8 Lecture 8, Jan 28

Announcements

- HW2 due next Tuesday. FAQs at `http://hua-zhou.github.io/teaching/biostatm280-2016winter/biostatm280winter2016/2016/01/26/hw2-hints.html`

Last time

- LU decomposition $\boldsymbol{A} = \boldsymbol{LU}$. $2/3n^3$ flops.
- LU with partial pivoting $\boldsymbol{PA} = \boldsymbol{LU}$. $2/3n^3$ flops.
- Cholesky decomposition $\boldsymbol{A} = \boldsymbol{LL}^T$. $1/3n^3$ flops.
- Cholesky decomposition with symmetric pivoting: $\boldsymbol{PAP}^T = \boldsymbol{LL}^T$. $1/3n^3$ flops. Yields the rank of the psd matrix.
- Note the actual implementation in LAPACK may be quite different from our description.
- *No inversion mentality*. Matrix inversion takes an extra of $4/3n^3$ flops, which is unnecessary in most applications.
- Multivariate normal density evaluation.

Today

- Linear regression by Cholesky.
- QR and linear regression.
- QR by (modified) Gram-Schmidt.
- QR by Householder.
- QR by Givens.
- Sweep operator.

Linear regression by Cholesky (method of normal equations)

Assume $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ has full column rank. (For rank deficient $\boldsymbol{X}$, use Cholesky with symmetric pivoting.)

- Cholesky on the augmented matrix $(\boldsymbol{X} \quad \boldsymbol{y})^T(\boldsymbol{X} \quad \boldsymbol{y})$ yields

$$\begin{pmatrix} \boldsymbol{X}^\top \boldsymbol{X} & \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{y}^\top \boldsymbol{X} & \boldsymbol{y}^\top \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{L} & \boldsymbol{0} \\ \boldsymbol{l}^\top & d \end{pmatrix} \begin{pmatrix} \boldsymbol{L}^\top & \boldsymbol{l} \\ \boldsymbol{0}^\top & d \end{pmatrix} = \begin{pmatrix} \boldsymbol{L}\boldsymbol{L}^\top & \boldsymbol{L}\boldsymbol{l} \\ \boldsymbol{l}^\top \boldsymbol{L}^\top & \|\boldsymbol{l}\|_2^2 + d^2 \end{pmatrix}.$$

  Normal equation $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^T\boldsymbol{y}$ implies the equation

$$\boldsymbol{L}\boldsymbol{L}^\top\boldsymbol{\beta} = \boldsymbol{L}\boldsymbol{l} \text{ or } \boldsymbol{L}^\top\boldsymbol{\beta} = \boldsymbol{l},$$

  which we can solve for $\boldsymbol{\beta}$ in p^2 flops. Since $\boldsymbol{l} = \boldsymbol{L}^{-1}\boldsymbol{X}^\top\boldsymbol{y}$, we have

$$\boldsymbol{l}^\top\boldsymbol{l} = \boldsymbol{y}\boldsymbol{X}(\boldsymbol{L}\boldsymbol{L}^\top)^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \boldsymbol{y}^\top\boldsymbol{P}_{\boldsymbol{X}}\boldsymbol{y} = \|\hat{\boldsymbol{y}}\|_2^2$$

  and

$$d^2 = \boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{l}^\top\boldsymbol{l} = \boldsymbol{y}^\top(\boldsymbol{I} - \boldsymbol{P}_{\boldsymbol{X}})\boldsymbol{y} = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \text{SSE}.$$

  If standard errors are needed, we need inversion $(\boldsymbol{X}^T\boldsymbol{X})^{-1} = (\boldsymbol{L}\boldsymbol{L}^T)^{-1} = \boldsymbol{L}^{-T}\boldsymbol{L}^{-1}$. Use `chol2inv()` in R function for this purpose.

