Solving an eigenvalue problem

Given $A \in \mathbb{R}^{n\times n}$ or $\mathbb{C}^{n\times n}$,

$$Ax = \lambda x$$

Goal: find *all* eigenvalues (and eigenvectors) of a matrix

- Look for Schur form $A = UTU^*$ $\mathrm{diag}(T) = \{\lambda_i\}_{i=1}^{n}$

We'll describe an algorithm called the QR algorithm that is used universally, e.g. by MATLAB's `eig`. It

- finds all eigenvalues (approximately but reliably) in $O(n^3)$ flops,
- is backward stable.

Sister problem: Given $A \in \mathbb{R}^{m\times n}$ or $\mathbb{C}^{m\times n}$, compute SVD $A = U\Sigma V^*$

- 'ok' algorithm: eig(A^TA) to find V, then normalise AV
- there's a better algorithm: Golub-Kahan bidiagonalisation

QR algorithm for eigenproblems

Set $A_1 = A$, and

$$\underset{\text{QR fact.}}{A_1 = Q_1R_1}, \quad \underset{\text{reverse}}{A_2 = R_1Q_1}, \quad \underset{\text{QR.}}{A_2 = Q_2R_2}, \quad \underset{\text{reverse}}{A_3 = R_2Q_2}, \quad \ldots$$

- A_k are all similar: $\underline{A_{k+1} = Q_k^T A_k Q_k}$
- We shall 'show' that $A_k \to$ triangular **triangular** as $k \to \infty$ (diagonal if A normal)
- Basically: QR(factorise)$\to RQ$(swap)$\to QR \to RQ \to \cdots$

$$A = A_1 = Q_1R_1$$

$$A_2 = R_1Q_1 = \underbrace{Q_1^T(Q_1}_{I} R_1)Q_1 = \underline{\underline{Q_1^T}}\, A_1 \underline{\underline{Q_1}}$$

So $A_2 \sim A_1$

$QR \to RQ$

$$\left[\ \mathrm{eig}(XY) = \mathrm{eig}(YX) \quad \text{"for } \lambda \neq 0\text{" (when } X, Y \text{ not square)}\ \right]$$

QR algorithm for eigenproblems

Set $A_1 = A$, and

$$A_1 = Q_1 R_1, \quad A_2 = R_1 Q_1, \quad A_2 = Q_2 R_2, \quad A_3 = R_2 Q_2, \quad \ldots$$

- A_k are all similar: $A_{k+1} = Q_k^T A_k Q_k$
- We shall 'show' that $A \to$ triangular **triangular** (diagonal if A normal)
- Basically: QR(factorise)$\to$ RQ(swap)$\to$ $QR \to RQ \to \cdots$

- Fundamental work by Francis (61,62) and Kublanovskaya (63)

- Truly Magical algorithm!
  - backward stable, as based on orthogonal transforms
  - always converges (with shifts), but global proof unavailable(!)
  - uses 'shifted inverse power method' (rational functions) without inversions

QR algorithm and power method

QR algorithm: $A_k = Q_kR_k$, $A_{k+1} = R_kQ_k$, repeat. Claims: for $k \ge 1$,

$$A^k = (Q_1 \cdots Q_k)(R_k \cdots R_1) =: \underbrace{Q^{(k)}}_{\text{orth}} R^{(k)}, \qquad A_{k+1} = (Q^{(k)})^T A Q^{(k)}.$$

Proof: recall $A_{k+1} = Q_k^T A_k Q_k$, repeat.

Proof by induction: $k = 1$ trivial.

