Schurova dekompozicija i primjene

Poljarević, Petar

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, School of Applied Mathematics and Informatics / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet primijenjene matematike i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:126:653934

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni diplomski studij matematike
(smjer: Matematika i računarstvo)

Petar Poljarević

Schurova dekompozicija i primjene

Diplomski rad

Osijek, 2023.

Sveučilište J. J. Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike
Sveučilišni diplomski studij matematike
(smjer: Matematika i računarstvo)

Petar Poljarević

Schurova dekompozicija i primjene

Diplomski rad

Mentor: izv. prof. dr. sc. Zoran Tomljanović

Osijek, 2023.

Sadržaj

1 Uvod

U ovom poglavlju navest ćemo neke osnovne pojmove, oznake i tvrdnje iz linearne algebre koje će nam biti važne u ostatku rada, a čitatelju možda nisu poznate od ranije.

Jedan od najvažnijih problema u linearnoj algebri je određivanje svojstvenih vrijednosti i svojstvenih vektora zadane matrice. Definirajmo ih.

Definicija 1.1 (Svojstvena vrijednost i svojstveni vektor)
*Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$. Ukoliko za neki broj $\lambda \in \mathbb{C}$ i neki vektor $x \in \mathbb{C}^n\backslash\{0\}$ vrijedi $Ax = \lambda x$, onda broj λ zovemo **svojstvena vrijednost** matrice A, a vektor x zovemo **svojstveni vektor** matrice A. Uređeni par (λ, x) zovemo **svojstveni par** matrice A.*

Iako je prilično jednostavna, ova definicija ne daje nam nikakav efikasan način da odredimo svojstvene vrijednosti matrice, osim da ih *pogodimo*. Zato ćemo sada definirati karakteristični polinom matrice te pokazati njegovu vezu sa svojstvenim vrijednostima.

Definicija 1.2 (Karakteristični polinom matrice)
*Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$. Polinom $k_A(z) = \det(zI - A)$ zovemo **karakteristični polinom** matrice A.*

Teorem 1.1 (Veza svojstvenih vrijednosti i karakterističnog polinoma)
Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$. Broj $\lambda \in \mathbb{C}$ je svojstvena vrijednost matrice A ako i samo ako poništava njen karakteristični polinom, tj. $\det(\lambda I - A) = 0$.

Dokaz. Neka su zadani $\lambda \in \mathbb{C}$ i $x \in \mathbb{C}^n\backslash\{0\}$, tako da vrijedi $Ax = \lambda x$.

$$Ax = \lambda x \iff \lambda x - Ax = 0 \iff (\lambda I - A)x = 0$$

Primijetimo da zadnja jednakost vrijedi ako i samo ako je matrica $(\lambda I - A)$ singularna (budući da x nije nul-vektor), a matrica je singularna ako i samo ako je njena determinanta jednaka nuli (vidi [2, str. 100]) pa slijedi

$$Ax = \lambda x \iff \det(\lambda I - A) = 0.$$

□

Pokazali smo da se problem računanja svojstvenih vrijednosti zadane matrice može svesti na računanje nultočaka njenog karakterističnog polinoma. Ova tvrdnja ima veliku važnost u teoriji, no u praksi se ne koristi, budući da ne postoji algebarska formula za nultočke polinoma kojemu je stupanj veći od 4 (vidi [1, str. 103]). S druge strane, ako nam je zadan polinom p i želimo izračunati njegove nultočke, onda možemo konstruirati matricu čiji je karakteristični polinom upravo p i nakon toga izračunati njene svojstvene vrijednosti korištenjem neke od numeričkih metoda, no više o ovome kasnije.

Definicija 1.3 (Spektar matrice)
*Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$. Skup svih svojstvenih vrijednosti matrice A zovemo **spektar** matrice A i označavamo ga sa $\sigma(A)$.*

Lako se pokaže da matrica $A \in \mathbb{C}^{n\times n}$ ima karakteristični polinom stupnja n. Budući da polinom stupnja n sadrži točno n nultočaka (ne nužno različitih), jasno je da matrica A ima točno n svojstvenih vrijednosti, tj. njen spektar sadrži točno n članova (ako uzmemo u obzir njihove višestrukosti).

Matrice često zapisujemo na drugačiji način, tj. u drugačijoj formi jer nam to značajno olakšava rad s njima. Najčešći načini zapisivanja matrice u drugoj formi je pomoću tzv. transformacija sličnosti. Definirajmo ih te pokažimo njihovu važnost.

Definicija 1.4 (Transformacija sličnosti)
Neka je $A \in \mathbb{C}^{n\times n}$. Kažemo da je matrica $\hat{A}$ ***slična*** *matrici A ako postoji regularna matrica $T \in \mathbb{C}^{n\times n}$ takva da je $\hat{A} = T^{-1}AT$. Matricu T u ovom kontekstu zovemo* ***transformacija sličnosti****. Dodatno, ako je T unitarna, onda kažemo da je $\hat{A}$* ***unitarno slična*** *matrici A.*

Primijetimo da u prethodnoj definiciji vrijedi i obratno, tj. ako je matrica $\hat{A}$ slična matrici A, onda je i matrica A slična matrici $\hat{A}$, odnosno matrice A i $\hat{A}$ su međusobno slične. Općenito, sličnost matrica je relacija ekvivalencije.

Lema 1.2 (Spektar slične matrice)
Ako su matrice $A \in \mathbb{C}^{n\times n}$ i $B \in \mathbb{C}^{n\times n}$ međusobno slične, onda vrijedi $\sigma(A) = \sigma(B)$.

Dokaz. Neka je $T \in \mathbb{C}^{n\times n}$ regularna matrica takva da je $B = T^{-1}AT$. Prema definiciji karakterističnog polinoma, vrijedi

$$\begin{aligned} k_B(z) &= \det(zI - B) = \det(zT^{-1}T - B) \\ &= \det(zT^{-1}IT - T^{-1}AT) \\ &= \det(T^{-1}(zI - A)T) \\ &= \det(T^{-1}) \cdot \det(zI - A) \cdot \det(T) \\ &= \det(T)^{-1} \cdot \det(zI - A) \cdot \det(T) \\ &= \det(zI - A) = k_A(z), \end{aligned}$$

pri čemu svojstva $\det(CD) = \det(C) \cdot \det(D)$ i $\det(T^{-1}) = \det(T)^{-1}$ slijede direktno iz Binet-Cauchy teorema (vidi [2, str. 98]). Pokazali smo da slične matrice imaju jednake karakteristične polinome, pa su im samim time i spektri međusobno jednaki. □

Dakle, ako pravom tranformacijom sličnosti uspijemo matricu svesti na neku specijalnu formu, to nam može značajno olakšati računanje njenih svojstvenih vrijednosti.

Najpoznatija forma za kvadratne matrice je Jordanova kanonska forma (vidi [4, str. 141]), koja je zbog svoje strukture vrlo korisna u teoriji, no može se pokazati da ona općenito nije numerički stabilna (vidi [4, str. 146]). Upravo zbog toga uvodimo Schurovu formu koja ima nešto složeniju strukturu od Jordanove, ali koristi unitarnu transformaciju sličnosti, što ju čini daleko boljom opcijom u praksi.

Specijalno, može se pokazati da normalne matrice imaju dijagonalnu Schurovu formu, što znači da su unitarno slične dijagonalnoj matrici (vidi [7, str. 314]). Općenito, ako je matrica slična nekoj dijagonalnoj matrici, onda kažemo da je matrica **dijagonalizabilna**.

Svojstvene vrijednosti realnih matrica definiraju se jednako kao i kod kompleksnih, pri čemu spektar svake realne matrice ima jedno zanimljivo svojstvo.

Lema 1.3 (Kompleksne svojstvene vrijednosti realne matrice)
Neka je $A \in \mathbb{R}^{n\times n}$. Tada vrijedi $\lambda \in \sigma(A) \iff \overline{\lambda} \in \sigma(A)$.

Dokaz. Budući da se matrica A sastoji isključivo od realnih brojeva, lako se pokaže da njen karakteristični polinom mora imati realne koeficijente. Nultočke polinoma s realnim koeficijentima su ili realne ili se pojavljuju u kompleksno-konjugiranim parovima. Traženu tvrdnju dobivamo primjenom teorema 1.1. □

Drugim riječima, kod realnih matrica, kompleksne svojstvene vrijednosti uvijek se pojavljuju u kompleksno-konjugiranim parovima. Ova će nam tvrdnja biti vrlo važna kada budemo razvijali algoritam za računanje realne Schurove dekompozcije.

Iako u praksi izbjegavamo korištenje karakterističnog polinoma za računanje svojstvenih vrijednosti, postoji specijalan slučaj kod kojeg će nam on ipak biti od koristi. Naime, ukoliko nam je zadana realna matrica tipa $(2,2)$, onda je računanje nultočaka njenog karakterističnog polinoma zapravo dosta stabilna procedura pa primjenom teorema 1.1 možemo dobiti dvije formule koje će nam kasnije biti važne za implementaciju QR algoritma.

Lema 1.4. *Neka je zadana matrica $A \in \mathbb{R}^{2\times 2}$ s elementima (a_{ij}). Neka je t trag matrice A, tj. $t = a_{11} + a_{22}$ te neka je $d = \det(A) = a_{11}a_{22} - a_{12}a_{22}$. Svojstvene vrijednosti matrice A tvore kompleksno-konjugirani par brojeva ako i samo ako vrijedi*

$$\left(\frac{t}{2}\right)^2 - d < 0. \tag{1.1}$$

Dodatno, svojstvene vrijednosti matrice A iznose

$$\lambda_{1,2} = \frac{t}{2} \pm \sqrt{\left(\frac{t}{2}\right)^2 - d}. \tag{1.2}$$

Dokaz. Iz teorema 1.1 slijedi da su λ_1 i λ_2 svojstvene vrijednosti matrice A ako i samo ako su ujedno i nultočke njenog karakterističnog polinoma.

$$\begin{aligned} k_A(z) &= \det(zI - A) = \begin{vmatrix} z - a_{11} & -a_{12} \\ -a_{21} & z - a_{22} \end{vmatrix} \\ &= (z - a_{11})(z - a_{22}) - (-a_{12})(-a_{21}) \\ &= z^2 - (a_{11} + a_{22})z + (a_{11}a_{22} - a_{12}a_{21}) \\ &= z^2 - tz + d \end{aligned}$$

Izjednačavanjem k_A s nulom dobivamo kvadratnu jednadžbu čija su rješenja $\lambda_{1,2}$ dana s

$$\begin{aligned} \lambda_{1,2} &= \frac{-(-t) \pm \sqrt{(-t)^2 - 4 \cdot 1 \cdot d}}{2 \cdot 1} = \frac{t}{2} \pm \sqrt{\frac{t^2 - 4d}{4}} \\ &= \frac{t}{2} \pm \sqrt{\frac{t^2}{4} - d} = \frac{t}{2} \pm \sqrt{\left(\frac{t}{2}\right)^2 - d}. \end{aligned}$$

Rješenja kvadratne jednadžbe tvore kompleksno-konjugirani par brojeva ako i samo ako je vrijednost pod korijenom negativna, tj. ako i samo ako je zadovoljen uvjet (1.1). □

Za razvoj algoritma koji računa Schurovu dekompoziciju potrebne su nam još definicija Hessenbergove matrice te neke tvrdnje koje ovdje nećemo dokazivati, ali dat ćemo ideju dokaza ili relevantnu literaturu.

Definicija 1.5 (Gornje Hessenbergova matrica)
Kažemo da je matrica $A \in \mathbb{C}^{n\times n}$ ***gornje Hessenbergova*** *ako $\forall i > j + 1$ vrijedi $a_{ij} = 0$. Dodatno, ako za takvu matricu vrijedi da je $a_{i+1,i} \neq 0$, $\forall i \in \{1, ..., n-1\}$, onda kažemo da je* ***nereducirana****.*

Analogno se definira donje Hessenbergova matrica, no ona nam nije od tolikog interesa. Više o Hessenbergovim matricama može se naći u [9, str. 35].

Lema 1.5 (Inverz trokutaste matrice)
Ako je gornje (donje) trokutasta matrica $T \in \mathbb{C}^{n\times n}$ regularna, onda je i njen inverz gornje (donje) trokutasta matrica.

Prethodnu tvrdnju je najlakše pokazati na sljedeći način: ako rastavimo trokutastu matricu T na četiri bloka tako da su gornji lijevi i donji desni blokovi kvadratne matrice, onda primjenom blok množenja na $TT^{-1} = I$ možemo dobiti eksplicitnu formulu za inverz blok-trokutaste matrice. Preciznije, ako su blokovi T_{11} i T_{22} regularne kvadratne matrice, onda se lako pokaže da vrijedi

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \implies T^{-1} = \begin{bmatrix} T_{11}^{-1} & -T_{11}^{-1}T_{12}T_{22}^{-1} \\ 0 & T_{22}^{-1} \end{bmatrix},$$

a iz ovoga ujedno i zaključujemo da je blok-trokutasta matrica regularna ako i samo ako su joj dijagonalni blokovi regularne matrice. Sada polaznu tvrdnju lako pokažemo indukcijom po n.

Lema 1.6 (Produkt trokutaste i Hessenbergove matrice)
Ako je matrica $A \in \mathbb{C}^{n\times n}$ gornje Hessenbergova, a matrica $T \in \mathbb{C}^{n\times n}$ gornje trokutasta, onda su matrice $B = AT$ i $C = TA$ gornje Hessenbergove.

Prethodnu tvrdnju se lako pokaže običnim matričnim množenjem. Potrebno je samo iskoristiti definicije za gornje trokutastu i gornje Hessenbergovu matricu kako bi se pokazalo da za proizvoljne $i > j + 1$ vrijedi $b_{ij} = c_{ij} = 0$.

Teorem 1.7 (QR dekompozicija)
Neka je zadana matrica $A \in \mathbb{R}^{m\times n}$, $m \geq n$. Tada postoje ortogonalna matrica $Q \in \mathbb{R}^{m\times m}$ i gornje trokutasta matrica $R \in \mathbb{R}^{m\times n}$ takve da vrijedi

$$A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1,$$

pri čemu je $R_1 \in \mathbb{R}^{n\times n}$ gornje trokutasta. Dodatno, ako je matrica A punog ranga, tada su matrice Q i R jedinstvene, a dijagonalne vrijednosti matrice R_1 su pozitivne.

Dokaz. Vidi [4, str. 107-108]. □

Napomena 1.1. *Dekompoziciju $A = QR$ zovemo* ***puna QR dekompozicija****, a dekompoziciju $A = Q_1R_1$ zovemo* ***reducirana QR dekompozicija.***

Napomena 1.2. *Teorem 1.7 može se lako proširiti na slučaj $A \in \mathbb{C}^{m\times n}$, pri čemu su u tom slučaju matrice Q i R kompleksne te je matrica Q unitarna (vidi [12, str. 78-81]).*

QR dekompozicija najčešće se računa pomoću Householderovih reflektora (vidi [7, str. 209-211, 224-225]) ili pomoću Givensovih rotacija (vidi [7, str. 215-217, 226-227]). Oba algoritma izvode se u $O(n^3)$ vremenu, no može se pokazati da, ako je na ulazu gornje Hessenbergova matrica, tada Givensov algoritam računa njenu QR dekompoziciju u $O(n^2)$ vremenu (vidi [7, str. 227-228]). Ovo će nam biti od velike važnosti kada budemo optimizirali algoritam za računanje realne Schurove dekompozicije.

