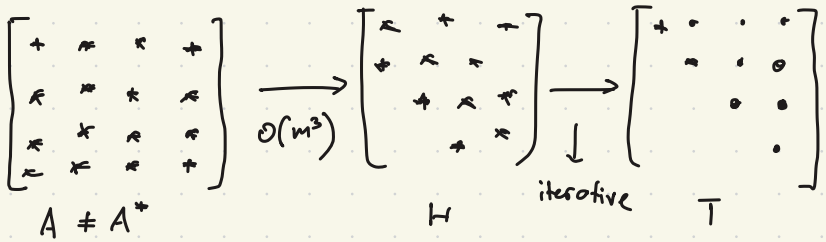
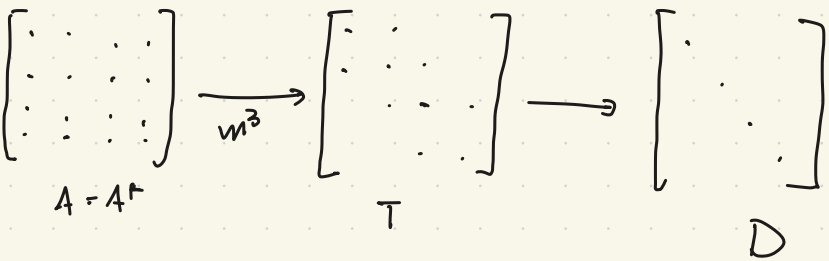
Eigenvalue Factorisation

We simply take matrices to their schur upper triangular form.



If A is hermitian:



We can go from (1) $\Rightarrow$ (3), but the in iteration, each step takes $O(m^3)$. (2) $\Rightarrow$ (3) takes $O(m^2)$. So if we iterate for $O(m)$, then (1) $\Rightarrow$ (3) is $O(m^4)$, but (2) $\Rightarrow$ (3) is $O(m^3)$

- In practice we iterate for $O(m)$

Method:

Apply Q_1^* s.t $Q_1^* A = \begin{bmatrix} * & * & * & * \\ \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \\ 0 & \cdot & \cdot & \cdot \end{bmatrix}$ *: unchanged, ·: changed

So just the householder reflection on A. We apply it to rows $2, \dots, m$, s.t 1st entry below 2 is 0.

Then doing this $m-2$ times, we get:

$$H = \begin{bmatrix} \cdot & \cdot & & & \\ \cdot & \cdot & \cdot & & \\ & \cdot & \cdot & \cdot & \\ & & \cdot & \cdot & \\ & & & \cdot & \cdot \end{bmatrix} = Q_{m-2}^* \cdots Q_2^* Q_1^* A Q_1 Q_2 \cdots Q_{m-2}$$

Rayleigh Quotient:

$$r(x) = \frac{x^T A x}{x^T x} \qquad (\mathbb{R}^m \to \mathbb{R})$$

$$\nabla r(x) = \frac{2}{x^T x}\left(Ax - r(x)x\right)$$

- So $\nabla r(x) = 0$ and $x \neq 0 \iff x$ is an e.w of A corresp to e.v $r(x)$

$$\Rightarrow r(x) - r(q_j) = O(\|x - q_j\|^2) \text{ as } x \to q_j \quad (*)$$

(q_j: j^{th} e.w)

Algorithm 27.1. Power Iteration

$v^{(0)} =$ some vector with $\|v^{(0)}\| = 1$
for $k = 1, 2, \ldots$

$w = Av^{(k-1)}$	apply A
$v^{(k)} = w/\|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$	Rayleigh quotient

Recall: Inverse Iteration (we have a prior for μ)

Algorithm 27.2. Inverse Iteration

$v^{(0)} =$ some vector with $\|v^{(0)}\| = 1$
for $k = 1, 2, \ldots$

Solve $(A - \mu I)w = v^{(k-1)}$ for w	apply $(A - \mu I)^{-1}$
$v^{(k)} = w/\|w\|$	normalize
$\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$	Rayleigh quotient

How to find eigenvalues:

Non-hermitian:

- Transform to upper Heisenberg
  → Householders, $O(m^3)$
- Shifted QR iteration on this matrix
  → Schur form, gives the eigenvalues

Wilkinson shifts used, because standard shifts can fail.

