Welcome to Ma221! (Mar 31)

Congratulations Jack Dongarra!
for winning the Turing Award!

Overview of Algorithms

(1) "Usual Accuracy"
backward stable: exact evals + evecs
of $A+E$, $\|E\| = O(\varepsilon)\cdot\|A\|$

(1.1) all evals (with or without evecs)
(1.2) just evals in $[x, y]$ (w or w/o evecs)
(1.3) just evals $\lambda_i, \dots, \lambda_j$
Eg $\lambda_1 \cdots \lambda_{10} \Rightarrow 10$ largest evals
(w or w/o evecs)

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

(2) "High Accuracy": get tiny evals
with more leading digits
Ex: if A well conditioned
i.e. all sing vals ~ same magnitude
then usual accuracy $\Rightarrow$ error bound
is $\pm O(\varepsilon)\cdot\|A\| \Rightarrow$ all computed
with correct leading digits

Consider $B = D \cdot A$ $\quad D = \mathrm{diag}(d_1, \dots, d_n)$

where some $d_i \gg$ other d_j

$\Rightarrow$ some $\sigma_i \gg$ other σ_j

$\Rightarrow$ usual accuracy $\Rightarrow$ no leading correct digits in small σ_j

$\exists$ perturbation theory and algs to get all σ_i with correct leading digits

other scaling too : DAD for $A = A^T$

(see links in class webpage for details)

(3) Updating: given evals and evecs of A, compute them for $A \pm xx^T$ more cheaply than starting from scratch

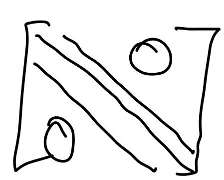
(basis of fast alg for all evals/evecs)

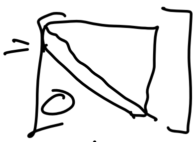
All these options apply to $A = A^T$ and SVD

Algorithms and costs

(1) Reduce $A = QTQ^T$ $\quad QQ^T = I$

$T = T^T$ tridiagonal



Same idea as for Hessenberg reduction in Chap 4: $A = QHQ^T$ $H =$ 

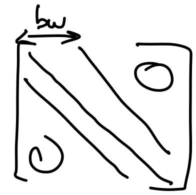
if $A = A^T \Rightarrow H = H^T \Rightarrow H$ tridiagonal

(Lapack: ssytrd)

all subsequent algorithms work on T

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

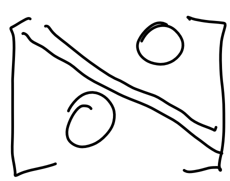
words moved fast $\leftrightarrow$ slow memory

if A banded 

cost drops from $O(n^3)$ to $O(n^2 \cdot bw)$

- SVD analogous: Reduce A to bidiagonal form $B = UAV^T$

U, V orthogonal

$B =$  , subsequent algs work on B

(1.1) Given T, find all evals
(w or w/o evecs): many algs
all cost $O(n^2)$ for all evals alone
If evecs 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 evecs, more
expensive than later algs
(CLAPACK: ssyev)

LAPACK 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 ($Ax \approx \lambda x$ for sure but x_i may not be orth to x_j if λ_i too ~~close~~ close to λ_j)

(1) compute evals $O(n^2)$

(2) compute each evec using inverse iteration:

$$x_{i+1} = (T - \lambda I)^{-1} x_i$$

x_i converges to evec of largest eval of $(T-\lambda I)^{-1}$

i.e. corresponds to eval of T closest to λ

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

if λ and λ' are very close, no guarantee that their evecs are orthogonal

(worst case: λ and λ' round to same floating point number)

Long Goal: find orthogonal evecs for $O(n^2)$ - solved by MRRR

(1.1.3) Divide and Conquer (CLAPACK: ssyevd)
(Cuppen, Gu + Eisenstat)

faster than QR, not as fast as inverse iteration, guarantees orthogonal evecs: cost $O(n^g)$, $2<g<3$

same idea used for updating
$\mathrm{eig}(A \pm vv^T)$

(1.1.4) MRRR = Multiple Relatively Robust Representations

like inverse iteration but guarantees orthog evecs (Parlett, Dhillon)
(see web page)
CLAPACK: ssyevr)

Extension to SVD by Paul Willems
but examples where sing vecs not fully orthogonal, ⇒
not in LAPACK, open problem

Beating $O(n^2)$ based on divide + conquer
$T = Z \Lambda Z^T$

Thm (Gu) One can compute evecs Z in $O(n \cdot \log^p n)$ (p small integer)
if represented implicitly
(can multiply $Z \cdot x$ cheaply)

but all evecs would cost $O(n^3)$
to get $A = QTQ^T = (QZ)\,\Lambda\,(Z^TQ^T)$

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

(1.2) or (1.3): few evals and evecs
Reduce $A = Q^T T Q$ as before,
use bisection to find few desired
evals of T (compute $T - xI = LDL^T$
to count # evals of $< x$ = # $D_{ii} < 0$,
use bisection to find evals in $[x, y]$)

