Linear Algebra

Orthogonal Matrices and QR Decomposition

Automotive Intelligence Lab.

Contents

Orthogonal matrices

Introduction of Orthogonal Matrices

- **Important and special matrices for several decompositions**
  - QR decomposition
  - Eigen decomposition
  - Singular value decomposition

- **Letter $\boldsymbol{Q}$**
  - Often used to indicate orthogonal matrices.

Mathematical Expression of Orthogonal Matrices

Two properties of orthogonal matrices

- Orthogonal columns
  - All columns are $\boxed{}$ orthogonal.
- Unit-norm columns
  - The norm (geometric length) of each column is exactly $\boxed{}$.

Translate those two properties into a mathematical expression.

- $\langle a, b \rangle$: alternative notation for the dot product
- q_i: i^{th} column of matrix

$$\langle q_i, q_j \rangle = \begin{cases} 0, & if \;\; i \neq j \\ 1, & if \;\; i = j \end{cases}$$

Mathematical expression of orthogonal matrices

- Dot product of a column with itself is 1.
- Dot product of a column with any other column is 0.

Characteristic of Orthogonal Matrices

- **Definition of matrix multiplication**
  - Dot products between all rows of the left matrix with all columns of the right matrix

- **Q^T is a matrix that multiplies Q to produce the identity matrix.**
  - Exact same definition as the **matrix** $\boxed{}$.
  - Inverse of an orthogonal matrix is its $\boxed{}$.
    - Matrix inverse: tedious and prone to numerical in accuracies.
    - Matrix transpose: fast and accurate.

- **Identity matrix is an example of an orthogonal matrix.**

$$Q^T Q = I$$

Characteristic of an orthogonal matrix

Example of Orthogonal Matrices

Practice in MATLAB with below matrices

- Does each column have unit length?
  - $\boxed{}$.
- Is each column orthogonal to other columns?
  - $\boxed{}$.
- Compute $\boldsymbol{QQ^T}$.
  - Is that still the identity matrix? Try it to find out!

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}, \qquad \frac{1}{3}\begin{bmatrix} 1 & 2 & 2 \\ 2 & 1 & -2 \\ -2 & 2 & -1 \end{bmatrix}$$

Example of an orthogonal matrices

```
% Clear workspace, command
window, and close all
figures
clc; clear; close all;

% Matrices Q1 and Q2
Q1 = [1 -1; 1 1]/sqrt(2);
Q2 = [1 2 2; 2 1 -2; -2 2 -
1]/3;

% Orthogonal matrices
Q1TQ1 = Q1' * Q1;
Q2TQ2 = Q2' * Q2;

% Display results
disp("Q1T * Q1");
disp(Q1TQ1);
disp("Q2T * Q2");
disp(Q2TQ2);
```

MATLAB code to compute Q^TQ

Gram-Schmidt

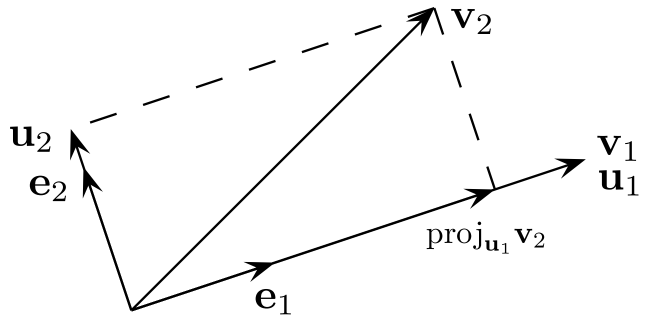
Process of Gram-Schmidt

- **Way of making two or more vectors perpendicular to each other**

- **Technical definition of Gram Schmidt**
  - Method of constructing an $\boxed{}$ basis
    - From a set of vectors in an **inner space**.
    - Most commonly Euclidean space $\boldsymbol{R^n}$ equipped with standard inner product.

- **Takes a finite, linearly independent set of vectors $\boldsymbol{S} = \{\boldsymbol{v_1}, \dots, \boldsymbol{v_k}\}$.**
  - Generate an orthogonal set $\boldsymbol{S'} = \{\boldsymbol{u_1}, \dots, \boldsymbol{u_k}\}$.
    - Spans the same $k-dimensional$ subspace of R^n as $\boldsymbol{S}$.