μ (for shifts) $= q_m^{(k)T} A q_m^{(k)}$

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

So $\mu = A_{mm}$. But what if $\mu = 0$?

Then this does not work

→ Use wilkinson shifts to break ties

⇒ let last 2x2 be:

$$B = \begin{bmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{bmatrix}.$$

$$\mu = a_m - \frac{\text{sign}(\delta) b_{m-1}^2}{\sqrt{|\delta| + b_{m-1}^2}} \qquad \delta = \frac{a_m - a_{m-1}}{2}$$

↳ sign(δ) = arbitrary when $\delta = 0$

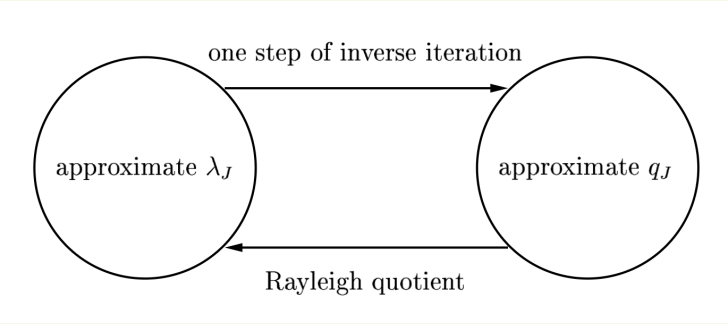
If $a_m = a_{m-1}$, then $\mu = a_m \pm |b_{m-1}|$

WLOG if $a_m >> a_{m-1}$ and $|\delta| >> b_{m-1}^2$:

$$\mu \approx a_m$$

Cubic Convergence?

- We combine inverse iteration with Rayleigh's quotient



Algorithm:

$v^{(0)}$ = unit $v \in V$

$\lambda^{(0)} = (v^{(0)})^T A v^{(0)}$ (Rayleigh)

for $k = 1, 2, \ldots$ (until cvg)

- Solve for w.
  - $(A - \lambda^{(k-1)} I) w = v^{(k-1)}$
- $v^{(k)} = w / \|w\|$
- $\lambda^{(k)} = v^{(k)T} A v^{(k)}$

Now, when $v^{(k)} \approx q_j$ (using (*))

$$\|v^{(k)} - q_j\| \leq \varepsilon \Rightarrow |\lambda^{(k)} - \lambda_j| \leq \varepsilon^2$$

Now, from inverse iteration:

$$\begin{aligned} \|v^{(k+1)} - q_j\| &= O(|\lambda^{(k)} - \lambda_j| \|v^{(k)} - q_j\|) \\ &= O(\varepsilon^3) \end{aligned}$$

So combining Rayleigh's and Inverse iteration:

- Cubic convergence!

Solving SVD:

We can go from $A \to A^*A$ (hermitian) and solve for e.v using methods above.

We know:

$$|s_i(A+\tilde{A}) - s_i(A)| \leq \|\tilde{A}\|$$

i.e if $\frac{\|\tilde{A}\|}{\|A\|} = O(\varepsilon)$, then $\tilde{s}_i = s_i(A+\tilde{A})$

$$|\tilde{s}_i(A) - s_i(A)| = O(\varepsilon \|A\|)$$

For hermitian $A, \tilde{A}$: ($\lambda_i^{\downarrow}$, i^{th} largest e.v)

$$|\lambda_i^{\downarrow}(A+\tilde{A}) - \lambda_i^{\downarrow}(A)| \leq \|\tilde{A}\|$$

So by using A^*A:

$$|\lambda_i^{\downarrow}(A^*A + \tilde{A}) - \lambda_i^{\downarrow}(A^*A)| \leq \|\tilde{A}\|$$

So $|\tilde{\lambda}_i^{\downarrow}(A^*A) - \lambda_i^{\downarrow}(A^*A)| = O(\varepsilon \|A^*A\|)$

$= O(\varepsilon \|A\|^2)$

Now using square roots:

$$|\tilde{s}_i - s_i| = O(|\tilde{\lambda}_i - \lambda_i| / \sqrt{\lambda_i}) = O(\varepsilon \|A\|^2 / s_i)$$

So using this method, we are infact very bad when we have $s_i \ll \|A\|$, because this gives us huge error.

