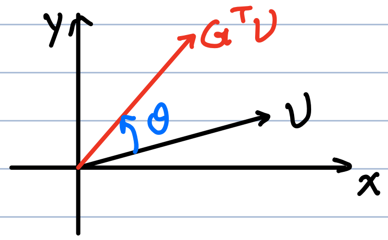
Last time: QR iteration

QR iteration is the standard solver in solving eig. val. problems for dense matrix

Today: Other eigenvalue solvers and SVD solvers

Previously: Householder transform ← zeroes out several coordinates using reflection

New: Givens transform ← zeros out a single coordinates using rotation



$$G := \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

rotate counterclockwise by θ

Choose c, s we can have $G^T v = \begin{bmatrix} r \\ 0 \end{bmatrix}$

In $\mathbb{R}^n$, define a Givens matrix

$$G(i,j,\theta) := \begin{array}{c} \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix} \end{array} \begin{array}{l} \\ \\ i^{th} \text{ row} \\ \\ j^{th} \text{ row} \\ \\ \\ \end{array}$$

$$c := \frac{x_i}{\sqrt{|x_i|^2 + |x_j|^2}}, \quad s := \frac{-x_j}{\sqrt{|x_i|^2 + |x_j|^2}}$$

$$G^T(i,j,\theta) \begin{bmatrix} x_1 \\ \vdots \\ x_i \\ \vdots \\ x_j \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_i' \\ \vdots \\ 0 \\ \vdots \\ x_n \end{bmatrix} \qquad \left(x_i' = \sqrt{x_i^2 + x_j^2} \right)$$

Complex case: $G := \begin{bmatrix} c & e^{i\phi}s \\ -e^{-i\phi}s & c \end{bmatrix} \in \mathbb{C}^{2\times 2}$

choose $c, s, \phi \in \mathbb{R}$ such that

$$G^T \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}, \quad \begin{array}{l} r^2 = |u|^2 + |v|^2 \\ r \in \mathbb{R} \end{array}$$

Note: G^T is equivalent to the Householder from $\begin{bmatrix} u \\ v \end{bmatrix}$ to $\begin{bmatrix} r \\ 0 \end{bmatrix}$.

and define Givens matrix similarly

$$G^T(i,j,\theta,\phi) \begin{bmatrix} v_1 \\ \vdots \\ v_i \\ \vdots \\ v_j \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1 \\ \vdots \\ v_i' \\ \vdots \\ 0 \\ \vdots \\ v_n \end{bmatrix} \quad \left(v_i' = \sqrt{|v_i|^2 + |v_j|^2} \right)$$

- Jacobi method

Let $A \in \mathbb{C}^{n\times n}$ be Hermitian

Goal: Perform similarity transform to make A diagonal

Idea: Let the sum of off-diagonal entries be

$$N(A) = \sum_{i\neq j} |a_{ij}|^2$$

Want to find $J \in \mathbb{C}^{n\times n}$ such that

$$B = G^{-1}AG \quad \text{and} \quad N(B) < N(A)$$

Choose G be Givens matrix

Analysis: Let $G = G(s,t,\theta,\phi)$

$$N(B) = \sum_{i\neq j} |b_{ij}|^2$$

$$= \|B\|_F^2 - \sum_{i=1}^{n} |b_{ii}|^2$$

when G is unitary

$\|G^T A G\|_F = \|A\|_F$

$a_{ii} = b_{ii} \quad \forall\, i \neq s,t$

$$= \|A\|_F^2 - \sum_{\substack{i=1 \\ i\neq s,t}}^{n} |a_{ii}|^2 - |b_{ss}|^2 - |b_{tt}|^2$$

Givens preserved the norm

$|b_{tt}|^2 + |b_{st}|^2 = |a_{tt}|^2 + |a_{st}|^2$

$|b_{ss}|^2 + |b_{ts}|^2 = |a_{ss}|^2 + |a_{ts}|^2$

$$= N(A) + |a_{ss}|^2 + |a_{tt}|^2 - |b_{ss}|^2 - |b_{tt}|^2$$

$$= N(A) + \underbrace{2|b_{st}|^2}_{\text{choose } \theta, \phi \text{ to zero out } b_{st}} - 2|a_{st}|^2$$

Implementation: (Classical Jacobi)

Let $A_1 = A$. Given a tolerance $\varepsilon > 0$

For $k = 1, 2, 3, \ldots$

Find $|a_{st}^{(k)}| = \max_{i \neq j} |a_{ij}^{(k)}|$ $\quad (A_k = (a_{ij}^{(k)})_{n \times n})$

If $|a_{st}^{(k)}| < \varepsilon$, return A_k

otherwise compute $G_k^T := G^T(s, t, \theta, \phi)$ to zero out $a_{st}^{(k)}$

compute $A_{k+1} = G_k^T A_k G_k$

Convergence: At each step

$$N(A_k) \leq (n^2 - n) \max_{i \neq j} |a_{ij}^{(k)}|^2 = n(n-1) |a_{st}^{(k)}|$$

$$N(A_{k+1}) = N(A_k) - 2|a_{st}^{(k)}|^2$$

$$\leq N(A_k) \underbrace{\left[1 - \frac{2}{n(n-1)}\right]}_{=: q_n \in [0,1)}$$

$$\leq \cdots \leq N(A_1) q_n^k \longrightarrow 0 \text{ as } k \to +\infty$$

The convergence is locally quadratic.

