Welcome to M221! (Mar 31)

Congratulations Jack Dongarra!
for winning the Turing Award!

Overview of Algorithms

(1) "Usual Accuracy"

backward stable: exact evals + evces of $A+E$, $\|E\| = O(\varepsilon)$, $\|A\|$

(1.1) all evals (with or without evces)
(1.2) just evals in $[x, y]$ (w or w/o evces)
(1.3) just evals $\lambda_1, \ldots, \lambda_j$

Eq. $\lambda_1 \cdots \lambda_j \Rightarrow 10$ largest evals (w or w/o evces)

(1.2) and (1.3) cheaper than (1.1)
when only few evals/evces desired

(2) "High Accuracy": get tiny evals with more leading digits

Ex: if $A$ well conditioned
i.e. all sing vals $\sim$ same magnitude
then usual accuracy $\Rightarrow$ errorbound
is $\pm O(\varepsilon)$, $\|A\| \Rightarrow$ all computed with correct leading digits
Consider \( B = D \cdot A \) \( D = \text{diag}(d_1, \ldots, d_n) \)
where some \( d_i \gg \text{other } d_j \)
\( \Rightarrow \) some \( \sigma_i \gg \text{other } \sigma_j \)
\( \Rightarrow \) usual accuracy \( \Rightarrow \) no leading correct digits in small \( \sigma_i \)

\( \Rightarrow \) perturbation theory and alg.

to get all \( \sigma_i \) with correct leading digits

Other scaling too: DAD \( A = A^T \)
(see links in class web page for details)

(3) Updating: given evals and evects of \( A \), compute them for \( A \pm xx^T \)
more cheaply than starting from scratch
(basis of fast alg for all evals/evects)

All these options apply to \( A = A^T \) and SVD

Algorithms and costs

(1) Reduce \( A = QTQ^T \)
\( T = T^T \) tridiagonal \( QTQ^T = I \)
Same idea as for Hessenberg reduction in Chap 4: \( A = Q H Q^T \) \( H = \begin{bmatrix} * \end{bmatrix} \)

if \( A = A^T \implies H = H^T \implies \) tridiagonal (Lapack: ssytrd)

all subsequent algorithms work on \( T \)

There is an alg for tridiagonal reduction that does \( O\left( \frac{n^3}{\sqrt{n \text{ fast mem size}}} \right) \)

words moved fast -> slow memory

if \( A \) banded \( \begin{bmatrix} * & * & & \cdots \end{bmatrix} \)

cost drops from \( O(n^3) \) to \( O(n^2 \cdot \text{bw}) \)

- SVD analogous : Reduce \( A \) to bidiagonal form \( B = U AV^T \)

\( U, V \) orthogonal

\( B = \begin{bmatrix} * & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \), subsequent alg's work on \( B \)
(1.1) Given \( T \), find all evals
(w or w/o evacs): many algs
all cost \( O(n^2) \) for all evals alone
If evacs too, costs range from
\( O(n^2) \) to \( O(n^3) \), varying
numerical stability

(1.1.1) Oldest is QR iteration
as in Chap 4

Thm (Wilkinson) With right shift, tridiagonal QR is
globally convergent, usually cubic (# correct digits triples
at each step)

Cost: \( O(n^2) \) for evals alone,
but \( O(n^3) \) for evacs, more
expensive than later algs
(CLAPACK: ssyev)

CLAPACK sgesvd uses a variant
of QR with additional property
that all sing values have correct
leading digits, no matter how small
(see notes)
(1.1.2) Improve cost of $O(n^3)$ for evecs to $O(n^2)$, but not guarantee evecs are orthogonal (Are $\mathbf{x}$ and $\mathbf{x}_i$ orthogonal? If $\mathbf{x}$ too close to $\mathbf{x}_i$ then $\mathbf{x}_i$ may not be ortho to $\mathbf{x}_j$)

1. Compute evals $O(n^2)$
2. Compute each evvec using inverse iteration:
   $$\mathbf{x}_i = (T - \lambda I)^{-1} \mathbf{x}_i$$

