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 $A = UTU^*$

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($A^T A$) to find V, then normalise AV
- there's a better algorithm: Golub-Kahan bidiagonalisation

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)$\to QR \to RQ \to \cdots$

QR algorithm for eigenproblems

Set $A_1 = A$, and

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

- 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: $A_k = Q_k R_k$, $A_{k+1} = R_k Q_k$, repeat. Claims: for $k \geq 1$,

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

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

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

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

$$A^k = AQ^{(k-1)} R^{(k-1)} = Q^{(k-1)} Q_k R_k R^{(k-1)} = Q^{(k)} R^{(k)} \square$$

QR algorithm and power method

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

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

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

In particular, consider $A^k[1, 0, \ldots, 0]^T = A^k e_n$:

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

Progress! But there is much better news

QR algorithm and inverse power method

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

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

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

Conjugate transpose: $(A^{-k})^* = Q^{(k)} (R^{(k)})^{-*}$

$\Rightarrow$ 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

- 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**

QR algorithm with shifts and shifted inverse power method

1. $A_k - s_k I = Q_k R_k$ (QR factorization)
2. $A_{k+1} = R_k Q_k + s_k I$, $\quad k \leftarrow k+1$, repeat.

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

QR algorithm with shifts and shifted inverse power method

1. $A_k - s_k I = Q_k R_k$ (QR factorization)
2. $A_{k+1} = R_k Q_k + s_k I, \quad k \leftarrow k+1$, repeat.

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

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

$$\prod_{i=1}^{k}(A - s_i I) = (A - s_k I)Q^{(k-1)} R^{(k-1)} = Q^{(k-1)} Q_k R_k R^{(k-1)} = Q^{(k)} R^{(k)}.$$

Inverse conjugate transpose: $\prod_{i=1}^{k}(A - s_i I)^{-*} = Q^{(k)}(R^{(k)})^{-*}$

- QR factorization of matrix with eigvals $r(\lambda_j) = \prod_{i=1}^{k} \frac{1}{\bar{\lambda}_j - s_i}$
- Ideally, choose $s_k \approx \lambda_n$
- 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)

$$A = \begin{bmatrix} * & * & * & * & * \\ \color{blue}{*} & * & * & * & * \\ \color{blue}{*} & * & * & * & * \\ \color{blue}{*} & * & * & * & * \\ \color{red}{*} & \color{red}{*} & \color{red}{*} & \color{red}{*} & * \end{bmatrix}$$

- 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 $O(n^3)$

QR algorithm preprocessing: Hessenberg reduction

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

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

Then $H_1 A H_1 = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix}$. Repeat with $H_2 = I - 2v_2 v_2^T, v_2 = [0, 0, *, *, *]^T$, ...:

$$H_2 H_1 A H_1 H_2 = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix}, \qquad H_3 H_2 H_1 A H_1 H_2 H_3 = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix},$$

Hessenberg reduction continued

$$A = \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} \overset{H_1}{\rightarrow} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{bmatrix} \overset{H_2}{\rightarrow} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{bmatrix} \overset{H_3}{\rightarrow} \cdots \overset{H_{n-2}}{\rightarrow} \begin{bmatrix} * & * & * & * & * \\ \color{red}{*} & * & * & * & * \\ & \color{red}{*} & * & * & * \\ & & \color{red}{*} & * & * \\ & & & \color{red}{*} & * \end{bmatrix}.$$

- 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 $\color{red}{*}$ to 0
- bottom-right $* \rightarrow \lambda_n$, can be used for shift s_k

Deflation

Once bottom-right $|{\color{red}*}| < \epsilon$,

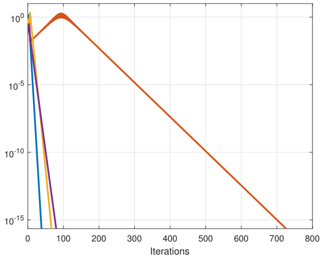
$$\begin{bmatrix} * & * & * & * & * \\ {\color{red}*} & * & * & * & * \\ & {\color{red}*} & * & * & * \\ & & {\color{red}*} & * & * \\ & & & {\color{red}*} & * \end{bmatrix} \approx \begin{bmatrix} * & * & * & * & * \\ {\color{red}*} & * & * & * & * \\ & {\color{red}*} & * & * & * \\ & & {\color{red}*}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

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

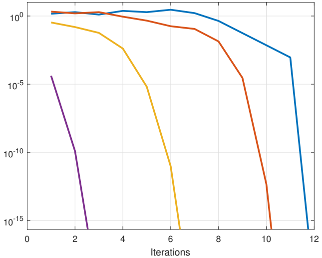
QR algorithm in action

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

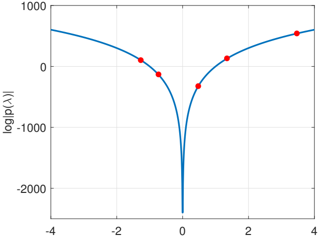
No shift (plain QR)

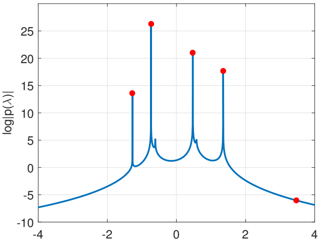


QR with shifts



underlying functions (red dots: eigvals)





QR algorithm: other improvements/simplifications

- Double-shift strategy for $A \in \mathbb{R}^{n\times n}$
  - $(A - sI)(A - \bar{s}I) = QR$ 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

- Initial reduction to Hessenberg form $\rightarrow$ 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)

QR algorithm for symmetric A

- Initial reduction to Hessenberg form $\rightarrow$ 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**

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

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

- $\sigma_i(A) = \sigma_i(B)$
- 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: $\approx 4mn^2$ flops for singvals Σ, $\approx 20mn^2$ flops for singvecs U, V

QZ algorithm for generalised eigenvalue problems

Generalised eigenvalue problem

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

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

QZ algorithm: look for unitary Q, Z s.t. QAZ, 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: $\approx 50n^3$
- 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 $k = 2$) $P(\lambda)x = (\lambda^2 A + \lambda B + C)x = 0$, $A, B, C \in \mathbb{C}^{n\times n}$:$\approx 20(nk)^3$
- 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)