- **Application to column vectors of full column rank matrix**
  - Yields the $\boldsymbol{QR}$ decomposition.
    - Decomposed into orthogonal and a triangular matrix.
      - We will study QR decomposition in next section!



Basic principles of the Gram-Schmidt process

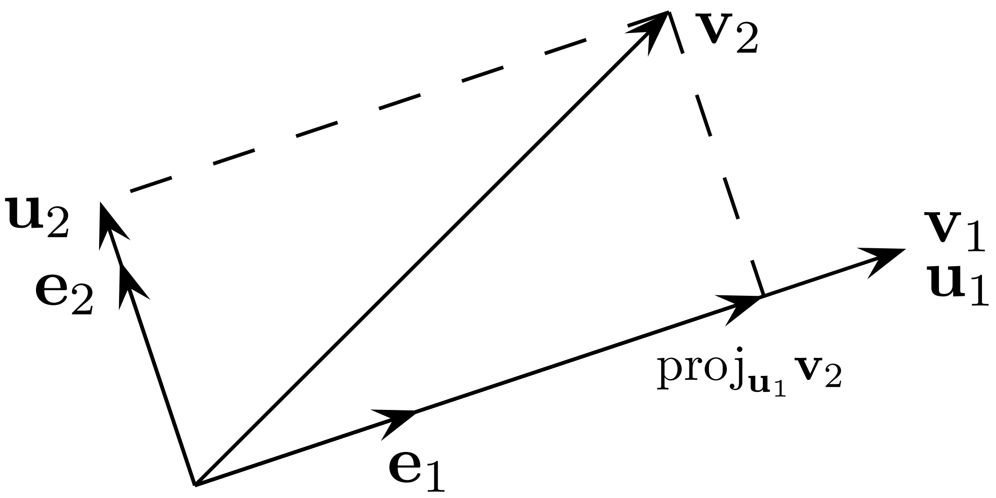
Reference: https://en.wikipedia.org/wiki/Gram%E2%80%93Schmidt_process

Vector Projection

- **Vector projection of a vector $\boldsymbol{v}$ on a nonzero vector $\boldsymbol{u}$.**
  - $<\boldsymbol{v},\boldsymbol{u}>$: inner product of vectors $\boldsymbol{v}$ and $\boldsymbol{u}$.
  - $proj_{\boldsymbol{u}}(\boldsymbol{v})$: orthogonal projection of $\boldsymbol{v}$ onto the line spanned by $\boldsymbol{u}$.
  - If $\boldsymbol{u}$ is zero vector,
    - $proj_u\boldsymbol{v}$ is defined as a zero vector.

$$\boldsymbol{proj_u}(\boldsymbol{v}) = \frac{<\boldsymbol{v},\boldsymbol{u}>}{<\boldsymbol{u},\boldsymbol{u}>}\boldsymbol{u}$$

Vector projection



Expression of Gram-Schmidt Using Vector Projection

- **Given k vectors** $v_1, \ldots, v_k$.
  - Gram-Schmidt process defines vectors $u_1, \ldots, u_k$ as shown in below expression.
    - $\boldsymbol{u_1}, \ldots, \boldsymbol{u_k}$ is required system of orthogonal vectors.
      - Known as Gram-Schmidt $\boxed{}$.
    - Normalized vector $\boldsymbol{e_1}, \ldots \boldsymbol{e_k}$ form an orthonormal set.
      - Known as Gram-Schmidt $\boxed{}$.

