(Trefethen-Bau, Lekcia 29)

V predošlej lekcii sme videli QR algoritmus, ktorý konvergoval lineárne. Pomocou vhodne zvolených posunov $A \to A - \mu I$ dosiahneme kubickú konvergenciu, podobne ako v inverznom iterovaní pomocou Rayleighových podielov – tie sa totiž v posunutom QR algoritme implicitne vyskytnú.

Inverzné iterovanie v QR algoritme

Nadalej predpokladajme, že $A \in \mathbb{R}_{m,m}(\mathbb{R})$ je reálna a symetrická, má reálne vl. hodnoty $\{\lambda_j\}$ a ortonormálne vl. vektory q_j. (Alg. 28.1)

V predošlej lekcii sme si ukázali, že QR algoritmus je ekvivalentný simultánnemu iterovaniu, keď začneme s maticou identity. ($A^k I$)

Špeciálne, prvý stĺpec výsledku zodpovedá mocninovému iterovaniu s počiatočným vektorom e_1.

V QR algoritme však môžeme objaviť aj <u>simultánne inverzné iterovanie</u> (t.j. násobenie maticou A^{-1}), ktoré pracuje s "preklopenou" identitou $P = \begin{pmatrix} 0 & & & 1 \\ & & 1 & \\ & \iddots & & \\ 1 & & & 0 \end{pmatrix}$, špeciálne m-tý stĺpec výsledku sa vyvíja ako inverzné mocninové iterovanie s počiatočným vektorom e_m.

Podme to ukázať:

Nech $Q^{(k)}$ je ortogonálna matica z k-tého kroku QR algoritmu a $\underline{Q}^{(k)} = Q^{(1)} Q^{(2)} \dots Q^{(k)}$ je akumulovaný súčin takýchto matíc:

$$\underline{Q}^{(k)} = \prod_{j=1}^{k} Q^{(j)} = \left[q_1^{(k)} \middle| q_2^{(k)} \middle| \dots \middle| q_m^{(k)} \right]$$

(v minulej prednáške som dával k vektorom $q_j^{(k)}$ podtrhnutie, ale zabudli sme sa ním bez neho...)

Matica $\underline{Q}^{(k)}$ sa vyskytla v k-tom kroku simultánneho iterovania, t.j. $A^k = \underline{Q}^{(k)} \underline{R}^{(k)}$.

Túto rovnosť môžeme invertovať:

$$A^{-k} = (\underline{R}^{(k)})^{-1} (\underline{Q}^{(k)})^T = \underline{Q}^{(k)} ((\underline{R}^{(k)})^{-1})^T \quad (*)$$

ale matica A^{-k} je aj symetrická, takže aj transponovaná

Toto nie je celkom QR rozklad matice A^{-k}, lebo $((R^{(k)})^{-1})^T$ je dolná trojuholníková. To sa však dá opraviť pomocou matice $P = \begin{pmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{pmatrix}$.

Platí $P^2 = I$, preto rovnosť (*) vieme prepísať ako

$$A^{-k} P = [\underline{Q}^{(k)} P][P((\underline{R}^{(k)})^{-1})^T P]$$

Potom: $\underline{Q}^{(k)} P$ je ortogonálna a $P((\underline{R}^{(k)})^{-1})^T P$ je horná trojuholníková, lebo zľava prehodíme poradie riadkov a sprava poradie stĺpcov, čo z dolnej trojuholníkovej vyrobí hornú trojuholníkovú.

Dostávame tak QR rozklad matice $A^{-k}P$, teda simultánne ~~inverzné~~ iterovanie maticou A^{-1}, t.j. inverzné simultánne iterovanie pre maticu A.

Potom pre prvý stĺpec $A^{-k}P$ máme ~~nie~~ inverzné iterovanie prvého stĺpca matice P – em, nedeviatkedok bude prvý stĺpec matice $\underline{Q}^{(k)} P$, t.j. posledný stĺpec matice $\underline{Q}^{(k)}$.

QR algoritmus s posunom a posunuté inverzné iterovanie

V lekcii 27 sme videli, že inverzné iterovanie má veľkú výhodu oproti mocninovému iterovaniu, keďže to prvé sa dá vapídne urýchliť voľbou posunu $\mu \approx \lambda_j$ a inverzným iterovaním $A - \mu I$.