2 Schurova dekompozicija

Prije nego što se krenemo baviti računanjem Schurove forme, trebamo pokazati da ona zaista postoji za proizvoljnu kvadratnu matricu. Nakon toga, dat ćemo jednostavan algoritam koji u $O(n^4)$ vremenu računa kompleksnu Schurovu formu, kao i algoritam koji u $O(n^3)$ vremenu računa Hessenbegovu formu. Zatim ćemo spojiti ova dva algoritma kako bismo mogli izračunati realnu Schurovu formu u $O(n^3)$ vremenu, uz linearnu brzinu konvergencije. Naposljetku ćemo razmotriti kako ubrzati konvergenciju dobivenog algoritma pomoću tzv. *shift*-ova te kako smanjiti broj potrebnih računskih operacija pomoću tzv. *decoupling*-a.

2.1 Egzistencija Schurove dekompozicije

Ovo potpoglavlje bit će obrađeno po [7, str. 311-313, 341-342].

Za početak ćemo pokazati jednu tvrdnju koja će nam dozvoliti da složene probleme pronalaska svojstvenih vrijednosti rastavimo na dva jednostavnija.

Lema 2.1 (Spektar blok-trokutaste matrice)
Neka je matrica $T \in \mathbb{C}^{n\times n}$ razdvojena na blokove na sljedeći način:

$$T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

pri čemu je $T_{11} \in \mathbb{C}^{p\times p}$. Tada vrijedi $\sigma(T) = \sigma(T_{11}) \cup \sigma(T_{22})$.

Dokaz. Neka je (λ, x) proizvoljni svojstveni par matrice T, tj.

$$Tx = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

pri čemu su $x_1 \in \mathbb{C}^p$ i $x_2 \in \mathbb{C}^{n-p}$. Promotrimo dva disjunktna slučaja: $x_2 = 0$ i $x_2 \neq 0$.

- Ukoliko je $x_2 = 0$, slijedi da je $T_{11}x_1 = \lambda x_1$. Budući da je x svojstveni vektor neke matrice, on po definiciji ne može biti nul-vektor pa slijedi da je $x_1 \neq 0$. Zaključujemo da je $\lambda \in \sigma(T_{11})$.
- Ukoliko je $x_2 \neq 0$, slijedi da je $T_{22}x_2 = \lambda x_2$, tj. $\lambda \in \sigma(T_{22})$.

Dakle, u oba slučaja vrijedi $\lambda \in \sigma(T_{11}) \cup \sigma(T_{22})$. Pokazali smo da ovo vrijedi za proizvoljni element $\lambda \in \sigma(T)$, što nas dovodi do zaključka $\sigma(T) \subseteq \sigma(T_{11}) \cup \sigma(T_{22})$. Promotrimo sada kardinalne brojeve ovih skupova. Već smo ranije spomenuli kako se može pokazati da spektar matrice reda n sadrži točno n članova ako uzmemo u obzir njihove višestrukosti. Zbog ovog slijedi

$$|\sigma(T_{11}) \cup \sigma(T_{22})| = |\sigma(T_{11})| + |\sigma(T_{22})| = p + (n-p) = n = |\sigma(T)|.$$

Budući da ova dva skupa sadrže isti broj članova, a jedan od njih je podskup drugoga, zaključujemo da su oni zapravo jednaki, tj. $\sigma(T) = \sigma(T_{11}) \cup \sigma(T_{22})$. □

Prethodna tvrdnja je od iznimne važnosti za pronalazak spektra blok-trokutaste matrice jer nam ona dozvoljava da umjesto računanja njenog spektra jednostavno pronađemo spektre njenih dijagonalnih blokova te na kraju njihovu uniju proglasimo spektrom polazne matrice. Također primijetimo da ako je polazna matrica trokutasta, tada se na njenoj dijagonali nalazi upravo njen spektar.

Prisjetimo se leme 1.2. Ona nam dozvoljava da transformacijama sličnosti svedemo proizvoljnu matricu na jednostavniju formu (npr. blok-trokutastu), što znači da prethodnu tvrdnju možemo upotrijebiti na praktički bilo kojoj kvadratnoj matrici ukoliko ju uspijemo svesti na blok-trokutastu.

Sljedeća će nam lema trebati da bismo pokazali egzistenciju Schurove dekompozicije, a korolar koji slijedi nakon nje trebat će nam da bismo pokazali egzistenciju realne Schurove dekompozicije.

Lema 2.2. *Neka su zadane matrice $A \in \mathbb{C}^{n\times n}$, $B \in \mathbb{C}^{p\times p}$ i $X \in \mathbb{C}^{n\times p}$ takve da vrijedi $AX = XB$, pri čemu je $p \leq n$, a matrica X je punog ranga p. Tada postoji unitarna matrica $Q \in \mathbb{C}^{n\times n}$ takva da je*

$$Q^*AQ = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

pri čemu je $T_{11} \in \mathbb{C}^{p\times p}$ i vrijedi $\sigma(T_{11}) = \sigma(A) \cap \sigma(B)$.

Dokaz. Neka je sa

$$X = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \quad Q \in \mathbb{C}^{n\times n},\ R_1 \in \mathbb{C}^{p\times p},$$

dana QR dekompozicija matrice X. Budući da je matrica X punog ranga, zbog teorema 1.7 slijedi da je matrica R_1 regularna. Nadalje, neka je

$$Q^*AQ = T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}, \quad T_{11} \in \mathbb{C}^{p\times p}.$$

Nadalje, ako izraz $AX = XB$ pomnožimo slijeva sa Q^* i supstituiramo $X = QR$, dobivamo

$$\begin{aligned} Q^*AQR = Q^*QRB \quad &\iff \quad \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = \begin{bmatrix} R_1 \\ 0 \end{bmatrix} B = \begin{bmatrix} R_1B \\ 0 \end{bmatrix} \\ &\implies \quad (T_{11}R_1 = R_1B) \wedge (T_{21}R_1 = 0). \end{aligned}$$

Budući da je R_1 regularna, slijedi da je $T_{21} = 0$ te da je $B = R_1^{-1}T_{11}R_1$, tj. matrice T_{11} i B su slične pa zbog leme 1.2 slijedi $\sigma(T_{11}) = \sigma(B)$. Primjenom leme 2.1 na matricu T dobivamo $\sigma(A) = \sigma(T) = \sigma(T_{11}) \cup \sigma(T_{22})$, odakle slijedi $\sigma(T_{11}) = \sigma(A) \cap \sigma(B)$. □

Sljedeći korolar je samo specijalan slučaj prethodne leme, a trebat će nam malo kasnije kako bismo mogli dokazati teorem vezan uz realnu Schurovu dekompoziciju.

Korolar 2.3. *Neka su zadane matrice $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{p\times p}$ i $X \in \mathbb{R}^{n\times p}$ takve da vrijedi $AX = XB$, pri čemu je $p \leq n$, a matrica X je punog ranga p. Tada postoji ortogonalna matrica $Q \in \mathbb{R}^{n\times n}$ takva da je*

$$Q^TAQ = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix},$$

pri čemu je $T_{11} \in \mathbb{R}^{p\times p}$ i vrijedi $\sigma(T_{11}) = \sigma(A) \cap \sigma(B)$.

Dokaz. Korolar se dokazuje gotovo identično kao i lema 2.2. □

Sada imamo sve što nam je potrebno kako bismo pokazali da se svaka kvadratna matrica može svesti na Schurovu dekompoziciju.

Teorem 2.4 (Schurova dekompozicija)
Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$ te neka je $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$. Tada postoji unitarna matrica $Q \in \mathbb{C}^{n\times n}$ takva da je

$$Q^*AQ \;=\; T \;=\; D + N,$$

pri čemu je $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ i $N \in \mathbb{C}^{n\times n}$ je strogo gornje trokutasta, tj. $n_{ij} = 0$, $\forall i \geq j$. Dodatno, matricu Q možemo odabrati tako da se svojstvene vrijednosti λ_i pojavljuju na dijagonali matrice T u proizvoljnom poretku.

Dokaz. Teorem dokazujemo indukcijom po n. Za $n = 1$ teorem očito vrijedi. Pretpostavimo sada da teorem vrijedi za svaku kvadratnu matricu reda $n-1$. Nadalje, neka je (λ, x) proizvoljan svojstveni par matrice A te neka je definirana matrica $B = [\lambda] \in \mathbb{C}^{1\times 1}$. Budući da je x svojstveni vektor, možemo ga promatrati kao matricu punog ranga 1 pa na jednakost $Ax = xB$ možemo primijeniti lemu 2.2, odakle slijedi da postoji unitarna matrica U takva da je

$$U^*AU \;=\; \begin{bmatrix} \lambda & w^* \\ 0 & C \end{bmatrix}, \quad w \in \mathbb{C}^{n-1}, \quad C \in \mathbb{C}^{(n-1)\times(n-1)}.$$

Zbog induktivne pretpostavke, postoji unitarna matrica $\tilde{U} \in \mathbb{C}^{(n-1)\times(n-1)}$ takva da je matrica $\tilde{U}^*C\tilde{U}$ gornje trokutasta. Ako sada definiramo matricu $Q = U\operatorname{diag}(1, \tilde{U})$, onda slijedi

$$Q^*AQ \;=\; \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U}^* \end{bmatrix} U^*AU \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix} \;=\; \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U}^* \end{bmatrix} \begin{bmatrix} \lambda & w^* \\ 0 & C \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & \tilde{U} \end{bmatrix} \;=\; \begin{bmatrix} \lambda & w^*\tilde{U} \\ 0 & \tilde{U}^*C\tilde{U} \end{bmatrix},$$

tj. postoji unitarna matrica Q takva da je Q^*AQ gornje trokutasta, što je i trebalo pokazati.

Dodatno, budući da smo svojstvenu vrijednost λ odabrali proizvoljno, zaključujemo da pravilnim odabirom matrice Q možemo dovesti bilo koju svojstvenu vrijednost na proizvoljno mjesto na dijagonali pa induktivno slijedi da pravilnim odabirom matrice Q možemo poredati svojstvene vrijednosti po dijagonali matrice T u proizvoljnom poretku. □

Pokazali smo da Schurova dekompozicija postoji za svaku kompleksnu kvadratnu matricu. Međutim, u praksi naša polazna matrica A često predstavlja nekakve podatke, tj. rezultate nekakvih mjerenja, a oni su najčešće realni brojevi. Budući da je skup $\mathbb{R}$ podskup skupa $\mathbb{C}$, očito je da lema 2.1 i teorem 2.4 vrijede i u slučaju kada je zadana matrica realna.

Međutim, svojstvene vrijednosti realne matrice ne moraju nužno biti realni brojevi te ćemo u tom slučaju izračunavanjem Schurove dekompozicije od realne matrice dobiti dvije kompleksne. Nažalost, ovo je nemoguće izbjeći ukoliko želimo da naša rezultirajuća matrica bude trokutasta, no, uz malu relaksaciju uvjeta na željenu formu matrice, možemo dobiti matricu koja je *skoro* trokutasta, ali ne zahtjeva uvođenje kompleksne aritmetike.

Preostaje nam pokazati još jednu manju tvrdnju te ćemo nakon toga imati sve što nam je potrebno da bismo pokazali egzistenciju realne Schurove dekompozicije.

Propozicija 2.5. *Neka su zadane matrice $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{2\times 2}$ i $X = [x_1, x_2] \in \mathbb{R}^{n\times 2}$, $n \geq 2$, takve da vrijedi*

$$AX = XB, \quad x_2 \neq 0, \quad B \;=\; \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}.$$

Ako je $\beta \neq 0$, onda je matrica X punog ranga.

Dokaz. Pretpostavimo suprotno, tj. neka je $\beta \neq 0$ i neka je $\text{rang}(X) < 2$. Budući da vrijedi $x_2 \neq 0$, odmah zaključujemo da je $\text{rang}(X) = 1$, tj. $x_1 = c\,x_2$, $c \in \mathbb{R}\setminus\{0\}$. Zbog $AX = XB$ slijedi

$$\begin{aligned}
A\begin{bmatrix} c\,x_2 & x_2 \end{bmatrix} = \begin{bmatrix} c\,x_2 & x_2 \end{bmatrix}\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} \quad &\Longleftrightarrow \quad \begin{cases} A(c\,x_2) &= c\alpha x_2 - \beta x_2 \\ Ax_2 &= c\beta x_2 + \alpha x_2 \end{cases} \\
&\Longleftrightarrow \quad \begin{cases} cAx_2 &= (c\alpha - \beta)x_2 \\ Ax_2 &= (c\beta + \alpha)x_2 \end{cases} \\
&\Longrightarrow \quad cAx_2 = c(c\beta + \alpha)x_2 = (c\alpha - \beta)x_2 \\
&\Longrightarrow \quad c^2\beta + c\alpha = c\alpha - \beta \\
&\Longrightarrow \quad c^2\beta = -\beta.
\end{aligned}$$

Zbog $c \in \mathbb{R}\setminus\{0\}$ zaključujemo da ne može vrijediti $c^2 = -1$ te iz toga slijedi $\beta = 0$, što je u kontradikciji s početnom pretpostavkom. □

Teorem 2.6 (Realna Schurova dekompozicija)
Neka je zadana matrica $A \in \mathbb{R}^{n\times n}$ te neka je $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$. Tada postoji ortogonalna matrica $Q \in \mathbb{R}^{n\times n}$ takva da je

$$Q^T AQ = \begin{bmatrix} R_{11} & R_{12} & \ldots & R_{1m} \\ 0 & R_{22} & \ldots & R_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & R_{mm} \end{bmatrix},$$

pri čemu je svaka matrica R_{ii} ili realan broj ili matrica tipa $(2,2)$ čije svojstvene vrijednosti tvore kompleksno-konjugirani par.

Dokaz. Zbog leme 1.3, znamo da se kod realnih matrica kompleksne svojstvene vrijednosti moraju pojavljivati u kompleksno-konjugiranim parovima. Neka je $k \in \mathbb{N}_0$ definiran kao broj kompleksno-konjugiranih parova u $\sigma(A)$. Dokaz teorema ćemo sada provesti indukcijom po k. Za $k = 0$ imamo iste uvjete kao u teoremu 2.4 pa taj slučaj očito vrijedi. Neka je $k \geq 1$ te pretpostavimo da teorem vrijedi za sve realne matrice koje sadrže manje od k kompleksno-konjugiranih parova u svome spektru.