Suppose $A^{k-1} = Q^{(k-1)}R^{(k-1)}$. We have

$$A_k = (Q^{(k-1)})^* A Q^{(k-1)} = Q_kR_k.$$

Then $AQ^{(k-1)} = Q^{(k-1)}Q_kR_k$, and so

$$\underset{AA^{k-1}}{A^k} = AQ^{(k-1)}R^{(k-1)} = \underbrace{Q^{(k-1)}Q_k}_{Q^{(k)}}\underbrace{R_kR^{(k-1)}}_{} = Q^{(k)}R^{(k)}\square$$

QR algorithm and power method

QR algorithm: $A_k = Q_k R_k$, $A_{k+1} = R_k Q_k$, repeat.

$$A^k = (Q_1 \cdots Q_k)(R_k \cdots R_1) =: Q^{(k)} R^{(k)}, \qquad A_{k+1} = (Q^{(k)})^T A Q^{(k)}.$$

$A^k e_1$

$Q^{(k)}R^{(k)}e_1 = Q^{(k)}\begin{bmatrix} R_{11} \\ 0 \end{bmatrix}$ = (1st col of $Q^{(k)}$)$\cdot R_{11}$

QR factorisation of A^k: 'dominated by leading eigenvector' x_1, where $Ax_1 = \lambda_1 x_1$ (recall power method)

$\frac{A^k v}{\|A^k v\|} \to x_1$ as $k \to \infty$

$v = [e_i]$ $\quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \leftarrow i$

In particular, consider $A^k[1, 0, \ldots, 0]^T = A^k e_{\overset{n}{1}}$:

- $A^k e_n = R^{(k)}(1,1) Q^{(k)}(:,1)$, parallel to 1st column of $Q^{(k)}$
- By power method, this implies $Q^{(k)}(:,1) \to x_1$
- Hence by $A_{k+1} = (Q^{(k)})^T A \underbrace{Q^{(k)}}_{}$, $A_k(:,1) \to [\lambda_1, 0, \ldots, 0]^T$

orth

$$\begin{bmatrix} x_1^T \\ * \end{bmatrix} A \begin{bmatrix} x_1 & * \end{bmatrix} = \begin{bmatrix} x_1^T \\ * \end{bmatrix} \begin{bmatrix} \lambda x_1 & * \end{bmatrix}$$

$$= \begin{bmatrix} \lambda_1 & * \\ 0 & \end{bmatrix} = \left[\begin{array}{c|c} \lambda_1 & * \\ \hline 0 & \underbrace{*}_{n-1} \end{array}\right] \Big\} n-1$$

Progress! But there is much better news

QR algorithm and inverse power method

$A_k \to \begin{bmatrix} \lambda_1 & * \\ 0 & * \end{bmatrix}$ at speed $\left|\frac{\lambda_2}{\lambda_1}\right|$

$\begin{bmatrix} * & \\ 0 & \lambda_n \end{bmatrix}$ at speed $\boxed{\left|\frac{\lambda_n}{\lambda_{n-1}}\right|}$ improve

QR algorithm: $A_k = Q_k R_k$, $A_{k+1} = R_k Q_k$, repeat.

$$A^k = (Q_1 \cdots Q_k)(R_k \cdots R_1) =: Q^{(k)} R^{(k)},$$

$$A_{k+1} = (Q^{(k)})^T A Q^{(k)}. \to \begin{bmatrix} * & * \\ 0 & \lambda_n \end{bmatrix}$$

→ left eigvec

$k \to \infty$.

Now take inverse: $A^{-k} = (R^{(k)})^{-1} (Q^{(k)})^*$,

Conjugate transpose: $(A^{-k})^* = \underbrace{Q^{(k)}}_{\text{orth}} \underbrace{(R^{(k)})^{-*}}_{\text{Lower triang}}$ ((QL))

$\lambda_i = \mathrm{eig}(A)$

$p(\lambda_i) = \mathrm{eig}(p(A))$

↳ poly (or any func)

$p(z) = r(z) = z^{-k}$

$\Rightarrow$ QR factorization of matrix $(A^{-k})^*$ with eigvals $r(\lambda_i) = \lambda_i^{-k}$

$\Rightarrow$ Connection also with (unshifted) inverse power method

NB no matrix inverse performed

$(A^{-k})^* \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} = Q^{(k)} (R^{(k)})^{-*} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$ // final col of $Q^{(k)}$