A práve sme videli, že QR algoritmus v sebe obsahuje mocninové iterovanie (prvý stĺpec $\underline{Q}^{(k)}$) a aj inverzné iterovanie (posledný stĺpec $\underline{Q}^{(k)}$).

V algoritme 28.2 sme už videli, ako vieme vložiť posuny do "čistého" QR algoritmu. Jednak tým dosiahneme posuny v simultánnom iterovaní ale aj posun v inverznom iterovaní — čo bude mať za následok kubickú konvergenciu.

—

Pre QR algoritmus sme mali:

$$A^{(k-1)} = Q^{(k)} R^{(k)}$$

$$A^{(k)} = R^{(k)} Q^{(k)}$$

z toho

$$A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)}$$

$$= \underline{Q}^{(k)T} A \underline{Q}^{(k)}$$

a tiež

$$A^k = \underline{Q}^{(k)} \underline{R}^{(k)}$$

pre posunuté

$$A^{(k-1)} - \mu^{(k)} I = Q^{(k)} R^{(k)}$$

$$A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$$

dá rovnaký ~~rekurrenciu~~ vzťah

$$A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)}$$

$$= \underline{Q}^{(k)T} A \underline{Q}^{(k)}$$

Ale táto posledná sa zmení na:

$$(A - \mu^{(k)} I)(A - \mu^{(k-1)} I) \dots (A - \mu^{(1)} I) = \underline{Q}^{(k)} \underline{R}^{(k)}.$$

(bez dôkazu - cvičenie)

Teda $\underline{Q}^{(k)} = \prod_{j=1}^{k} Q^{(j)}$ bude tentoraz ortogonalizácia matice $\prod_{j=k}^{1} (A - \mu^{(j)} I)$.

Prvý stĺpec $\underline{Q}^{(k)}$ bude zodpredať posunutým mocninovým iteráciám so začiatkom e_1 a posledný $(n-1)$ stĺpec zodpredá

inverznou iteráciu s tými istými posunmi, so začiatkom e_m. Ak sú posuny dobrou aproximáciou vl. hodnoty, potom posledný stĺpec $Q^{(k)}$ konverguje rýchlo k vl. vektoru.

QR algoritmus a Rayleighov podiel (s posunom)

V predošlej diskusii sme zistili, že v QR algoritme sa ukrýva aj inverzné iterovanie (s posunom). Na dokompletizie myšlienky potrebujeme rozumne vyberať tieto posuny.

Môžeme začať s Rayleighovým podielom
- odhadovaný vl. vektor zodpovedá poslednému stĺpcu $Q^{(k)}$, teda vektoru $q_m^{(k)}$. Počítame sa na Rayleighov podiel pre tento vektor, a berieme ho ako posun:

$$r(q_m^{(k)}) = \frac{(q_m^{(k)})^T A\, q_m^{(k)}}{(q_m^{(k)})^T q_m^{(k)}} = (q_m^{(k)})^T A\, q_m^{(k)} = \mu^{(k)}.$$

$\rightarrow$ takto zvolené $\mu^{(k)}$ a $q_m^{(k)}$ dá inverzné iterovanie s Rayleighovým podielom pre počiatočný vektor e_m. $\rightarrow$ máme kubickú konvergenciu (až to konverguje)

V skutočnosti $r(q_m^{(k)})$ nebolo počítať, lebo so vzťahom $A^{(k)} = (Q^{(k)})^T A\, Q^{(k)}$ máme pre pozíciu (m,m) nasledovné:

$$A^{(k)}_{(m,m)} = e_m^T A^{(k)} e_m = e_m^T (Q^{(k)})^T A\, Q^{(k)} e_m = (q_m^{(k)})^T A\, q_m^{(k)}.$$

Preto stačí voliť $\mu^{(k)} = A^{(k)}_{m,m}$.

Takýto posun sa nazýva **posun o Rayleighov podiel**.

Wilkinsonov posun

V generickom prípade vedie Rayleighov posun v QR algoritme ku kubickej konvergencii, ale existujú počiatočné podmienky, kedy to tak nebude.

Zoberme maticu $A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

Pre QR algoritmus bez posunu máme:

$$A = Q^{(1)} R^{(1)} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

a

$$A^{(1)} = R^{(1)} Q^{(1)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = A.$$

Teda konvergenciu k diagonálnej matici nemáme. (ďalej $|\lambda_1| = |\lambda_2|$)

Podobne Rayleighov podiel vyjde $\mu = A_{m,m} = 0$, teda ani „posun" nič nezlepší.