Neka je (λ, x) svojstveni par matrice $A \in \mathbb{R}^{n\times n}$ koja sadrži k kompleksno-konjugiranih parova u svome spektru, pri čemu vrijedi $\lambda = \alpha + i\beta$, $\alpha \in \mathbb{R}$, $\beta \in \mathbb{R}\setminus\{0\}$ te $x = y + iz$, $y \in \mathbb{R}^n$, $z \in \mathbb{R}^n\setminus\{0\}$, tj.

$$\begin{aligned}
Ax = \lambda x \quad &\Longleftrightarrow \quad A(y+iz) = (\alpha + i\beta)(y+iz) \\
&\Longleftrightarrow \quad \begin{cases} Ay &= \alpha y - \beta z \\ Az &= \beta y + \alpha z \end{cases} \\
&\Longleftrightarrow \quad A\begin{bmatrix} y & z \end{bmatrix} = \begin{bmatrix} y & z \end{bmatrix}\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} \\
&\Longleftrightarrow \quad AX = XB, \quad X = \begin{bmatrix} y & z \end{bmatrix}, \quad B = \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}.
\end{aligned}$$

Zbog $\beta \neq 0$, $z \neq 0$ i propozicije 2.5 slijedi da je matrica X punog ranga 2. Sada na jednakost $AX = XB$ možemo primijeniti korolar 2.3, odakle slijedi da postoji ortogonalna matrica $U \in \mathbb{R}^{n\times n}$ takva da je

$$U^T AU = T = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix}, \quad T_{11} \in \mathbb{R}^{2\times 2}, \quad \sigma(T_{11}) = \sigma(A) \cap \sigma(B),$$

pri čemu je očito da vrijedi $\sigma(B) = \{\lambda, \overline{\lambda}\}$ pa odatle slijedi da je $\sigma(T_{11}) = \{\lambda, \overline{\lambda}\}$. Znamo da A sadrži k kompleksno-konjugiranih parova u svome spektru, što znači da ih i T sadrži isto toliko, a budući da se jedan od njih nalazi u bloku T_{11}, zaključujemo da ih T_{22} sadrži $k-1$, tj. manje od k. Prema pretpostavci indukcije, postoji ortogonalna matrica $\tilde{U}$ takva da $\tilde{U}^T T_{22} \tilde{U}$ ima traženu strukturu. Slično kao u dokazu teorema 2.4, ako postavimo $Q = U \operatorname{diag}(I_2, \tilde{U})$, matrica $Q^T A Q$ će imati traženu strukturu. □

Dakle, ako nam je zadana realna kvadratna matrica, imamo dvije opcije: možemo ju svesti na običnu Schurovu formu te time dobiti gornje trokutastu matricu (ali bi matrica mogla postati kompleksna) ili ju možemo svesti na realnu Schurovu formu te time dobiti blok-trokutastu matricu (ali će matrica ostati realna). U literaturi se 2×2 blokovi realne Schurove forme često još nazivaju ***Schur bumps.***

Općenito ne postoji *bolji* izbor između ovih dviju formi, nego se u praksi koriste obje, a na nama je da odaberemo koja je od njih prikladnija za problem kojeg trenutno rješavamo. Također je moguće napraviti i kombinaciju, tj. prvo izračunamo realnu Schurovu dekompoziciju zadane matrice, a zatim za svaki *bump* zasebno izračunamo njegove svojstvene vrijednosti.

2.2 Računanje Schurove forme

Ovo potpoglavlje bit će obrađeno po [4, str. 153-166] i [7, str. 330-345].

Prije nego što damo osnovni algoritam za računanje Schurove forme, kratko ćemo spomenuti još dva algoritma vezana uz problem određivanja svojstvenih vrijednosti i svojstvenih vektora.

Neka je zadana matrica $A \in \mathbb{C}^{n \times n}$ sa spektrom $\sigma(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$ i svojstvenim vektorima $s_1, s_2, \ldots, s_n$, pri čemu vrijedi

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|, \qquad As_j = \lambda_j s_j,\ s_j \neq 0, \quad j = 1, \ldots, n.$$

Nadalje, odaberimo *početni* vektor $x^{(0)}$ tako da vrijedi $s_1^* x^{(0)} \neq 0$, tj. tako da nije okomit na vektor s_1. Zapišimo sada $x^{(0)}$ kao linearnu kombinaciju svojstvenih vektora matrice A.

$$x^{(0)} = \alpha_1 s_1 + \alpha_2 s_2 + \cdots + \alpha_n s_n.$$

Ako smo dobro odabrali $x^{(0)}$, onda će vrijediti $\alpha_1 \neq 0$. Pomnožimo vektor $x^{(0)}$ matricom A.

$$\begin{aligned} Ax^{(0)} &= \alpha_1 As_1 + \alpha_2 As_2 + \cdots + \alpha_n As_n \\ &= \alpha_1 \lambda_1 s_1 + \alpha_2 \lambda_2 s_2 + \cdots + \alpha_n \lambda_n s_n \end{aligned}$$

Induktivno se pokaže da vrijedi

$$\begin{aligned} A^k x^{(0)} &= \alpha_1 A^k s_1 + \alpha_2 A^k s_2 + \cdots + \alpha_n A^k s_n \\ &= \alpha_1 \lambda_1^k s_1 + \alpha_2 \lambda_2^k s_2 + \cdots + \alpha_n^k \lambda_n s_n \\ &= \lambda_1^k \left(\alpha_1 s_1 + \alpha_2 \frac{\lambda_2^k}{\lambda_1^k} s_2 + \cdots + \alpha_n^k \frac{\lambda_n^k}{\lambda_1^k} s_n \right) \\ &= \lambda_1^k \left[\alpha_1 s_1 + \alpha_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k s_2 + \cdots + \alpha_n \left(\frac{\lambda_n}{\lambda_1} \right)^k s_n \right] \end{aligned}$$

Budući da je $|\lambda_1| > |\lambda_j|\ \forall j = 2, \ldots, n$, lako se pokaže da je

$$\lim_{k\to\infty} \left(\frac{\lambda_j}{\lambda_1}\right)^k = 0, \quad \forall j = 2, \ldots n$$

Sada zbog $|\lambda_1| > 0$ i $\alpha_1 \neq 0$ dobivamo

$$\lim_{k\to\infty} \frac{A^k x^{(0)}}{\alpha_1 \lambda_1^k} = s_1.$$

Ovim postupkom dobili smo metodu za računanje svojstvenog vektora ulazne matrice koji odgovara svojstvenoj vrijednosti s najvećom apsolutnom vrijednošću. Ukoliko normiramo dobiveni vektor s_1, možemo dobiti i eksplicitnu formulu za njegovu pripadnu svojstvenu vrijednost λ_1, tj. $\lambda_1 = s_1^* A s_1$. Ova se metoda u literaturi zove ***metoda potencija.*** Zapišimo ju sada formalno (algoritam preuzet iz [7, str. 330]).

Algoritam 2.1 (Metoda potencija)
Ulaz: $A \in \mathbb{C}^{n\times n}$, $x^{(0)} \in \mathbb{C}^n$
Izlaz: (λ, x), *t.d. je* $|\lambda| = \max\{|\lambda_i| : \lambda_i \in \sigma(A)\}$
1. $x \leftarrow x^{(0)}$
2. ***while*** *ne konvergira*
3. $\quad y \leftarrow Ax$
4. $\quad x \leftarrow y/\|y\|$
5. $\quad \lambda \leftarrow x^* Ax$
6. ***end while***

Valja primijetiti da brzina konvergencije ovog algoritma ovisi o omjeru $|\lambda_2|/|\lambda_1|$, tj. algoritam će brže konvergirati ako matrica na ulazu ima jednu "jako dominantnu" svojstvenu vrijednost.

Dodatno, algoritam se može modificirati tako da računa neku drugu svojstvenu vrijednost umjesto one s najvećom apsolutnom vrijednošću. Ovo se može postići tzv. ***algoritmom inverznih iteracija sa shiftom***, o kojem zainteresirani čitatelj može više vidjeti u [4, str. 155-156].

Sljedeći algoritam služi kao poopćenje algoritma 2.1 te se koristi kada želimo izračunati tzv. invarijantne potprostore zadane matrice, pri čemu je `reduced_QR` jedna od metoda koja računa reduciranu QR dekompoziciju dane matrice (npr. Householderova ili Givensova). Algoritam je preuzet iz [7, str. 332].

Algoritam 2.2 (Metoda ortogonalnih iteracija)
Ulaz: $A \in \mathbb{C}^{n\times n}$, $Q_0 \in \mathbb{C}^{n\times r}$ *ortogonalna*
Izlaz: $Q \in \mathbb{C}^{n\times r}$
1. $k \leftarrow 0$
2. ***while*** *ne konvergira*
3. $\quad Y_k \leftarrow AQ_{k-1}$
4. $\quad (Q_k, R_k) \leftarrow$ `reduced_QR`(Y_k)
5. $\quad Q \leftarrow Q_k$
6. $\quad k \leftarrow k + 1$
7. ***end while***

Prethodni algoritam vraća matricu Q čiji stupci razapinju r-dimenzionalni potprostor matrice A, no ovdje nam to nije od tolikog interesa za $r < n$. Ono što nam je trenutno od

interesa je slučaj kada postavimo $r = n$, jer će tada niz matrica (T_k) definiran sa

$$T_k = Q_k^* A Q_k, \quad k = 0, 1, 2, \ldots \tag{2.1}$$

konvergirati prema Schurovoj formi matrice A. Dokaz da niz (T_k) zaista konvergira prema Schurovoj formi vrlo je složen. Ideja dokaza može se pronaći u [4, str. 157-158], a puni dokaz može se pronaći u [7, str. 336-339]. Primijetimo da ovaj algoritam ima mali problem, a to je da nakon računanja QR dekompozicije matricu R_k uopće ne koristimo.

Promotrimo sada u kakvoj su vezi matrice T_k i T_{k-1} definirane kao u (2.1). Primijetimo da matricu T_{k-1} možemo zapisati kao

$$\begin{aligned} T_{k-1} &= Q_{k-1}^* A Q_{k-1} = Q_{k-1}^* (A Q_{k-1}) = Q_{k-1}^* Y_k \\ &= Q_{k-1}^* (Q_k R_k) = (Q_{k-1}^* Q_k) R_k =: \tilde{Q}_k R_k \end{aligned}$$

dok matricu T_k možemo zapisati kao

$$\begin{aligned} T_k &= Q_k^* A Q_k = Q_k^* A (Q_{k-1} Q_{k-1}^*) Q_k \\ &= Q_k^* (A Q_{k-1})(Q_{k-1}^* Q_k) = Q_k^* Y_k \tilde{Q}_k \\ &= Q_k^* (Q_k R_k) \tilde{Q}_k = (Q_k^* Q_k) R_k \tilde{Q}_k = R_k \tilde{Q}_k. \end{aligned}$$

Drugim riječima, ukoliko nam je poznata QR dekompozicija matrice T_{k-1}, tada sljedeću iteraciju T_k možemo dobiti tako da jednostavno zamijenimo mjesta matricama Q i R te ih pomnožimo. Zapišimo sada ovaj postupak na formalan način (algoritam preuzet iz [7, str. 330]).

Algoritam 2.3 (Osnovni QR algoritam)
Ulaz: $A \in \mathbb{C}^{n \times n}$, $Q_0 \in \mathbb{C}^{n \times n}$ *unitarna*
Izlaz: $Q \in \mathbb{C}^{n \times n}, R \in \mathbb{C}^{n \times n}$, *t.d. je* $R = Q^* A Q$ *Schurova dekompozicija od* A
1. $k \leftarrow 0$
2. $Q \leftarrow Q_0, \ T_0 \leftarrow Q_0^* A Q_0$
3. ***while*** *ne konvergira*
4. $\quad (Q_k, R_k) \leftarrow$ **`full_QR`**(T_{k-1})
5. $\quad T_k \leftarrow R_k Q_k$
6. $\quad Q \leftarrow Q \cdot Q_k, \ R \leftarrow T_k$
7. $\quad k \leftarrow k + 1$
8. ***end while***

Slično kao kod algoritma 2.2, **`full_QR`** je jedna od metoda koja računa punu QR dekompoziciju dane matrice, npr. Householderova ili Givensova. Za razliku od algoritma 2.2, algoritam 2.3 zapravo iskoristi obje matrice iz QR dekompozicije, umjesto da jednu od njih samo odbaci tijekom sljedeće iteracije.

Iako je prilično jednostavan, glavni nedostatak algoritma 2.3 je njegova vremenska složenost. Kao što smo već ranije spomenuli, računanje QR dekompozicije proizvoljne matrice odvija se u $O(n^3)$ vremenu, a lako se pokaže da se i množenje dviju proizvoljnih matrica također odvija u $O(n^3)$ vremenu. U kombinaciji sa *for* petljom koja se u praksi pozove $O(n)$ puta, dobivamo algoritam čija je vremenska složenost $O(n^4)$, što je za praktične primjene jednostavno presporo.

Iako algoritam 2.3 ima preveliku vremensku složenost, ipak ga možemo značajno ubrzati ukoliko je matrica na ulazu gornje Hessenbergova, budući da se QR dekompozicija takvih

matrica može izračunati u $O(n^2)$ vremenu. Kako bi uopće imalo smisla razmatrati ovakve varijante algoritma, prvo se trebamo uvjeriti da se svaka realna matrica može svesti na gornje Hessenbergovu korištenjem ortogonalne transformacije sličnosti. Dokaz ćemo provesti pomoću Householderovih matrica o kojima zainteresirani čitatelj može više pronaći na [3, str. 85-88], [4, str. 119-121] i [7, str. 209-211]. U ovom trenutku nam nije toliko važno kako se one računaju, nego je samo važno napomenuti da su Householderove matrice ortogonalne.

Teorem 2.7 (Gornje Hessenbergova dekompozicija)
Neka je zadana matrica $A \in \mathbb{R}^{n\times n}$ te neka je definiran niz matrica $A^{(0)}, \ldots, A^{(n-2)}$ kao

$$A^{(0)} = A, \quad A^{(k)} = P_k A^{(k-1)} P_k^T, \quad k = 1, 2, \ldots, n-2,$$

pri čemu je $P_k = \operatorname{diag}(I_k, H_k)$, a svaka od matrica $H_k \in \mathbb{R}^{(n-k)\times(n-k)}$ je Householderova matrica koja poništava elemente ispod $a_{k+1,k}^{(k-1)}$, tj.

$$\tilde{A}_{k-1} = \begin{bmatrix} a_{k+1,k}^{(k-1)} \\ a_{k+2,k}^{(k-1)} \\ \vdots \\ a_{n,k}^{(k-1)} \end{bmatrix} \quad \Longrightarrow \quad H_k \tilde{A}_{k-1} = \begin{bmatrix} \|\tilde{A}_{k-1}\| \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad k = 1, 2, \ldots, n-2.$$

Tada je matrica $A^{(n-2)}$ gornje Hessenbergova.