- In summary, linear regression by Cholesky, aka the method of normal equations:
  - Form the lower triangular part of $(\boldsymbol{X}, \boldsymbol{y})^T(\boldsymbol{X}, \boldsymbol{y})$.
    $n(p+1)^2 \approx np^2$ flops.
  - Cholesky decomposition of the augmented system $\begin{pmatrix} \boldsymbol{X}^\top \boldsymbol{X} & \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{y}^\top \boldsymbol{X} & \boldsymbol{y}^\top \boldsymbol{y} \end{pmatrix}$.
    $(p+1)^3/3 \approx p^3/3$ flops.
  - Solve $\boldsymbol{L}^T\boldsymbol{\beta} = \boldsymbol{l}$ for regression coefficients $\hat{\boldsymbol{\beta}}$.
    p^2 flops.
  - If want standard errors, estimate σ^2 by $\hat{\sigma}^2 = d^2/(n-p)$ and compute $\hat{\sigma}^2(\boldsymbol{X}^T\boldsymbol{X})^{-1} = \hat{\sigma}^2(\boldsymbol{L}\boldsymbol{L}^T)^{-1}$.
    $4p^3/3$ flops.

  Total cost is $p^3/3 + np^2$ flops (without s.e.) or $5p^3/3 + np^2$ flops (with s.e.).

QR decomposition and linear regression

A second approach for linear regression uses QR decomposition. This is how the `lm()` function in R does linear regression. Assume $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ has full column rank.

- QR decomposition: $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$, $\boldsymbol{Q}^\intercal \boldsymbol{Q} = \boldsymbol{I}_n$, and $\boldsymbol{R} \in \mathbb{R}^{n \times p}$.
  - first p columns of $\boldsymbol{Q}$ form an orthonormal basis of $\mathcal{C}(\boldsymbol{X})$
  - last $n - p$ columns of $\boldsymbol{Q}$ form an orthonormal basis of $\mathcal{N}(\boldsymbol{X}^T)$
- (Thin/Skinny QR) Then $\boldsymbol{X} = \boldsymbol{Q}_1 \boldsymbol{R}_1$, where $\boldsymbol{Q}_1 \in \mathbb{R}^{n \times p}$ has orthogonal columns, $\boldsymbol{Q}_1^\intercal \boldsymbol{Q}_1 = \boldsymbol{I}_p$, and $\boldsymbol{R}_1 \in \mathbb{R}^{p \times p}$ is an invertible upper triangular matrix with positive diagonal entries.
- For linear regression, we only need skinny QR.
  Note $\boldsymbol{X}^\intercal \boldsymbol{X} = \boldsymbol{R}_1^\intercal \boldsymbol{R}_1$ yields the Cholesky decomposition of $\boldsymbol{X}^\intercal \boldsymbol{X}$.
- (Skinny) QR on the augmented matrix yields

$$\begin{pmatrix} \boldsymbol{X} & \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} \boldsymbol{Q} & \boldsymbol{q} \end{pmatrix} \begin{pmatrix} \boldsymbol{R} & \boldsymbol{r} \\ \boldsymbol{0}_p^\intercal & d \end{pmatrix} = \begin{pmatrix} \boldsymbol{Q}\boldsymbol{R} & \boldsymbol{Q}\boldsymbol{r} + d\boldsymbol{q} \end{pmatrix}.$$

  Normal equation $\boldsymbol{X}^\intercal \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^\intercal \boldsymbol{y}$ implies

$$\boldsymbol{R}\boldsymbol{\beta} = \boldsymbol{R}^{-T}\boldsymbol{X}^\intercal \boldsymbol{y} = \boldsymbol{R}^{-T}\boldsymbol{R}^\intercal \boldsymbol{Q}^\intercal \boldsymbol{y} = \boldsymbol{Q}^\intercal \boldsymbol{y} = \boldsymbol{r},$$

  which is easy to solve for $\boldsymbol{\beta}$. The fitted value is $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{Q}\boldsymbol{R}\boldsymbol{R}^{-1}\boldsymbol{r} = \boldsymbol{Q}\boldsymbol{r}$. The residual is