V najhoršom prípade QR algoritmus s posunom 0 Rayleighov podiel zlyhávа.

Pozrime sa na príčinu zlyhania: jedna vl. hodnota je $+1$, druhá -1, odhad $\mu = A_{2,2}$ je 0 – presne v strede medzi nimi, pričom aj posun zostáva 0 - sme uviaznutí medzi nimi...

Treba nejako túto symetriu narušiť. To sa dá pomocou tzv. Wilkinsonovho posunu: Namiesto $A^{(k)}_{m,m}$ sa pozrieme na 2×2 podmatici $B = \begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix}$ matice $A^{(k)}$.

(ľavý dolný)

Vl. hodnoty

$$\chi = \lambda^2 - (a_m + a_{m-1})\lambda + (a_m a_{m-1} - b_{m-1}^2)$$

$$\lambda_{1,2} = \frac{+(a_m + a_{m-1}) \pm \sqrt{(a_m + a_{m-1})^2 - 4(a_m a_{m-1} - b_{m-1}^2)}}{2}$$

$$= \frac{a_m + a_{m-1}}{2} \pm \sqrt{\left(\frac{a_m - a_{m-1}}{2}\right)^2 + b_{m-1}^2}$$

$$= a_m + \frac{a_{m-1} - a_m}{2} \pm \sqrt{\left(\frac{a_{m-1} - a_m}{2}\right)^2 + b_{m-1}^2} = a_m + \delta \pm \sqrt{\delta^2 + b_{m-1}^2}$$

$$-b_{m-1}^2 = |\delta|^2 - (\delta^2 + b_{m-1}^2) = \left(|\delta| + \sqrt{\delta^2+b_{m-1}^2}\right)\left(|\delta| - \sqrt{\delta^2+b_{m-1}^2}\right)$$

takže $$|\delta| - \sqrt{\delta^2+b_{m-1}^2} = \frac{-b_{m-1}^2}{|\delta| + \sqrt{\delta^2+b_{m-1}^2}}$$

→ to vedie k vzorcu

$$\mu = a_m - \operatorname{sign}(\delta)\, b_{m-1}^2 \Big/ \left(|\delta| + \sqrt{\delta^2+b_{m-1}^2}\right),$$

ktorý je numericky stabilný. $\delta = \frac{a_{m-1}-a_m}{2}$.

Pre $\delta = 0$ sa dá zvoliť $\operatorname{sign}(\delta)$ 0 alebo 1, potom

$$\mu = a_m \pm b_{m-1}.$$

Podobne ako posun o Rayleighov podiel, Wilkinsonov posun dáva ~~ge~~ v generickom prípade ~~pozorne~~ kubickú konvergenciu. V najhoršom prípade bude konvergencia kvadratická, teda QR algoritmus s Wilkinsonovým posunom vždy konverguje (v presnej aritmetike).

V príklade, ktorý robil problém je posun $\mu = 0 \pm 1 = \pm 1$, Wilk.
teda symetria sa naruší a konvergencia dosiahnuje na jeden krok.

Stabilita a presnosť

Týmto sme ukončili diskusiu základného mechanizmu QR algoritmu, aj keď mnohé praktické detaily sa vynechali (podmienky deflácie, stratégie posunov ...)

Patrilo by sa povedať niečo o stabilite a presnosti. Keďže používame ortogonálne matice, ich podmienenosť je 1, takže ~~kde~~ podľa očakávania dostaneme

spätnú stabilitu algoritmu
Tento výsledok sa dá stromulovať nasledovne:

$\tilde{\Lambda}$ je diagonalizácia A spočítaná vo floating point aritmetike,

$\tilde{Q}$ je (presne) ortogonálna matica, ktorú dostaneme ako súčin Householderových reflexií (resp. Givensových rotácií), ktoré používame v QR rozkladoch počas výpočtu.

Potom máme:

Veta Pre reálnu, symetrickú maticu A diagonalizovanú QR algoritmom na počítači spĺňajúcom axiómy 13.5 a 13.7 s výsledkom $\tilde{\Lambda}$, $\tilde{Q}$ platí:

$$\tilde{Q}\tilde{\Lambda}\tilde{Q}^* = A + \delta A \quad \text{s} \quad \frac{\|\delta A\|}{\|A\|} = O(\varepsilon_{machine})$$

pre nejaké $\delta A \in \mathbb{M}_{m\times m}(\mathbb{C})$.