Cost Let $N(A_k) \sim O(\varepsilon_{mach})$

$$\Rightarrow k \sim \log(\varepsilon_{mach}) / \log q_n \sim n^2 \log(\varepsilon_{mach})$$

Each step requires $O(n)$ flops.

Total cost $\approx O(n^3 \log(\varepsilon_{mach}))$

Remark: 1) Searching for the largest off-diagonal entry is expansive. Can improve by a lot of different trick.

e.g. eliminate all off diagonal entries one-by-one,

or, set a threshod such that as long as we scan through an entry whose magnitude is larger than the threshold, elimination is implemented.

2) Jacobi method is easily parallelizable with each thread eliminating different row / column

3) Sparsity is not preserved in Jacobi method

4) Not used in practice, slower than standard QR in many cases

---

Other methods:

- Bisection method / Strum sequence methods:
  - compute a $\underbrace{\text{specific}}_{}$ (small) subset of eig. values. e.g. p^{th} eig. val.
  - much faster (O(n) cost) to find a small subset of eig. val. of A
- Divide-and-Conquer
  - Tear the tridiagonal Schur form in half, compute each in parallel, and then combine them together
  - Fast in practice

- Computing the SVD

<u>Goal</u>: $A \in \mathbb{C}^{m\times n}$

Find unitary $U \in \mathbb{C}^{m\times m}$, $V \in \mathbb{C}^{n\times n}$,

$\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_{\min\{m,n\}}) \in \mathbb{R}^{m\times n}$

such that $A = U\Sigma V^*$

<u>Naive algorithm</u>

Step 1: Compute $C = A^*A$

Step 2: Apply QR iteration to C

Bad idea because : 1) Forming A^*A is costly

2) Information is lost in A^*A

ex. $A = \begin{bmatrix} 1 & 1 \\ \sqrt{\varepsilon_{mach}} & 0 \\ 0 & \sqrt{\varepsilon_{mach}} \end{bmatrix}$, $\kappa_2(A) = \sqrt{\varepsilon_{mach}}$

compute on floating point system $\rightarrow$ $fl(A^*A) = fl\left(\begin{bmatrix} 1 & \sqrt{\varepsilon_{mach}} & 0 \\ 1 & 0 & \sqrt{\varepsilon_{mach}} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ \sqrt{\varepsilon_{mach}} & 0 \\ 0 & \sqrt{\varepsilon_{mach}} \end{bmatrix} \right)$

$= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $\leftarrow$ singular!

Mimicking the two-phase method of QR iteration, the standard SVD solver for dense A is the following

<u>Golub-Kahan Method</u>

Idea: Implicitly apply QR iteration to A^*A

Step 1 Reduce A to upper bidiagonal form

Goal: Apply different unitary matrix on left and right of A

$$A = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \underset{F_1}{\xRightarrow{U_1^* \cdot}} \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}}_{U_1^* A} \underset{\begin{bmatrix} 1 & \\ & F_2 \end{bmatrix}}{\xRightarrow{\cdot V_1}} \underbrace{\begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{bmatrix}}_{U_1^* A V_1}$$

$$\underset{\begin{bmatrix} I_2 & \\ & F_3 \end{bmatrix}}{\xRightarrow{U_2^* \cdot}} \underbrace{\begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}}_{U_2^* U_1^* A V_1} \underset{\begin{bmatrix} I_3 & \\ & F_4 \end{bmatrix}}{\xRightarrow{\cdot V_2}} \underbrace{\begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}}_{U_2^* U_1^* A V_1 V_2} \Rightarrow \cdots \Rightarrow \underbrace{\begin{bmatrix} \times & \times & 0 & 0 \\ 0 & \times & \times & 0 \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{U_4^* U_3^* U_2^* U_1^* A V_1 V_2 =: B}$$

Work for Golub-Kahan bidiagonalization

$\approx$ twice of QR fact.

$\approx 2\left(2mn^2 - \frac{2}{3}n^3\right)$ complex flops

Remark: Cost can be reduced by first computing QR fact. of $A = QR$, then perform bidiagonalization to R. This is saving when $m \gg n$.

(R-SVD) (Lawson-Hanson-Chan Bidiagonalization)

Step 2: Apply QR iter. to $B^T B$

bidiagonal form

Again, forming $B^T B$ is not a wise choice form numerical standpoint.

Need to apply QR and RQ step implicitly.

Details are omitted (Implicit Q theorem ......)

Remark: Jacobi method can be extended similarly.

Mature algorithms can be found in LAPACK