$$\boldsymbol{u_1} = \boldsymbol{v_1} \qquad\qquad \boldsymbol{e_1} = \frac{\boldsymbol{u_1}}{\|\boldsymbol{u_1}\|}$$

$$\boldsymbol{u_2} = \boldsymbol{v_2} - \boldsymbol{proj_{u_1}}(\boldsymbol{v_2}) \qquad\qquad \boldsymbol{e_2} = \frac{\boldsymbol{u_2}}{\|\boldsymbol{u_2}\|}$$

$$\boldsymbol{u_3} = \boldsymbol{v_3} - \boldsymbol{proj_{u_1}}(\boldsymbol{v_3}) - \boldsymbol{proj_{u_2}}(\boldsymbol{v_3}) \qquad\qquad \boldsymbol{e_2} = \frac{\boldsymbol{u_2}}{\|\boldsymbol{u_2}\|}$$

$$\vdots \qquad\qquad \vdots$$

$$\boldsymbol{u_k} = \boldsymbol{v_k} - \sum_{j=1}^{k-1} \boldsymbol{proj_{u_j}}(\boldsymbol{v_k}) \qquad\qquad \boldsymbol{e_k} = \frac{\boldsymbol{u_k}}{\|\boldsymbol{u_k}\|}$$

Expression of Gram-Schmidt using vector projection

Check Formula Validity

$$u_1 = v_1 \qquad e_1 = \frac{u_1}{\|u_1\|}$$

$$u_2 = v_2 - proj_{u_1}(v_2) \qquad e_2 = \frac{u_2}{\|u_2\|}$$

$$u_3 = v_3 - proj_{u_1}(v_3) - proj_{u_2}(v_3) \qquad e_2 = \frac{u_2}{\|u_2\|}$$

$$\vdots \qquad \vdots$$

$$u_k = v_k - \sum_{j=1}^{k-1} proj_{u_j}(v_k) \qquad e_k = \frac{u_k}{\|u_k\|}$$

Expression of Gram-Schmidt using vector projection

- **First, compute $< u_1, u_2 >$ and check the result is zero.**
  - Substituting previous formula for u_2.
    - $u_2 = v_2 - proj_{u_1}(v_2)$
- **Then, compute $< u_1, u_3 >$ and check the result is zero.**
  - Substituting previous formula for u_3.
    - $u_3 = v_3 - proj_{u_1}(v_3) - proj_{u_2}(v_3)$

Geometrically Check Formula Validity

$$u_1 = v_1 \qquad e_1 = \frac{u_1}{\|u_1\|}$$

$$u_2 = v_2 - proj_{u_1}(v_2) \qquad e_2 = \frac{u_2}{\|u_2\|}$$

$$u_3 = v_3 - proj_{u_1}(v_3) - proj_{u_2}(v_3) \qquad e_2 = \frac{u_2}{\|u_2\|}$$

$$\vdots \qquad \vdots$$

$$u_k = v_k - \sum_{j=1}^{k-1} proj_{u_j}(v_k) \qquad e_k = \frac{u_k}{\|u_k\|}$$

Expression of Gram Schmidt using vector projection

- **To compute $\boldsymbol{u_i}$,**
  - Project $\boldsymbol{v_i}$ orthogonally onto subspace U.
    - U: generated by $u_1, \dots, u_{i-1}$
      - Same as subspace generated by $v_1, \dots, v_{i-1}$
    - Vector $\boldsymbol{u_i}$ defined to be the difference between $\boldsymbol{v_i}$.
  - This projection is guaranteed to be **orthogonal to all vectors in the subspace $\boldsymbol{U}$**.

Euclidean Space

- **Consider following set of vectors in R^2 as Eq 1..**
  - With conventional inner product.
- **Then, perform Gram-Schmidt as Eq 2..**
  - To obtain orthogonal set of vectors!

$$\boldsymbol{S} = \left\{\boldsymbol{v}_1 = \begin{bmatrix}3\\1\end{bmatrix}, \boldsymbol{v}_2 = \begin{bmatrix}2\\2\end{bmatrix}\right\}$$

Eq 1. Set of vectors

$$\boldsymbol{u}_1 = \boldsymbol{v}_1 = \begin{bmatrix}3\\1\end{bmatrix}$$

$$\boldsymbol{u}_2 = \boldsymbol{v}_2 - proj_{\boldsymbol{u}_1}(\boldsymbol{v}_2) = \begin{bmatrix}2\\2\end{bmatrix} - proj_{\begin{bmatrix}3\\1\end{bmatrix}}\begin{bmatrix}2\\2\end{bmatrix} = \begin{bmatrix}2\\2\end{bmatrix} - \frac{8}{10}\begin{bmatrix}3\\1\end{bmatrix} = \begin{bmatrix}-2/5\\6/5\end{bmatrix}$$

