Stability of algorithms

Note Title 2024-11-22

$$\frac{\|\tilde{x}-x\|}{\|x\|} \leq \kappa_{problem} \cdot u$$

backward stability: an algorithm is **backward stable** if $\tilde{x} = f(\hat{y})$ for a modified input $\hat{y}$ close to the real one y, $\frac{\|\hat{y}-y\|}{\|y\|} = O(u)$

backward stability $\Rightarrow \frac{\|\tilde{x}-x\|}{\|x\|} = \kappa_{problem} \cdot O(u)$

A-posteriori bounds: use **residual** $(\|A-QR\|, \|Ax-b\|, \ldots)$ to estimate the error.

Stability analysis via $a \oplus b = (a+b)(1+\delta)$ for scalar products can be used to prove backward stability "a priori". We did that for scalar products.

Ex: backward stability for the problem of solving triangular linear systems, $Ux = y$

$\tilde{x} = \hat{U}^{-1}\hat{y}$, $\frac{\|\hat{U}-U\|}{\|U\|} = O(u)$, $\frac{\|\hat{y}-y\|}{\|y\|} = O(u)$.

QR factorization with Householder reflectors is backward stable. $\rightarrow$ we shall prove it today

Stability of matrix products:

$C = A \cdot B \qquad \tilde{C}$

Using results for matrix products:

$$|\tilde{C}_{ij} - C_{ij}| \leq [|a_{i1}| \cdots |a_{in}|] \cdot \begin{bmatrix} |b_{1j}| \\ \vdots \\ |b_{nj}| \end{bmatrix} O(u)$$

This leads to a norm bound for e.g. the Frobenius norm:

$$\|\tilde{C} - C\| \leq O(u) \|A\| \cdot \|B\|$$

Note that $\frac{\|\tilde{C} - C\|}{\|C\|}$ is <u>not</u> $O(u)$, because $\|A\| \cdot \|B\|$ can be much larger than $\|C\|$, if there is cancellation.

If A is square invertible, we can give a backward error bound:

$$\tilde{C} = AB + F \qquad \|F\| = \|\tilde{C} - C\| = O(u) \cdot \|A\| \cdot \|B\|$$

$$= A\underbrace{(B + A^{-1}F)}_{\hat{B}} \qquad \frac{\|\hat{B} - B\|}{\|B\|} = \frac{\|A^{-1}F\|}{\|B\|} \leq \frac{\|A^{-1}\| \cdot O(u) \|A\| \cdot \|B\|}{\|B\|}$$

$$= O(u) \cdot \underbrace{\|A\| \cdot \|A^{-1}\|}_{\kappa(A)}$$

Backward stability holds only if $\kappa(A) \approx 1$

In particular, if A is <u>orthogonal</u>:

$\|Q\|_2 = \|Q^T\|_2 = \|Q^{-1}\|_2 = 1$, and $\sqrt{n}$ for the Frobenius norm.

Let Q be orthogonal, B any matrix: then computing $C = QB$ is backward stable:

$$\tilde{C} = Q\hat{B} \qquad \frac{\|\hat{B} - B\|}{\|B\|} = O(u)$$

The same can be shown to hold for fast arithmetic with Householder matrices: $H = I - 2uu^T \quad \|u\| = 1$

$C = HB = B - 2u(u^T B)$ can be computed in $O(\text{size}(B))$

$$\tilde{C} = H\hat{B} \qquad \frac{\|\hat{B} - B\|}{\|B\|} = O(u)$$

QR factorization: compute $u_1, u_2 \ldots u_n$ such that

$$A = Q(u_1)\,Q(u_2) \cdots Q(u_n)\,R \qquad R = \begin{bmatrix} \times & \cdots & \times \\ & \ddots & \vdots \\ 0 & & \times \end{bmatrix}$$

$$Q(u_i) = \left[\begin{array}{c|c} I_{i-1} & \\ \hline & I - 2u_i u_i^T \end{array}\right]$$

Stability: the computed $\tilde{A}, \tilde{u}_1, \ldots \tilde{u}_n$ satisfy

$$\tilde{A} = \underbrace{Q(\tilde{u}_1)\,Q(\tilde{u}_2) \cdots Q(\tilde{u}_n)}_{\text{exactly orthogonal}} \cdot \hat{R} \quad \text{for a given } \hat{R} \text{ with } \frac{\|\hat{R} - R\|}{\|R\|} = O(u)$$

Proof: QR produces (in exact arithmetic)

$$R_0 = A$$

$$R_1 = \begin{bmatrix} \times & \cdots & \times \\ 0 & & \vdots \\ \vdots & & \vdots \\ 0 & \cdots & \times \end{bmatrix} \qquad R_i = Q(u_i) R_{i-1}$$

$$R_2 = \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ 0 & 0 & & \vdots \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \times \end{bmatrix}$$

$$\vdots$$

$$R = R_n = \begin{bmatrix} \times & \cdots & \times \\ & \ddots & \vdots \\ 0 & & \times \end{bmatrix}$$

