AMS526: Numerical Analysis I (Numerical Linear Algebra for Computational and Data Sciences)

Lecture 13: Stability of Householder Triangularization; Other Methods for Least Squares Problems; Linear Algebra Software

Xiangmin Jiao

Stony Brook University

Outline

1. Stability of Householder Triangularization (NLA§16,19)
2. Solution of Least Squares Problems
3. Rank-Deficient Least Squares Problems
4. Software for Linear Algebra

Solution of Least Squares Problems

- An efficient and robust approach is to use QR factorization $A = \hat{Q}\hat{R}$
  - b can be projected onto range(A) by $P = \hat{Q}\hat{Q}^T$, and therefore $\hat{Q}\hat{R}x = \hat{Q}\hat{Q}^T b$
  - Left-multiply by $\hat{Q}^T$ and we get $\hat{R}x = \hat{Q}^T b$ (note $A^+ = \hat{R}^{-1}\hat{Q}^T$)

> Least squares via QR Factorization
>
> Compute reduced QR factorization $A = \hat{Q}\hat{R}$
> Compute vector $c = \hat{Q}^T b$
> Solve upper-triangular system $\hat{R}x = c$ for x

- Computation is dominated by QR factorization ($2mn^2 - \frac{2}{3}n^3$)
- What about stability?

Backward Stability of Householder Triangularization

- For a QR factorization $A = QR$ computed by Householder triangularization, the factors $\tilde{Q}$ and $\tilde{R}$ satisfy

$$\tilde{Q}\tilde{R} = A + \delta A, \quad \|\delta A\|/\|A\| = O(\epsilon_{\text{machine}}),$$

  i.e., exact QR factorization of a slightly perturbed A
- $\tilde{R}$ is R computed by algorithm using floating points
- However, $\tilde{Q}$ is product of *exactly orthogonal* reflectors

$$\tilde{Q} = \tilde{Q}_1 \tilde{Q}_2 \dots \tilde{Q}_n$$

  where $\tilde{Q}_k$ is given by computed $\tilde{v}_k$, since Q is not formed explicitly

Backward Stability of Solving $Ax = b$ with QR

Algorithm: Solving $Ax = b$ by QR Factorization

Compute $A = QR$ using Householder, represent Q by reflectors
Compute vector $y = Q^T b$ implicitly using reflectors
Solve upper-triangular system $R_{1:n,1:n}x = y_{1:n}$ for x

- All three steps are backward stable
- Overall, we can show that

$$(A + \Delta A)\tilde{x} = b, \quad \|\Delta A\|/\|A\| = O(\epsilon_{\text{machine}})$$

as we prove next

Backward Stability of Solving $Ax = b$ with Householder Triangularization

Proof: Step 2 gives

$$(\tilde{Q} + \delta Q)\tilde{y} = b, \quad \|\delta Q\| = O(\epsilon_{\text{machine}})$$

Step 3 gives

$$(\tilde{R} + \delta R)\tilde{x} = \tilde{y}, \quad \|\delta R\|/\|\tilde{R}\| = O(\epsilon_{\text{machine}})$$

Therefore,

$$b = (\tilde{Q} + \delta Q)(\tilde{R} + \delta R)\tilde{x} = \left[\tilde{Q}\tilde{R} + (\delta Q)\tilde{R} + \tilde{Q}(\delta R) + (\delta Q)(\delta R)\right]\tilde{x}$$

Step 1 gives

$$b = \left[A + \underbrace{\delta A + (\delta Q)\tilde{R} + \tilde{Q}(\delta R) + (\delta Q)(\delta R)}_{\Delta A}\right]\tilde{x}$$

where $\tilde{Q}\tilde{R} = A + \delta A$

Proof of Backward Stability Cont'd

$\tilde{Q}\tilde{R} = A + \delta A$ where $\|\delta A\|/\|A\| = O(\epsilon_{\text{machine}})$, and therefore

$$\frac{\|\tilde{R}\|}{\|A\|} \leq \|\tilde{Q}^T\| \frac{\|A + \delta A\|}{\|A\|} = O(1)$$