Dokaz. Dokaz ćemo provesti indukcijom po k. Preciznije, pokazat ćemo da ako matrica P_k poništi sve elemente ispod $a_{k+1,k}^{(k-1)}$, onda će ti elementi i dalje ostati nule nakon što matricu $P_k A^{(k-1)}$ zdesna pomnožimo matricom P_k^T, tj. matrica $A^{(k)}$ će biti Hessenbergova u prvih k stupaca. Pogledajmo prvo kako matrica P_1 djeluje na matricu A.

$$P_1 A P_1^T = \begin{bmatrix} I_1 & 0 \\ 0 & H_1 \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} I_1 & 0 \\ 0 & H_1^T \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12}H_1^T \\ H_1 A_{21} & H_1 A_{22} H_1^T \end{bmatrix} = \begin{bmatrix} a_{11} & A_{12}H_1^T \\ H_1\tilde{A}_1 & H_1 A_{22} H_1^T \end{bmatrix}$$

Dakle, ako matricom P_1 poništimo sve elemente ispod a_{12} te zatim dobivenu matricu zdesna pomnožimo matricom P_1^T, dobivena matrica $A^{(1)}$ bit će Hessenbergova u prvom stupcu.

Pretpostavimo da je matrica $A^{(k-1)}$ Hessenbergova u prvih $k-1$ stupaca, tj.

$$A^{(k-1)} = \begin{bmatrix} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ 0 & B_{32} & B_{33} \end{bmatrix}, \quad B_{11} \in \mathbb{R}^{(k-1)\times(k-1)},\ B_{22} \in \mathbb{R}^{1\times 1},\ B_{33} \in \mathbb{R}^{(n-k)\times(n-k)},$$

pri čemu je očito da je $B_{32} = \tilde{A}_{k-1}$. Pokažimo da je matrica $A^{(k)}$ Hessenbergova u prvih k stupaca. Vrijedi

$$\begin{aligned} P_k A^{(k-1)} P_k^T &= \begin{bmatrix} I_k & 0 \\ 0 & H_k \end{bmatrix} \left[\begin{array}{cc:c} B_{11} & B_{12} & B_{13} \\ B_{21} & B_{22} & B_{23} \\ \hdashline 0 & B_{32} & B_{33} \end{array}\right] \begin{bmatrix} I_k & 0 \\ 0 & H_k^T \end{bmatrix} \\ &= \begin{bmatrix} B_{11} & B_{12} & B_{13}H_k^T \\ B_{21} & B_{22} & B_{23}H_k^T \\ 0 & H_k B_{32} & H_k B_{33} H_k^T \end{bmatrix}, \end{aligned}$$

pri čemu je $H_k B_{32} = H_k \tilde{A}_{k-1}$, što znači da su svi elementi tog vektora ispod prvog jednaki nuli. Time smo pokazali da je matrica $A^{(k)}$ Hessenbergova u prvih k stupaca pa zbog indukcije slijedi da je matrica $A^{(n-2)}$ Hessenbergova u prvih $n-2$ stupaca, tj. $A^{(n-2)}$ je gornje Hessenbergova. □

Dakle, ako proizvoljnu matricu $A \in \mathbb{R}^{n\times n}$ svedemo na gornje Hessenbergovu sukcesivnom primjenom Householderovih transformacija P te zatim istim transformacijama djelujemo na dobivenu matricu PA s druge strane, matrica će ostati gornje Hessenbergova. Ovo je važno upravo zato što smo ovim postupkom dobili matricu koja je ortogonalno slična polaznoj, a nalazi se u formi koja nam je vrlo pogodna za primjenu QR algoritma.

Kako bismo dovršili naš algoritam, potrebno nam je još samo pokazati da, ako Givensovim rotacijama izračunamo QR dekompoziciju gornje Hessenbergove matrice $T_{k-1} = Q_k R_k$, tada će matrica $T_k = R_k Q_k$ također biti gornje Hessenbergova. Ovu tvrdnju nećemo formalno dokazivati, nego ćemo samo objasniti zašto ona vrijedi.

Naime, u [4, str. 122] ili u [7, str. 215] možemo vidjeti da su za $i \neq j$ Givensove rotacije $G(i, j, \theta)$ definirane kao da smo uzeli jediničnu matricu G te *prepravili* njene elemente na sljedeći način: $g_{ii} = g_{jj} = \cos(\theta)$, $g_{ij} = \sin(\theta)$, $g_{ji} = -\sin(\theta)$. MATLAB kod koji izračunava vrijednosti $\sin(\theta)$ i $\cos(\theta)$ može se naći u dodatku A pod algoritmom A.1. Dakle, ako u k-toj iteraciji Givensovog algoritma definiramo matricu G_k kao

$$G_k := G(k, k+1, \theta) = \begin{bmatrix} I_{k-1} & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & I_{n-k-1} \end{bmatrix}, \quad k = 1, \ldots, n-1,$$

onda će ona poništiti element $a_{k,k+1}$, dok će sve nule ispod njega ostati netaknute. Nakon završetka izvođenja Givensovog algoritma, krajnji rezultat bit će matrica $R = Q^T A$, pri čemu je $Q = G_1 G_2 \cdots G_{n-2} G_{n-1}$, a vrlo lako se može pokazati da je tada matrica Q gornje Hessenbergova. Sada zbog leme 1.6 slijedi da je i matrica RQ gornje Hessenbergova.

Dakle, algoritam 2.3 možemo svesti na vremensku složenost $O(n^3)$ na sljedeći način:

1. Izračunamo Hessenbergovu formu polazne matrice A kao $H_0 = Q_0^T A Q_0$ (npr. možemo koristiti MATLAB naredbu `[Q_0, H_0] = hess(A)`).

2. Korištenjem Givensovog algoritma izračunamo QR dekompoziciju matrice H_{k-1} kao $H_{k-1} = Q_k R_k$, pri čemu je Q_k gornje Hessenbergova. Izračunamo $H_k = R_k Q_k$, koja je ponovno gornje Hessenbergova. Ovaj se korak u literaturi često zove ***QR step,*** a budući da je matrica na ulazu gornja Hessenbergova, ova specijalna varijanta zove se ***Hessenberg-QR step*** (MATLAB kod može se naći u dodatku A pod algoritmom A.2).

3. Ponavljamo korak 2 dok svi elementi $h_{i,i+1}$, $i = 1, \ldots, n-1$, ne budu dovoljno blizu nuli (npr. $|h_{i,i+1}| < 10^{-12}$), osim eventualno *bump*-ova - njih možemo vrlo lako prepoznati tijekom izvođenja algoritma ako na blokove `H(i:i+1, i:i+1)` primjenimo lemu 1.4, s time što moramo paziti na to da ni gore lijevo ni dolje desno od *bump*-a ne možemo imati još jedan *bump*.

U sljedećem potpoglavlju navest ćemo još jedan način da se konvergencija ovog algoritma dodatno ubrza te jedan način pomoću kojeg možemo malo smanjiti broj računskih operacija.

2.3 *Decoupling* i *shiftovi*

Ovo potpoglavlje bit će obrađeno po [4, str. 167-173] i [7, str. 352-359].

Uvođenjem Hessenbergove forme već smo značajno smanjili broj računskih operacija potrebnih za izvođenje algoritma 2.3, no, jednim jednostavnim trikom možemo taj broj još malo smanjiti. Naime, u praksi se nakon određenog broja iteracija dogodi da neki element $h_{i,i+1}$ postane nula ili nešto vrlo blizu nuli te time matrica H postane blok-trokutasta.

Prisjetimo se leme 2.1. Ona nam dozvoljava da problem određivanja svojstvenih vrijednosti blok-trokutaste matrice razdvojimo na dva potproblema te njihovu uniju proglasimo našim konačnim rješenjem. Preciznije, ako imamo matricu

$$H = \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix},$$

pri čemu su H_{11} i H_{22} kvadratne matrice, onda možemo pronaći unitarne matrice Q_1 i Q_2 koje redom svode matrice H_{11} i H_{22} na Schurovu formu te će time Schurova dekompozicija naše *glavne* matrice H biti dana sa

$$T = Q^*HQ = \begin{bmatrix} Q_1^* & 0 \\ 0 & Q_2^* \end{bmatrix} \begin{bmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{bmatrix} \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} = \begin{bmatrix} Q_1^*H_{11}Q_1 & Q_1^*H_{12}Q_2 \\ 0 & Q_2^*H_{22}Q_2 \end{bmatrix}.$$

Ovaj se postupak zove ***decoupling***, tj. ***razdvajanje***. Budući da u praksi ne znamo tijekom koje iteracije će ovo biti moguće izvesti niti znamo koji će se element dovoljno približiti nuli, potrebno je nakon svake iteracije provjeriti sve subdijagonalne elemente trenutne matrice te napraviti *decoupling* ukoliko je neki od njih dovoljno blizu nuli. Za ovu provjeru potrebno je samo $O(n)$ operacija, što gotovo nimalo ne utječe na vrijeme izvođenja algoritma, a može značajno smanjiti dimenzije problema s kojim radimo.

Preostaje nam još kratko prodiskutirati kako možemo ubrzati konvergenciju QR algoritma. Pretpostavimo da želimo izračunati Schurovu dekompoziciju zadane matrice te u tu svrhu koristimo sljedeći algoritam:

Algoritam 2.4 (QR algoritam sa *shiftom*)
Ulaz: $A \in \mathbb{R}^{n\times n}$
Izlaz: $Q \in \mathbb{C}^{n\times n}, R \in \mathbb{C}^{n\times n}$, *t.d. je* $R = Q^*AQ$ *Schurova dekompozicija od* A
1. $(Q_0, H_0) \leftarrow$ **`hess`**(A), $Q \leftarrow Q_0$
2. ***for*** $k = 1, 2, \ldots$
3. *Izračunaj skalar* $\mu \in \mathbb{C}$.
4. $(Q_k, R_k) \leftarrow$ **`hessQRstep`**$(H_{k-1} - \mu I)$
5. $H_k \leftarrow R_k + \mu I$
6. $Q \leftarrow Q \cdot Q_k$, $R \leftarrow H_k$
7. ***end for***

MATLAB kod funkcije **`hessQRstep`** može se naći u dodatku A pod algoritmom A.2. Naime, ako je $H_{k-1} - \mu I = Q_kR_kQ_k^*$, onda slijedi

$$\begin{aligned} H_k &= R_k + \mu I = (Q_k^*Q_k)R_k(Q_k^*Q_k) + (\mu I)(Q_k^*Q_k) \\ &= Q_k^*(Q_kR_kQ_k^*)Q_k + Q_k^*(\mu I)Q_k = Q_k^*(Q_kR_kQ_k^* + \mu I)Q_k \\ &= Q_k^*(H_{k-1} - \mu I + \mu I)Q_k = Q_k^*H_{k-1}Q_k, \end{aligned} \tag{2.2}$$

tj. matrica H_k unitarno je slična matrici H_{k-1}. Ova strategija zove se *shift*-anje, tj. *pomicanje*, budući da oduzimanjem $A - \mu I$ pomičemo cijeli spektar matrice A za vrijednost μ ulijevo (dokaz slijedi iz teorema 1.1). Skalar μ u ovom kontekstu zovemo ***shift.*** Može

se pokazati da, ako za *shift* postavimo $\mu \in \sigma(A)$, onda će se na kraju trenutne iteracije algoritma dogoditi *decoupling*, i to na zadnjem bloku (vidi [7, str. 353]). Budući da nema smisla računati svojstvene vrijednosti matrice kako bismo ubrzali algoritam koji upravo služi računanju svojstvenih vrijednosti, *shiftove* ćemo morati nekako aproksimirati. Može se pokazati da je $\mu = h_{nn}$ prilično dobra aproksimacija jer će u tom slučaju brzina konvergencije QR algoritma porasti s linearne na kvadratnu (vidi [7, str. 354-355]).

Promotrimo sada slučaj kada je matrica na ulazu realna te želimo dobiti njenu realnu Schurovu dekompoziciju. Budući da svojstvene vrijednosti realne matrice mogu biti kompleksni brojevi, aproksimacija $\mu = h_{nn}$ može nam stvarati probleme ukoliko su svojstvene vrijednosti bloka

$$H' = \begin{bmatrix} h_{mm} & h_{mn} \\ h_{nm} & h_{nn} \end{bmatrix}, \quad m = n - 1$$

kompleksno-konjugirani par brojeva. Kako bismo izbjegli ovo, uvodimo tzv. ***double shift*** strategiju na sljedeći način:

$$\begin{aligned} H - \mu_1 I &= Q_1 R_1; \\ H_1 &= R_1 Q_1 + \mu_1 I; \\ H_1 - \mu_2 I &= Q_2 R_2; \\ H_2 &= R_2 Q_2 + \mu_2 I, \end{aligned}$$

pri čemu su μ_1 i μ_2 svojstvene vrijednosti bloka H'. Primijetimo da vrijedi

$$H_2 \overset{(2.2)}{=} Q_2^* H_1 Q_2 \overset{(2.2)}{=} Q_2^*(Q_1^* H Q_1) Q_2 = (Q_1 Q_2)^* H (Q_1 Q_2) =: Q^* H Q,$$

tj. matrica H_2 unitarno je slična matrici H, što znači da uz pravi izbor matrice Q možemo ove dvije iteracije sa shiftovima zamijeniti jednom običnom iteracijom QR algoritma, a da pritom zadržimo kvadratnu brzinu konvergencije. Pogledajmo sada kako izvesti ovo bez uvođenja kompleksne aritmetike. Ako definiramo matricu $M = (H - \mu_1 I)(H - \mu_2 I)$, onda se lako može pokazati da vrijedi $M = (Q_1 Q_2)(R_2 R_1)$. Dodatno, može se pokazati da su njeni elementi realni, neovisno o tome jesu li μ_1 i μ_2 dva realna broja ili kompleksno-konjugirani par (vidi [4, str. 171] ili [7, str. 355-356]). Konačno, matricu H_2 možemo dobiti tako da izračunamo QR dekompoziciju matrice $M = QR$ te postavimo $H_2 = RQ$.

Nažalost, može se pokazati da eksplicitno izračunavanje matrice M ima složenost $O(n^3)$, što ponovno vraća cjelokupni algoritam na složenost $O(n^4)$ (vidi [7, str. 356]). Međutim, eksplicitno izračunavanje matrice M može se izbjeći korištenjem tzv. *implicitnih shiftova*, ali ta strategija izlazi izvan ovkira ovog rada. Više o implicitnim *shiftovima* može se pročitati u [4, str. 169-172] i [7, str. 356-358].

Iako se jednostruki *shiftovi* u praksi izbjegavaju, to ne znači da oni nemaju apsolutno nikakvu primjenu. Naime, izračunavanjem QR dekompozicije realne matrice uvijek ćemo dobiti dvije realne matrice Q i R, što znači da algoritam 2.3 zapravo nikada neće ni moći izračunati kompleksnu Schurovu dekompoziciju realne matrice. Ovo se može zaobići tako da koristimo jednostruki *shift* $\mu \in \mathbb{C} \setminus \mathbb{R}$. Preciznije, ako odredimo jednu svojstvenu vrijednost nekog *bump*-a korištenjem leme 1.4, možemo tu vrijednost upotrijebiti kao *shift* te će zbog toga *bump* konvergirati na kraju trenutne iteracije. Naposljetku korištenjem *decoupling*-a ažuriramo relevantne retke i stupce matrice R_k te eventualno relevantne stupce matrice Q.