$\mathbf{x}_i$ converges to evvec of largest eval of $(T - \lambda I)^{-1}$

i.e. corresponds to eval of $T$ closest to $\lambda$

(Power iteration from Chap 4 applied to $(T - \lambda I)^{-1}$)

If $\lambda$ and $\lambda'$ are very close, no guarantee that their evvecs are orthogonal

(Worst case: $\lambda$ and $\lambda'$ round to same floating point number)

Long goal: find orthogonal evvecs for $O(n^2)$ - solved by MRRR
(1.13) Divide and Conquer (LAPACK):
(Cuppen, Gru Bisection) ssyevd

faster than QR, not as fast as inverse iteration, guarantees orthogonal evcs: cost \( O(n^3) \), \( 2 \leq g < 3 \)
same idea used for updating
\[ \text{eig} \left( A \pm vv^T \right) \]

(1.14) MRRR = Multiple Relatively Robust Representations

like inverse iteration but guarantees orthogonal evcs (Parlett, Dhillon)
(see web page)
(LAPACK: ssyevr)

Extension to SVD by Paul Willems
but examples where sing evcs not fully orthogonal, \( \exists \)
not in LAPACK, open problem

Beating \( O(n^2) \) based on divide+conquer
\[ T = Z A Z^T \]

Thm (Gv) One can compute evcs \( Z \) in
\( O(n \cdot \log^2 n) \) (p small integer)
if represented implicitly
(can multiply \( Z \cdot x \) cheaply)
but all evacs would cost $O(n^3)$

to get $A = QTQ^T = (QZ)T \Lambda (Z^TQ^T)$

evacs of $A$ would cost $O(n^3)$

(1.2) or (1.3): few evacs and evacs

Reduce $A = Q^T T Q$ as before,

use bisection to find few desired evacs of $T$ (compute $T-xI = LDL^T$

to count # evacs of $<x = \pm \text{Diag}$, use bisection to find evacs in $[x, y]$)

Given evacs, use inverse iteration to get evacs, or use MRRR

(2) High Accuracy: Based on

Jacobi's method: (historically oldest): LAPACK (SVD only) sgesvdx

(3) Updating $A = Q \Lambda Q^T$ to $A \pm vv^T$

will use later for divide + conquer

---

QR iteration

(matlab demo for cubic convergence)

code in typemnotes
Cubic convergence follows from analysis of Rayleigh Quotient Iteration

\[ i = 0 \]

choose unit vector \( x_0 \)

repeat

\[ s_i = p(x_i, A) = x_i^T A x_i \]

\[ y = (A - s_i I)^{-1} x_i \]

\[ x_{i+1} = y / \| y \|^2 \]

\[ i = i + 1 \]

until convergence

\( (s_i \text{ or } x_i \text{ stop changing}) \)

Inverse iteration using Rayleigh Quotient as shift, best available approx eval given approx evoc \( x_i \)

Suppose \( A g = \lambda g \) , \( \| g \|_2 = 1 \) \( \| x_i - g \|_2 = e^{-k} \)

Want to show \( \| x_{i+1} - g \|_2 = O(e^3) \)

Bound \( | s_i - \lambda | \)

\[ s_i = x_i^T A x_i = (x_i - g + g)^T A (x_i - g + g) \]

\[ = (x_i - g)^T A (x_i - g) + g^T A (x_i - g) \]

\[ + (x_i - g)^T A g + g^T A g \]

\[ \frac{d g}{\lambda} \]

\[ \frac{1}{\lambda} \]
\[ s_c - \lambda = (x_c - q)^T A (x_c - q) + 2A (x_c - q)^T q \]
\[ |s_c - \lambda| \leq O(\|x_c - q\|_2^2) + O(\|x_c - q\|_2) \]

want tighter bound

\[ |s_c - \lambda| \leq \| A x_c - s_q x_i \|_2^2 / \text{gap} \]