$$\hat{\boldsymbol{e}} = \boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{y} - \boldsymbol{Q}\boldsymbol{r} = d\boldsymbol{q}$$

  and SSE $= \|\hat{\boldsymbol{e}}\|_2^2 = d^2$. The projection matrix is

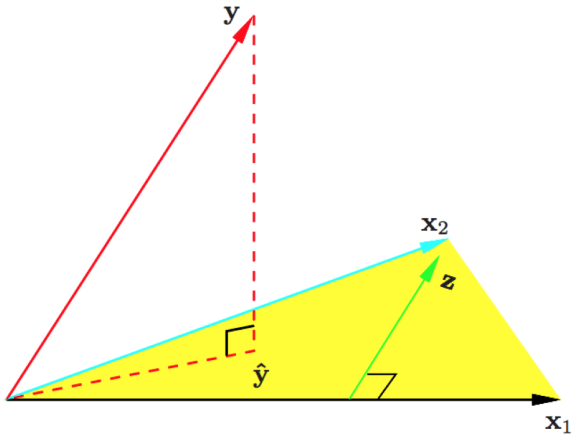
$$\boldsymbol{X}(\boldsymbol{X}^\intercal \boldsymbol{X})^{-1}\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}(\boldsymbol{R}^\intercal \boldsymbol{R})^{-1}\boldsymbol{R}^\intercal \boldsymbol{Q}^\intercal = \boldsymbol{Q}\boldsymbol{Q}^\intercal.$$

- Three numerical methods to compute QR: (modified) Gram-Schmidt, Householder transform, (fast) Givens transform.

QR by (modified) Gram-Schmidt

Jørgen Pedersen Gram in an undated photo

Erhard Schmidt (courtesy MFO)



Assume $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) \in \mathbb{R}^{n \times p}$ has full column rank.

- Gram-Schmidt (GS) algorithm produces the skinny QR: $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$, where $\boldsymbol{Q} \in \mathbb{R}^{n \times p}$ has orthogonal columns and $\boldsymbol{R} \in \mathbb{R}^{p \times p}$ is an upper triangular matrix.
- Gram-Schmidt algorithm orthonormalizes a set of non-zero, *linearly independent* vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p$. Initialize $\boldsymbol{q}_1 = \boldsymbol{x}_1 / \|\boldsymbol{x}_1\|_2$; then for $k = 2, \ldots, p$,

$$\begin{aligned} \boldsymbol{v}_k &= \boldsymbol{x}_k - \boldsymbol{P}_{\mathcal{C}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_{k-1}\}} \boldsymbol{x}_k = \boldsymbol{x}_k - \sum_{j=1}^{k-1} \langle \boldsymbol{x}_k, \boldsymbol{q}_j \rangle \boldsymbol{q}_j \\ \boldsymbol{q}_k &= \boldsymbol{v}_k / \|\boldsymbol{v}_k\|_2 \end{aligned}$$

- For $j = 1, \ldots, p$, $\mathcal{C}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_j\} = \mathcal{C}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$, and $\boldsymbol{q}_j \perp \mathcal{C}\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_{j-1}\}$.
- Collectively, we have $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$ (skinny QR), where
  - $\boldsymbol{Q} \in \mathbb{R}^{n \times p}$ has orthonormal columns $\boldsymbol{q}_k$ and thus $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}_p$.
  - Where $\boldsymbol{R}$? $\boldsymbol{R} = \boldsymbol{Q}^T\boldsymbol{X} \in \mathbb{R}^{p \times p}$ has entries $r_{jk} = \langle \boldsymbol{q}_j, \boldsymbol{x}_k \rangle$, which are available from the algorithm. Note $r_{jk} = 0$ for $j > k$. Thus $\boldsymbol{R}$ is upper triangular.
- In GS algorithm, $\boldsymbol{X}$ is over-written by $\boldsymbol{Q}$ and $\boldsymbol{R}$ is stored in a separate array.
- The regular Gram-Schmidt is *unstable* (we loose orthogonality due to roundoff errors) when columns of $\boldsymbol{X}$ are collinear.
- *Modified Gram-Schmidt* (MGS): after each normalization step of $\boldsymbol{v}_k$, we replace $\tilde{\boldsymbol{x}}_j$, $j > k$, by its residual.
- Why MGS is better than GS? Read `http://cavern.uark.edu/~arnold/4353/CGSMGS.pdf`
- Computational cost of GS and MGS is $\sum_{k=1}^{p} 4n(k-1) \approx 2np^2$.