U dodatku A, pod algoritmom A.4, može se pronaći algoritam koji objedinjuje sve dosad spomenute modifikacije osnovnog QR algoritma, osim dvostrukih *shiftova.*

3 Primjene Schurove dekompozicije

U ovom poglavlju navest ćemo neke važnije primjene Schurove dekompozicije. Glavna motivacija za njeno računanje je traženje spektra zadane matrice na numerički stabilan način, no, osim toga, spomenut ćemo još neke važne primjene: traženje nultočaka polinoma proizvoljnog stupnja, izračun matrične eksponencijalne funkcije te još nekih problema koji su usko vezani uz teoriju upravljanja.

3.1 Računanje nultočaka polinoma i svojstvenih vrijednosti

Prisjetimo se teorema 1.1. Ona nam dozvoljava da problem određivanja svojstvenih vrijednosti svedemo na problem računanja nultočaka karakterističnog polinoma. Već smo ranije spomenuli da ne postoje egzaktne algebarske formule za nultočke polinoma kojima je stupanj veći od 4, što znači da se ovaj problem mora rješavati iterativnim numeričkim metodama, koje obično nisu pretjerano efikasne ukoliko je polinom visokog stupnja. Međutim, kroz prethodnih desetak stranica upravo smo izgradili numerički stabilnu i efikasnu metodu za određivanje svojstvenih vrijednosti proizvoljne kvadratne matrice bez potrebe za korištenjem karakterističnog polinoma, osim da bismo prebacili matricu iz realne Schurove forme u kompleksnu.

Što ako imamo obrnuti problem, tj. ako nam nije zadana matrica kojoj treba izračunati svojstvene vrijednosti, nego nam je baš zadan polinom kojemu želimo izračunati nultočke? Odgovor na to daje nam tzv. ***matrica pratilac*** (eng. *companion matrix form*). Nadalje u tekstu koristit ćemo naziv *companion matrix* forma.

Teorem 3.1 (Companion matrix forma)
Neka je zadan polinom

$$p(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0$$

te neka je njegova ***companion matrix forma*** *definirana kao*

$$C = \begin{bmatrix} 0 & 0 & \dots & 0 & -a_0 \\ 1 & 0 & \dots & 0 & -a_1 \\ 0 & 1 & \dots & 0 & -a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & -a_{n-1} \end{bmatrix}.$$

Tada je p karakteristični polinom matrice C.

Dokaz. Teorem ćemo dokazati indukcijom po n, gdje je n stupanj polinoma p.

Za $n = 1$ imamo polinom $p_1(z) = z + a_0$ i $C_1 = [-a_0]$, pa je $k_{C_1}(z) = \det(z + a_0) = z + a_0$. Očito je $p_1 = k_{C_1}$. Pretpostavimo sada da teorem vrijedi za svaki polinom stupnja $n - 1$.

Neka je p polinom stupnja n definiran kao

$$p(z) = z^n + a_{n-1}z^{n-1} + \cdots + a_1 z + a_0.$$

Razvojem determinante po prvom retku slijedi

$$k_C(z) = \det(zI - C) = \begin{vmatrix} z & 0 & \dots & 0 & a_0 \\ -1 & z & \dots & 0 & a_1 \\ 0 & -1 & \dots & 0 & a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & z + a_{n-1} \end{vmatrix}$$

$$= z \begin{vmatrix} z & 0 & \dots & 0 & a_1 \\ -1 & z & \dots & 0 & a_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & -1 & z + a_{n-1} \end{vmatrix} + (-1)^{n+1} a_0 \begin{vmatrix} -1 & z & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & -1 \end{vmatrix}.$$

Ako sada iskoristimo pretpostavku indukcije te svojstvo da je determinanta trokutaste matrice jednaka umnošku njenih dijagonalnih elemenata, onda dobivamo

$$\begin{aligned} k_C(z) &= z \cdot (z^{n-1} + a_{n-1} z^{n-2} + \dots + a_2 z + a_1) + (-1)^{2n} \cdot a_0 \\ &= z^n + a_{n-1} z^{n-1} + \dots + a_2 z^2 + a_1 z + a_0 \\ &= p(z). \end{aligned}$$

□

Drugim riječima, ako želimo pronaći nultočke polinoma proizvoljnog stupnja, možemo definirati matricu C koja odgovora njegovoj *companion matrix* formi te zatim izračunati svojstvene vrijednosti matrice C tako da ju svedemo na Schurovu formu. Primijetimo da je *companion matrix* forma svakog polinoma gornje Hessenbergova, što nam je vrlo pogodna situacija za izvođenje QR algoritma. Prethodnim teoremom pokazali smo da su određivanje svojstvenih vrijednosti matrice i računanje nultočaka polinoma ekvivalentni problemi.

Pogledajmo kako ova metoda radi na nekoliko primjera, uz napomenu da se MATLAB kod funkcije `mySchur` može naći u dodatku A pod algoritmom A.4. Prvo ćemo konstruirati jedan manji primjer kako bismo na njemu malo detaljnije proučili cijelu proceduru.

Primjer 3.1.1. Odredimo nultočke polinoma zadanog sa

$$p(x) = (x-2)(x-7)(x-8) = x^3 - 17x^2 + 86x - 112.$$

Zapišimo prvo polinom p u njegovoj *companion matrix* formi.

$$C = \begin{bmatrix} 0 & 0 & 112 \\ 1 & 0 & -86 \\ 0 & 1 & 17 \end{bmatrix}$$

Rješavanje originalnog problema sada se svodi na određivanje svojstvenih vrijednosti matrice C, a njih možemo dobiti tako da matricu C svedemo na kompleksnu Schurovu formu te iščitamo njene dijagonalne vrijednosti. Iako nam za rješavanje ovog problema nije potrebno eksplicitno formiranje matrice Q, ipak ćemo ju izračunati kako bismo se uvjerili u ispravnost implementacije. Pozivanjem funkcije `[R, Q] = mySchur(C,'complex')` dobivamo sljedeće matrice:

$$R = \begin{bmatrix} 8.0000 & -68.7386 & 123.4052 \\ 0.0000 & 7.0000 & -12.6549 \\ 0 & 0.0000 & 2.0000 \end{bmatrix}, \quad Q = \begin{bmatrix} 0.8397 & 0.4973 & 0.2182 \\ -0.5398 & 0.7198 & 0.4364 \\ 0.0600 & -0.4843 & 0.8729 \end{bmatrix}.$$

Definirajmo vektor r_1 kao dijagonalu matrice R te zatim evaluirajmo polinom p u izračunatim nultočkama korištenjem MATLAB naredbe `r2 = polyval(p, r1)`. Dobivamo sljedeće rezultate:

- $\|Q^*Q\|_2 - 1 \approx 9.77 \cdot 10^{-15}$, $\|QQ^*\|_2 - 1 \approx 9.33 \cdot 10^{-15}$, tj. $\|Q^*Q\|_2 \approx 1$, $\|QQ^*\|_2 \approx 1$, čime smo pokazali da je matrica Q zaista unitarna;
- norma reziduala Schurove dekompozicije iznosi $\|C - QRQ^*\|_2 = 1.8477 \cdot 10^{-12}$;
- $\|r_2\|_\infty = 4.0359 \cdot 10^{-12}$, čime smo pokazali da su sve izračunate nultočke polinoma p točne barem u prvih 11 decimalnih mjesta.

Pogledajmo koliko dobro naša implementacija radi kada je polinom na ulazu reda većeg od 4 i ima kompleksne nultočke.

Primjer 3.1.2. Odredimo nultočke polinoma zadanog sa

$$p(x) = x^5 - x - 1.$$

Ponavljamo istu proceduru kao i u prethodnom primjeru: zapisujemo polinom p u njegovoj *companion matrix* formi C, izračunavamo kompleksnu Schurovu dekompoziciju matrice $C = QRQ^*$ te evaluiramo polinom p u točkama koje se nalaze na dijagonali matrice R. Dobivamo sljedeće rezultate:

- $\|Q^*Q\|_2 - 1 \approx 1.73 \cdot 10^{-14}$, $\|QQ^*\|_2 - 1 \approx 1.71 \cdot 10^{-14}$, tj. $\|Q^*Q\|_2 \approx 1$, $\|QQ^*\|_2 \approx 1$, čime smo pokazali da je matrica Q zaista unitarna;
- norma reziduala Schurove dekompozicije iznosi $\|C - QRQ^*\|_2 = 4.5274 \cdot 10^{-14}$;
- svojstvene vrijednosti matrice C, odnosno nultočke polinoma p iznose

$$r_1 = \begin{bmatrix} 1.1673 + 0.0000i \\ 0.1812 + 1.0840i \\ 0.1812 - 1.0840i \\ -0.7649 + 0.3525i \\ -0.7649 - 0.3525i \end{bmatrix};$$

- $\|r_2\|_\infty = 1.1362 \cdot 10^{-13}$, čime smo pokazali da su sve izračunate nultočke polinoma p točne barem u prvih 12 decimalnih mjesta.

Odmaknimo se na trenutak od polinoma te pogledajmo koliko dobro naša implementacija računa svojstvene vrijednosti jedne matrice iz prakse.

Primjer 3.1.3. Neka je zadana matrica $A \in \mathbb{R}^{48\times 48}$ iz *Building Modela* (podaci dostupni na [13]). Izračunajmo njene svojstvene vrijednosti.

Značenje ovih podataka objasnit ćemo u jednom kasnijem primjeru. Prvo ćemo pozvati funkciju `[R, Q] = mySchur(C,'complex')`, zatim ćemo dohvatiti dijagonalne vrijednosti matrice T i spremiti ih u vektor r_1 te ćemo naposljetku izračunati "prave" svojstvene vrijednosti matrice A pozivanjem MATLAB naredbe `r2 = eig(A)`. Elemente vektora r_1 i r_2 sortirat ćemo uzlazno. Dobivamo sljedeće rezultate:

- $\|Q^*Q\|_2 - 1 \approx 5.77 \cdot 10^{-14}$, $\|QQ^*\|_2 - 1 \approx 5.86 \cdot 10^{-14}$, tj. $\|Q^*Q\|_2 \approx 1$, $\|QQ^*\|_2 \approx 1$, čime smo pokazali da je matrica Q zaista unitarna;
- norma reziduala Schurove dekompozicije iznosi $\|A - QRQ^*\|_2 = 3.2691 \cdot 10^{-10}$;
- $\|r_1 - r_2\|_\infty = 4.1095 \cdot 10^{-11}$, čime smo pokazali da su sve izračunate svojstvene vrijednosti matrice A točne barem u prvih 10 decimalnih mjesta.

3.2 LTI sustavi i matrična eksponencijalna funkcija

Ovo potpoglavlje bit će obrađeno po [3, str. 152-166].

Jedan od osnovnih koncepata u teoriji upravljanja su tzv. LTI sustavi (eng. ***L**inear **T**ime **I**nvariant system*). Da bismo ih znali riješiti, prvo moramo uvesti pojam matrične eksponencijalne funkcije. Definirajmo ju i pokažimo neka njena svojstva.

Definicija 3.1 (Matrična eksponencijalna funkcija)
*Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$. **Matrična eksponencijalna funkcija** definirana je kao*

$$e^{At} := \sum_{k=0}^{\infty} \frac{(At)^k}{k!}.$$

Matrična eksponencijalna funkcija ima nekoliko zanimljivih svojstava koja se mogu naći u [3, str. 156-157], no, mi ćemo ovdje navesti samo dva.

Lema 3.2 (Svojstva matrične eksponencijalne funkcije)
Neka je zadana matrica $A \in \mathbb{C}^{n\times n}$ i regularna matrica $P \in \mathbb{C}^{n\times n}$. Tada vrijedi:

1. $\frac{d}{dt}\left(e^{At}\right) = Ae^{At} = e^{At}A.$
2. $e^{P^{-1}APt} = P^{-1}e^{At}P.$

Iako je glavna motivacija za uvođenje matrične eksponencijalne funkcije rješavanje linearnih sustava ODJ, ovdje ćemo se baviti rješavanjem LTI sustava koji nisu ništa drugo nego poopćenje linearnih sustava ODJ.

Teorem 3.3 (Rješenje LTI sustava)
Neka je dan LTI sustav

$$\dot{x}(t) = Ax(t) + Bu(t), \quad x_0 = x(0) \tag{3.1}$$
$$y(t) = Cx(t) + Du(t), \tag{3.2}$$

pri čemu je $t \geq 0$. Tada je rješenje sustava dano sa

$$x(t) = e^{At}x_0 + \int_0^t e^{A(t-s)}Bu(s)ds, \tag{3.3}$$
$$y(t) = Ce^{At}x_0 + \int_0^t Ce^{A(t-s)}Bu(s)ds + Du(t). \tag{3.4}$$

Dokaz. Radi jednostavnosti, razdvojit ćemo izraz (3.3) na vektore $x_1(t)$ i $x_2(t)$:

$$x_1(t) := e^{At}x_0, \quad x_2(t) := \int_0^t e^{A(t-s)}Bu(s)ds$$

Derivirajmo x_2 korištenjem Leibnizovog integralnog pravila (vidi [6]).

$$\dot{x}_2(t) = e^{A(t-t)}Bu(t) \cdot \frac{d}{dt}(t) - e^{A(t-0)}Bu(0) \cdot \frac{d}{dt}(0) + \int_0^t \frac{d}{dt}\left[e^{A(t-s)}Bu(s)\right] ds$$

Sređivanjem prethodnog izraza dobivamo

$$\dot{x}_2(t) = Bu(t) + A\int_0^t e^{A(t-s)}Bu(s)ds = Bu(t) + Ax_2(t).$$

Derivirajmo sada izraz (3.3).

$$\begin{aligned}
\dot{x}(t) &= \frac{d}{dt}\left[x_1(t) + x_2(t)\right] = \dot{x}_1(t) + \dot{x}_2(t) \\
&= \frac{d}{dt}\left(e^{At}x_0\right) + \left[Bu(t) + Ax_2(t)\right] \\
&= Ae^{At}x_0 + Bu(t) + Ax_2(t) \\
&= A\left[e^{At}x_0 + x_2(t)\right] + Bu(t) \\
&= Ax(t) + Bu(t)
\end{aligned}$$

Sada uvrštavanjem $t = 0$ u (3.3) dobivamo $x(0) = x_0$. Time smo dokazali ispravnost izraza (3.3), a njegovim direktnim uvrštavanjem u (3.2) dobit ćemo upravo izraz (3.4). □

Preostaje nam još samo smisliti kako efikasno izračunati e^A za neku proizvoljnu matricu A. Korištenje definicije 3.1 nam očigledno nije opcija jer će izraz presporo konvergirati, a i sama procedura je numerički vrlo nestabilna. Postoje razne metode za računanje o kojima se više može pročitati u [3, str. 158-166], no mi ćemo se ovdje baviti isključivo Schurovom metodom.