In machine arithmetic: at each step, start from $\tilde{R}_{i-1}$ (that contains errors at previous steps), produce $\tilde{u}_i$ and then compute $\tilde{R}_i = Q(\tilde{u}_i) \odot \tilde{R}_{i-1}$

$$= Q(\tilde{u}_i)\, \hat{R}_{i-1}, \quad \text{with } \hat{R}_{i-1} = \tilde{R}_{i-1} + \Delta_{i-1}$$

Step 1: $R_0 = A$

$$\tilde{R}_1 = Q(\tilde{u}_1)(A + \Delta_0)$$

Step 2: $\tilde{R}_2 = Q(\tilde{u}_2)(\tilde{R}_1 + \Delta_1) = Q(\tilde{u}_2) Q(\tilde{u}_1)\left(A + \Delta_0 + Q(\tilde{u}_1)^T \Delta_1\right)$

Step 3: $\tilde{R}_3 = Q(\tilde{u}_3) Q(\tilde{u}_2) Q(\tilde{u}_1)\left(A + \Delta_0 + Q(\tilde{u}_1)^T \Delta_1 + Q(\tilde{u}_2)^T Q(\tilde{u}_1)^T \Delta_2\right)$

$\vdots$

the error at each step can be seen as a perturbation to the original matrix A.

$$\|\Delta_0\| = O(u)\|R_0\| = O(u)\|A\|$$

$$\|\Delta_1\| = O(u)\|\tilde{R}_1\| = O(u)\|R_1\| + O(u^2)$$

$$= O(u)\|A\| + O(u^2)$$

All errors Δ_i can be proved to be $\|A\| \cdot O(u)$

Backward stability of each step + orthogonal transformations at each step = backward stability

$$\tilde{R}_n = Q(\tilde{u}_n) \cdots Q(\tilde{u}_1)\Big(A + \underbrace{\Delta_0 + Q_1^T \Delta_1 + \cdots}_{\text{n errors of magnitude } O(u)\|A\|}\Big)$$

$\hat{A}$

$$\frac{\|\hat{A}-A\|}{\|A\|} = \|\Delta_0 + Q_1^T\Delta_1 + Q_2^T Q_1^T \Delta_2 + \dots\| = n\,O(u)\,\|A\|$$

To solve least squares problems with QR:

$[Q_0, R_0] = qr(A)$ back. stable

$c = Q_0^T y$ $\|c\| = \|y\|$ back. stable

$x = R_0 \backslash c$ by back substitution $\|R_0\| = \|A\|$, back. stable

$\Rightarrow$ the computed solution $\tilde{x}$ is the exact solution of $\min \|\hat{A}\tilde{x} - \hat{y}\|$, with $\frac{\|\hat{A}-A\|}{\|A\|}, \frac{\|\hat{y}-y\|}{\|y\|} = O(u)$.

$\frac{\|\tilde{x}-x\|}{\|x\|} = O(u) \cdot$ cond. number of this problem.

What about other algorithms?

- SVD: $[U_0, S_0, V] = svd(A)$

$c = U_0^T y$

$d = S_0^{-1} c$ $\quad S_0 = \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_n \end{bmatrix}$

$x = Vd$

Again, backward stable steps + orthogonal transformations $\Rightarrow$ backward stab.

- Normal equations:

$C = A^T A$

$d = A^T y$

$x = C^{-1} d$ via Cholesky factorization / LU / ...

The transformation $Ax-y \rightarrow A^T(Ax-y)=0$ is <u>not</u> orthogonal

$\kappa(A^TA) = \kappa(A)^2$ (A^TA has sing. values $\sigma_1^2, \dots \sigma_n^2$)

Even if we just consider the error incurred in storing A, this amplified by a factor $\kappa(C) = \kappa(A)^2$

If $\vartheta \approx 0$, then

$\kappa_{LS\ problem} \approx \kappa(A)$.

NE can only deliver at most an error bound

$$\frac{\|\tilde{x}-x\|}{\|x\|} = \kappa(A)^2 O(u)$$

→ Normal equations can have much larger error than the other algorithms.

In our earlier example:

QR	→	$\kappa(A)u \approx 10^8 \cdot 10^{-16}$
SVD	→	$\kappa(A)u \approx 10^8 \cdot 10^{-16}$
NE	→	$\kappa(A)^2 u \approx 10^{16} \cdot 10^{-16}$ ⚠

NE

cost when $m \gg n$: mn^2

cost when $m \approx n$: $\frac{1}{3}n^3$

Stability can be poor!

