SI231 Matrix Analysis and Computations
Topic 4: Orthogonalization and QR Decomposition

Ziping Zhao

Spring Term 2020–2021

School of Information Science and Technology
ShanghaiTech University, Shanghai, China

Topic 4: Orthogonalization and QR Decomposition

- QR decomposition
- Solving LS via QR decomposition
- Gram-Schmidt QR
- Householder QR
- Givens QR
- Solving Underdetermined Linear Systems via QR decomposition

Summary

QR decomposition/factorization: Any $\mathbf{A} \in \mathbb{R}^{m \times n}$ admits a decomposition

$$\mathbf{A} = \mathbf{Q}\mathbf{R},$$

where $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is orthogonal, $\mathbf{R} \in \mathbb{R}^{m \times n}$ takes an upper triangular form. $(\mathbf{Q}, \mathbf{R})$ is called a QR factor of $\mathbf{A}$. (see Theorem 5.2.1 in **[Golub-Van Loan'13]**)

- efficient to compute
  - done algorithmically by either Gram-Schmidt, Householder reflections, or Givens rotations
- can be used to compute (thread for most of the algortihms in matrix computations)
  - a basis for $\mathcal{R}(\mathbf{A})$ or for $\mathcal{R}(\mathbf{A})^{\perp}$;
  - LS solutions;
  - linear systems (not the standard method).
- a building block for QR iteration algorithm—a popular numerical method for solving eigenvalue problem (all eigenvalues) (cf. Topic 5) and computing SVD (cf. Topic 7)
- for complex $\mathbf{A} \in \mathbb{C}^{m \times n}$, $\mathbf{Q} \in \mathbb{C}^{m \times m}$ is unitary

Thin QR Decomposition for Tall or Square $\mathbf{A}$

- for $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m \geq n$,

$$\mathbf{A} = \mathbf{Q}\mathbf{R} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1,$$

  where $\mathbf{Q}_1 \in \mathbb{R}^{m\times n}$, $\mathbf{Q}_2 \in \mathbb{R}^{m\times(m-n)}$, $\mathbf{R}_1 \in \mathbb{R}^{n\times n}$ which is upper triangular

- the decomposition $\mathbf{A} = \mathbf{Q}_1\mathbf{R}_1$ is called the thin QR (reduced/economic QR) decomposition of $\mathbf{A}$; $(\mathbf{Q}_1, \mathbf{R}_1)$ is called a thin QR factor of $\mathbf{A}$

- in contrast, the QR in the previous page is also called full QR decomposition

- properties under thin QR and $m \geq n$:
  - $\mathbf{A}$ has full column rank if and only if $r_{ii} \neq 0$ for all i;
  - if $\mathbf{A}$ has full column rank (Quiz),

$$\mathcal{R}(\mathbf{A}) = \mathcal{R}(\mathbf{Q}_1), \qquad \mathcal{R}(\mathbf{A})^\perp = \mathcal{R}(\mathbf{Q}_2)$$

  - see Theorem 5.2.2 in **[Golub-Van Loan'13]**

QR Decomposition for Full Column-Rank Matrices

Theorem 4.1. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$ be a full column-rank matrix. Then $\mathbf{A}$ admits a decomposition

$$\mathbf{A} = \mathbf{Q}_1 \mathbf{R}_1,$$

where $\mathbf{Q}_1 \in \mathbb{R}^{m \times n}$ is semi-orthogonal; $\mathbf{R}_1 \in \mathbb{R}^{n \times n}$ is upper triangular. If we restrict $r_{ii} > 0$ for all i, then $(\mathbf{Q}_1, \mathbf{R}_1)$ is unique.

- Proof:
  1. let $\mathbf{C} = \mathbf{A}^T \mathbf{A}$, which is PD if $\mathbf{A}$ has full column rank
  2. since $\mathbf{C}$ is PD, it admits the Cholesky decomposition $\mathbf{C} = \mathbf{R}_1^T \mathbf{R}_1$
  3. $\mathbf{R}_1$, as the upper triangular Cholesky factor, is unique (cf. Theorem 2.3)
  4. let $\mathbf{Q}_1 = \mathbf{A}\mathbf{R}_1^{-1}$. It can be verified that $\mathbf{Q}_1^T \mathbf{Q}_1 = \mathbf{I}, \mathbf{Q}_1 \mathbf{R}_1 = \mathbf{A}$
- see Theorem 5.2.3 in **[Golub-Van Loan'13]**
- Remark: the proof above reveals that thin QR may be computed via Cholesky decomposition, but this is not what we usually do in practice