Teda QR algoritmus vypočíta presné riešenie mierne perturbovaného systému.
Spojením s podobnou vetou pre tri-diagonalizáciu máme, že tridiagonalizácia nasledovaná QR iteráciami je spätne stabilný algoritmus pre výpočet vl. hodnôt.
Spolu s výsledkom cvičenia 26.4 o perturbáciách vl. hodnôt (DÚ) (keďže reálne symetrické matice sú normálne) máme:

$$\frac{|\tilde{\lambda}_j - \lambda_j|}{\|A\|} = O(\varepsilon_{machine}).$$

Pritom cena za takúto presnosť je $\sim \frac{4}{3}m^3$ flopov v prvej fáze a $O(m^2)$ flopov v druhej (lebo tridiag.).
Pritom na súčin dvoch $m\times m$ plných matíc potrebujeme $2m^3$ flopov.
Čiže za 2/3 ceny násobenia matíc máme vl. hodnoty. Nie zle...

(Trefethen–Bau, Lekcia 30)

Ďalšie algoritmy na výpočet vl. hodnôt

Okrem QR algoritmu existujú aj iné algoritmy na výpočet vl. hodnôt, ktoré stoja za zmienku:

- Jacobiho algoritmus
- bisekcia
- rozdeľuj a panuj (divide & conquer)

Jacobi

Tento algoritmus pochádza od Jacobiho (1845). Z pohľadu numeriky je zaujímavý preto, lebo sa dá počítať pomocou paralelných výpočtov, ale stále nedokáže konkurovať iným...

Myšlienka je nasledovná: vieme, že pre 5×5 matice neexistuje (a všeobecne) dostať vl. hodnoty priamym výpočtom.

Pre 2×2, 3×3 a 4×4 to však ide. Myšlienka teda bude diagonalizovať malé podmatice v matici A, potom ďalšiu a dúfať, že dosiahneme diagonalizáciu celej matice.

Existujú implementácie pre 4×4 matice, ale klasicky sa to robí pre 2×2:

2×2 reálna symetrická matica sa dá diagonalizovať:

$$J^T \begin{bmatrix} a & d \\ d & b \end{bmatrix} J = \begin{bmatrix} * & 0 \\ 0 & * \end{bmatrix},$$

kde J je ortogonálna. J sa dá vybrať viacerými spôsobmi – obsahuje vl. vektory matice $\begin{bmatrix} a & d \\ d & b \end{bmatrix}$ (normalizované), ale stále máme voľnosť vo výbere štvorice.

Jednou možnosťou je

$$F=\begin{bmatrix} -c & s \\ s & c \end{bmatrix},$$

kde $s=\sin\theta$ a $c=\cos\theta$, potom $\det F=-1$ a ide o reflexiu.

Môžme však v prvom stĺpci zmeniť znamienko a dostať

$$J=\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

s $\det J=1$, teda ide o rotáciu zachovávajúcu orientáciu.

Tento výber bude štandardný v Jacobiho algoritme.

Dá sa ukázať, že správne θ pre diagonalizáciu $\begin{bmatrix} a & d \\ d & b \end{bmatrix}$ spĺňa

$$\tan(2\theta)=\frac{2d}{b-a}.$$

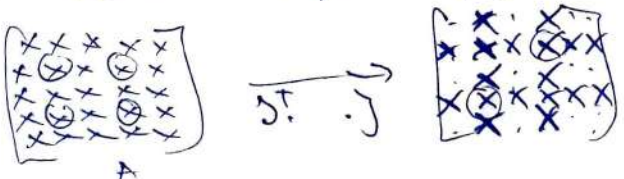
Matica s takouto rotáciou s takýmto θ sa nazýva Jacobiho rotácia.

(takáto matica má rovnaký tvar ako Givensova rotácia - lekcia 10, resp. MNLA, ktorá sa používala pri ortogonálnej triangularizácii J^TA)

Nech $A\in M_{m\times m}(\mathbb{R})$ je symetrická (typu $m\times m$).

Jacobiho algoritmus pozostáva z opakovaného aplikovania podobnostnej transformácie J^TAJ. Matica J je teraz $m\times m$ identita okrem štyroch zložiek $J_{ii}, J_{ij}, J_{ji}, J_{jj}$, kde obsahuje $\begin{bmatrix} c & s \\ -s & c \end{bmatrix}$.