Intuition for Actual Method:

Compute $H = \begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix}$. E.v of H are $\pm s_i(A)$

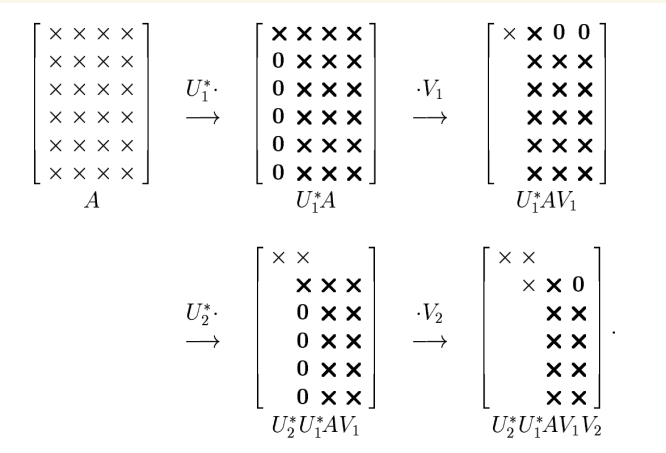
This keeps us stable.

- However we never explicitly use an $m+n \times m+n$ matrix.

Technique: Let $A \in \mathbb{R}^{m \times n}$, $m \geq n$

$$A \xrightarrow[O(nm^2)]{} \text{Bidiagonal} \xrightarrow[\substack{O(m) \\ \text{each step}}]{\text{iterative}} \text{Diagonal}$$

Golub-Kahan Bidiagonalisation:



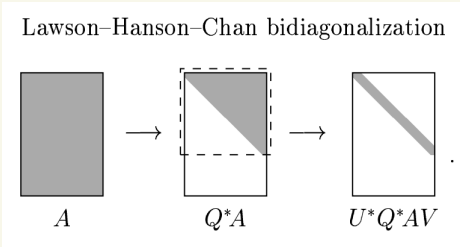
- Unlike the e.v problem, we can use different unitary matrices on the left and right.

  m matrices on the left, n-2 on the right.

$$\sim 4mn^2 - \frac{4}{3}n^3 \text{ flops}$$

But: what if $m \gg n$? This is too expensive

LHC (Lawson, Hanson, Chan) Method:



$A = QR$

$R = Q^*A$

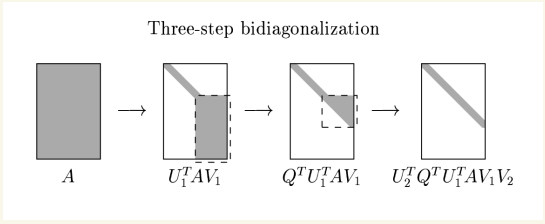
QR on A requires $2mn^2 - \frac{2}{3}n^3$ flops.

We only need to bidiagonalise $n \times n$ matrix R.

$$B = U^*RV = U^*Q^*AV$$

$\sim 2mn^2 + 2n^3$ flops total

Cheaper when $m > \frac{5}{3}n$



Now we have this bidiagonal matrix. But how do we find singular values?

We note that B^TB will be tridiagonal.

→ Can do QR. But again, same issue as A^TA (loss of precision)

So thus, can we solve somehow without B^TB?