QR by Householder

Photograph by Paul Halmos

Assume $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_p) \in \mathbb{R}^{n \times p}$ has full column rank.

- Idea: $\boldsymbol{H}_p \cdots \boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}$, where $\boldsymbol{H}_j \in \mathbb{R}^{n \times n}$ are the Householder transformation matrix. It yields the full QR where $\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_p \in \mathbb{R}^{n \times n}$. Recall that GS/MGS only produces the thin QR decomposition.

- For arbitrary $\boldsymbol{v}, \boldsymbol{w} \in \mathbb{R}^n$ with $\|\boldsymbol{v}\|_2 = \|\boldsymbol{w}\|_2$, we can construct a *Householder matrix* $\boldsymbol{H} = \boldsymbol{I}_n - 2\boldsymbol{u}\boldsymbol{u}^\intercal$, $\boldsymbol{u} = -\frac{1}{\|\boldsymbol{v}-\boldsymbol{w}\|_2}(\boldsymbol{v} - \boldsymbol{w})$, that carries $\boldsymbol{v}$ to $\boldsymbol{w}$:

$$\boldsymbol{H}\boldsymbol{v} = \boldsymbol{w}.$$

  $\boldsymbol{H}$ is symmetric and orthogonal. Calculation of Householder vector $\boldsymbol{u}$ costs $2n$ flops.

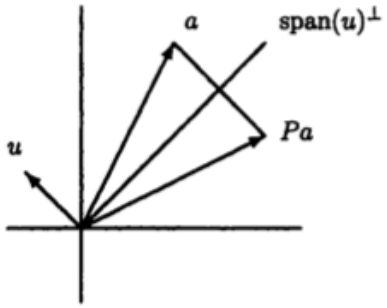


FIG. 2.3.1. *Reflection of a vector a in a hyperplane with normal u.*

- Now choose $\boldsymbol{H}_1$ to zero the first column of $\boldsymbol{X}$ below diagonal

$$\boldsymbol{H}_1 \boldsymbol{x}_1 = \begin{pmatrix} \|\boldsymbol{x}_1\|_2 \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

  Take $\boldsymbol{H}_2$ to zero the second column below diagonal; ...

$$\boldsymbol{H}_2 \boldsymbol{H}_1 \boldsymbol{A} = \begin{bmatrix} \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \\ 0 & 0 & \boxtimes & \times & \times \end{bmatrix}$$

In general, choose the j-th Householder transform $\boldsymbol{H}_j = \boldsymbol{I}_n - 2\boldsymbol{u}_j\boldsymbol{u}_j^\top$, where $\boldsymbol{u}_j = \begin{pmatrix} \boldsymbol{0}_{j-1} \\ \tilde{\boldsymbol{u}}_j \end{pmatrix}$, $\tilde{\boldsymbol{u}}_j \in \mathbb{R}^{n-j+1}$, to zero the j-th column below diagonal. $\boldsymbol{H}_j$ takes the form

$$\boldsymbol{H}_j = \begin{pmatrix} \boldsymbol{I}_{j-1} & \\ & \boldsymbol{I}_{n-j+1} - 2\tilde{\boldsymbol{u}}_j\tilde{\boldsymbol{u}}_j^T \end{pmatrix} = \begin{pmatrix} \boldsymbol{I}_{j-1} & \\ & \tilde{\boldsymbol{H}}_j \end{pmatrix}.$$