QR

$2mn^2$

$\frac{4}{3}n^3$

back. stable

$x = A \backslash y$

SVD

$2mn^2$

$\approx 13n^3$

back. stable

gives info on how close the problem is to singularity!

For dense problems. Sparse: lsqr CGNE

"conjugate gradient on the normal equations"

No more details!

We now switch to a different problem:

Large, sparse linear systems with generic A (non necessarily symm. pos. def.)

We already know CG: $Ax = y \Leftrightarrow \min \frac{1}{2} x^T A x - y^T x + \text{Const}$ if $A > 0$

Main ideas:

1. Interact with A only with matrix products: 1 matvec Ad_k at each step

2. We produce at each step j iterates in

$$K_j(A, y) = \text{span}\left(y, Ay, A^2y, \dots A^{j-1}y\right)$$

(Krylov space)

We want to apply these ideas to a general matrix.

Idea: • compute a orthonormal basis of $K_j(A, y)$

• look for an iterate x_j inside this space.

The columns of $V = \left[y \; Ay, A^2y, \dots A^{j-1}y\right]$ are already a basis of $K_j(A, y)$, but they form a basis matrix V that is very ill-conditioned.

We want an orthonormal basis: vectors $q_1, q_2, \dots q_j$ such that $q_i^\dagger q_j = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$

$Q_j = [q_1 | q_2 | \dots q_j]$ $\quad Q_j^\dagger Q_j = I_j$

$Q_j \in \mathbb{R}^{m \times j}$ if $A \in \mathbb{R}^{m \times m}$, $y \in \mathbb{R}^m$

and $\text{span}(q_1, q_2 \dots q_j) = K_j(A, y)$.

This is a problem then we can solve inductively:

assume we have already computed

$q_1, \dots q_j$ s.t. $\text{span}(q_1, \dots q_j) = K_j(A, y) = \text{span}(y, Ay, \dots A^{j-1}y)$

we would like to add a new vector q_{j+1} s.t.

$\text{span}(q_1, q_2 \dots q_{j+1}) = K_{j+1}(A, y) = \text{span}(y, Ay, \dots A^{j-1}y, A^j y)$

Arnoldi algorithm

Solves the problem with an additional invariant:

q_j is a vector of degree exactly $j-1$

If $q_j \in K_j(A, y)$, then $q_j = \alpha_0 y + \alpha_1 Ay + \dots + \alpha_{j-1} A^{j-1} y$

we see q_j has degree $j-1$ if $\alpha_{j-1} \neq 0$

$= (\alpha_0 I + \alpha_1 A + \dots + \alpha_{j-1} A^{j-1}) y$

$= p(A) y$

Arnoldi algorithm: $\boxed{j=1}$ $\quad K_1(A, y) = \text{span}\{y\}$

$q_1 = \frac{1}{\|y\|} \cdot y$

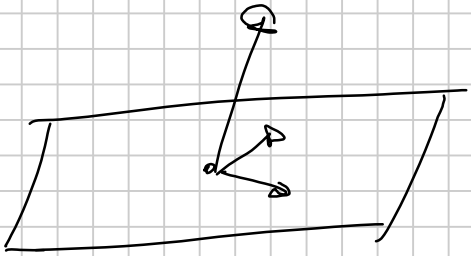
inductive step $j \to j+1$: Assume you have

$q_1, q_2, \dots q_j$ orthonormal,

$\text{span}(q_1 \dots q_j) = K_j(A, y) = \text{span}(y, Ay, \dots A^{j-1}y)$

$q_j = \alpha_0 y + \dots + \alpha_{j-1} A^{j-1} y$ with $\alpha_{j-1} \neq 0$.

We extend $q_1, \dots q_j$ with a vector $w \in K_{j+1}(A, y) \setminus K_j(A, y)$



A valid choice is $w = Aq_j$

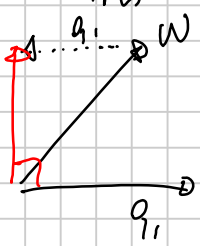
because q_j has degree $j-1$ $\Rightarrow$ Aq_j has degree j.

$\text{span}(q_1, q_2 \dots q_j, w) = K_{j+1}(A, y)$

But w is not necessarily orthonormal to the previous vectors!

We can make w orthogonal to q_1 with

$w \leftarrow w - q_1(q_1^T w)$



And we do the same for all q_i:

$w = Aq_j$

for $i = 1 \dots j$

$w = w - q_i(q_i^T w)$

end

At the end of the for loop, the resulting w satisfies

$q_i^T w = 0$ for $i = 1, \dots j$

$q_{j+1} \frac{1}{\|w\|} \cdot w.$