$\frac{|\lambda_2|}{|\lambda_1|}$

- This means final column of $Q^{(k)}$ converges to minimum left eigenvector x_n with rate $\frac{|\lambda_{n-1}|}{|\lambda_n|}$, hence $A_k(n,:) \to [0, \ldots, 0, \lambda_n]$
- (Very) fast convergence if $|\lambda_n| \ll |\lambda_{n-1}|$
- Can we force this situation? **Yes by shifts**

$\left((A^{-k})^* \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} \right)^* = [0, 0, \ldots, 1] (A^{-k})$

→ dominant left eigvec of A^{-1}

QR algorithm with shifts and shifted inverse power method

1. $A_k - s_kI = Q_kR_k$ (QR factorization)
2. $A_{k+1} = R_kQ_k + s_kI$, $\quad k \leftarrow k+1$, repeat.

similar

Roughly, if $s_k \approx \lambda_n$, then $A_{k+1} \approx \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ & & & & \lambda_n \end{bmatrix}$ by argument just made.

convergence of last row:

rate $\left|\frac{\lambda_n}{\lambda_{n-1}}\right|$ without shift.

with shift:

rate $\left|\frac{\lambda_n - s_k}{\lambda_{n-1} - s_k}\right| \to 0$ if $\lambda_n \approx s_k$

QR algorithm with shifts and shifted inverse power method

1. $A_k - s_kI = Q_kR_k$ (QR factorization)
2. $A_{k+1} = R_kQ_k + s_kI, \quad k \leftarrow k+1$, repeat.

$$\prod_{i=1}^{k}(A - s_iI) = Q^{(k)}R^{(k)} \;(= (Q_1\cdots Q_k)(R_k\cdots R_1))$$

recall $A^k = Q^{(k)}R^{(k)}$ w/o shifts

Proof: Suppose true for $k-1$. Then QR alg. computes $(Q^{(k-1)})^*(A - s_kI)Q^{(k-1)} = Q_kR_k$, so $(A - s_kI)Q^{(k-1)} = Q^{(k-1)}Q_kR_k$, hence

$$\prod_{i=1}^{k}(A - s_iI) = (A - s_kI)Q^{(k-1)}R^{(k-1)} = Q^{(k-1)}Q_kR_kR^{(k-1)} = Q^{(k)}R^{(k)}.$$

Inverse conjugate transpose: $\prod_{i=1}^{k}(A - s_iI)^{-*} \begin{bmatrix}0\\ \vdots\\ 0\\ 1\end{bmatrix} = Q^{(k)}(R^{(k)})^{-*} \begin{bmatrix}0\\ \vdots\\ 0\\ 1\end{bmatrix} \rightarrow$ dominant left eigvec of $\prod(A - s_iI)^{-*}$ w/ speed $\left|\frac{\bar{\lambda}_j - s_i}{\bar{\lambda}_{\pm j} - s_i}\right|$

- QR factorization of matrix with eigvals $r(\lambda_j) = \prod_{i=1}^{k}\frac{1}{\bar{\lambda}_j - s_i}$
- Ideally, choose $s_k \approx \lambda_n$ $\quad\begin{bmatrix}\times & \times\\ & \times\end{bmatrix}$
- Connection with shifted inverse power method, hence rational approximation

QR algorithm preprocessing

We've seen the QR iterations drives colored entries to 0 (esp. red ones)

by power. $\left(\frac{\lambda_2}{\lambda_1}\right)$

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

inv power $\left(\frac{\lambda_n}{\lambda_{n-1}}\right)$

- Hence $A_{n,n} \to \lambda_n$, so choosing $s_k = A_{n,n}$ is sensible
- This reduces #QR iterations to $O(n)$ (empirical but reliable estimate)
- But each iteration is $O(n^3)$ for QR, overall $O(n^4)$
- We next discuss a preprocessing technique to reduce to $O(n^3)$

if eig(A) fails, report to YN or MATHWORK.

QR algorithm preprocessing: Hessenberg reduction

To improve cost of QR factorisation, first reduce via orthogonal Householder transformations

not similar to A!