Solving (Well-determined) Linear Systems via QR

- Problem: compute the solution to

$$\mathbf{A}\mathbf{x} = \mathbf{y}$$

  with nonsingular $\mathbf{A} \in \mathbb{R}^{n \times n}$

- if $\mathbf{A} = \mathbf{Q}\mathbf{R}$ is a QR factorization, we have $\mathbf{Q}\mathbf{R}\mathbf{x} = \mathbf{y}$ or

$$\mathbf{R}\mathbf{x} = \mathbf{Q}^T\mathbf{y}$$

- Solution (computational):
  1. factorize $\mathbf{A}$ as $\mathbf{A} = \mathbf{Q}\mathbf{R}$, $\mathcal{O}(2n^3)$ (to be shown next)
  2. compute $\mathbf{z} = \mathbf{Q}^T\mathbf{y}$, $\mathcal{O}(2n^2)$
  3. solve $\mathbf{R}\mathbf{x} = \mathbf{z}$ via backward substitution, $\mathcal{O}(n^2)$

- more expensive than Gauss elimination and LU decompositions, and hence not the standard method...

Solving LS via QR

- Problem: compute the solution to

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2,$$

  with $\mathbf{A}$ being of full column rank
- can be used to solve a well-determined or an overdetermined linear system
- observe ($\|\mathbf{Q}^T\mathbf{z}\|_2 = \|\mathbf{z}\|_2$)

$$\begin{aligned}\|\mathbf{y}-\mathbf{A}\mathbf{x}\|_2^2 &= \|\mathbf{Q}^T\mathbf{y}-\mathbf{Q}^T\mathbf{A}\mathbf{x}\|_2^2 = \|\mathbf{Q}^T\mathbf{y}-\mathbf{R}\mathbf{x}\|_2^2 \\ &= \left\|\begin{bmatrix}\mathbf{Q}_1^T\mathbf{y}\\ \mathbf{Q}_2^T\mathbf{y}\end{bmatrix} - \begin{bmatrix}\mathbf{R}_1\mathbf{x}\\ \mathbf{0}\end{bmatrix}\right\|_2^2 = \|\mathbf{Q}_1^T\mathbf{y}-\mathbf{R}_1\mathbf{x}\|_2^2 + \|\mathbf{Q}_2^T\mathbf{y}\|_2^2\end{aligned}$$

- it reduces to solve $\mathbf{R}_1\mathbf{x} = \mathbf{Q}_1^T\mathbf{y}$
- Solution (computational):
  1. compute the thin QR factor $(\mathbf{Q}_1, \mathbf{R}_1)$ of $\mathbf{A}$;
  2. compute $\mathbf{z} = \mathbf{Q}_1^T\mathbf{y}$
  3. solve $\mathbf{R}_1\mathbf{x} = \mathbf{z}$ via backward substitution.

Gram-Schmidt for Computing Thin QR

Recall the (classical) Gram-Schmidt (GS) orthogonalization procedure in Topic 1:

Algorithm: Gram-Schmidt orthogonalization
input: a collection of linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$
$\tilde{\mathbf{q}}_1 = \mathbf{a}_1$
$\mathbf{q}_1 = \tilde{\mathbf{q}}_1 / \|\tilde{\mathbf{q}}_1\|_2$
for $i = 2, \ldots, n$
  $\tilde{\mathbf{q}}_i = \mathbf{a}_i - \sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i) \mathbf{q}_j$
  $\mathbf{q}_i = \tilde{\mathbf{q}}_i / \|\tilde{\mathbf{q}}_i\|_2$
end
output: $\mathbf{q}_1, \ldots, \mathbf{q}_n$

Gram-Schmidt for Computing Thin QR

- let $r_{ii} = \|\tilde{\mathbf{q}}_i\|_2$, $r_{ji} = \mathbf{q}_j^T \mathbf{a}_i$ for $j = 1, \ldots, i-1$
- we see that $\mathbf{a}_i = \sum_{j=1}^{i} r_{ji} \mathbf{q}_j$ for all i, or, equivalently,

$$\underbrace{\begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \ldots & \mathbf{a}_n \end{bmatrix}}_{=\mathbf{A}} = \underbrace{\begin{bmatrix} \mathbf{q}_1 & \mathbf{q}_2 & \ldots & \mathbf{q}_n \end{bmatrix}}_{=\mathbf{Q}_1} \underbrace{\begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{bmatrix}}_{=\mathbf{R}_1}$$

i.e.,

$$\mathbf{A} = \mathbf{Q}_1 \mathbf{R}_1,$$

where $\mathbf{Q}_1 = [\ \mathbf{q}_1, \ldots, \mathbf{q}_n\ ]$; $\mathbf{R}_1$ is upper triangular with $[\mathbf{R}_1]_{ji} = r_{ji}$ for $j \leq i$