... gap = distance from \( s_i \) to second closest eval

... Thm 5.5 in book, sketched last time

\[ = \| A (x_c - q + q) - s_c (x_i - q + q) \|_2^2 / \text{gap} \]
\[ = \| (A - s_c I) (x_i - q) + (A - s_c) q_c \|_2^2 / \text{gap} \]
\[ \leq (\| (A - s_c I) (x_i - q) \|_2 + \| (A - s_c) q_c \|_2)^2 / \text{gap} \]
\[ = O(\varepsilon) / \text{gap} \]

Use analysis from Chap 4 of one step of inverse iteration:

\[ \| x_{i+1} - q \|_2 \leq \| x_i - q \|_2 \cdot \| s_i - \lambda \| / \text{gap} \]
\[ \leq O(\varepsilon) / O(\varepsilon^2) \]

= \( O(\varepsilon^3) \) if gap not too small

Show that QR iteration doing Rayleigh Quotient iteration implicitly
\[ T - s_i \mathbf{1} = QR \rightarrow \text{new } T = RQ + s_i \mathbf{1} \]

\[(T - s_i \mathbf{1})^{-1} = R^{-1} Q^{-1} \]

\[= R^{-1} Q^\top \quad \text{symmetric} \]

\[= (R^{-1} Q^\top)^\top \]

\[= QR^{-1} \]

\[(T - s_i \mathbf{1})^{-1} R^\top = Q \]

last column: \((T - s_i \mathbf{1})^{-1} e_n \mathbf{R}_{nn} = \text{last col of } Q \)

\[s_i = T(n,n) = e_n^\top T e_n = p(e_n, T) \]

\[\Rightarrow q_n = \text{last col of } Q = \text{result of step of Rayleigh quotient iteration} \]

starting with \(e_n\)

\[\text{new } T = RQ + s_i \mathbf{1} = Q^\top T Q \]

\[(\text{new } T)(n,n) = q_n^\top T q_n = p(q_n, T) \]

\[\Rightarrow \text{QR iteration doing Rayleigh Quotient iteration} \]

Actual implementation uses “bogus chasing” as in Chap 4, cost \(O(n)\) per iteration since \(T\) tridiagonal

\[\Rightarrow \text{cost } = O(n) \text{ per eval} \]

\[\text{cost } = O(n^2) \text{ for all evals} \]

But cost to compute evens by taking products of all Givens rotations \(= O(n^3)\), want to be faster
Next algorithm to beat $O(n^3)$:

Divide + Conquer:
used for computing all evals + evecs

Main ingredient: cheaply updating $A = Q \Lambda Q^T$ for $A + \alpha uv^T$

$$A + \alpha uv^T = Q \Lambda Q^T + \alpha uv^T$$
$$= Q \left( \Lambda + \alpha (v^T v) I \right) Q^T$$
$$= Q \left( \Lambda + \alpha v \cdot v^T \right) Q^T$$

need evals + evecs of $\Lambda + \alpha v v^T$
= diagonal + rank-1

use characteristic polynomial

Lemma (HW5.14) $\det(I + xy^T) = 1 + y^T x$

$$\det \left( \Lambda + \alpha vv^T - \lambda I \right)$$
$$= \det \left( \left( \Lambda - \lambda I \right) \left( I + \alpha \left( \Lambda - \lambda I \right)^{-1} v v^T \right) \right)$$
$$= \prod_{\lambda_i} \left( 1 + \alpha \left( \frac{v_i^2}{\lambda_i - \lambda} \right) \right)^{\text{rank} 1}$$

$f(\lambda) = \text{find roots}$

$$f(\lambda) = 1 + \frac{0.5}{1-\lambda} + \frac{0.5}{2-\lambda} + \frac{0.5}{3-\lambda} + \frac{0.5}{4-\lambda}$$

Fig 5.2 in text

$$f(\lambda) = 1 + \frac{0.001}{1-\lambda} + \frac{0.001}{2-\lambda} + \frac{0.001}{3-\lambda} + \frac{0.001}{4-\lambda}$$

Fig 5.3 in text
can't use plain Newton to solve $f(u) = 0$