Eq 2. Gram-Schmidt

Check Whether Orthogonal or Not

- **Check that vectors $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ are indeed orthogonal as Eq 1..**
  - If dot product of two vectors is $\mathbf{0}$, then they are $\boxed{}$.
- **In case of non-zero vectors,**
  - We can normalize vectors by dividing out their sizes as Eq 2..

$$\langle \boldsymbol{u}_1, \boldsymbol{u}_2 \rangle = \left\langle \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \begin{bmatrix} -2/5 \\ 6/5 \end{bmatrix} \right\rangle = -\frac{6}{5} + \frac{6}{5} = 0$$

Eq 1. Dot product of two vectors

$$\boldsymbol{e}_1 = \frac{1}{\sqrt{10}} \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

$$\boldsymbol{e}_2 = \frac{1}{\sqrt{\frac{40}{25}}} \begin{bmatrix} -2/5 \\ 6/5 \end{bmatrix} = \frac{1}{\sqrt{10}} \begin{bmatrix} -1 \\ 3 \end{bmatrix}$$

Eq 2. Normalizing vectors

Code Exercise of Gram-Schmidt algorithm using MATLAB

Code Exercise (09_01)

- Follow the order of Gram-Schmidt algorithm in previous slide.

```
%% Gram-Schmidt Algorithm

% Clear workspace, command window, and close all figures
clc; clear; close all;

% Initialize the matrices
A = [8 1 6; 3 5 7; 4 9 2];
Q = zeros(3);

% Perform the Gram-Schmidt process
for i = 1:size(A, 2)
    % Start with the original vector
    v = A(:, i);

    % Subtract the projections onto all previously obtained orthogonal vectors
    for j = 1:i-1
        v = v - (Q(:, j)' * A(:, i)) / (Q(:, j)' * Q(:, j)) * Q(:, j);
    end

    % Normalize the vector to make it orthogonal
    Q(:, i) = v / norm(v);
end

% Display the original and orthogonalized matrices
disp('Original Matrix A:');
disp(A);
disp('Orthogonalized Matrix Q:');
disp(Q);

% Verify orthogonality by computing dot proudct
disp('Dot products between different vectors of Q (should be close to zero):');
for i = 1:size(Q, 2)
    for j = i+1:size(Q, 2)
        fprintf('Dot product between Q(:, %d) and Q(:, %d): %f\n', i, j, dot(Q(:, i), Q(:, j)));
    end
end
```

MATLAB code

QR decomposition

Definition of QR Decomposition

- **Decomposition matrix with $\boxed{}$ vector which is found using Gram-Schmidt.**

- **Matrix $\boldsymbol{Q}$**
  - A set of standard orthogonal basis $q_1, \cdots, q_n$ obtained through the Gram-Schmidt
  - $\boldsymbol{Q}$ is obviously different from the original matrix.
    - Assuming original matrix was not orthogonal.
    - $\boxed{}$ information about that matrix.

- **Fortunately, lost information can be retrieved and stored in another matrix $\boldsymbol{R}$.**
  - $\boldsymbol{R}$ multiplied to $\boldsymbol{Q}$.
  - Then…, how to create $\boldsymbol{R}$?

Creating R

- **Comes right from the definition of QR.**

$$A = QR$$

$$Q^T A = Q^T QR$$

$$Q^T A = R$$

Definition of QR

- **Advantage of orthogonal matrices that can be seen from the above definition.**
  - ▶ Solve matrix equations **without** worrying about **computing the inverse**.

$$\begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} a_1 \cdot q_1 & a_2 \cdot q_1 & \cdots & a_n \cdot q_1 \\ a_1 \cdot q_2 & a_2 \cdot q_2 & \cdots & a_n \cdot q_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_1 \cdot q_n & a_2 \cdot q_n & \cdots & a_n \cdot q_n \end{bmatrix}$$

Overall form of QR decomposition

Simplification of QR Decomposition

- **Consider $\boldsymbol{a_1 \cdot q_2}$.**
  - $a_1 \cdot q_2 = 0$ because a_1 is orthogonal to q_2 .