Gram-Schmidt for Computing Thin QR

Algorithm: (classical) Gram-Schmidt iteration for thin QR
input: full column-rank $\mathbf{A}$
$\mathbf{Q}_1 = \mathbf{0}$, $\mathbf{R}_1 = \mathbf{0}$
$\mathbf{z} = \mathbf{A}(:,1)$, $\mathbf{R}_1(1,1) = \|\mathbf{z}\|_2$, $\mathbf{Q}_1(:,1) = \mathbf{z}/\mathbf{R}_1(1,1)$
for $i = 2, \ldots, n$
    $\mathbf{R}_1(1:i-1,i) = \mathbf{Q}_1(:,1:i-1)^T\mathbf{A}(:,i)$ % $(2m-1)i$ flops
    $\mathbf{z} = \mathbf{A}(:,i) - \mathbf{Q}_1(:,1:i-1)\mathbf{R}_1(1:i-1,i)$ % $(2i-1)m+m$ flops
    $\mathbf{R}_1(i,i) = \|\mathbf{z}\|_2$ % $2m$ flops
    $\mathbf{Q}_1(:,i) = \mathbf{z}/\mathbf{R}_1(i,i)$ % m flops
end
output: $\mathbf{Q}_1$ and $\mathbf{R}_1$

- complexity of Gram-Schmidt iteration: $\mathcal{O}(mn^2)$ ($\sum_{i=2}^{n}(4m-1)i \sim 2mn^2$)
- in the ith iteration, the ith columns of both $\mathbf{Q}$ and $\mathbf{R}$ are generated
- what if $\mathbf{A}$ is not full column-rank?
  - i.e., $\mathbf{a}_1, ..., \mathbf{a}_n$ are linear dependent, and we can find $\mathbf{z} = \mathbf{0}$ for some i, which means $\mathbf{a}_i$ is linearly dependent on $\mathbf{a}_1, ..., \mathbf{a}_{i-1}$

Algorithm: general (classical) Gram-Schmidt iteration for thin QR
input: $\mathbf{A}$
$\mathbf{Q}_1 = \mathbf{0}$, $\mathbf{R}_1 = \mathbf{0}$
$\mathbf{z} = \mathbf{A}(:,1)$
if $\mathbf{z} \neq \mathbf{0}$
    $\mathbf{R}_1(1,1) = \|\mathbf{z}\|_2$, $\mathbf{Q}_1(:,1) = \mathbf{z}/\mathbf{R}_1(1,1)$
else
    $\mathbf{R}_1(1,1) = 0$, $\mathbf{Q}_1(:,1) = \mathbf{0}$
end
for $i = 2, \ldots, n$
    $\mathbf{R}_1(1:i-1, i) = \mathbf{Q}_1(:, 1:i-1)^T \mathbf{A}(:,i)$
    $\mathbf{z} = \mathbf{A}(:,i) - \mathbf{Q}_1(:,1:i-1)\mathbf{R}_1(1:i-1,i)$
    if $\mathbf{z} \neq \mathbf{0}$
        $\mathbf{R}_1(i,i) = \|\mathbf{z}\|_2$, $\mathbf{Q}_1(:,i) = \mathbf{z}/\mathbf{R}_1(i,i)$
    else
        $\mathbf{R}_1(i,i) = 0$, $\mathbf{Q}_1(:,i) = \mathbf{0}$
    end
end
replace the $\mathbf{0}$-columns in $\mathbf{Q}_1$ to make it form a basis of $\mathbb{R}^m$
output: $\mathbf{Q}_1$ and $\mathbf{R}_1$

Gram-Schmidt for Computing Thin QR

- GS is numerically unstable due to computer rounding errors
  - say, what if $\mathbf{z}$ is close to $\mathbf{0}$?
- there are several variants with the Gram-Schmidt procedure