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}, \quad H_1 A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ 0 & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix}, \quad H_1 = I - 2\underline{v_1 v_1^T}, \; v_1 = \begin{bmatrix} 0 \\ * \\ * \\ * \\ * \end{bmatrix}$$

$$(H_1A)\begin{bmatrix} 1 & 0 \\ 0 & X \end{bmatrix} = \begin{bmatrix} (H_1A)_1 & X \end{bmatrix}$$

$$\text{Then } \overset{\sim A}{H_1 A \underset{\overset{\|}{H_1^T}}{\underline{H_1}}} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ 0 & * & * & * & * \\ & * & * & * & * \end{bmatrix}. \text{ Repeat with } H_2 = I - 2v_2v_2^T, v_2 = [0, 0, *, *, *]^T, \ldots:$$

$$\underset{I-2v_2v_2^T}{\underline{H_2}} H_1 A H_1 \underline{H_2} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & 0 & * & * & * \\ & & * & * & * \end{bmatrix}, \qquad \underbrace{\overset{X}{\overbrace{H_3H_2H_1}} \overset{A}{A} \overset{X^{-1}}{\overbrace{H_1H_2H_3}}}_{\text{orthogonal transformation}} = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix},$$

$v_2 = [0, 0, *, *, *]^T$

upper Hessenberg

$A_{ij} = 0$ if $i > j+1$

Hessenberg reduction continued

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \xrightarrow{H_1} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \xrightarrow{H_2} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix} \xrightarrow{H_3} \cdots \xrightarrow{H_{n-2}} \mathrm{eig}\Bigg(\overset{A_1}{\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}}\Bigg) = \mathrm{eig}(A).$$

- QR iterations preserve structure: if $A_1 = QR$ Hessenberg, then so is $A_2 = RQ$
- using Givens rotations, each QR iter is $O(n^2)$ (not $O(n^3)$)
- overall shifted QR algorithm cost is $O(n^3), \approx 25n^3$ flops

- Remaining task (done by shifted QR): drive subdiagonal $*$ to 0
- bottom-right $* \to \lambda_n$, can be used for shift s_k

$$G_3 G_2 G_1 A_1 = R$$

$$\Longleftrightarrow A_1 = \underbrace{G_1^T G_2^T G_3^T}_{Q} R$$

$$A_2 = RQ = \boxed{R G_1^T G_2^T G_3^T}$$

upper Hessenberg.

$$G_3 \underbrace{G_2}_{\text{Givens}} G_1 \Bigg(\underbrace{\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ & x & x & x \\ & & x & x \end{bmatrix}}_{A_1} (G_1^T G_2^T G_3^T) = G_3 G_2 \Bigg(\begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ & x & x & x \\ & & x & x \end{bmatrix} = G_3 \begin{bmatrix} x & x & x & x \\ & x & x & x \\ & & x & x \\ & & 0 & x \end{bmatrix} = \begin{bmatrix} x & x & x \\ x & x & x \\ & x & x & x \\ & & x & x \end{bmatrix} = R.$$

$G_1 G_2 G_3$

Deflation

Once bottom-right $|*| < \epsilon$,

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix} \approx \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & & * \end{bmatrix}$$

block upper triangular.

$$\mathrm{eig}\left(\begin{array}{c|cc} & n_1 & \\ n_1 & A_1 & A_{12} \\ & 0 & A_2 \end{array}\right) = \mathrm{eig}(A_1) \cup \mathrm{eig}(A_2).$$

(10^{-16})