- **For $\boldsymbol{a_i \cdot q_j, i < j}$**
  - $a_i \cdot q_j = 0$
    - Because a_i is orthogonal to q_j for $\boldsymbol{i < j}$.

$$\boldsymbol{q_1 = a_1}$$

$$\boldsymbol{q_2 = a_2 - proj_{q_1}(a_2)}$$

$$\boldsymbol{q_3 = a_3 - proj_{q_1}(a_3) - proj_{q_2}(a_3)}$$

$$\vdots$$

$$\boldsymbol{q_k = a_k - \sum_{j=1}^{k-1} proj_{q_j}(a_k)}$$

$$= \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} a_1 \cdot q_1 & a_2 \cdot q_1 & \cdots & a_n \cdot q_1 \\ 0 & a_2 \cdot q_2 & \cdots & a_n \cdot q_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_n \cdot q_n \end{bmatrix}$$

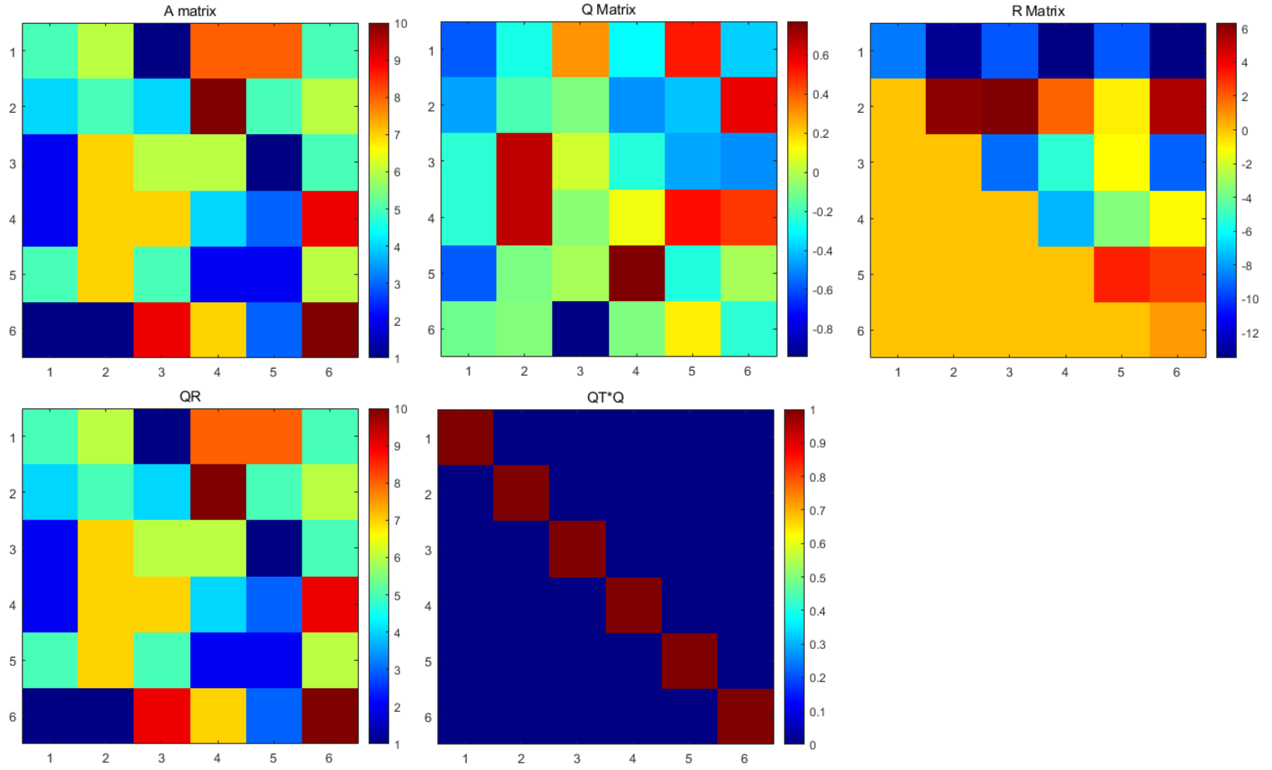
Simplification of QR decomposition

Features of QR Decomposition

- $A = QR$
  - $A - QR$ is zeros matrix.
- Q **times its transpose gives the identity matrix.**
- R **matrix: always upper triangular**
  - It will be explained in the next section.

```
% Clear workspace, command window, and close
all figures
clc; clear; close all;

% Random integer matrix A
A = randi(10, 6);

% QR decomposition
[Q,R] = qr(A);

% Visualize the results
figure;
imagesc(A); % Display the matrix as a color
image
title('A matrix');
colorbar; % Show a color scale
colormap jet; % Use the jet color map
axis equal tight;  % Adjust axes to fit the
data

figure;
imagesc(Q);
title('Q Matrix');
colorbar;
colormap jet;
axis equal tight;

figure;
imagesc(R);
title('R Matrix');
colorbar;
colormap jet;
axis equal tight;

figure;
imagesc(Q*R);
title('QR');
colorbar;
colormap jet;
axis equal tight;

figure;
imagesc(Q' * Q);
title('QT*Q');
colorbar;
colormap jet;
axis equal tight;
```

MATLAB code



QR decomposition of a random numbers matrix

Sizes of Q and R

- **Depend on the size of to-be-decomposed matrix A.**

- **Whether QR decomposition is economy or full.**
  - **Economy** called reduced.
  - **Full** called complete.

Overview of All Possible Sizes of Q and R