Násobenie J^T zľava mení riadky i a j, násobenie J sprava mení stĺpce i, j.



v každom kroku vyrobíme nuly na pozíciách (i,j) a (j,i)

ale predošlé nuly sa mohli zničiť.
Zvyčajne však veľkosť týchto nenulových zložiek klesne.

- ktoré a_{ij} vybrať na vynulovanie?
- ponúka sa výber najväčšia mimodiagonálna zložka.

V tom prípade dostaneme konvergenciu, lebo súčet štvorcov mimo diag. zložiek klesne aspoň o faktor

$$1 - \frac{2}{(n^2-n)}$$ (cvičenie 30.3).

Po $O(n^2)$ opakovaniach (každé $O(n)$ operácií) máme pokles o konšt. faktor, teda konvergencia k presnosti $\varepsilon_{machine}$ sa dosiahne s $O(n^3 \log(\varepsilon_{machine}))$ operáciami.

V skutočnosti bude konvergencia skôr kvadratická ako lineárna, teda počet operácií bude $O(n^3 \log(|\log(\varepsilon_{machine})|))$.

Typicky sa však zvykne eliminovať mimo diagonály cyklicky – aby sme sa vyhli prehľadávaniu maxima ktoré je $O(n^2)$ dlhé. Napr. sa vezme $n(n-1)/2$ zložiek nad diagonálou v poradí $a_{12}, a_{13}, \ldots$

V takomto prípade sa tiež dosiahne asymptotická konvergencia.

Po jednom "zametení" $n(n-1)/2$ párov mimodiagonálnych zložiek sa typicky dosiahne lepšie ako (zlepšenie o) konštantný faktor, konvergencia zvykne byť kvadratická.

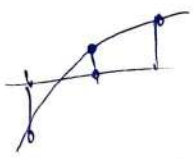
Výhody Jacobiho algoritmu: v jednom kroku sa pracuje iba s párom riadkov a párom stĺpcov → dá sa paralelizovať.

Matica nie je tridiagonalizovaná (Jacobiho rotácie

by túto štruktúru zničili.
Typicky sa pre $n \leq 1000$ dosiahne konvergencia na menej než 10 zameraní, pamäť bývа väčšia ako pri QR algoritme.
Rýchlosť však nie je tiež dobrá ako QR algoritmus, v praxi je asi 10x pomalší.

Bisekcia

Ďalší z algoritmov na hľadanie vl. hodnôt - bisekcia - má praktický význam, keď sa hľadajú iba niektoré z vl. hodnôt. (hlavne tridiagonalizovanej matice).
Napr. 10% najväčších, najmenších 30, všetky v danom intervale.
Postupuje sa tak, že sa hľadajú korene char. polynómu $p(x) = \det(A - xI)$ pôlením intervalu - bisekcia.
- lebo matica je reálna symetrická, korene sú reálne.
Toto znie ako zlý nápad, lebo hľadanie koreňov polynómu je nestabilný algoritmus, ako sme už spomenuli.
Lenže toto platí vtedy, keď hľadáme korene z koeficientov.
Pri bisekcii priamo hľadáme hodnotu $p(x)$ pre rôzne hodnoty x - bez hľadania koeficientov polynómu a potom hľadáme korene podľa hodnôt:



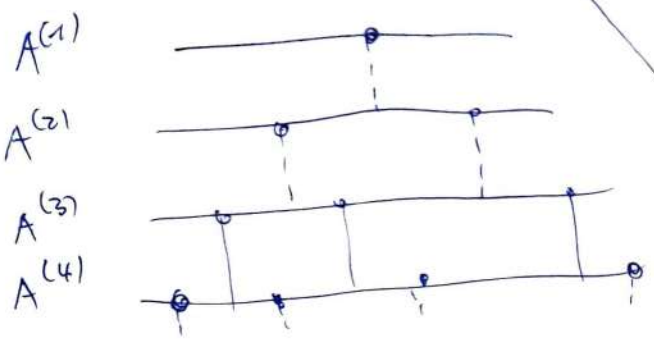
hodnotu $p(x) = \det(A - xI)$ ju hľadáme Gaussovou eliminačnou s pivotovaním, algoritmus tak bude stabilný.

Toto nestrie veľmi náročné a sofistikované, pokiaľ nevyužijeme dodatočné vlastnosti vl. hodnôt a determinantov, ktoré nie sú ešte známe.