- Applying a Householder transform $\boldsymbol{H} = \boldsymbol{I} - 2\boldsymbol{u}\boldsymbol{u}^T$ to a matrix $\boldsymbol{X} \in \mathbb{R}^{n\times p}$

  $$\boldsymbol{H}\boldsymbol{X} = \boldsymbol{X} - 2\boldsymbol{u}(\boldsymbol{u}^\top\boldsymbol{X})$$

  costs $4np$ flops. *We never explicitly form the Householder matrices.*

  Note applying $\boldsymbol{H}_j$ to $\boldsymbol{X}$ only needs $4(n-j+1)(p-j+1)$ flops.

- QR by Householder: $\boldsymbol{H}_p \cdots \boldsymbol{H}_1\boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}$.

- The process is done in place. Upper triangular part of $\boldsymbol{X}$ is overwritten by $\boldsymbol{R}_1$ and the essential Householder vectors ($\tilde{u}_{j1}$ is normalized to 1) are stored in $\boldsymbol{X}(j:n, j)$.

- At j-th stage

  1. computing the Householder vector $\tilde{\boldsymbol{u}}_j$ costs $2(n-j+1)$ flops
  2. applying the Householder transform $\tilde{\boldsymbol{H}}_j$ to the $\boldsymbol{X}(j:n, j:p)$ block costs $4(n-j+1)(p-j+1)$ flops

  In total we need $\sum_{j=1}^{p}[2(n-j+1)+4(n-j+1)(p-j+1)] \approx 2np^2 - \frac{2}{3}p^3$ flops.

- Where is $\boldsymbol{Q}$? $\boldsymbol{Q} = \boldsymbol{H}_1 \cdots \boldsymbol{H}_p$.

  In some applications, it's necessary to form the orthogonal matrix $\boldsymbol{Q}$.

  Accumulating $\boldsymbol{Q}$ costs another $2np^2 - \frac{2}{3}p^3$ flops.

- When computing $\boldsymbol{Q}^\top\boldsymbol{v}$ or $\boldsymbol{Q}\boldsymbol{v}$ as in some applications (e.g., solve linear equation using QR), no need to form $\boldsymbol{Q}$. Simply apply Householder transforms successively.

  `qr.qy()` and `qr.qty()` in R do this.

- Computational cost of Householder QR for linear regression: $2np^2 - \frac{2}{3}p^3$ (regression coefficients and $\hat{\sigma}^2$) or more (fitted values, s.e., ...).

Rank deficient X: Householder QR with column pivoting

$\boldsymbol{X} \in \mathbb{R}^{n \times p}$ may not have full column rank.

- Idea (due to Businger and Golub 1965): at the j-th stage, swap the column in $\boldsymbol{X}(j:n, j:p)$ with maximum ℓ_2 norm to be the pivot column. If the maximum ℓ_2 norm is 0, it stops, ending with

$$\boldsymbol{X}\boldsymbol{\Pi} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R}_{11} & \boldsymbol{R}_{12} \\ \mathbf{0}_{(n-r)\times r} & \mathbf{0}_{(n-r)\times(p-r)} \end{pmatrix},$$

  where $\boldsymbol{\Pi} \in \mathbb{R}^{p \times p}$ is a permutation matrix and r is the rank of $\boldsymbol{X}$. QR with column pivoting is rank revealing.

- The overhead of re-computing the column norms can be reduced by the property

$$\boldsymbol{Q}\boldsymbol{z} = \begin{pmatrix} \alpha \\ \boldsymbol{\omega} \end{pmatrix} \Rightarrow \|\boldsymbol{\omega}\|_2^2 = \|\boldsymbol{z}\|_2^2 - \alpha^2$$

  for any orthogonal matrix $\boldsymbol{Q}$.

