STA 141C - Big Data & High Performance Statistical Computing **Spring 2022**

Lecture 7: Householder and givens transformations

Lecturer: Bo Y.-C. Ning *January 27, 2022*

Disclaimer: *My notes may contain errors, distribution outside this class is allowed only with the permission of the Instructor.*

Announcement

- Team member list due this Sunday
- hw2 due next Wednesday

Last time

- Cholesky decomposition
- QR by GS and MGS

Today

- QR by householder
- QR by givens

1 Housesholder transformation

Let's first take a look at an example for getting QR for $A \in \mathbb{R}^{5\times 4}$ using the Householder transformation. Given a matrix A, where

$$A = \begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix}$$

In Step 1, we choose H_1 such that

$$A = \begin{pmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{pmatrix} \longrightarrow H_1 \begin{pmatrix} \times \\ \times \\ \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} \times \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Thus,

$$H_1 A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix}$$

In Step 2, we choose $\tilde{H}_2$ such that

$$H_1 A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \end{pmatrix} \longrightarrow \tilde{H}_2 \begin{pmatrix} \times \\ \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} \times \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

and let

$$H_2 = \begin{pmatrix} 1 & 0_4' \\ 0_4 & \tilde{H}_2 \end{pmatrix}$$

We then obtain

$$H_2 H_1 A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix}$$

In Step 3, we choose $\tilde{H}_3$ such that

$$H_2 H_1 A = \begin{pmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{pmatrix} \longrightarrow \tilde{H}_3 \begin{pmatrix} \times \\ \times \\ \times \end{pmatrix} = \begin{pmatrix} \times \\ 0 \\ 0 \end{pmatrix}.$$

We let

$$H_3 = \begin{pmatrix} 1 & 0 & 0_3' \\ 0 & 1 & 0_3' \\ 0_3 & 0_3 & \tilde{H}_3 \end{pmatrix}$$

We continue this for the last column, finally we obtain $R = H_4 H_3 H_2 H_1 A$, which is an upper triangular matrix and $Q = H_1 H_2 H_3 H_4$, which is an orthogonal matrix. Can you verify $A = QR$?

In general, we have the following definition for the householder matrix.

Definition 7.1 (Householder matrix) *Let v be a $k \times 1$ vector. The Householder matrix associated to v is the $k \times k$ matrix H_k defined as follows:*

$$H_k = I_k - \frac{2vv'}{\|v\|^2},$$

where I is the $k \times k$ identity matrix, $\|v\|$ is the norm of v.

Assume $A = (a_1, \ldots, a_p) \in \mathbb{R}^{n\times p}$ has full column rank, we construct matrices $H_1, \ldots, H_p$ in $\mathbb{R}^{n\times n}$ such that

$$H_p \ldots H_2 H_1 A = \begin{pmatrix} R_1 \\ 0 \end{pmatrix},$$

then the matrices H_js are called *Householder matrices*.

How many flops for QR via Householder transformation? The total is $2p^2(n - p/3)$ flops for an $n \times p$ matrix.

QR via Householder transformation is built in LAPACK. In python: scipy.linalg.qr is a wrapper of the LAPACK routines dgeqrf, zgeqrf, dorgqr, and zungqr, note that dgeqrf uses Householder transformation for real matrices In R, the default option for qr is DQRDC in LINPACK, which also uses the Householder transformation.

2 Givens rotation

Again, assuming $A \in \mathbb{R}^{n\times p}$ has full column rank, Givens rotation provides another approach for getting $A = QR$.

The next example illustrates how givens rotation works (from Page 252 of Golub & van Loan, 4th edition): Consider a 4×3 matrix,

$$\begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{pmatrix} \xrightarrow{(1;\ 3,4)} \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{pmatrix} \xrightarrow{(1;\ 2,3)} \begin{pmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix} \xrightarrow{(1;\ 1,2)}$$
$$\begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{pmatrix} \xrightarrow{(2;\ 3,4)} \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{pmatrix} \xrightarrow{(2;\ 2,3)} \begin{pmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{pmatrix} \xrightarrow{(3;\ 3,4)} R$$

Definition 7.2 *Given a vector with length two, the Givens rotation matrix is given by G such that*

$$G' \begin{pmatrix} a & b \end{pmatrix} = \begin{pmatrix} \gamma & -\sigma \\ \sigma & \gamma \end{pmatrix}' \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix},$$

where $r = \sqrt{a^2 + b^2}$, $\sigma = \sin\theta$, and $\gamma = \cos\theta$. To compute γ and σ, a straightforward approach is

$$\gamma = \frac{a}{\sqrt{a^2 + b^2}}, \quad \sigma = \frac{b}{\sqrt{a^2 + b^2}}.$$

In practice, we should avoid calculating $\sqrt{a^2 + b^2}$ especially when a, b are small. Instead, if $a^2 > b^2$, we shall take $\tau = b/a < 1$, $\gamma = \frac{1}{\sqrt{1+\tau^2}}$, and $\sigma = \gamma\tau$; and if $a^2 < b^2$, we shall take $\tau = a/b < 1$, $\sigma = \frac{1}{\sqrt{1+\tau^2}}$, and $\gamma = \sigma\tau$.