Pre $m \times m$ reálnu symetrickú maticu označme $A^{(1)}, A^{(2)}, \dots, A^{(m)}$ jej ľavé (horné) podmatice veľkosti $1\times 1, 2\times 2, \dots, m\times m$.

Potom vlastné hodnoty týchto matíc sú **prepletené**:

(zhrnutie) $\lambda_j^{(k+1)} \leq \lambda_j^{(k)} < \lambda_{j+1}^{(k)}$



Ak A je tridiagonálna, bez núl na vedľajších diagonálach, potom majú $A^{(k)}$ ostré vl. hodnoty:

$\lambda_1^{(k)} < \lambda_2^{(k)} < \dots < \lambda_k^{(k)}$,

ktoré sú **striktne prepletené**

Vďaka tomuto prepleteniu vieme získať presné počty vl. hodnôt v ľubovoľnom intervale. Napr. pre maticu

$$A = \begin{bmatrix} 1 & 1 & & \\ 1 & 0 & 1 & \\ & 1 & 2 & 1 \\ & & 1 & -1 \end{bmatrix}$$

máme $\det(A^{(1)}) = 1$, $\det(A^{(2)}) = -1$, $\det(A^{(3)}) = -3$, $\det(A^{(4)}) = 4$

→ preto počet záporných vl. hodnôt pre $A^{(1)}$ je: 0

$A^{(2)}$ 1

$A^{(3)}$ 1

$A^{(4)}$ 2

Vo všeobecnosti, symetrická tridiagonálna $m\times m$ matica A má počet záporných vl. hodnôt rovný počtu zmien znamienok v postupnosti $1, \det(A^{(1)}), \det(A^{(2)}), \dots, \det(A^{(m)})$. – Sturmova postupnosť.

Toto funguje aj s nulovými determinantami, pričom "zmenu znamienka" definujeme aj ako $\begin{pmatrix}+\\0\end{pmatrix} \to -$, resp. $\begin{pmatrix}-\\0\end{pmatrix} \to +$ ale nie $\pm \to 0$.

Posunutím A o aI, bI vieme ľahko zistiť počet vl. hodnôt v intervale $\langle a,b)$ – počet v $(-\infty,b)$ mínus $(-\infty,a)$.

Ešte jedno pozorovanie k bisekcii: $\begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & b_2 & \\ & b_2 & \ddots & b_{m-1} \\ & & b_{m-1} & a_m \end{bmatrix}$

pre tridiagonálnu maticu vieme počítať determinant pomocou Laplaceovho rozvoja, ktorý dá rekurentný vzťah s troma členmi:

$$\det(A^{(k)}) = a_k \det A^{(k-1)} - b_{k-1}^2 \det A^{(k-2)}$$

Spolu s posunom xI a zápisom $p^{(k)} = \det(A^{(k)} - xI)$ máme

$$p^{(k)}(x) = (a_m - x)\, p^{(k-1)}(x) - b_{k-1}^2\, p^{(k-2)}(x). \quad (*)$$

Dodefinovaním $p^{(-1)}(x) = 0$ a $p^{(0)}(x) = 1$ dostaneme aj rovnosť pre $k = 1$ a 2.

Počítaním $(*)$ pre rôzne hodnoty x a sledovaním počtu zmien znamienok vieme zistiť počet vl. hodnôt v ľubovoľne malom intervale.

Cena je iba $O(m)$, teda $O(m \log(\varepsilon_{machine}))$ pre nájdenie vl. hodnoty s presnosťou $\varepsilon_{machine}$.

Ak treba iba málo vl. hodnôt, toto je úspora oproti $O(m^2)$ ďalšej iteratívnej časti QR algoritmu.

– Opäť pri viacerých procesoroch môžeme hľadať vl. hodnoty paralelne.

Divide-and-Conquer

- najväčší pokrok od QR algoritmu zo 60-tych rokov

1981 - Cuppen - pre výpočet vl. vektorov 2x taký rýchly

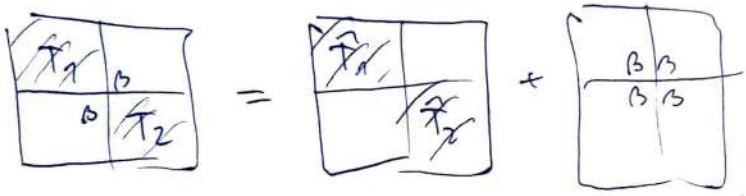
(ba prehľad: (detaily sú však dôležité, lebo algoritmus nie je stabilný pokiaľ sa nenastavia správne
- doladenie trvalo 10 rokov...)