Prije nego što damo algoritam koji računa e^A, valja napomenuti kako on neće raditi ukoliko su dvije svojstvene vrijednosti matrice A jednake ili vrlo blizu jedna drugoj. Radi jednostavnosti, također ćemo se ograničiti na slučajeve kada je matrica A realna i sve su joj svojstvtene vrijednosti realne. U [10, str. 118] može se pronaći izraz za izračunavanje $f(R)$, gdje je R blok-gornje trokutasta matrica, a f je proizvoljna (skalarna) analitička funkcija za koju je $f(R)$ dobro definirana. Ako zapišemo taj izraz s oznakom $B = f(R) = [b_{ij}]$ te indeksima i, j, p, dobivamo

$$\begin{aligned}
r_{ii}b_{ij} - b_{ij}r_{jj} &= \sum_{p=0}^{j-i-1} \left(b_{i,i+p} \cdot r_{i+p,j} - r_{i,j-p} \cdot b_{j-p,j}\right) \\
&= \sum_{p=i}^{j-1} \left(b_{ip} \cdot r_{pj} - r_{i,i+j-p} \cdot b_{i+j-p,j}\right) \\
&= \left(b_{ii}r_{ij} - r_{ij}b_{jj}\right) + \sum_{p=i+1}^{j-1} \left(b_{ip} \cdot r_{pj} - r_{i,i+j-p} \cdot b_{i+j-p,j}\right) \\
&= r_{ij}\left(b_{ii} - b_{jj}\right) + \sum_{p=i+1}^{j-1} \left(b_{ip} \cdot r_{pj} - r_{ip} \cdot b_{pj}\right), \quad i < j.
\end{aligned}$$

Zbog pretpostavke da su sve svojstvene vrijednosti različite, možemo podijeliti cijeli izraz sa $(r_{ii} - r_{jj})$ te time dobivamo eksplicitnu formulu za b_{ij}, $i < j$.

$$b_{ij} = \frac{1}{r_{ii} - r_{jj}} \cdot \left[r_{ij}\left(b_{ii} - b_{jj}\right) + \sum_{p=i+1}^{j-1} \left(b_{ip}r_{pj} - r_{ip}b_{pj}\right)\right] \tag{3.5}$$

Ako malo bolje promotrimo ovu formulu, možemo zaključiti kako elementi na prvoj superdijagonali (tj. neposredno iznad glavne dijagonale) ovise isključivo o elementima na dijagonali, elementi na drugoj superdijagonali ovise isključivo o elementima na prvoj i na glavnoj dijagonali itd. Drugim riječima, najprije trebamo izračunati elemente na glavnoj dijagonali te zatim iteriramo po superdijagonalama $k = 1, \ldots, n-1$ i formulom (3.5) dobivamo njihove vrijednosti.

Preostaje nam još samo doći do formule za dijagonalne elemente matrice B. Može se pokazati (vidi [11, str. 6]) da za proizvoljnu blok-dijagonalnu matricu D i proizvoljnu analitičku funkciju f za koju je $f(D)$ dobro definirana vrijedi

$$f(D) = f\left(\operatorname{diag}(D_1, D_2, \ldots, D_n)\right) = \operatorname{diag}\left(f(D_1), f(D_2), \ldots, f(D_n)\right), \tag{3.6}$$

tj. blok-dijagonalne matrice komutiraju s analitičkim funkcijama ako su one dobro definirane za matricu na ulazu. U našem slučaju, funkcija f je $f(x) \equiv e^x$, a matrica D je dijagonalna. Korištenjem formula (3.5) i (3.6) dobivamo algoritam koji izračunava e^A, pri čemu je **`exp`** metoda koja računa e^x za ulaz $x \in \mathbb{R}$, a **`mySchur`** je metoda čiju se MATLAB implementaciju može naći u dodatku A pod algoritmom A.4.

Algoritam 3.1 (Schurov algoritam za e^A)
Ulaz: $A \in \mathbb{R}^{n\times n}$, *t.d. je* $\sigma(A) \subset \mathbb{R}$
Izlaz: $B = e^A$
1. $(R, Q) \leftarrow$ **`mySchur`**(A)
2. ***for*** $i = 1, \ldots, n$
3. $\quad b_{ii} \leftarrow$ **`exp`**(r_{ii})
4. ***end for***
5. ***for*** $k = 1, \ldots, n-1$
6. $\quad$ ***for*** $i = 1, \ldots, n-k$
7. $\quad\quad j \leftarrow i + k$
8. $\quad\quad b_{ij} \leftarrow 1/(r_{ii} - r_{jj}) \cdot \left[r_{ij}\left(b_{ii} - b_{jj}\right) + \sum_{p=i+1}^{j-1}\left(b_{ip}r_{pj} - r_{ip}b_{pj}\right)\right]$
9. $\quad$ ***end for***
10. ***end for***
11. $B \leftarrow QBQ^T$

Napomena 3.1. *Prethodni algoritam može se modificirati tako da radi i za matrice koje imaju kompleksne svojstvene vrijednosti, ali se u tom slučaju formula* (3.5) *mora modificirati tako da radi s blokovima dimenzija* 1×1, 1×2, 2×1 *i* 2×2, *a računanje vrijednosti na dijagonalnim blokovima postaje značajno kompliciranije. Računanje* e^A *pomoću kompleksne Schurove forme generalno nije dobra opcija jer se zbog uvođenja kompleksne aritmetike mogu dogoditi velike pogreške zaokruživanja.*

Napomena 3.2. *Algoritam 3.1 može se vrlo lako implementirati u MATLAB-u tako da ga praktički samo prepišemo.*

Pogledajmo kako algoritam 3.1 radi na nekoliko primjera. Kao i u prethodnom potpoglavlju, prvo ćemo se baviti jednim manjim primjerom kako bismo detaljnije proučili cijelu proceduru.

Primjer 3.2.1. Izračunajmo e^A, pri čemu je matrica $A \in \mathbb{R}^{3\times 3}$ nasumično generirana MATLAB naredbom `A = randi([1, 5], 3, 3)`, tj.

$$A = \begin{bmatrix} 2 & 4 & 1 \\ 4 & 1 & 1 \\ 1 & 2 & 5 \end{bmatrix}.$$

Prvi korak je računanje Schurove dekompozicije matrice A.

$$Q = \begin{bmatrix} 0.5150 & 0.5477 & 0.6594 \\ 0.4635 & 0.4691 & -0.7517 \\ 0.7210 & -0.6928 & 0.0122 \end{bmatrix}, \quad R = Q^T A Q = \begin{bmatrix} 7.0000 & 0.6594 & -0.5477 \\ -0.0000 & 3.5414 & 0.5150 \\ 0 & -0.0000 & -2.5414 \end{bmatrix}$$

Drugi korak je postavljanje dijagonalnih vrijednosti matrice B na $\exp(r_{ii})$.

$$b_{11} = 1096.6, \quad b_{22} = 34.515, \quad b_{33} = 0.0789$$

Treći korak je računanje vrijednosti iznad dijagonale pomoću *for* petlje.

- $k = 1,\ i = 1 \implies j = 2; \quad b_{12} = 202.5$
- $k = 1,\ i = 2 \implies j = 3; \quad b_{23} = 2.9157$
- $k = 2,\ i = 1 \implies j = 3; \quad b_{13} = -52.213$

Dakle, trenutna vrijednost matrice B je

$$B = \begin{bmatrix} 1096.6 & 202.5 & -52.213 \\ 0 & 34.515 & 2.9157 \\ 0 & 0 & 0.0789 \end{bmatrix}.$$

Četvrti i posljednji korak je "vraćanje" matrice B iz Schurove forme u originalnu formu, tj.

$$e^A = QBQ^T = \begin{bmatrix} 341.7093 & 338.5656 & 321.5820 \\ 306.9736 & 304.4560 & 289.9899 \\ 447.9503 & 453.6115 & 485.0612 \end{bmatrix}.$$

Spremimo sada dobiveni rezultat u matricu E_1 te izračunajmo "stvarnu" vrijednost e^A pozivanjem MATLAB naredbe `E2 = expm(A)`. Dobivamo sljedeće rezultate:

- $\|Q^*Q\|_2 - 1 \approx 3.55 \cdot 10^{-15}$, $\|QQ^*\|_2 - 1 \approx 2.89 \cdot 10^{-15}$, tj. $\|Q^*Q\|_2 \approx 1$, $\|QQ^*\|_2 \approx 1$, čime smo pokazali da je matrica Q zaista unitarna;
- norma reziduala Schurove dekompozicije iznosi $\|A - QRQ^*\|_2 = 2.6036 \cdot 10^{-14}$;
- apsolutne pogreške iznose

$$\|E_1 - E_2\|_2 = 1.2123 \cdot 10^{-11}, \quad \|E_1 - E_2\|_\infty = 1.2903 \cdot 10^{-11};$$

- relativne pogreške iznose

$$\frac{\|E_1 - E_2\|_2}{\|E_1\|_2} = 1.0859 \cdot 10^{-14}, \quad \frac{\|E_1 - E_2\|_\infty}{\|E_1\|_\infty} = 9.3057 \cdot 10^{-15}.$$

Sad kada smo vidjeli kako cijela ova procedura radi, možemo ju isprobati na jednom malo većem primjeru.

Primjer 3.2.2. Izračunajmo e^A, pri čemu je $A \in \mathbb{R}^{10\times 10}$ simetrična cjelobrojna matrica s elementima $a_{ij} \in [-5, 5] \cap \mathbb{Z}$ nasumično generirana u MATLAB-u. Preciznije,

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

Razlog zašto smo zahtijevali simetričnu matricu je zato što se lako može pokazati da simetrične realne matrice imaju isključivo realne svojstvene vrijednosti. Dodatno, generirali smo matricu koja ima male svojstvene vrijednosti, a malo kasnije ćemo objasniti i zašto. Pozivanjem MATLAB naredbe `eig(A)` možemo se uvjeriti da su joj sve svojstvene vrijednosti različite i nalaze se u intervalu $[-16, 15]$. Ponavljamo proceduru iz prethodnog primjera s istim oznakama te time dobivamo sljedeće rezultate:

- $\|Q^*Q\|_2 - 1 \approx 3.69 \cdot 10^{-14}$, $\|QQ^*\|_2 - 1 \approx 3.51 \cdot 10^{-14}$, tj. $\|Q^*Q\|_2 \approx 1$, $\|QQ^*\|_2 \approx 1$, čime smo pokazali da je matrica Q zaista unitarna;
- norma reziduala Schurove dekompozicije iznosi $\|A - QRQ^*\|_2 = 1.1412 \cdot 10^{-12}$;
- apsolutne pogreške iznose

$$\|E_1 - E_2\|_2 = 1.6407 \cdot 10^{-6}, \quad \|E_1 - E_2\|_\infty = 2.9034 \cdot 10^{-6};$$

- relativne pogreške iznose

$$\frac{\|E_1 - E_2\|_2}{\|E_1\|_2} = 5.6877 \cdot 10^{-13}, \quad \frac{\|E_1 - E_2\|_\infty}{\|E_1\|_\infty} = 5.6980 \cdot 10^{-13}.$$

Primijetimo da su apsolutne pogreške $\|E_1 - E_2\|_2$ i $\|E_1 - E_2\|_\infty$ daleko veće nego u prethodnom primjeru, što je i za očekivati zbog velikih dimenzija matrice A. Ove pogreške još više rastu ako je matrica na ulazu još većih dimenzija ili ako su joj svojstvene vrijednosti jako velike jer će u tom slučaju dijagonalni elementi matrice e^A *eksplodirati,* a to će zatim utjecati i na njene preostale elemente.

Velike dimenzije nam zapravo u praksi ne stvaraju toliki problem jer su velike matrice najčešće vrlo *prazne* (eng. *sparse*), tj. sadrže jako puno nul-elemenata. S druge strane, matrice s jako velikim pozitivnim svojstvenim vrijednostima λ_k i dalje mogu prouzročiti probleme pri računanju e^{λ_k}. No, u praksi nam pozitivne svojstvene vrijednosti daju do znanja da je LTI sustav s kojim radimo nestabilan te se takvi sustavi najčešće prvo moraju stabilizirati kako bismo mogli dalje raditi s njima (vidi [8, str. 63-70, 123-131] za detalje).

3.3 Rješavanje matričnih jednadžbi

Ovo potpoglavlje bit će obrađeno po [3, str. 291-293, 309-318].

Rješavanje matričnih jednadžbi kao što su Sylvesterova i Ljapunovljeva je problem koji ima vrlo široke primjene u teoriji upravljanja i redukciji modela. Ovdje se nećemo previše fokusirati niti na jednu od tih primjena, nego ćemo samo izgraditi jedan numerički algoritam za njihovo rješavanje pod nazivom ***Bartels-Stewart algoritam*** te ćemo spomenuti jednu njegovu primjenu vezanu uz teoriju upravljanja. Ovaj se algoritam zapravo jako često koristi u praksi zbog svoje numeričke stabilnosti te male vremenske složenosti $O(n^3)$, a izračunavanje Schurove dekompozicije jedan je od najvažnijih koraka u tom algoritmu. Prije nego što krenemo u izgradnju Bartels-Stewart algoritma, potrebno je navesti uvjete pod kojima Sylvesterova i Ljapunovljeva jednadžba uopće imaju rješenje.

Teorem 3.4 (Egzistencija rješenja Sylvesterove jednadžbe)
Neka su zadane matrice $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{m\times m}$ i $C \in \mathbb{R}^{m\times n}$. Tada Sylvesterova jednadžba

$$XA + BX = C \tag{3.7}$$

ima jedinstveno rješenje $X \in \mathbb{R}^{m\times n}$ ako i samo ako vrijedi $\sigma(A) \cap \sigma(-B) = \varnothing$.

Dokaz. Vidi [3, str. 292-293]. □

Korolar 3.5 (Egzistencija rješenja Ljapunovljeve jednadžbe)
Neka su zadane matrice $A, C \in \mathbb{R}^{n\times n}$. Tada Ljapunovljeva jednadžba

$$XA + A^T X = C \tag{3.8}$$

ima jedinstveno rješenje $X \in \mathbb{R}^{n\times n}$ ako i samo ako vrijedi $\lambda \in \sigma(A) \Longrightarrow -\lambda \notin \sigma(A)$.

Dokaz. Lako se pokaže da vrijedi $\sigma(A^T) = \sigma(A)$ i $\sigma(-A) = \{-\lambda_i : \lambda_i \in \sigma(A)\}$. Traženu tvrdnju dobivamo primjenom teorema 3.4 na jednadžbu (3.8). □

U ovome radu bavit ćemo se samo jednostavnijim slučajem, tj. Ljapunovljevom jednadžbom. Uz nekoliko jednostavnih modifikacija možemo dobiti algoritam koji rješava Sylvesterove jednadžbe (vidi [3, str. 313-315]).

Prvi korak je izračunavanje realne Schurove dekompozicije matrice A^T. Neka je ona dana sa $R = U^T A^T U$. Nadalje, vrijedi

$$\begin{aligned} XA + A^T X = C &\iff X(UR^TU^T) + (URU^T)X = C \\ &\iff U^TX(UR^TU^T)U + U^T(URU^T)XU = U^TCU \\ &\iff (U^TXU)R^T(U^TU) + (U^TU)R(U^TXU) = U^TCU \\ &\iff YR^T + RY = \hat{C}, \quad Y := U^TXU,\ \hat{C} := U^TCU. \end{aligned} \tag{3.9}$$

Jednadžbu (3.8) nadalje ćemo zvati početni problem, a jednadžbu (3.9) zvat ćemo reducirani problem.