Modfied Gram-Schmidt for Computing Thin QR

- GS can lead to nonorthogonal $\mathbf{q}_i$'s
- denote the ith row of $\mathbf{R}_1$ as $\tilde{\mathbf{r}}_i^T$, then we define the matrix $\mathbf{A}^{(i)}_{(:,i:n)} \in \mathbb{R}^{m\times(n-i+1)}$

$$\left[\mathbf{0} \mid \mathbf{A}^{(i)}_{(:,i:n)}\right] = \mathbf{A} - \sum_{k=1}^{i-1} \mathbf{q}_k \tilde{\mathbf{r}}_k^T = \sum_{k=i}^{n} \mathbf{q}_k \tilde{\mathbf{r}}_k^T$$

or

$$\left[\mathbf{0} \mid \mathbf{a}_i^{(i)} \ \ \ldots \ \ \mathbf{a}_n^{(i)}\right] = \left[\mathbf{0} \mid \mathbf{q}_i \ \ \ldots \ \ \mathbf{q}_n\right] \begin{bmatrix} r_{ii} & \cdots & r_{in} \\ & \ddots & \vdots \\ & & r_{nn} \end{bmatrix}$$

- it follows if $\mathbf{z} = \mathbf{a}_i^{(i)}$ then $r_{ii} = \|\mathbf{z}\|_2$, $\mathbf{q}_i = \mathbf{z}/r_{ii}$, and $[\tilde{\mathbf{r}}_i^T]_{(i+1:n)} = [r_{i,i+1}, \ldots, r_{i,n}] = \mathbf{q}_i^T[\mathbf{a}_{i+1}^{(i)}, \ldots, \mathbf{a}_n^{(i)}]$
- we can compute $\mathbf{A}^{(i+1)}_{(:,i+1:n)} = [\mathbf{A}^{(i)}]_{(:,i+1:n)} - \mathbf{q}_i[\tilde{\mathbf{r}}_i^T]_{(i+1:n)}$

Modfied Gram-Schmidt for Computing Thin QR

the modfied Gram-Schmidt (MGS) iteratiom is

Algorithm: modfied Gram-Schmidt iteration for thin QR
input: full column-rank $\mathbf{A}$
$\mathbf{Q}_1 = \mathbf{0}$, $\mathbf{R}_1 = \mathbf{0}$
for $i = 1, \ldots, n$
  $\mathbf{z} = \mathbf{A}(:, i)$
  $\mathbf{R}_1(i, i) = \|\mathbf{z}\|_2$ % $2m$ flops
  $\mathbf{Q}_1(:, i) = \mathbf{z}/\mathbf{R}_1(i, i)$ % m flops
  $\mathbf{R}_1(i, i+1 : i) = \mathbf{Q}_1(:, i)^T \mathbf{A}(:, i+1 : n)$ % $(2m-1)(n-i)$ flops
  $\mathbf{A}(:, i+1 : n) = \mathbf{A}(:, i+1 : n) - \mathbf{Q}_1(:, i)\mathbf{R}_1(i, i+1 : n)$ % $m(n-i)+m$
end
output: $\mathbf{Q}_1$ and $\mathbf{R}_1$

- complexity of modified Gram-Schmidt: $\mathcal{O}(mn^2)$
- in the ith iteration, the ith column of $\mathbf{Q}$ and the ith row of $\mathbf{R}$ are generated
- GS and MGS tell us how we may compute the thin QR, but not the full QR

A Second Look on GS and MGS via Orthogonal Projections

- in classical GS, we have

$$\tilde{\mathbf{q}}_i = \mathbf{a}_i - \sum_{j=1}^{i-1} (\mathbf{q}_j^T \mathbf{a}_i)\mathbf{q}_j$$

- observe that

$$\begin{aligned}
\tilde{\mathbf{q}}_i &= \mathbf{a}_i - (\mathbf{q}_1^T \mathbf{a}_i)\mathbf{q}_1 - (\mathbf{q}_2^T \mathbf{a}_i)\mathbf{q}_2 - \ldots - (\mathbf{q}_{i-1}^T \mathbf{a}_i)\mathbf{q}_{i-1} \\
&= \mathbf{a}_i - \mathbf{q}_1\mathbf{q}_1^T \mathbf{a}_i - \mathbf{q}_2\mathbf{q}_2^T \mathbf{a}_i - \ldots - \mathbf{q}_{i-1}\mathbf{q}_{i-1}^T \mathbf{a}_i \\
&= (\mathbf{I} - \mathbf{q}_1\mathbf{q}_1^T - \mathbf{q}_2\mathbf{q}_2^T - \ldots - \mathbf{q}_{i-1}\mathbf{q}_{i-1}^T)\mathbf{a}_i \\
&= (\mathbf{I} - \mathbf{q}_{i-1}\mathbf{q}_{i-1}^T) \ldots (\mathbf{I} - \mathbf{q}_2\mathbf{q}_2^T)(\mathbf{I} - \mathbf{q}_1\mathbf{q}_1^T)\mathbf{a}_i
\end{aligned}$$