$T \in \mathbb{R}^{m,m}(\mathbb{R})$ je symetrická, reálna, tridiagonálna, ireducibilná

(žiadne nuly na vedľajších diagonálach)

Potom pre n $1 \le n \le m$ T sa dá rozdeliť

takto:

$T =$  $=$ $+$

T_1 je ľavá-horná hlavná $n \times n$ podmatica, T_2 je pravá-dolná

$(m-n) \times (m-n)$ a $\beta = t_{n+1,n} = t_{n,n+1} \neq 0$.

Rozdiel medzi T_1 a $\hat{T}_1$ je v zložke n,n ; $t_{n,n}$ sa nahradí $t_{n,n} - \beta$.

T_2 a $\hat{T}_2$ sa líšia v $n+1, n+1$: $t_{n+1,n+1} \dots t_{n+1,n+1} - \beta$

Zmeny v T_1 a T_2 sa robia tak, aby matica napravo mala hodnosť 1.

Tridiagonálna matica T sa dá napísať ako redukcibilná tridiagonálna matica (dva tridiagonálne bloky) spolu s kovekciou hodnosti 1.

Algoritmus rozdeľuj & panuj rozdelí T na časti $\approx m/2$.

Ak poznáme vl. hodnoty (a vektory) $\hat{T}_1, \hat{T}_2$, chceme

nájsť súvislosť s vl. hodnotami matice T.

Rekurzívne rozdelíme $\hat{T}_1$ a $\hat{T}_2$ na menšie bloky až dôjdeme k 1×1 blokom (a korekciám) alebo dostatočne malým blokom veľkosti, na ktoré pustíme QR algoritmus (toto je efektívnejšie)

Ako teda prepojiť vl. hodnoty matíc $\hat{T}_1, \hat{T}_2$ a matice T?

Majme diagonalizácie $\hat{T}_1 = Q_1 D_1 Q_1^T$ a $\hat{T}_2 = Q_2 D_2 Q_2^T$.

Potom $$T = \begin{bmatrix} Q_1 & \\ & Q_2 \end{bmatrix} \left(\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \beta z z^T \right) \begin{bmatrix} Q_1^T & \\ & Q_2^T \end{bmatrix},$$

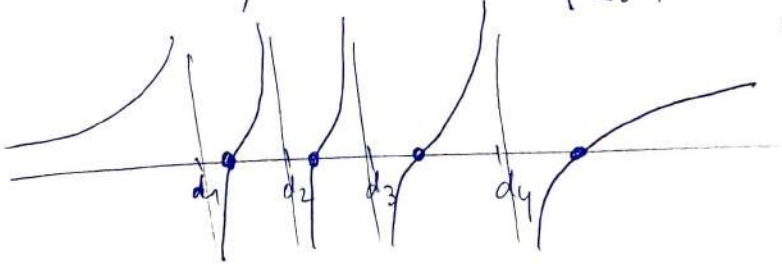
kde $z^T = (q_1^T, q_2^T)$ pričom q_1^T je posledný riadok Q_1 a q_2^T je prvý riadok Q_2.

Treba teda nájsť vl. hodnoty diagonálnej matice s korekciou hodnosti 1.

Zjednodušme si značenie: chceme nájsť vl. hodnoty matice $D + w w^T$, kde $D \in \mathbb{R}^{m,m}$ je diagonálna s rôznymi zložkami $\{d_j\}$ a $w \in \mathbb{R}^m$ je vektor.

(ak $\beta > 0$ berieme so znamienkom +, ak $\beta < 0$ zobrali by sme $D - w w^T$.