Now show that each term in ΔA is small

$$\frac{\|(\delta Q)\tilde{R}\|}{\|A\|} \leq \|(\delta Q)\| \frac{\|\tilde{R}\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

$$\frac{\|\tilde{Q}(\delta R)\|}{\|A\|} \leq \|\tilde{Q}\| \frac{\|\delta R\|}{\|\tilde{R}\|} \frac{\|\tilde{R}\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

$$\frac{\|(\delta Q)(\delta R)\|}{\|A\|} \leq \|\delta Q\| \frac{\|\delta R\|}{\|A\|} = O(\epsilon_{\text{machine}}^2)$$

Overall,

$$\frac{\|\Delta A\|}{\|A\|} \leq \frac{\|\delta A\|}{\|A\|} + \frac{\|(\delta Q)\tilde{R}\|}{\|A\|} + \frac{\|\tilde{Q}(\delta R)\|}{\|A\|} + \frac{\|(\delta Q)(\delta R)\|}{\|A\|} = O(\epsilon_{\text{machine}})$$

Since the algorithm is backward stable, it is also accurate.

Backward Stability of Householder Triangularization

Theorem

Let the full-rank least squares problem be solved using Householder triangularization on a computer satisfying the two axioms of floating point numbers. The algorithm is backward stable in the sense that the computed solution $\tilde{x}$ has the property

$$\|(A + \delta A)\tilde{x} - b)\| = \min, \quad \frac{\|\delta A\|}{\|A\|} = O(\epsilon_{machine})$$

for some $\delta A \in \mathbb{R}^{m \times n}$.

- Backward stability of the algorithm is true whether $\hat{Q}^T b$ is computed via explicit formation of $\hat{Q}$ or computed implicitly
- Backward stability also holds for Householder triangularization with arbitrary column pivoting $AP = \hat{Q}\hat{R}$

Outline

Algorithms for Solving Least Squares Problems

- There are many variants of algorithms for solving least squares problems
  - Householder QR (with/without pivoting, explicit or implicit Q): **Backward stable**
  - Classical Gram-Schmidt: **Unstable**
  - Modified Gram-Schmidt with explicit Q: **Unstable**
  - Modified Gram-Schmidt with augmented system of equations with implicit Q: **Backward stable**
  - Normal equations (solve $A^T Ax = A^T b$): **Unstable**
  - Singular value decomposition: **Stable and most accurate**

Stability of Gram-Schmidt Orthogonalization

- Gram-Schmidt QR is unstable, due to loss of orthogonality
- Gram-Schmidt can be stabilized using augmented system of equations
  1. Compute QR factorization of augmented matrix: [Q,R1]=mgs([A,b])
  2. Extract R and $\hat{Q}^T b$ from $R1$: R=R1(1:n,1:n); Qb=R1(1:n,n+1)
  3. Back solve: x=R\Qb

Theorem

The solution of the full-rank least squares problem by Gram-Schmidt orthogonality is backward stable in the sense that the computed solution $\tilde{x}$ has the property

$$\|(A+\delta A)\tilde{x}-b\| = \min, \quad \frac{\|\delta A\|}{\|A\|} = O(\epsilon_{machine})$$

for some $\delta A \in \mathbb{R}^{m\times n}$, provided that $\hat{Q}^T b$ is formed implicitly.

Other Methods

- The method of *normal equation* solves $x = (A^T A)^{-1} A^T b$, due to squaring of condition number of A

Theorem

The solution of the full-rank least squares problem via normal equation is unstable. Stability can be achieved, however, by restriction to a class of problems in which $\kappa(A)$ is uniformly bounded above.

- Another method is to SVD

Solution by SVD

- Using $A = \hat{U}\hat{\Sigma}V^T$, b can be projected onto range(A) by $P = \hat{U}\hat{U}^T$, and therefore $\hat{U}\hat{\Sigma}V^T x = \hat{U}\hat{U}^T b$
- Left-multiply by $\hat{U}^T$ and we get $\hat{\Sigma}V^T x = \hat{U}^T b$

> Least squares via SVD
>
> Compute reduced SVD factorization $A = \hat{U}\hat{\Sigma}V^T$
> Compute vector $c = \hat{U}^T b$
> Solve diagonal system $\hat{\Sigma}w = c$ for w
> Set $x = Vw$

- Work is dominated by SVD, which is $\sim 2mn^2 + 11n^3$ flops, very expensive if $m \approx n$
- Question: If A is rank deficient, how to solve $Ax \approx b$?