Given evals, use inverse iteration
to get evecs, or use MRRR

(2) High Accuracy! Based on
Jacobi's method: (historically
oldest): LAPACK (SVD only) sgesvj

(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 typednotes

Cubic convergence follows from analysis of Rayleigh Quotient Iteration

$i = 0$
choose unit vector x_0
repeat
$s_i = \rho(x_i, A) = x_i^T A x_i$
$y = (A - s_i I)^{-1} x_i$
$x_{i+1} = y / \|y\|_2$
$i = i+1$
until convergence
(s_i or x_i stop changing)

Inverse iteration using Rayleigh Quotient as shift, best available approx eval given approx evec x_i

Suppose $Aq = \lambda q$, $\|q\|_2 = 1$ $\|x_i - q\|_2 = e \ll 1$
Want to show $\|x_{i+1} - q\|_2 = O(e^3)$

Bound $|s_i - \lambda|$

$$s_i = x_i^T A x_i = (x_i - q + q)^T A (x_i - q + q)$$
$$= (x_i - q)^T A (x_i - q) + \underbrace{q^T A}_{\lambda q^T} (x_i - q)$$
$$+ (x_i - q)^T \underbrace{Aq}_{\lambda q} + \underbrace{q^T A q}_{\lambda}$$

$$s_i - \lambda = (x_i - q)^T A (x_i - q) + 2\lambda (x_i - q)^T q$$

$$|s_i - \lambda| \le O(\|x_i - q\|_2^2) + O(\|x_i - q\|_2)$$
$$= O(\|x_i - q\|_2) = O(e)$$

want tighter bound

$$|s_i - \lambda| \le \|A x_i - s_i x_i\|_2^2 / gap$$

... gap = distance from s_i to second closest eval

... Thm 5.5 in book, sketched last time

$$= \|A(x_i - q + q) - s_i(x_i - q + q)\|_2^2 / gap$$
$$= \|(A - s_i I)(x_i - q) + (\lambda - s_i) q_i\|_2^2 / gap$$
$$\le (\|(A - s_i I)\underbrace{(x_i - q)}_{O(e)}\|_2 + \|\underbrace{(\lambda - s_i)}_{O(e)} \cdot q_i\|_2)^2 / gap$$
$$= O(e^2) / gap$$

Use analysis from Chap 4 of one step of inverse iteration

$$\|x_{i+1} - q\|_2 \le \underbrace{\|x_i - q\|_2}_{O(e)} \cdot \underbrace{|s_i - \lambda|}_{O(e^2)} / gap$$
$$= O(e^3) \text{ if gap not too small}$$

Show that QR iteration doing Rayleigh Quotient iteration implicitly

$T - s_i I = QR \rightarrow \text{new}T = RQ + s_i I$

$(T - s_i I)^{-1} = R^{-1} Q^{-1}$

$= R^{-1} Q^T$ symmetric

$= (R^{-1} Q^T)^T$

$= Q R^{-T}$

$(T - s_i I)^{-1} R^T = Q$

last column: $(T - s_i I)^{-1} e_n R_{nn}$ = last col of Q

$s_i = T(n,n) = e_n^T T e_n = \rho(e_n, T)$

$\Rightarrow q_n$ = last col of Q = result of step of Rayleigh quotient iteration starting with e_n

$\text{new}T = RQ + s_i I = Q^T T Q$

$(\text{new}T)(n,n) = q_n^T T q_n = \rho(q_n, T)$

$\Rightarrow$ QR iteration doing Rayleigh Quotient iteration

Actual implementation uses "bulge chasing" as in Chap 4, cost $O(n)$ per iteration since T tridiagonal

$\Rightarrow$ cost $= O(n)$ per eval

cost $= O(n^2)$ for all evals

But cost to compute evecs 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 $A + \alpha u u^T$

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

need evals + evecs of $\Lambda + \alpha v v^T$

= diagonal + rank-1

use characteristic polynomial

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

$$\det(\Lambda + \alpha v v^T - \lambda I)$$
$$= \det\left((\Lambda - \lambda I)(I + \alpha \underbrace{(\Lambda - \lambda I)^{-1} v v^T}_{\text{rank 1}})\right)$$
$$= \prod_{i=1}^{n} (\lambda - \lambda_i) \cdot \underbrace{\left(1 + \alpha \sum_{i=1}^{n} \frac{v_i^2}{\lambda_i - \lambda}\right)}_{f(\lambda)\text{ : find roots}}$$

$$f(\lambda) = 1 + \frac{.5}{1-\lambda} + \frac{.5}{2-\lambda} + \frac{.5}{3-\lambda} + \frac{.5}{4-\lambda}$$

Fig 5.2 in text

$$f(\lambda) = 1 + \frac{.001}{1-\lambda} + \frac{.001}{2-\lambda} + \frac{.001}{3-\lambda} + \frac{.001}{4-\lambda}$$

Fig 5.3 in text

can't use plain Newton to solve $f(\lambda)=0$