  defining $\mathbf{a}_i^{(1)} = \mathbf{a}_i$, for $j \leq i$

$$\mathbf{a}_i^{(j)} = (\mathbf{I} - \mathbf{q}_{j-1}\mathbf{q}_{j-1}^T)\mathbf{a}_i^{(j-1)} = \mathbf{a}_i^{(j-1)} - (\mathbf{q}_{j-1}^T \mathbf{a}_i^{(j-1)})\mathbf{q}_{j-1}$$

  and $\tilde{\mathbf{q}}_i = \mathbf{a}_i^{(i)}$, we obtain the update steps in MGS

  – we have rearranged the calculation in contrast to GS

Classical Gram-Schmidt vs. Modfied Gram-Schmidt

Algorithm: Classical Gram-Schmidt
input: a collection of linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$
for $i = 1, \ldots, n$
    $\tilde{\mathbf{q}}_i = \mathbf{a}_i$
end
for $i = 1, \ldots, n$
    for $j = 1, \ldots, i-1$
        $\tilde{\mathbf{q}}_i = \tilde{\mathbf{q}}_i - (\mathbf{q}_j^T \mathbf{a}_i)\mathbf{q}_j$
    end
    $\mathbf{q}_i = \tilde{\mathbf{q}}_i / \|\tilde{\mathbf{q}}_i\|_2$
end
output: $\mathbf{q}_1, \ldots, \mathbf{q}_n$
	Algorithm: Modified Gram-Schmidt
	input: a collection of linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_n$
	for $i = 1, \ldots, n$
	    $\tilde{\mathbf{q}}_i = \mathbf{a}_i$
	end
	for $i = 1, \ldots, n$
	    $\mathbf{q}_i = \tilde{\mathbf{q}}_i / \|\tilde{\mathbf{q}}_i\|_2$
	    for $j = i+1, \ldots, n$
	        $\tilde{\mathbf{q}}_j = \tilde{\mathbf{q}}_j - (\mathbf{q}_i^T \tilde{\mathbf{q}}_j)\mathbf{q}_i$
	    end
	end
	output: $\mathbf{q}_1, \ldots, \mathbf{q}_n$

Gram-Schmidt as Triangular Orthogonalization

- GS iteration is a process of "triangular orthogonalization" - making the columns of a matrix orthogonal via a sequence of matrix operations that can be intepretaed as multiplications on the right by upper-triangular matrices

$$\underbrace{\begin{bmatrix} \cdots & \mathbf{q}_{i-1} & \mathbf{a}_i^{(i)} & \ldots & \mathbf{a}_n^{(i)} \end{bmatrix}}_{=\mathbf{A}^{(i)}} \underbrace{\begin{bmatrix} \ddots & & & & \\ & 1 & & & \\ & & \frac{1}{r_{ii}} & \cdots & \frac{-r_{in}}{r_{ii}} \\ & & & 1 & \\ & & & & \ddots \end{bmatrix}}_{=\tilde{\mathbf{R}}_1^{(i)}} = \underbrace{\begin{bmatrix} \cdots & \mathbf{q}_i & \mathbf{a}_{i+1}^{(i+1)} & \ldots & \mathbf{a}_n^{(i+1)} \end{bmatrix}}_{=\mathbf{A}^{(i+1)}}$$

and hence

$$\mathbf{A}\tilde{\mathbf{R}}_1^{(1)}\tilde{\mathbf{R}}_1^{(2)}\cdots\tilde{\mathbf{R}}_1^{(n)} = \mathbf{Q}_1$$

where $\tilde{\mathbf{R}}_1^{(1)}\tilde{\mathbf{R}}_1^{(2)}\cdots\tilde{\mathbf{R}}_1^{(n)} = \mathbf{R}_1^{-1}$

- in practice, we do not form these $\tilde{\mathbf{R}}_1^{(i)}$'s, it just helps to get insight in to the structure of GS
- the above procedure looks similar to the Gauss eliminiation as well as LU decomposition in which case is a "triangular triangularization" procedure

Reflection Matrices

- a matrix $\mathbf{H} \in \mathbb{R}^{m \times m}$ is called a reflection matrix if

$$\mathbf{H} = \mathbf{I} - 2\mathbf{P},$$

  where $\mathbf{P}$ is an orthogonal projector.