- In R, the `qr()` function is a wrapper for various LINPACK (default) and LAPACK routines. It performs Householder QR with column pivoting and returns

  - `$qr`: a matrix of same size as input matrix
  - `$rank`: rank of the input matrix
  - `$pivot`: pivot vector
  - `$aux`: normalizing constants of Householder vectors

  Auxiliary functions `qr.coef()`, `qr.resid()`, `qr.qy()`, `qr.qty()`, `qr.solve()`, ... are very helpful.

QR by Givens rotation

- Householder transform $\boldsymbol{H}_j$ introduces batch of zeros into a vector.
- Givens transform (aka Givens rotation, Jacobi rotation, plane rotation) selectively zeros one element of a vector.
- Overall QR by Givens rotation is less efficient than the Householder method, but is better suited for matrices with structured patterns of nonzero elements.
- Givens/Jacobi rotations:

$$\boldsymbol{G}(i,k,\theta) = \begin{pmatrix} 1 & & 0 & & 0 & & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & & c & & s & & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & & -s & & c & & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & & 0 & & 0 & & 1 \end{pmatrix},$$

  where $c = \cos(\theta)$ and $s = \sin(\theta)$. $\boldsymbol{G}(i,k,\theta)$ is orthogonal.
- Pre-multiplication by $\boldsymbol{G}(i,k,\theta)^T$ rotates counterclockwise θ radians in the (i,k) coordinate plane. If $\boldsymbol{x} \in \mathbb{R}^n$ and $\boldsymbol{y} = \boldsymbol{G}(i,k,\theta)^T\boldsymbol{x}$, then

$$y_j = \begin{cases} cx_i - sx_k & j = i \\ sx_i + cx_k & j = k \\ x_j & j \neq i,k \end{cases}.$$

Apparently if we choose $\tan(\theta) = -x_k/x_i$, or equivalently,

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

then $y_k = 0$.

- Pre-applying Givens transform $\boldsymbol{G}(i,k,\theta)^T \in \mathbb{R}^{n\times n}$ to a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ only effects two rows of $\boldsymbol{A}$:

$$\boldsymbol{A}([i,k],:) \leftarrow \begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \boldsymbol{A}([i,k],:),$$

costing $3m$ flops.

- Post-applying Givens transform $\boldsymbol{G}(i,k,\theta) \in \mathbb{R}^{m\times m}$ to a matrix $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ only effects two columns of $\boldsymbol{A}$:

$$\boldsymbol{A}(:,[i,k]) \leftarrow \boldsymbol{A}(:,[i,k]) \begin{pmatrix} c & s \\ -s & c \end{pmatrix},$$

costing $3n$ flops.

- QR by Givens: $\boldsymbol{G}_t^T \cdots \boldsymbol{G}_1^T \boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_1 \\ \boldsymbol{0} \end{pmatrix}$

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

$$\begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{(3,4)} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{(2,3)} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{(3,4)} \boldsymbol{R}$$

- Zeros in $\boldsymbol{X}$ can also be introduced row-by-row.
- If $\boldsymbol{X} \in \mathbb{R}^{n\times p}$, the total cost is $3np^2 - p^3$ flops and $O(np)$ square roots.
- Note each Givens transform can be summarized by a single number, which is stored in the zeroed entry of $\boldsymbol{X}$.
- Fast Givens transform avoids taking square roots.

Sweep operator

Assume $\boldsymbol{A} \succeq \boldsymbol{0}_{n\times n}$.

- KL 7.4-7.6; Also see "*A tutorial on the SWEEP operator*" by James H. Goodnight. `http://www.jstor.org/stable/2683825`
- *Sweep* on the k-th diagonal entry $a_{kk} \neq 0$ yields $\hat{\boldsymbol{A}}$ with entries

$$\begin{aligned}
\hat{a}_{kk} &= -\frac{1}{a_{kk}} \\
\hat{a}_{ik} &= \frac{a_{ik}}{a_{kk}} \\
\hat{a}_{kj} &= \frac{a_{kj}}{a_{kk}} \\
\hat{a}_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i \neq k, j \neq k.
\end{aligned}$$

  n^2 flops (taking into account of symmetry).