Drugi korak rješavanja Ljapunovljeve jednadžbe je rješavanje reduciranog problema. Ideja ovog koraka slična je algoritmu povratnih supstitucija za sustave linearnih jednadžbi (vidi [12, str. 64]). Pri definiranju ovog postupka koristit ćemo sljedeću notaciju:

$$Y = \begin{bmatrix} y_1, & y_2, & \dots, & y_n \end{bmatrix}, \quad \hat{C} = \begin{bmatrix} c_1, & c_2, & \dots, & c_n \end{bmatrix}, \quad R = [r_{ij}]. \tag{3.10}$$

Postupak se svodi na n iteracija, pri čemu u svakoj iteraciji računamo y_k, s time što krećemo od n-tog stupca i idemo prema prvom. Pretpostavimo da smo izračunali stupce y_{k+1} do y_n i želimo izračunati stupac y_k. Budući da nismo uvodili nikakvu pretpostavku na spektar matrice A^T, postoji mogućnost da njena realna Schurova forma sadrži *bumpove*. Zbog toga u ovom koraku algoritma razlikujemo dva slučaja:

1. $r_{k,k-1} = 0$, tj. k-ti stupac od R ne sadrži *bump*. U ovom slučaju trebamo jednostavno riješiti sustav linearnih jednadžbi dimenzije $n \times n$, a to možemo izvesti korištenjem sljedeće formule:
$$(R + r_{kk}I)\, y_k = c_k - \sum_{j=k+1}^{n} (r_{kj} \cdot y_j)\,. \tag{3.11}$$
Naime, ako sa $(\,\cdot\,)_k$ označimo operaciju dohvaćanja k-tog stupca ulazne matrice, onda zbog (3.9) i (3.10) imamo
$$\begin{aligned} \left(YR^T + RY\right)_k = \left(\hat{C}\right)_k \quad &\Longleftrightarrow \quad \left(YR^T\right)_k + R \cdot y_k = c_k \\ &\Longleftrightarrow \quad \sum_{j=1}^{n} \left[\left(R^T\right)_{jk} \cdot y_j\right] + R \cdot y_k = c_k \\ &\Longleftrightarrow \quad \sum_{j=1}^{n} (r_{kj} \cdot y_j) + R \cdot y_k = c_k, \end{aligned}$$
a budući da je R gornje Hessenbergova i vrijedi $r_{k,k-1} = 0$, to znači da za sve $j < k$ vrijedi $r_{kj} = 0$. Odatle slijedi
$$\begin{aligned} \sum_{j=1}^{n} (r_{kj} \cdot y_j) + R \cdot y_k = c_k \quad &\Longleftrightarrow \quad \sum_{j=k}^{n} (r_{kj} \cdot y_j) + R \cdot y_k = c_k \\ &\Longleftrightarrow \quad \sum_{j=k+1}^{n} (r_{kj} \cdot y_j) + R \cdot y_k + r_{kk} \cdot y_k = c_k \\ &\Longleftrightarrow \quad (R + r_{kk}I)\, y_k = c_k - \sum_{j=k+1}^{n} (r_{kj} \cdot y_j)\,. \end{aligned}$$

2. $r_{k,k-1} \neq 0$, tj. k-ti stupac od R sadrži *bump*. Ovaj slučaj je malo kompliciraniji jer moramo riješiti sustav dimenzije $2n \times 2n$, ali ćemo time istovremeno izračunati stupce y_k i y_{k-1}. Za ovaj slučaj uvodimo $i = k - 1$ te koristimo sljedeću formulu:
$$R \begin{bmatrix} y_i, & y_k \end{bmatrix} + \begin{bmatrix} y_i, & y_k \end{bmatrix} \begin{bmatrix} r_{ii} & r_{ki} \\ r_{ik} & r_{kk} \end{bmatrix} = \begin{bmatrix} c_i, & c_k \end{bmatrix} - \sum_{j=k+1}^{n} \begin{bmatrix} r_{ij} \cdot y_j, & r_{kj} \cdot y_j \end{bmatrix}.$$
Prethodna formula izvede se na sličan način kao i formula (3.11), a još se može zapisati i na sljedeći način:
$$\begin{bmatrix} R + r_{ii} \cdot I_n & r_{ik} \cdot I_n \\ r_{ki} \cdot I_n & R + r_{kk} \cdot I_n \end{bmatrix} \begin{bmatrix} y_i \\ y_k \end{bmatrix} = \begin{bmatrix} c_i \\ c_k \end{bmatrix} - \sum_{j=k+1}^{n} \begin{bmatrix} r_{ij} \cdot y_j \\ r_{kj} \cdot y_j \end{bmatrix}. \tag{3.12}$$
Očito je da se ovaj slučaj ne može dogoditi za $k = 1$ te da u sljedećoj iteraciji umjesto $k \leftarrow k - 1$ postavljamo $k \leftarrow k - 2$.

Treći korak je jednostavan: izračunamo $X = UYU^T$ te vratimo X kao rješenje početnog problema. Cjelokupni algoritam ima vremensku složenost $O(n^3)$, pri čemu prvi korak zahtijeva daleko najviše vremena. Detaljniju analizu može se pronaći u [3, str. 313], a MATLAB kod algoritma može se pronaći u dodatku A pod algoritmom A.5.

Pogledajmo kako ovaj algoritam radi na nekoliko primjera. Kao i dosad, prvo ćemo testirati metodu na jednom manjem primjeru kako bismo detaljnije proučili njen rad. Sljedeći primjer preuzet je iz [3, str. 311]).

Primjer 3.3.1. Riješimo Ljapunovljevu jednadžbu $XA + A^TX = C$, gdje je

$$A = \begin{bmatrix} 0 & 2 & -1 \\ -3 & -2 & 2 \\ -2 & 1 & -1 \end{bmatrix}, \quad C = \begin{bmatrix} -2 & 2 & -3 \\ -8 & -6 & -5 \\ 11 & 13 & -2 \end{bmatrix}.$$

Prvi korak je svođenje početnog problema na reducirani problem. Korištenjem MATLAB funkcije `[R, U] = mySchur(A', 'real')` dobivamo sljedeće matrice:

$$R = \begin{bmatrix} -2.5160 & 1.6449 & 2.7174 \\ -0.0000 & -0.2526 & -0.8298 \\ 0 & 3.2824 & -0.2314 \end{bmatrix}, \quad U = \begin{bmatrix} 0.1972 & -0.0747 & 0.9775 \\ 0.6529 & -0.7338 & -0.1878 \\ -0.7313 & -0.6753 & 0.0959 \end{bmatrix},$$

a transformacijom $\hat{C} = U^TCU$ dobivamo

$$\hat{C} = \begin{bmatrix} -9.4514 & 12.1985 & -11.1371 \\ -7.0672 & -0.1150 & -0.0239 \\ 4.5432 & -1.7457 & -0.4336 \end{bmatrix}.$$

Provjerimo točnost dobivenih rezultata:

- $\|U^TU\|_2 \approx 1,\ \|UU^T\|_2 \approx 1;$
- $\|A^T - URU^T\|_2 = 2.8596 \cdot 10^{-14};$
- $\|C - U\hat{C}U^T\|_2 = 1.3099 \cdot 10^{-13}.$

Drugi korak je rješavanje reduciranog problema. U prvoj iteraciji imamo $k = 3$, što znači da računamo stupac y_3. Budući da na poziciji r_{23} imamo *bump,* moramo istovremeno računati i stupac y_2. Korištenjem formule (3.12) dobivamo da vrijedi

$$y_2 = \begin{bmatrix} -2.8696 \\ 0.1058 \\ 1.8157 \end{bmatrix}, \quad y_3 = \begin{bmatrix} 1.0295 \\ -1.7415 \\ 1.4629 \end{bmatrix}.$$

Budući da smo imali *bump,* u sljedećoj iteraciji postavljamo $k = 1$, što znači da ne možemo imati *bump* te zbog toga automatski koristimo formulu (3.11). Dobivamo da vrijedi

$$y_1 = \begin{bmatrix} 2.4313 \\ 0.4729 \\ 1.4454 \end{bmatrix}.$$

Došli smo do kraja drugog koraka te nam preostaje samo transformirati rješenje reduciranog problema u rješenje početnog problema.

$$X = UYU^T = \begin{bmatrix} 2.0000 & 0.0000 & -2.0000 \\ 2.0000 & 2.0000 & 1.0000 \\ -0.0000 & -3.0000 & 0.0000 \end{bmatrix}$$

Spremimo sada dobiveno rješenje u matricu X_1 te izračunajmo "pravo" rješenje pozovanjem MATLAB funkcije `X2 = lyap(A', -C)`. Usporedimo norme reziduala dobivenih rješenja:

- $\|X_1A + A^TX_1 - C\|_2 = 6.9097 \cdot 10^{-14}$, $\|X_1A + A^TX_1 - C\|_\infty = 8.4821 \cdot 10^{-14}$;
- $\|X_2A + A^TX_2 - C\|_2 = 1.4312 \cdot 10^{-14}$, $\|X_2A + A^TX_2 - C\|_\infty = 1.5099 \cdot 10^{-14}$.

Zaključujemo da algoritam A.5 radi malo lošije nego MATLAB-ov *state-of-the-art* algoritam, ali i dalje radi prilično dobro.

Prije nego što se krenemo baviti nekim stvarnim primjerima, moramo definirati još dva važna pojma iz teorije upravljanja.

Definicija 3.2 (Gramijani upravljivosti i osmotrivosti)
Neka je dan LTI sustav

$$\dot{x}(t) = Ax(t) + Bu(t), \quad x_0 = x(0)$$
$$y(t) = Cx(t) + Du(t),$$

pri čemu je $t \geq 0$. *Matricu* C_G *koja je rješenje Ljapunovljeve jednadžbe*

$$AC_G + C_GA^T = -BB^T$$

zovemo ***gramijan upravljivosti,*** *a matricu* O_G *koja je rješenje Ljapunovljeve jednadžbe*

$$O_GA + A^TO_G = -C^TC$$

zovemo ***gramijan osmotrivosti.***

Ovdje nećemo ulaziti u detalje vezane uz teoriju upravljanja, nego ćemo samo napomenuti da su gramijani upravljivosti i osmotrivosti vrlo važne matrice svakog LTI sustava, ukoliko one uopće postoje. Više detalja može se pronaći u npr. [8, str. 95-117, 135-147].

Provjerimo sada koliko dobro algoritam A.5 radi na nekim većim, stvarnim primjerima.

Primjer 3.3.2. Neka je zadan LTI sustav *Building Model* (podaci dostupni na [13]). Ovaj sustav opisuje pomicanje višekatne zgrade tijekom potresa, a dobiven je svođenjem sustava ODJ drugoga reda

$$\ddot{x}(t) + E\dot{x}(t) + Kx(t) = Bu(t), \quad E, K \in \mathbb{R}^{24\times 24}, \quad B \in \mathbb{R}^{24\times 1}$$
$$y(t) = C_v\dot{x}(t), \quad C_v \in \mathbb{R}^{1\times 24}$$

na sustav ODJ prvoga reda, tj. na LTI sustav

$$\dot{x}(t) = Ax(t) + Bu(t)$$
$$y(t) = Cx(t),$$

pri čemu su matrice ovog sustava $A \in \mathbb{R}^{48\times 48}$, $B \in \mathbb{R}^{48\times 1}$, $C \in \mathbb{R}^{1\times 48}$. Izračunajmo gramijane upravljivosti i osmotrivosti ovog LTI sustava. Prvo ćemo izračunati gramijan upravljivosti. Rješenje koje dobijemo korištenjem algoritma A.5 označit ćemo sa X_1. Dobivamo sljedeće norme reziduala:

- $\|AX_1 + X_1A^T + BB^T\|_2 = 9.1101 \cdot 10^{-17}$;
- $\|AX_1 + X_1A^T + BB^T\|_\infty = 3.0138 \cdot 10^{-16}$.

Za usporedbu, pozivanjem MATLAB naredbe `X2 = lyap(A, B*B')` dobivamo "točno" rješenje koje ima sljedeće norme reziduala:

- $\|AX_2 + X_2A^T + BB^T\|_2 = 7.9078 \cdot 10^{-18}$;
- $\|AX_2 + X_2A^T + BB^T\|_\infty = 2.2178 \cdot 10^{-17}$.

Ponovimo isti postupak za gramijan osmotrivosti. Rješenje koje dobijemo korištenjem algoritma A.5 označit ćemo sa Y_1. Dobivamo sljedeće norme reziduala:

- $\|Y_1A + A^TY_1 + C^TC\|_2 = 1.3690 \cdot 10^{-10}$;
- $\|Y_1A + A^TY_1 + C^TC\|_\infty = 4.0915 \cdot 10^{-10}$.

Za usporedbu, pozivanjem MATLAB naredbe `Y2 = lyap(A', C'*C)` dobivamo "točno" rješenje koje ima sljedeće norme reziduala:

- $\|Y_2A + A^TY_2 + C^TC\|_2 = 6.7896 \cdot 10^{-12}$;
- $\|Y_2A + A^TY_2 + C^TC\|_\infty = 1.8357 \cdot 10^{-11}$.

U usporedbi s MATLAB-ovim algoritmom `lyap`, algoritam A.5 zapravo radi prilično dobro na ovom primjeru.

Primjer 3.3.3. Neka je zadan LTI sustav *CD Player* (podaci dostupni na [14]). Ovaj sustav modelira zakretnu ruku CD playera koja drži optičku leću koja se može pomicati po horizontalnoj ravnini. Sustav je zadan sa

$$\begin{aligned} \dot{x}(t) &= Ax(t) + Bu(t) \\ y(t) &= Cx(t), \end{aligned}$$

pri čemu su njegove matrice $A \in \mathbb{R}^{120\times120}$, $B \in \mathbb{R}^{120\times2}$, $C \in \mathbb{R}^{2\times120}$. Izračunajmo gramijane upravljivosti i osmotrivosti.

U ovom primjeru koristit ćemo tri različita algoritma za izračunavanje gramijana. Matrice X_1, X_2 i X_3 odgovarat će gramijanima upravljivosti, a matrice Y_1, Y_2 i Y_3 odgovarat će gramijanima osmotrivosti.