- interpretation: denote $\mathbf{P}^{\perp} = \mathbf{I} - \mathbf{P}$, and observe

$$\mathbf{x} = \mathbf{P}\mathbf{x} + \mathbf{P}^{\perp}\mathbf{x}, \qquad \mathbf{H}\mathbf{x} = -\mathbf{P}\mathbf{x} + \mathbf{P}^{\perp}\mathbf{x}.$$

  The vector $\mathbf{H}\mathbf{x}$ is a reflected version of $\mathbf{x}$, with $\mathcal{R}(\mathbf{P}^{\perp})$ being the "mirror"

- a reflection matrix is orthogonal:

$$\mathbf{H}^T\mathbf{H} = (\mathbf{I} - 2\mathbf{P})(\mathbf{I} - 2\mathbf{P}) = \mathbf{I} - 4\mathbf{P} + 4\mathbf{P}^2 = \mathbf{I} - 4\mathbf{P} + 4\mathbf{P} = \mathbf{I}$$

Householder Reflections

- Problem: given $\mathbf{x} \in \mathbb{R}^m$, find an orthogonal $\mathbf{H} \in \mathbb{R}^{m \times m}$ such that

$$\mathbf{H}\mathbf{x} = \begin{bmatrix} \beta \\ \mathbf{0} \end{bmatrix} = \beta \mathbf{e}_1, \qquad \text{for some } \beta \in \mathbb{R}.$$

- Householder reflection/transformation: let $\mathbf{v} \in \mathbb{R}^m$ (Householder vector), $\mathbf{v} \neq \mathbf{0}$. Let

$$\mathbf{H} = \mathbf{I} - \frac{2}{\|\mathbf{v}\|_2^2} \mathbf{v}\mathbf{v}^T,$$

which is a reflection matrix with $\mathbf{P} = \mathbf{v}\mathbf{v}^T / \|\mathbf{v}\|_2^2$

- it can be verified that (try)

$$\mathbf{v} = \mathbf{x} \mp \|\mathbf{x}\|_2 \mathbf{e}_1 \quad \Longrightarrow \quad \mathbf{H}\mathbf{x} = \pm \|\mathbf{x}\|_2 \mathbf{e}_1;$$

the sign above may be determined to be the one that maximizes $\|\mathbf{v}\|_2$, for the sake of numerical stability

Householder QR

- let $\mathbf{H}_1 \in \mathbb{R}^{m\times m}$ be the Householder reflection w.r.t. $\mathbf{a}_1$. Transform $\mathbf{A} \in \mathbb{R}^{m\times n}$ as

$$\mathbf{A}^{(1)} = \mathbf{H}_1\mathbf{A} = \begin{bmatrix} \times & \times & \dots & \times \\ 0 & {\color{red}\times} & \dots & \times \\ \vdots & \vdots & & \vdots \\ 0 & {\color{red}\times} & \dots & \times \end{bmatrix}$$

- let $\tilde{\mathbf{H}}_2 \in \mathbb{R}^{(m-1)\times(m-1)}$ be the Householder reflection w.r.t. $\mathbf{A}^{(1)}_{2:m,2}$ (marked red above). Transform $\mathbf{A}^{(1)}$ as

$$\mathbf{A}^{(2)} = \underbrace{\begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{H}}_2 \end{bmatrix}}_{=\mathbf{H}_2} \mathbf{A}^{(1)} = \begin{bmatrix} \times & \times \quad \dots \times \\ \mathbf{0} & \tilde{\mathbf{H}}_2\mathbf{A}^{(1)}_{2:m,2:n} \end{bmatrix} = \begin{bmatrix} \times & \times & \times & \dots & \times \\ 0 & \times & \times & \dots & \times \\ \vdots & 0 & \times & \dots & \times \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & \times & \dots & \times \end{bmatrix}$$

- by repeatedly applying the trick above, we can transform $\mathbf{A}$ as the desired $\mathbf{R}$

Householder QR

- assume $m \geq n$, without loss of generality (why?)

