedy od skutečné hodnoty o $1.38 \cdot 10^{-6}$, λ_{12} vyšlo 0.0310285167912, od skutečné hodnoty se liší o $0.455 \cdot 10^{-6}$.*

2. *rozdíl spočítaného největšího a nejmenšího vlastního čísla a jejich přesných hodnot byl menší než ε?*

   K tomu jsme potřebovali ještě jednu iteraci. Rozdíl pro λ_1 činil $0.868 \cdot 10^{-6}$, pro λ_{12} činil $0.285 \cdot 10^{-6}$.

3. *rozdíl extrapolovaného největšího vlastního čísla a jeho přesné hodnoty byl menší než ε?*

   *Po **19** iteracích jsme získali tuto extrapolovanou hodnotu:* 32.22889215848012. *Od skutečné hodnoty se liší o $0.656 \cdot 10^{-6}$. Abychom tuto přesnost největšího*

[1] podle knihy [10]

vlastního čísla získali bez extrapolace, potřebovali bychom k tomu 36 iterací QR-algoritmu.

4. *rozdíl extrapolovaného nejmenšího vlastního čísla a jeho přesné hodnoty byl menší než ε?*

   *V tomto případě jsme potřebnou přesnost obdrželi po **24** iteracích. Při výpočtu bez extrapolace bychom potřebovali 34 iterací.*

Závěr

Cílem této bakalářské práce bylo studovat jeden z nejpoužívanějších způsobů výpočtu vlastních čísel, tj. QR-algoritmus.

Nejdříve bylo potřeba prozkoumat QR-rozklad, který je základem QR-algoritmu. Porovnávali jsme Gram-Schmidtův, modifikovaný Gram-Schmidtův, Householderův a Givensův QR-rozklad. Zjišťovali jsme, který z těchto rozkladů je pro QR-algoritmus nejvhodnější. Jednotlivé QR-rozklady se lišily zejména ve kvalitě ortogonality matice Q. Zjistili jsme, že přesnost ortogonality matice Q má na výpočet vlastních čísel pomocí QR-algoritmu malý vliv.

Přehledně jsme sestavili a podrobně vysvětlili důkaz konvergence QR-algoritmu k horní trojúhelníkové matici. Zde jsme se pokusili ukázat asymptotiku posloupnosti matic A_k. Chtěli jsme předvést, že diagonální prvky konvergují k vlastním číslům logaritmicky a že by šlo využít Aitkenova δ^2-procesu.

Zkoumali jsme po jednotlivých iteracích velikost odchylek spočítaných vlastních čísel od skutečných jejich hodnot. Kreslili jsme si průběh logaritmu těchto odchylek a zde si ověřili teoretické úvahy, že se tento graf vždycky podobá přímce. Využili jsme tedy Aitkenova δ^2-procesu a tímto způsobem jsme vlastní čísla extrapolovali. Mnohdy nám ke stejné přesnosti vlastních čísel stačilo poloviční množství iterací.

Naráželi jsme na matice, u kterých QR-algoritmus konvergoval velice pomalu. K odstanění tohoto problému jsme využívali posunutí, čímž jsme změnili poměr mezi hodnotami vlastních čísel, který právě rychlost konvergence ovlivňuje.

Ukázali jsme, jakým způsobem se konvergence časově urychluje. Urychlení se provádí převodem matice na speciální tvar (Hessenbergův v případě nesymetrické nebo třídiagonální v případě symetrické matice). QR-rozklad matice v tomto tvaru vyžaduje podstatně méně operací, proto se QR-algoritmus výrazně urychlí. Vlastností QR-algortimu je, že pokud má matice Hessenbergův nebo třídiagonální tvar, tak i matice v následující iteraci QR-algoritmu bude tento tvar zachovávat. Proto se tento převod provádí pouze jedinkrát, a to před započetím QR-algoritmu.

Veškeré postupy jsme prakticky zkoušeli na množství různých matic. Jednotlivé algoritmy jsme naprogramovali ve Fortranu a zkoušeli jsme, jak se tyto algoritmy chovají v praxi. Zajímalo nás, jakých výsledků dosáhneme, spočítáme-li rychle a přesně vlastní čísla. Kromě toho jsme také chtěli porovnat naše výsledky s výsledky komerčních programů a s výsledky programu na výpočet vlastních čísel z balíku Lapack. Proto uvádíme i příklady s ukázkami našich výpočtů.

– uvedl bych, co se dá ještě zkoumat –
– taky to, co se nám neporařilo zjistit, nebo co by nás zajímalo –
– přínos této práce —

Literatura

[1] Golub, G.H. - Van Loan, Ch.F.: *Matrix Computation*, The Johns Hopkins University Press, Baltimore, third edition, 1996

[2] Fiedler, M.: *Speciální matice a jejich použití v numerické matematice*, SNTL Praha, 1981

[3] Wilkinson, J.H.: *Convergence of the LR a QR and Related Algorithms*, The Computer Journal, April 1965, Vol 8, No 1.

[4] Saad, Y.: *Iterative Methods for Sparse Linear Systems*, second edition, January 2000

[5] Parlett, B.N.: *The Symmetric Eigenvalue Problem*, Society for Industrial and Applied Mathematics, Philadelphia, 1998

[6] Hřebíček, J. a kol.: *Programovací jazyk FORTRAN 77 a vědeckotechnické výpočty*, Academia, Praha, 1989

[7] Giraud, L. - Langou J.: *When Modified Gram-Schmidt Generates a Well-Conditioned Set of Vectors*

[8] Björk Å.: *Solving linear least squares problems using Gram-Schmidt orthogonalization*, 1967

[9] Björk Å. - Paige C.C.: *Loss and recapture of orthogonality in the modified Gram-Schmidt Algorithm*, 1992

[10] Gregory, R.T. - Karney D.L.: *A Collection of Matrices for Testing Computational Algorithms*, John Wiley & sons, 1969