- Rješenja X_1 i Y_1 izračunat ćemo korištenjem algoritma A.5, pri čemu unutar funkcije **`hess2schur`** nećemo koristiti *decoupling.*
- Rješenja X_2 i Y_2 izračunat ćemo korištenjem algoritma A.5 s varijantom funkcije **`hess2schur`** koja koristi *decoupling,* tj. koristit ćemo baš funkciju A.3.
- Rješenja X_3 i Y_3 izračunat ćemo korištenjem MATLAB funkcije **`lyap`.**

Zbog velikih dimenzija zadanog LTI sustava, korištenje *decoupling*-a imat će značajan utjecaj na brzinu izvođenja algoritma, kao i na opterećenje procesora. Preciznije, izračunavanjem matrica X_i, $i = 1, 2, 3$, te definiranjem matrica $R_i = AX_i + X_iA^T + BB^T$, $i = 1, 2, 3$, dobivamo rezultate prikazane u sljedećoj tablici.

algoritam	$\|R_i\|_2 \,/\, \|X_i\|_2$	$\|R_i\|_\infty \,/\, \|X_i\|_\infty$	vrijeme izvršavanja
bez *decoupling*-a	$1.4429 \cdot 10^{-11}$	$4.5741 \cdot 10^{-11}$	173.31 sekundi
sa *decoupling*-om	$9.1760 \cdot 10^{-12}$	$3.4796 \cdot 10^{-11}$	27.22 sekundi
`lyap`	$9.1333 \cdot 10^{-12}$	$3.4673 \cdot 10^{-11}$	0.094 sekundi

Tablica 1: Usporedba algoritama za računanje gramijana upravljivosti.

Iz prethodne tablice možemo zaključiti kako *decoupling* za ovaj primjer smanjuje vrijeme izvršavanja algoritma za gotovo 85%, a osim toga daje i malo točnije rješenje nego kada ga ne koristimo. U usporedbi s MATLAB funkcijom `lyap`, algoritam sa *decoupling*-om je puno sporiji, ali daje gotovo jednako dobro rješenje. Nadalje, izračunavanjem matrica Y_i, $i = 1, 2, 3$, te definiranjem matrica $\tilde{R}_i = Y_i A + A^T Y_i + C^T C$, $i = 1, 2, 3$, dobivamo rezultate prikazane u sljedećoj tablici.

algoritam	$\|\tilde{R}_i\|_2 \,/\, \|Y_i\|_2$	$\|\tilde{R}_i\|_\infty \,/\, \|Y_i\|_\infty$	vrijeme izvršavanja
bez *decoupling*-a	$1.8092 \cdot 10^{-11}$	$5.3087 \cdot 10^{-11}$	176.19 sekundi
sa *decoupling*-om	$1.0751 \cdot 10^{-11}$	$3.6594 \cdot 10^{-11}$	27.33 sekundi
`lyap`	$1.0753 \cdot 10^{-11}$	$3.6598 \cdot 10^{-11}$	0.009 sekundi

Tablica 2: Usporedba algoritama za računanje gramijana osmotrivosti.

Iz prethodne tablice možemo zaključiti kako *decoupling* i u ovom slučaju smanjuje vrijeme izvršavanja algoritma za gotovo 85%, a osim toga daje i točnije rješenje nego u slučaju kada ga ne koristimo. Varijanta algoritma sa *decoupling*-om je i dalje puno sporija od MATLAB funkcije `lyap`, ali u slučaju gramijana osmotrivosti daje čak i malo bolje rješenje.

A Dodatak: popis MATLAB kodova

Algoritam A.1 (Givensova rotacija)
Algoritam je preuzet iz [7, str. 216], a služi za računanje Givensove rotacijske matrice.

```
function [c, s] = givens(a, b)
  if b == 0
    c = 1; s = 0;
  else
    if abs(b) > abs(a)
      tau = -a / b;
      s = 1 / sqrt(1 + abs(tau)^2);
      c = s * tau;
    else
      tau = -b / a;
      c = 1 / sqrt(1 + abs(tau)^2);
      s = c * tau;
    end
  end
end
```

Algoritam A.2 (Hessenberg-QR step)
Algoritam je većinski preuzet iz [7, str. 343], a vraća unitarnu matricu Q koja poništava subdijagonalne elemente matrice H te matricu R za koju vrijedi vrijedi $R = Q^*HQ$. Modifikacija za kompleksne matrice preuzeta je iz [5].

```
function [R, Q] = hessQRstep(H)
  n = length(H);
  c = zeros(n,1);
  s = zeros(n,1);
  % if Q is required
  if nargout > 1
    Q = eye(n);
  end
  for k=1:n-1
    [c(k), s(k)] = givens(H(k,k), H(k+1,k));
    G_k = [c(k), s(k); -s(k)', c(k)'];
    H(k:k+1, k:n) = G_k' * H(k:k+1, k:n);
  end
  for k=1:n-1
    G_k = [c(k), s(k); -s(k)', c(k)'];
    H(1:k+1, k:k+1) = H(1:k+1, k:k+1) * G_k;
    % if Q is required
    if nargout > 1
      Q(1:k+1, k:k+1) = Q(1:k+1, k:k+1) * G_k;
    end
  end
  R = H;
end
```

Algoritam A.3 (Hessenberg-to-Schur)
Ovaj MATLAB kod predstavlja glavni dio algoritma 2.3, a služi za računanje Schurove dekompozicije gornje Hessenbergove matrice. Dodatno, implementirana je strategija *decoupling* kako bi se ubrzalo izvođenje algoritma te smanjilo opterećenje procesora. Kompleksnu Schurovu dekompoziciju dobivamo iz realne tako što za svaki *bump* izračunamo jednu njegovu svojstvenu vrijednost λ korištenjem formule (1.2) te zatim napravimo jednostruki *shift* λ.

```
function [R, Q] = hess2schur(H, flag)
  input_real = isreal(H);
  tol = 1e-14;
  n = length(H);
  if nargin < 2 || ~input_real
    flag = 'real';
  end
  R = H;
  % if Q is required
  if nargout > 1
    Q = eye(n);
  end
  zf = zeros(n-1, 1); %zero flags
  cf = zeros(n-1, 1); %complex flags
  converge = false;
  while ~converge
    for j=1:n-1
      % check if R(j+1, j) is zero
      if abs(R(j+1, j)) < tol
        zf(j) = 1;
        cf(j) = 0;
      else
        zf(j) = 0;
        if input_real
          % possible bump
          B = R(j:j+1, j:j+1);
          tp = (B(1,1) + B(2,2))/2;
          d = B(1,1)*B(2,2) - B(1,2)*B(2,1);
          if (tp^2 - d) < 0
            if j > 1
              cf(j-1) = 0;
            end
            cf(j) = 1;
            if j < n-1
              cf(j+1) = 0;
            end
          else
            cf(j) = 0;
          end
        end
      end
    end
```

```
    converge = all(or(zf, cf));
    if ~converge
      % find next non-bump block
      e = 0;
      while ~isempty(e)
        s = find(zf(e+1:n-1)==0, 1, 'first') + e;
        if isempty(s)
          e = [];
        else
          e = find(zf(s+1:n-1), 1, 'first') + s;
          if isempty(e)
            e = n;
          end
          if ~((e - s < 2) && (cf(s) == 1))
            [R(s:e, s:e), Q_j] = hessQRstep(R(s:e, s:e));
            R(s:e, e+1:n) = Q_j' * R(s:e, e+1:n);
            R(1:s-1, s:e) = R(1:s-1, s:e) * Q_j;
            if nargout > 1
              Q(:, s:e) = Q(:, s:e) * Q_j;
            end
          end
        end
      end
    end
  end
  if ~all(zf) && input_real && strcmp(flag, 'complex')
    % check all bumps
    for j=1:n-1
      if cf(j) == 1
        B = R(j:j+1, j:j+1);
        tp = (B(1,1) + B(2,2))/2;
        d = B(1,1)*B(2,2) - B(1,2)*B(2,1);
        shift = tp + sqrt(tp^2 - d);
        if abs(B(1,1)) < abs(B(2,1))
          % keep eigenvalue with +bi above one with -bi
          shift = shift';
        end
        [B, Q_j] = hessQRstep(B - shift*eye(2));
        R(j:j+1, j:j+1) = B + shift*eye(2);
        R(j:j+1, j+2:n) = Q_j' * R(j:j+1, j+2:n);
        R(1:j-1, j:j+1) = R(1:j-1, j:j+1) * Q_j;
        if nargout > 1
          Q(:, j:j+1) = Q(:, j:j+1) * Q_j;
        end
      end
    end
  end
end
```

Algoritam A.4 (My Schur)
Ovaj MATLAB kod služi kao glavna funkcija za QR algoritam jer prvo pretvara matricu na ulazu u gornje Hessenbergovu, a zatim nju prosljeđuje funkciji `hess2schur`.

```
function [T, U] = mySchur(A, flag)
  if nargin < 2
    flag = 'real';
  end
  if nargout > 1
    [U_0, T] = hess(A);
    [T, U] = hess2schur(T, flag);
    U = U_0 * U;
  else
    T = hess(A);
    T = hess2schur(T, flag);
  end
end
```

Algoritam A.5 (Bartels-Stewart za Ljapunovljevu jednadžbu)
Ovaj MATLAB kod služi kao implementacija algoritma opisanog u potpoglavlju 3.3.

```
function X = bartelsStewartLyap(A, C, tol)
  n = length(A);
  I = eye(n);
  opts.UHESS = true;
  if nargin < 3
    tol = 1e-14;
  end

  % 1. Schurova dekompozicija
  [R, U] = mySchur(A', 'real');
  C_hat = U' * C * U;
  Y = zeros(n,n);

  % 2. Rjesavanje reduciranog problema
  k = n;
  while k > 0
    bump = false;
    if k > 1
      if abs(R(k, k-1)) >= tol
        bump = true;
      end
    end
    if ~bump
      A_ = R + R(k,k) * I;
      b = C_hat(:,k);
      for j=k+1:n
        b = b - R(k,j) * Y(:,j);
      end
      Y(:,k) = linsolve(A_, b, opts);
```

```
30      else
31        A_ = [R + R(k-1,k-1)*I, R(k-1,k)*I;
32              R(k,k-1)*I, R + R(k,k)*I];
33        b = [C_hat(:,k-1); C_hat(:,k)];
34        for j=k+1:n
35          b(1:n) = b(1:n) - R(k-1,j)*Y(:,j);
36          b(n+1:2*n) = b(n+1:2*n) - R(k,j)*Y(:,j);
37        end
38        sol = linsolve(A_, b);
39        Y(:,k-1) = sol(1:n);
40        Y(:,k) = sol(n+1:2*n);
41        k = k - 1;
42      end
43      k = k - 1;
44    end
45
46    % 3. Povratak na pocetni problem
47    X = U * Y * U';
48  end
```

Literatura

[1] V. B. Alekseev, *Abel's Theorem in Problems and Solutions*, Moscow State University, 2004.

[2] D. Bakić, *Linearna algebra*, Školska knjiga, Zagreb, 2008.

[3] B. N. Datta, *Numerical Methods for Linear Control Systems*, Northern Illinois University, 2003.

[4] J. W. Demmel, *Applied Numerical Linear Algebra*, University of California, 1997.

[5] W. Fan, A. Alimohammad, *Givens rotation-based QR decomposition for MIMO systems*, IET Communications, vol. 11, 2017.

[6] H. Flanders, *Differentiation Under the Integral Sign*, The American Mathematical Monthly, vol. 80, no. 6, str. 615-627, Mathematical Association of America, 1973.

[7] G. H. Golub, C. F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1996.

[8] J. Hespanha, *Linear Systems Theory*, Princeton University Press, 2009.

[9] R. A. Horn, C. R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.

[10] B. N. Parlett, *A recurrence among the elements of functions of triangular matrices*, Linear Algebra and its Applications, vol. 29, str. 323-346, 1976.

[11] B. N. Parlett, *Computation of Functions of Triangular Matrices*, EECS Department, University of California, Berkeley, 1974.

[12] N. Truhar, *Numerička linearna algebra*, Sveučilište J. J. Strossmayera u Osijeku, Odjel za matematiku, Osijek, 2010.

[13] Building Model, MOR wiki. Dostupno na: `https://morwiki.mpi-magdeburg.mpg.de/morwiki/index.php/Building_Model` (13. rujan 2023.)

[14] CD Player, MOR wiki. Dostupno na: `https://morwiki.mpi-magdeburg.mpg.de/morwiki/index.php/CD_Player` (14. rujan 2023.)

Sažetak

U ovome radu bavit ćemo se Schurovom dekompozicijom kvadratnih matrica te njenim najvažnijim praktičnim primjenama. U prvom dijelu rada navest ćemo neke osnovne pojmove i tvrdnje iz područja linearne algebre koje će nam biti važne za razumijevanje ostatka rada. U drugom dijelu pokazat ćemo egzistenciju Schurove dekompozicije za proizvoljnu kvadratnu matricu, kao i egzistenciju realne Schurove dekompozicije za proizvoljnu realnu kvadratnu matricu. Nakon toga, izgradit ćemo numerički stabilan i učinkovit algoritam za računanje Schurove dekompozicije pod nazivom *QR algoritam* te ćemo navesti nekoliko ideja za njegovo ubrzanje. U trećem i posljednjem dijelu rada pokazat ćemo neke fundamentalne primjene Schurove dekompozicije, kao što su računanje svojstvenih vrijednosti zadane matrice, računanje nultočaka polinoma proizvoljnog stupnja, računanje matrične eksponencijalne funkcije te rješavanje Ljapunovljevih i Sylvesterovih matričnih jednadžbi.

Ključne riječi

Schurova dekompozicija, QR algoritam, svojstveni problem, nultočke polinoma, matrična eksponencijalna funkcija, Ljapunovljeva jednadžba, Sylvesterova jednadžba, Bartels-Stewart algoritam

Schur decomposition and its applications

Abstract

In this paper, we will observe Schur decomposition of square matrices, as well as its most important practical applications. In the first chapter of this paper, we will list some basic terms and statements from the field of linear algebra that will be important during the remaining chapters. In the second chapter, we will show the existence of Schur decomposition for any given square matrix, as well as the existence of real Schur decomposition for any given real square matrix. After that, we will introduce a numerically stable and efficient algorithm used for computing the Schur decomposition called *The QR algorithm* and we will also list a few ideas we can use to make the algorithm run faster. In the third and the final chapter of this paper, we will show some fundamental applications of Schur decomposition, such as computing the eigenvalues of a given matrix, computing the roots of a polynomial of any degree, computing the matrix exponential function, and solving Lyapunov and Sylvester matrix equations.

Keywords

Schur decomposition, QR algorithm, eigenvalue problem, polynomial roots, matrix exponential function, Lyapunov equation, Sylvester equation, Bartels-Stewart algorithm

Životopis

Rođen sam u Osijeku 2000. godine. Od 2006. do 2010. godine pohađao sam PŠ Punitovci, a od 2010. do 2014. godine Osnovnu školu "Josip Kozarac" u Josipovcu Punitovačkom. Po završetku osnovne škole upisujem III. gimnaziju Osijek koju završavam 2018. godine te nakon toga upisujem Preddiplomski sveučilišni studij matematike i računarstva na tadašnjem Odjelu za matematiku Sveučilišta J. J. Strossmayera u Osijeku. Preddiplomski studij završavam 2021. godine te odmah po završetku upisujem Diplomski sveučilišni studij matematike, smjer Matematika i računarstvo, na istom odjelu, sadašnjem Fakultetu primijenjene matematike i informatike.