Predpokladajme, že $w_j \neq 0$ pre $\forall j$, inak by bol problém redukcibilný. Potom vl. hodnoty $D + w w^T$ sú korene racionálnej funkcie $f(\lambda) = 1 + \sum_{j=1}^{m} \frac{w_j^2}{d_j - \lambda}$



to sa dá vyjadriť z nasledovného:

$(D + ww^T)q = \lambda q$ pre $q \neq 0$, teda

$(D - \lambda I)q + w(w^T q) = 0$

teda $q + (D - \lambda I)^{-1} w(w^T q) = 0$

$w^T q + w^T (D - \lambda I)^{-1} w (w^T q) = 0$

čo je $f(\lambda)(w^T q) = 0.$

Ak by $w^T q = 0$, potom $(D + ww^T)q = Dq = \lambda q \rightarrow$

q je vl. vektor D, teda vieme ej ale to dá $w^T q = w_j \neq 0$

Preto ak je q vl. vektor $D + ww^T$, s vl. hodnotou λ,

potom $f(\lambda)$ musí byť 0.

Naopak, $f(\lambda)$ má presne m koreňov.

$f(\lambda) = 0$ sa nazýva sekulárna rovnica.

V každom kroku divide-and-conquer algoritmu sa

korene $f(\lambda)$ nájdu Newtonovou metódou –

stačí $O(1)$ iterácií (resp. $O(\log(\log(\varepsilon_{machine})))$ – ak $\varepsilon_{machine}$ berieme ako premennú)

na jeden koreň – teda $O(m)$ operácií floprov pre jeden

koreň $m \times m$ matice, resp. $O(m^2)$ celkovo.

Ak uvažujeme o delení na prvé polovice, celkovo

potrebujeme $O\left(m^2 + 2\left(\frac{m}{2}\right)^2 + 4\left(\frac{m}{4}\right)^2 + 8\left(\frac{m}{8}\right)^2 + \dots + m\left(\frac{m}{m}\right)^2\right)$

floprov, čo dáva v limite $O(m^2)$ (a nie $O(m^2 \log m)$)

Teda počet operácií bude riadovo rovnaký ako ~~v~~ v QR algoritme.

Zaliať nevidno prečo by divide-and-conquer ~~nie~~ priniesol úsporu.

- Vo fáze 1 (Lekcia 25) potrebujeme $\frac{4}{3}m^3$ flopov

a vo fáze 2 sa zdá zlepšenie $O(m^2)$ nepodstatné.

Účelom však zase bude hrať, keď budeme počítať nielen vl. hodnoty ale aj vl. vektory.

Potom si musíme pri QR algoritme pamätať aj maticu $\underline{Q}^{(k)}$, resp. Q^0 z fázy 1

Tým počet operácií stúpne na $\frac{8}{3}m^3$ vo fáze 1.

a vo fáze 2 treba $O(m^3)$ (násobíme $\underline{Q}^{(k)} = Q^{(1)}Q^{(2)}\dots Q^{(k)}$)

(v praxi $6m^3$)

Divide-and-conquer tu vyhráva, lebo v každom kroku stačí počítať skalárnu funkciu $f(\lambda)$ a nie ortogonálnu maticu $Q_z^{(j)}$, pričom QR algoritmus manipuluje s $Q^{(s)}$

- ak sa počíta počet operácií - $O(m^3)$ v divide-and-conquer zodpovedá násobeniu $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}\left(\begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix} + \beta zz^T\right)\begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix}$

- Sumovaním cez všetky kroky dostaneme $\frac{4}{3}m^3$ flopov, zlepšenie oproti $\approx 6m^3$. Spolu s $\frac{8}{3}m^3$ flopov vo fáze 1 to je zlepšenie z $\approx 9m^3$ na $4m^3$.

V skutočnosti divide-and-conquer v praxi funguje ešte lepšie – matice Q_j a vektory z vyhádzajú numericky riedke – zložky majú hodnoty nižšie ako istá hodnota riedkosti strojová presnosť. – môžeme použiť tzv. numerickú deflačiu – rozdeliť na tridiagonálne problémy hľadania vl. hodnôt pre matice menších rozmerov.

V typickom prípade bude vo fáze 2 potrebných nájsť cca n^3 flopov, čím sa dosiahne kombinačné $\frac{8}{3}n^3$ flopov.

Fáza 1 + Fáza 2

Dôkazom tohto fenoménu je tzv. "exponenciálne lokalizované" vl. vektory pre tridiagonálne matice (cvič. 30.7).

Existuje veľa variant divide-and-conquer algoritmov, lebo sa dajú robiť iné kovekcie
- iné hodnosti 1 kvôli stabilite
- ~~vzdialené~~ aktualizácie hodnosti 2
- rôzne metódy hľadania koreňov $f(\lambda)$
- pre veľké m sa inak násobí Q_j (multipole expansions)

ale ...