$\mathbf{A}^{(0)} = \mathbf{A}$
for $k = 1, \ldots, n$
$\quad \mathbf{A}^{(k)} = \mathbf{H}_k \mathbf{A}^{(k-1)}$, where

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{I}_{k-1} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{H}}_k \end{bmatrix},$$

$\quad \mathbf{I}_k$ is the $k \times k$ identity matrix; $\tilde{\mathbf{H}}_k$ is the Householder reflection of $\mathbf{A}^{(k-1)}_{k:m,k}$
end

- the above procedure results in

$$\mathbf{A}^{(n)} = \mathbf{H}_n \cdots \mathbf{H}_2 \mathbf{H}_1 \mathbf{A}, \quad \mathbf{A}^{(n)} \text{ taking an upper triangular form}$$

- letting $\mathbf{R} = \mathbf{A}^{(n)}$, $\mathbf{Q} = (\mathbf{H}_n \cdots \mathbf{H}_2 \mathbf{H}_1)^T = \mathbf{H}_1 \mathbf{H}_2 \cdots \mathbf{H}_n$, we obtain the full QR
- the Householder QR procedure is a process of "orthogonal triangularization"
- a popularly used method for QR (used as "qr" in MATLAB and Julia)

Householder QR

$\mathbf{A}^{(0)} = \mathbf{A}$
for $k = 1, \ldots, n$
$\quad \mathbf{A}^{(k)} = \mathbf{H}_k \mathbf{A}^{(k-1)}$, where

$$\mathbf{H}_k = \begin{bmatrix} \mathbf{I}_{k-1} & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{H}}_k \end{bmatrix},$$

$\quad \mathbf{I}_k$ is the $k \times k$ identity matrix; $\tilde{\mathbf{H}}_k$ is the Householder reflection of $\mathbf{A}^{(k-1)}_{k:m,k}$
end

- the complexity (for $m \geq n$):
  - $\mathcal{O}(2n^2(m - n/3))$ for $\mathbf{R}$ only
    * a direct implementation of the above Householder pseudo-code does not lead us to this complexity; structures of $\mathbf{H}_k$ are exploited in the implementations to lead to this complexity (cf. matrix computation tricks in Topic 1)
  - $\mathcal{O}(4(m^2 n - mn^2 + n^3/3))$ if $\mathbf{Q}$ is also wanted

Givens Rotations

- Example: Let

$$\mathbf{J} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$$

where $c = \cos(\theta), s = \sin(\theta)$ for some θ. Consider $\mathbf{y} = \mathbf{J}\mathbf{x}$:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} cx_1 + sx_2 \\ -sx_1 + cx_2 \end{bmatrix}.$$

It can be verified that

  - $\mathbf{J}$ is orthogonal;
  - $y_2 = 0$ if $\theta = \tan^{-1}(x_2/x_1)$, or equivalently if

$$c = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}, \quad s = \frac{x_2}{\sqrt{x_1^2 + x_2^2}}.$$

Givens Rotations

- Givens rotations:

$$\begin{array}{cc} & \begin{array}{ccccc} & i & & k & \\ & \downarrow & & \downarrow & \end{array} \\ \mathbf{J}(i,k,\theta) = & \begin{bmatrix} \mathbf{I} & & & & \\ & c & & s & \\ & & \mathbf{I} & & \\ & -s & & c & \\ & & & & \mathbf{I} \end{bmatrix} \begin{array}{c} \\ \leftarrow i \\ \\ \leftarrow k \\ \\ \end{array} \end{array}$$

where $c = \cos(\theta)$, $s = \sin(\theta)$.

- $\mathbf{J}(i,k,\theta)$ is orthogonal
- let $\mathbf{y} = \mathbf{J}(i,k,\theta)\mathbf{x}$. It holds that

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

- y_k is forced to zero if we choose $\theta = \tan^{-1}(x_k/x_i)$.

Givens QR

- Example: consider a 4×3 matrix. Givens QR (from top to bottom) can be

$$
\mathbf{A} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{1,2}^{(1)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{1,3}^{(1)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{1,4}^{(1)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}
$$

$$
\xrightarrow{\mathbf{J}_{2,3}^{(2)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{2,4}^{(2)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{3,4}^{(3)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix} = \mathbf{R}
$$

Givens QR

or (from bottom to top)

$$\mathbf{A} = \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{3,4}^{(1)}} \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{2,3}^{(1)}} \begin{bmatrix} \times & \times & \times \\ \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{1,2}^{(1)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \end{bmatrix}$$

$$\xrightarrow{\mathbf{J}_{3,4}^{(2)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{2,3}^{(2)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & \times \end{bmatrix} \xrightarrow{\mathbf{J}_{3,4}^{(3)}} \begin{bmatrix} \times & \times & \times \\ 0 & \times & \times \\ 0 & 0 & \times \\ 0 & 0 & 0 \end{bmatrix} = \mathbf{R}$$

where $\mathbf{B} \xrightarrow{\mathbf{J}} \mathbf{C}$ means $\mathbf{B} = \mathbf{J}\mathbf{C}$; $\mathbf{J}_{i,k}^{(j)} = \mathbf{J}^{(j)}(i, k, \theta)$, with θ chosen to zero out the (k, j)th entry of the matrix transformed by $\mathbf{J}_{i,k}^{(j)}$.