λ_n

and continue with shifted QR on $(n-1) \times (n-1)$ block, repeat

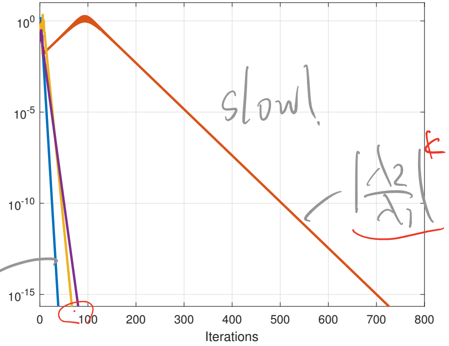
work on A_1

QR algorithm in action

Convergence of $|A_{i+1,i}|$

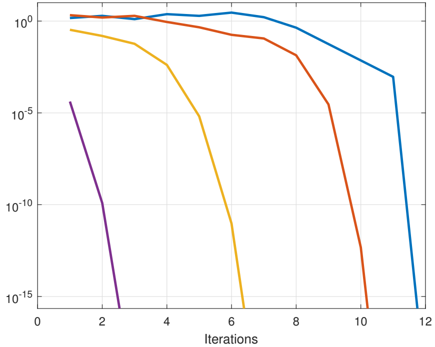
No shift (plain QR)

QR with shifts

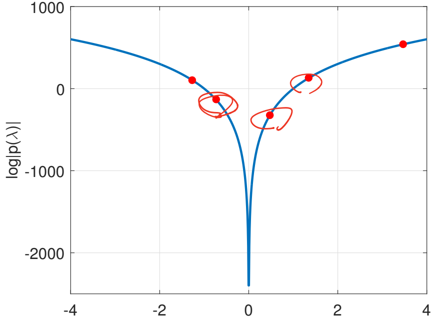


$\left|\frac{\lambda_2}{\lambda_1}\right|^k$

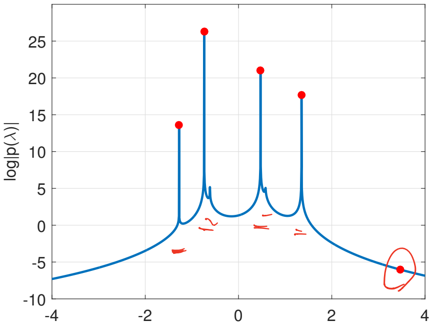
$\left|\frac{\lambda_n}{\lambda_{n-1}}\right|^k$



underlying functions (red dots: eigvals)



x^k

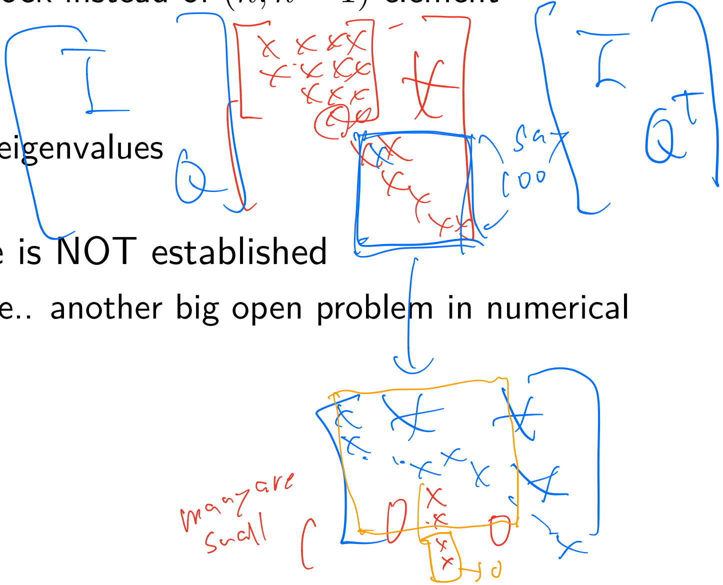


$\prod_{k=1}^{12}(x-s_k)^{-1}$

$\begin{bmatrix} x & * \\ x & * \end{bmatrix} \xrightarrow{QR} \begin{bmatrix} * & x \\ & x \end{bmatrix}$

QR algorithm: other improvements/simplifications

- Double-shift strategy for $A \in \mathbb{R}^{n\times n}$ $s \in \mathbb{C}$
  - $\underline{(A - sI)}\underline{(A - \bar{s}I)} = QR$ using only real arithmetic
- Aggressive early deflation [Braman-Byers-Mathias 2002]
  - Examine lower-right (say 100×100) block instead of $(n, n-1)$ element
  - dramatic speedup ($\approx \times 10$)
- Balancing $A \leftarrow DAD^{-1}$, D: diagonal
  - reduce $\|DAD^{-1}\|$: better-conditioned eigenvalues



- For nonsymmetric A, global convergence is NOT established
  - of course it always converges in practice.. another big open problem in numerical linear algebra

QR algorithm for symmetric A

$H_1 A H_1$ sym.