Outline

Rank-Deficient Least Squares Problems

- Least squares problems $Ax \approx b$ is the most challenging if A is (nearly) rank deficient
- If A is rank deficient, there are an infinite number of x that minimizes $\|b - Ax\|$. This is because if $y \in \text{null}(A)$, for any x that minimizes $\|b - Ax\|$, $x + y$ also minimizes $\|b - Ax\|$
- "Uniqueness" is recovered by requiring $x \perp \text{null}(A)$. Or equivalently, minimize $\|x\|$ subject to $(b - Ax) \perp \text{range}(A)$
- In practice, however, we often have near rank deficiency instead of exact rank deficiency
- For rank deficiency, (left or right) null space is the space span by (left or right) singular vectors corresponding to zero singular values
- For nearly rank deficient least squares problem, define "numerical null space" to be singular vectors corresponding to smallest singular values

Solving Rank-Deficient Least Squares Problems by SVD

- If A is full rank, $A = \hat{U}\hat{\Sigma}V^T = \sum_{j=1}^{\min\{m,n\}} \sigma_j u_j v_j^T$, and $A^+ = \sum_{j=1}^{\min\{m,n\}} \frac{1}{\sigma_j} v_j u_j^T$
- If A is rank deficient, $A^+ = \sum_{j=1}^{r} \frac{1}{\sigma_j} v_j u_j^T$, where r is rank of A
- If A is nearly rank deficient, $\tilde{A}^+ = \sum_{j=1}^{r} \frac{1}{\sigma_j} v_j u_j^T$, where r is *numerical rank* of A, i.e., largest j such that $\sigma_j \geq \epsilon\sigma_1$ for some small ϵ. This is called *truncated SVD*
- $\tilde{A} = \sum_{j=1}^{r} \sigma_j u_j v_j^T$ is a low-rank approximation to A

Rank-deficient least squares via truncated SVD

Compute reduced SVD factorization $A = \hat{U}\hat{\Sigma}V^T$ and estimate r

Compute vector $c = \left(\hat{U}_{:,1:r}\right)^T b$

Solve diagonal system $\hat{\Sigma}_{1:r,1:r} w = c$ for w

Set $x = V_{1:m,1:r} w$

A Note on Pseudoinverse

- If $A \in \mathbb{R}^{m \times n}$ is rank deficient, the pseudoinverse of A is defined as

$$A^{+} = \sum_{j=1}^{r} \frac{1}{\sigma_j} v_j u_j^T,$$

  where r is rank of A
- It is unique minimum Frobenius norm solution to

$$\min_{X \in \mathbb{R}^{n \times m}} \|AX - I_m\|_F$$

- It is also unique matrix $X \in \mathbb{R}^{n \times m}$ that satisfies four *Moore-Penrose conditions*:
  1. $AXA = A$
  2. $XAX = X$
  3. $(AX)^T = AX$
  4. $(XA)^T = XA$

QR with Column Pivoting

- Another approach is to use QR with column pivoting, or QRCP
- Suppose $A \in \mathbb{R}^{m \times n}$, and r be its rank. In **exact arithmetic**, QR with column pivoting is rank revealing if

$$Q^T A \Pi = \begin{array}{cc} \left[\begin{array}{cc} R_{11} & R_{12} \\ 0 & 0 \end{array}\right] & \begin{array}{c} r \\ m-r \end{array} \\ \begin{array}{cc} r & n-r \end{array} & \end{array}$$

  where Π is a permutation matrix. $\text{range}(A) = \text{span}\{q_1, \dots, q_r\}$
- In exact arithmetic, a rank-revealing QRCP is obtained by permuting columns such that diagonal entry in R is maximized at each step
- In particular, at kth step,

$$(Q_{k-1} \cdots Q_1) A (\Pi_1 \cdots \Pi_{k-1}) = R^{(k-1)} = \begin{array}{cc} \left[\begin{array}{cc} R_{11}^{(k-1)} & R_{12}^{(k-1)} \\ 0 & R_{22}^{(k-1)} \end{array}\right] & \begin{array}{c} k-1 \\ m-k+ \end{array} \\ \begin{array}{cc} k-1 & n-k+1 \end{array} & \end{array}$$