Defn: Properly Bidiagonal

let $B = \begin{bmatrix} \beta_1 & \gamma_1 & & \\ & \beta_2 & \ddots & \\ & & \ddots & \gamma_{m-1} \\ & & & \beta_m \end{bmatrix}$. B properly diagonal if $\beta_i \neq 0$ and $\gamma_i \neq 0$ $\forall i$

B properly bidiagonal $\Leftrightarrow$ B^TB and BB^T properly tridiagonal

Note: If B not properly bidiagonal, we can solve smaller problems:

$$B = \begin{bmatrix} B_1 & 0 \\ 0 & B_2 \end{bmatrix}$$

Say we do QR on B^TB and BB^T:

$$B^TB - \rho I = \hat{Q}R$$

$$BB^T - \rho I = \hat{P}S$$

Let $\hat{B} = \hat{P}^T B \hat{Q}$. (1)

Then $\hat{B}^T\hat{B} = \hat{Q}^T B^T B \hat{Q} = \hat{Q}^T(\hat{Q}R + \rho I)\hat{Q}$

$= R\hat{Q} + \rho I$

and similarly: $\hat{B}\hat{B}^T = S\hat{P} + \rho I$

So doing (1) in fact is implicitly shifting

BB^T and B^TB!

- How do we get $\hat{P}$ and $\hat{Q}$?
- Don't want to use BB^T and B^TB

$$B = \begin{bmatrix} a_1 & b_1 & & & \\ & a_2 & b_2 & & \\ & & \ddots & \ddots & \\ & & & b_{n-1} & \\ & & & a_n & \end{bmatrix}$$

$$[B^T B]_{ii} = a_i^2 + b_i^2 \qquad (b_n := 0)$$

$$[B^T B]_{i,i+1} = a_{i+1} b_i$$

Let's look at first 2x2 block of $B^T B$:

$$B^T B - \rho I \qquad \begin{bmatrix} a_1^2 + b_1^2 - \rho & a_2 b_1 \\ a_2 b_1 & a_2^2 + b_2^2 - \rho \end{bmatrix}$$

Doing QR on this, we want G orthogonal

s.t $$G^T \begin{bmatrix} a_1^2 + b_1^2 - \rho \\ a_2 b_1 \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}$$

→ Cannot use householders as that operates on ALL rows below: our tridiagonal is gone.

So G must be I everywhere, but the first 2x2

→ All we thus do is rotate. G has to be a rotation matrix.

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

Now, we basically want to solve s, c so that $G^T v = \begin{bmatrix} * \\ 0 \end{bmatrix}$

We know a_1, a_2, b_1, and ρ so we dont ever use B^TB!

Call this rotation matrix Q_1.

$$Q_1 = \begin{bmatrix} c & s & & \\ -s & c & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix}, \quad B = \begin{bmatrix} a_1 & b_1 & & \\ & a_2 & \ddots & \\ & & \ddots & b_{n-1} \\ & & & a_n \end{bmatrix}$$

Then $$BQ_1 = \left[cB_1 - sB_2 \mid sB_1 + cB_2 \mid B_3 \mid \cdots \mid B_n \right]$$

$$= \begin{bmatrix} ca_1 - sb_1 & sa_1 + cb_1 & & & \\ -sa_2 & ca_2 & b_2 & & \\ \uparrow & & a_3 & \ddots & \\ \text{"bulge"} & & & & b_{n-1} \\ & & & & a_n \end{bmatrix}$$

Now we choose another rotation P_1 that just operates on rows 1 and 2 s.t the bulge is gone $\left(P_1 \begin{pmatrix} \cdot \\ \cdot \end{pmatrix} = \begin{pmatrix} \cdot \\ 0 \end{pmatrix}\right)$

$$P_1^T B Q_1 = \begin{bmatrix} * & * & & & \\ * & * & & & \\ & * & * & & \\ & & & \ddots & * \\ & & & & * \end{bmatrix}$$

Now, we apply Q_2 on the right that is again a 2×2 iteration, which does the same B^TB logic. Only operates on columns 2 and 3.

$$\Rightarrow P_1^T B Q_1 Q_2 = \begin{bmatrix} * & * & & & \\ * & * & & & \\ & * & * & & \\ & & \ddots & * & \\ & & & * & \end{bmatrix}$$

Now get P_2.

This: $B^{(1)} = \hat{P}^T B \hat{Q}$, where

$$\hat{P} = P_1 P_2 \cdots P_{n-1}$$

$$\hat{Q} = Q_1 Q_2 \cdots Q_{n-1}$$

This is one QR step. We iterate on this until convergence.