- *Inverse sweep* sends $\boldsymbol{A}$ to $\check{\boldsymbol{A}}$ with entries

$$\begin{aligned}
\check{a}_{kk} &= -\frac{1}{a_{kk}} \\
\check{a}_{ik} &= -\frac{a_{ik}}{a_{kk}} \\
\check{a}_{kj} &= -\frac{a_{kj}}{a_{kk}} \\
\check{a}_{ij} &= a_{ij} - \frac{a_{ik}a_{kj}}{a_{kk}}, \quad i \neq k, j \neq k.
\end{aligned}$$

  n^2 flops (taking into account of symmetry).

- $\check{\hat{\boldsymbol{A}}} = \boldsymbol{A}$.
- Block form of sweep: Let the symmetric matrix $\boldsymbol{A}$ be partitioned as $\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{pmatrix}$. If possible, sweep on the diagonal entries of $\boldsymbol{A}_{11}$ yields

$$\hat{\boldsymbol{A}} = \begin{pmatrix} -\boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \\ \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1} & \boldsymbol{A}_{22} - \boldsymbol{A}_{21}\boldsymbol{A}_{11}^{-1}\boldsymbol{A}_{12} \end{pmatrix}.$$

  Order dose *not* matter.

- Pd and determinant: $\boldsymbol{A}$ is pd 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 $\boldsymbol{A}$ is swept, it becomes negative and remains negative thereafter. Taking the product of diagonal entries just before each is swept yields the determinant of $\boldsymbol{A}$.
- Linear regression by sweep. Sweep on $\begin{pmatrix} \boldsymbol{X}^\top \boldsymbol{X} & \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{y}^\top \boldsymbol{X} & \boldsymbol{y}^\top \boldsymbol{y} \end{pmatrix}$ yields

  $$\begin{pmatrix} -(\boldsymbol{X}^\top \boldsymbol{X})^{-1} & (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \\ \boldsymbol{y}^\top \boldsymbol{X} (\boldsymbol{X}\boldsymbol{X})^{-1} & \boldsymbol{y}^\top \boldsymbol{y} - \boldsymbol{y}^\top \boldsymbol{X} (\boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y} \end{pmatrix} = \begin{pmatrix} -\frac{1}{\sigma^2} \mathrm{Var}(\hat{\boldsymbol{\beta}}) & \hat{\boldsymbol{\beta}} \\ \hat{\boldsymbol{\beta}}^\top & \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 \end{pmatrix}.$$

  In total $np^2 + p^3$ flops.
- Sweep is useful for stepwise regression, (conditional) multivariate normal density calculation, MANOVA, ...
- Warning: the `sweep()` function in R has nothing do to with the sweep operator here.
- Demo code: `http://hua-zhou.github.io/teaching/biostatm280-2016winter/sweep.html`

Summary of linear regression: Table on KL p105

Method	Flops	Remarks	Software	Stability
Sweep	$np^2 + p^3$	$(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ available	SAS	less stable
Cholesky	$np^2 + p^3/3$			less stable
QR by Householder	$2np^2 - 2/3p^3$		R	stable
QR by MGS	$2np^2$	$\boldsymbol{Q}_1$ available		more stable

Table 1: Numerical methods for linear regression. In order of stability.

Remarks:

- When $n \gg p$, sweep and Cholesky are twice faster than QR and need less space.

  But QR methods are more stable and produce numerically more accurate solution.

- Although sweep is slower than Cholesky, it yields standard errors and so on.
- Sweep is useful for stepwise regression, multivariate normal calculation, and numerous other statistical applications.
- MGS appears slower than Householder, but it yields $\boldsymbol{Q}_1$.
- Sweep and Cholesky is based on the Gram matrix $\boldsymbol{X}^T\boldsymbol{X}$, which can be dynamically updated with incoming data. They can easily handle huge n, moderate p data sets that cannot fit into memory.

  "There is simply no such thing as a universal 'gold standard' when it comes to algorithms".