Notice that $Q = G_1 \ldots, G_t$ with t number of total rotations. QR by givens rotation takes $3p^2(n - p/3)$ flops.

In R, givens() function in the pracma package (see here) uses givens rotation.

In python, the build-in function scipy.linalg.qr uses Householder as default. The givens implementation is available here

3 Sweep operator

In statistics, linear regression is the most commonly applied procedure. The matrices that appear in linear regression and multivariate analysis are almost invariably symmetric (and positive definite). Sweeping exploits this symmetry. Although there are faster and numerically more stable algorithms for inverting a matrix or solving a least-squares problem, no algorithm matches the conceptual simplicity and utility of sweeping.

Recall the GS algorithm we learned in the last lecture, the idea for Gauss-Jordan elimination is similar to the GS algorithm but is directly applied to solve the linear system $X'X\beta = X'y$.

Suppose $X'X = \begin{pmatrix} 4 & 2 \\ 2 & 6 \end{pmatrix}$ and $X'y = \begin{pmatrix} 6 \\ 10 \end{pmatrix}$.

- Step1: write it as the augmented matrix:

$$\left[\begin{array}{cc|c} 4 & 2 & 6 \\ 2 & 6 & 10 \end{array}\right]$$

- Step 2: Multiply row by 1/4

$$\left[\begin{array}{cc|c} 1 & 1/2 & 3/2 \\ 2 & 6 & 10 \end{array}\right]$$

- Step 3: Add -2 times row 1 to row 2

$$\left[\begin{array}{cc|c} 1 & 1/2 & 3/2 \\ 0 & 5 & 7 \end{array}\right]$$

- Step 4: Multiply row 2 by 1/5

$$\left[\begin{array}{cc|c} 1 & 1/2 & 3/2 \\ 0 & 1 & 7/5 \end{array}\right]$$

- Step 5: add $-1/2$ times row 2 to row 1

$$\left[\begin{array}{cc|c} 1 & 0 & 4/5 \\ 0 & 1 & 7/5 \end{array}\right]$$

Thus, to solve full-rank regression equations, form the augmented matrix $[X'X|X'y]$. One then does not need to solve $(X'X)^{-1}$.

For an $n \times n$ matrix, the sweep operator takes $n^2/2$ flops. The error sum of squares may also be simultaneously computed by choosing an augmented matrix

$$\left[\begin{array}{c|c} X'X & X'y \\ \hline X'y & y'y \end{array}\right]$$

Applying the sweep operator, one obtain

$$\left[\begin{array}{c|c} I & (X'X)^{-1}X'y \\ \hline 0 & y'y - y'X(X'X)^{-1}X'y \end{array}\right]$$

For a symmetric p.d. matrix A, write A into the form

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

Do sweep on the diagonal entries of A_{11}, one gets

$$\begin{pmatrix} -A_{11}^{-1} & A_{11}^{-1}A_{12} \\ A_{21}A_{11}^{-1} & A_{22} - A_{21}A_{11}^{-1}A_{12} \end{pmatrix}$$

Apply to the linear regression equation $A_{11} = X'X$, $A_{12} = X'y$, $A_{22} = y'y$, we got

$$\begin{pmatrix} -(X'X)^{-1} & (X'X)^{-1}X'y \\ y'X(X'X)^{-1} & y'y - y'X(X'X)^{-1}X'y \end{pmatrix} = \begin{pmatrix} -\mathrm{Var}(\hat{\beta})/\sigma^2 & \hat{\beta} \\ \hat{\beta}' & \hat{e}'\hat{e} \end{pmatrix}$$

A few notes:

- A is p.d. if and only if each diagonal entry can be swept in succession and is positive until it is swept. When a diagonal entry of a pd matrix A is swept, it becomes negative and remains negative thereafter. Taking the product of diagonal entries just before each is swept yields the determinant of A.
- SAS automatically implement it to solve linear equations
- the `sweep()` in R has nothing to do with the sweep operator
- See the implement code in R on Piazza
- See Section 7.4 of Kenneth Lange's book *Numerical Analysis for Statisticians*.

4 Review of numerical linear algebra

Consider solving the problem $Ax = b$, A is symmetric and p.s.d.

So far, we have learned GE/LU, Cholesky, QR, sweep, and soon we will learn SVD.

Let's summarize the numerical methods we learned for solving linear regression problem

Method	Flops	Software	Stability
Sweep	$np^2/2 + p^3/2$	SAS	less stable
Cholesky	$np^2/2 + p^3/2$	R & Python	less stable
QR by HH	$2np^2 - 2p^3/3$	R & Python	stable
QR by Givens	$3np^2 - p^3$	R & Python	stable
QR by MGS	np^2		more stable

A few notes:

- When $n \gg p$, sweep and Cholesky are twice faster than QR and need less space
- QR are more stable and produce numerically more accurate solution
- Sweep can yields standard errors and so on, while Cholesky cannot
- MGS is slower than HH but yields Q_1 (thin QR)
- There is simply no such thing as a universal 'gold standard' when it comes to algorithms.
- Flop counts is not everything. GE/LU has a higher memory traffic and vectorization overheads.
- QR is comparable in efficiency and more stable