  permute column with maximum 2-norm in $R_{22}^{(k-1)}$ to kth column

Solving Rank-Deficient Least Squares Problems by QRCP

- With rounding errors, one terminates if the computed $R_{22}^{(k-1)}$ ($\tilde{R}_{22}^{(k-1)}$) has a sufficient small 2-norm compared to that of A
  - If $\tilde{R}_{22}^{(k-1)}$ is small, then A is (numerically) rank deficient
  - However, if $rank(A) = k$, it does not follow that $\tilde{R}_{22}^{(k-1)}$ is small, so it may not reveal rank deficiency (and still lead to instability)
- In practice, QRCP needs to be coupled with a condition number estimator to help reveal the rank

Rank-deficient least squares via truncated QRCP

Compute QRCP $AP = QR$ and estimate r
Compute vector $c = \left(Q_{:,1:r}\right)^T b$
Solve triangular system $R_{1:r,1:r}y = c$ for y
Set $x = P_{1:m,1:r}y$

- Truncated QRCP is far less expensive than truncated SVD, and is robust with a good condition number estimator
- Unlike SVD, QRCP uses a subset of columns of A

Outline

Software for Linear Algebra

- LAPACK: Linear Algebra PACKage (`www.netlib.org/lapack/lug`)
  - Standard library for solving linear systems and eigenvalue problems
  - Successor of LINPACK (`www.netlib.org/linpack`) and EISPACK (`www.netlib.org/eispack`)
  - Depends on BLAS (Basic Linear Algebra Subprograms)
  - Parallel extensions include ScaLAPACK and PLAPACK (with MPI)
  - Note: Uses Fortran conventions for matrix arrangements
- MATLAB
  - Factorization A: lu(A) and chol(A)
  - Solve $Ax = b$: $x = A\backslash b$
    - Uses back/forward substitution for triangular matrices
    - Uses Cholesky factorization for positive-definite matrices
    - Uses LU factorization with partial pivoting for nonsymmetric matrices
    - Uses Householder QR for least squares problems
    - Uses some special routines for matrices with special sparsity patterns
  - Uses LAPACK and other packages internally
- Direct solvers for sparse matrices (e.g., SuperLU, SuiteSparse, MUMPS)

Some Commonly Used Functions

Example BLAS routines: Matrix-vector multip.: dgemv; Matrix-matrix multip: dgemm

	LU Factorization		Solve linear system		Est. cond
	General	Symmetric	General	Symmetric	
LAPACK	dgetrf	dpotrf/dsytrf	**dgesv**	**dposv/dposvx**	dgecon
LINPACK	dgefa	dpofa/dsifa	dgesl	dposl/dsisl	dgeco
MATLAB	lu	chol	\	\	rcond

	Linear least squares			Eigenvalue/vector		SVD
	QR	Solve	Rank-deficient	General	Sym.	
LAPACK	dgeqrf	**dgels**	**dgelsy/s/d**	**dgeev**	**dsyev**	**dgesvd**
LINPACK	dqrdc	dqrsl	dqrst	-	-	dsvdc
MATLAB	qr	\	\	eig	eig	svd

For BLAS, LINPACK, and LAPACK, first letter s stands for single-precision real, d for double-precision real, c for single-precision complex, and z for double-precision complex. Boldface LAPACK routines are **driver** routines; others are **computational** routines.

Using LAPACK Routines in C Programs

- LAPACK was written in Fortran 77. Special attention is required when calling from C.
- Key differences between C and Fortran
  1. Storage of matrices: column major (Fortran) versus row major (C/C++)
  2. Argument passing for subroutines in C and Fortran: pass by reference (Fortran) and pass by value (C/C++)
- Example C code (example.c) for solving linear system using sgesv
  - See class website for sample code
  - To compile, issue command "cc -o example example.c -llapack -lblas"
- Hint: To find a function name, refer to LAPACK Users' Guide
- To find out arguments for a given function, search on netlib.org