- Initial reduction to Hessenberg form → tridiagonal

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \xrightarrow{Q_1} \begin{bmatrix} * & * & & 0 & \\ * & * & * & * & * \\ 0 & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \xrightarrow{Q_2} \begin{bmatrix} * & * & & 0 & \\ * & * & * & & \\ & * & * & * & * \\ 0 & & * & * & * \\ & & * & * & * \end{bmatrix} \xrightarrow{Q_3} \begin{bmatrix} * & * & & 0 & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ 0 & & & * & * \end{bmatrix}$$

tridiagonal!

- QR steps for tridiagonal: $\underline{O(n)}$ instead of $\underline{\underline{O(n^2)}}$ per step
- Powerful alternatives available for tridiagonal eigenproblem (divide-conquer [Gu-Eisenstat 95], HODLR [Kressner-Susnjara 19],...)
- Cost: $\underline{\frac{4}{3}n^3}$ flops for eigvals, $\approx \underline{10n^3}$ for eigvecs (store Givens rotations)

nonsymmetric A: $\sim 10n^3$ for eigvals,

$\underbrace{20\text{-}30}_{\text{empirical}}\ n^3$ eigvecs

QR alg is $\underline{\text{direct}}$ method

as opposed to iterative (e.g. Krylov).

QR algorithm for symmetric A

- Initial reduction to Hessenberg form → tridiagonal

$$A=\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \overset{Q_1}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \overset{Q_2}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix} \overset{Q_3}{\rightarrow} \begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & * & \\ & & * & * & * \\ & & & * & * \end{bmatrix}$$

- QR steps for tridiagonal: $O(n)$ instead of $O(n^2)$ per step
- Powerful alternatives available for tridiagonal eigenproblem (divide-conquer [Gu-Eisenstat 95], HODLR [Kressner-Susnjara 19],...)
- Cost: $\frac{4}{3}n^3$ flops for eigvals, $\approx 10n^3$ for eigvecs (store Givens rotations)
- Self advertisement (nonexaminable): spectral divide-and-conquer (w/ Freund, Higham); which is all about rational approximation

$$A=\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \overset{V_1}{\rightarrow} \begin{bmatrix} * & * & * & & \\ * & * & * & & \\ * & * & * & & \\ & & & * & * \\ & & & * & * \end{bmatrix} \overset{V_2}{\rightarrow} \begin{bmatrix} * & & & & \\ & * & * & & \\ & * & * & & \\ & & & * & \\ & & & & * \end{bmatrix} \overset{V_3}{\rightarrow} \begin{bmatrix} * & & & & \\ & * & & & \\ & & * & & \\ & & & * & \\ & & & & * \end{bmatrix} = \Lambda.$$

Golub-Kahan for SVD

Apply Householder reflectors from left and right (different ones) to **bidiagonalize**

$$A \to B = \overbrace{H_{L,n} \cdots H_{L,1}}^{\text{orth}} A \overbrace{H_{R,1} H_{R,2} \cdots H_{R,n-2}}^{\text{orth}}$$

not similarity transform

$$A \xrightarrow{H_{L,1}} \begin{bmatrix} \star & \star & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \end{bmatrix} \xrightarrow{H_{R,1}} \begin{bmatrix} \star & \star & & \\ & \star & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \\ & \star & \star & \star \end{bmatrix} \xrightarrow{H_{L,2}} \begin{bmatrix} \star & \star & & \\ & \star & \star & \star \\ & & \star & \star \\ & & \star & \star \\ & & \star & \star \end{bmatrix} \xrightarrow{H_{R,2}} \begin{bmatrix} \star & \star & & \\ & \star & \star & \\ & & \star & \star \\ & & \star & \star \\ & & \star & \star \end{bmatrix} \xrightarrow{H_{L,3}} \begin{bmatrix} \star & \star & & \\ & \star & \star & \\ & & \star & \star \\ & & & \star \\ & & & \star \end{bmatrix} \xrightarrow{H_{L,4}} B,$$