Givens QR

- **Givens QR:** assume $m \geq n$. Perform a sequence of Givens rotations to annihilate the lower triangular parts of $\mathbf{A}$ to obtain $\mathbf{R}$, say

$$\underbrace{(\mathbf{J}_{n,m}^{(n)} \ldots \mathbf{J}_{n,n+2}^{(n)} \mathbf{J}_{n,n+1}^{(n)}) \ldots (\mathbf{J}_{2m}^{(2)} \ldots \mathbf{J}_{24}^{(2)} \mathbf{J}_{23}^{(2)})(\mathbf{J}_{1m}^{(1)} \ldots \mathbf{J}_{13}^{(1)} \mathbf{J}_{12}^{(1)})}_{=\mathbf{Q}^T} \mathbf{A} = \mathbf{R}$$

  where $\mathbf{R}$ takes the upper triangular form, and $\mathbf{Q}$ is orthogonal.

- the Givens QR procedure is a process of "orthogonal triangularization"
- complexity (for $m \geq n$): $\mathcal{O}(3n^2(m - n/3))$ for $\mathbf{R}$ only
- not as efficient as Householder QR for general (and dense) $\mathbf{A}$'s
  - the flop count for Householder QR is $2n^2(m - n/3)$ (for $\mathbf{R}$ and for $m \geq n$)
  - the flop count for Givens QR is $3n^2(m - n/3)$
- can be faster than Householder QR if $\mathbf{A}$ has certain sparse structures and we exploit them

Method of Normal Equations vs. QR for LS

- In terms of complexity, method of normal equations only needs half of the arithmetic compared to QR decompostion when $m \gg n$.

- Method of normal equations can be easy for implementation, however, it is not recommended due to its numerical instability.
  - By forming the product $\mathbf{A}^T\mathbf{A}$, we square the condition number of $\mathbf{A}$. (cf. Topic 7: SVD)

- Thus, using the QR decomposition yields a better least-squares estimate than the normal equations in terms of solution quality.

Solving Underdetermined Systems by QR

For $\mathbf{A} \in \mathbb{R}^{m \times n}$ with $m < n$ and $\operatorname{rank}(\mathbf{A}) = m$, we have

$$\mathbf{A}^T = \mathbf{Q}\mathbf{R} = \begin{bmatrix} \mathbf{Q}_1 & \mathbf{Q}_2 \end{bmatrix} \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1 + \mathbf{Q}_2\mathbf{0}$$

- note

  $$\mathbf{A}\mathbf{x} = \mathbf{R}_1^T\mathbf{Q}_1^T\mathbf{x} + \mathbf{0}^T\mathbf{Q}_2^T\mathbf{x} = \mathbf{b}$$

  which indicates

  $$\mathbf{Q}_1^T\mathbf{x} = \mathbf{R}_1^{-T}\mathbf{b}$$

  and $\mathbf{Q}_2^T\mathbf{x}$ can be anything, which we set to be $\mathbf{d}$. Then we have

  $$\begin{bmatrix} \mathbf{Q}_1^T\mathbf{x} \\ \mathbf{Q}_2^T\mathbf{x} \end{bmatrix} = \mathbf{Q}^T\mathbf{x} = \begin{bmatrix} \mathbf{R}_1^{-T}\mathbf{b} \\ \mathbf{d} \end{bmatrix}$$

- the solution is

  $$\mathbf{x} = \mathbf{Q} \begin{bmatrix} \mathbf{R}_1^{-T}\mathbf{b} \\ \mathbf{d} \end{bmatrix} = \mathbf{Q}_1\mathbf{R}_1^{-T}\mathbf{b} + \mathbf{Q}_2\mathbf{d}$$

  where to get the minimum norm solution, we can set $\mathbf{d} = \mathbf{0}$.

Other Contents on QR

- QR with column pivoting (cf. Section 5.4.2 in **[Golub-Van Loan'13]**)
- QR algorithm for computing eigenvalues (cf. Topic 5)
- QR algorithm for computing SVD (cf. Topic 7)

References

[Golub-Van Loan'13] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th edition, JHU Press, 2013.