$\sigma_i(B) = \sigma_i(A)$

$A = U\Sigma V^T$

$QAW = \underline{QU}\Sigma \underline{V^T W}$

- $\sigma_i(A) = \sigma_i(B)$
- Once bidiagonalized,
  - Mathematically, do QR alg on $B^T B$ (symmetric tridiagonal)
  - More elegant: divide-and-conquer [Gu-Eisenstat 1995] or dqds algorithm [Fernando-Parlett 1994]; nonexaminable
- Cost: $\underline{\approx 4mn^2}$ flops for singvals Σ, $\underline{\approx 20mn^2}$ flops for singvecs U, V

$B_1 = LU$ $B_2 = UL$

→ stable+fast

QZ algorithm for generalised eigenvalue problems

Generalised eigenvalue problem

$$\boxed{Ax} = \boxed{\lambda Bx}, \qquad A, B \in \mathbb{C}^{n\times n}$$

$x \neq 0$

(λ, x) n eigvals.

($\lambda = \infty$ is possible)

$$\left[\ \underset{\neq 0}{Ax} = \lambda \underset{\overset{\vee}{0}}{Bx}\ \right]$$

- A, B given, find eigenvalues λ and eigenvector x
- n eigenvalues, roots of $\det(A - \lambda B)$
- (Important case: A, B symmetric, $\underset{>0}{B}$ positive definite: λ all real)

variant of QR.

QZ algorithm: look for unitary $\underline{Q}, \underline{Z}$ s.t. $\underset{\nabla}{QAZ}, \underset{\nabla}{QBZ}$ both upper triangular

- then $\mathrm{diag}(QAZ)/\mathrm{diag}(QBZ)$ are eigenvalues
- Algorithm: first reduce A, B to Hessenberg-triangular form
- then implicitly do QR to $B^{-1}A$ (without inverting B)
- Cost: $\approx 50n^3$ QZ is Backward stable.
- See [Golub-Van Loan] for details

Tractable eigenvalue problems

- Standard eigenvalue problems $Ax = \lambda x$
  - symmetric ($4/3n^3$ flops for eigvals, $+9n^3$ for eigvecs)
  - nonsymmetric ($10n^3$ flops for eigvals, $+15n^3$ for eigvecs)
- SVD $A = U\Sigma V^*$ for $A \in \mathbb{C}^{m\times n}$: ($\frac{8}{3}mn^2$ flops for singvals, $+20mn^2$ for singvecs)
- Generalized eigenvalue problems $Ax = \lambda Bx$, $A, B \in \mathbb{C}^{n\times n}$
- Polynomial eigenvalue problems, e.g. (degree $k = 2$) $P(\lambda)x = (\lambda^2 A + \lambda B + C)x = 0$, $A, B, C \in \mathbb{C}^{n\times n}$: $\approx 20(nk)^3$

  $\det(\lambda^2 A + \lambda B + C) = 0$.

  linearisation.

  $$\operatorname{eig}\left(\begin{bmatrix} -A^{-1}B & -A^{-1}C \\ I & 0 \end{bmatrix}\right)$$

  2n

  2n
- Nonlinear problems, e.g. $N(\lambda)x = (A\exp(\lambda) + B)x = 0$
  - often solved via approximating by polynomial $N(\lambda) \approx P(\lambda)$
  - more difficult: $A(x)x = \lambda x$: eigenvector nonlinearity

Further speedup when structure present (e.g. sparse, low-rank)