- **Fig 1. shows an overview of all possible sizes.**
- **“?” indicates that the matrix elements depend on values in A.**
  - Not identity matrix.

	A	Q	Q^TQ	QQ^T	R
Square full-rank	$M \times M$ $r = M$	$M \times M$ $r = M$	I_M	I_M	$M \times M$ $r = M$
Square singular	$M \times M$ $r = K < M$	$M \times M$ $r = M$	I_M	I_M	$M \times M$ $r = k$
Tall “full”	$M > N$ $r = K$	$M \times M$ $r = M$	I_M	I_M	$M \times M$ $r = k$
Tall “economy”	$M > N$ $r = K$	$M \times N$ $r = N$	I_N	?	$M \times N$ $r = K$
Wide	$M < N$ $r = K$	$M \times M$ $r = M$	I_M	I_M	$M \times N$ $r = K$

Fig 1. Sizes of Q and R depending on size of A

Code Exercise of Orthogonal Matrix using MATLAB

Code Exercise (09_02)

- Notice optional second input ‘complete’, which produces a full QR decomposition.
- Setting that to ‘reduced’, gives economy-mode QR decomposition,
  in which $\boldsymbol{Q}$ is same size as $\boldsymbol{A}$.

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Matrix A
A = [1; -1];
[Q,R] = qr(A); % Full QR decomposition
[Q_econ,R_econ] = qr(A, "econ"); % Economy-mode QR decomposition, Q is same size as matrix A

% Scale to make integer matrix
Q = Q*sqrt(2);
Q_econ = Q_econ*sqrt(2);

% Display the results
disp("Q")
disp(Q);
disp("Q_econ")
disp(Q_econ);
```

MATLAB code of orthogonal matrix

Rank of Orthogonal Matrix

- **Rank of $\boldsymbol{Q}$ is always maximum possible rank.**
  - It is possible to craft more than $M > N$ orthogonal vectors from a matrix with N columns.

- **Rank of $\boldsymbol{Q}$**
  - $\boldsymbol{M}$ for all square $\boldsymbol{Q}$ matrices
  - $\boldsymbol{N}$ for economy $\boldsymbol{Q}$ matrices

- **Rank of $\boldsymbol{R}$**
  - Same as rank of $\boldsymbol{A}$.

- **Difference in rank between $\boldsymbol{Q}$ and $\boldsymbol{A}$ resulting from orthogonalization**
  - Q spans all of $\mathbb{R}^{\square}$ even if the column space of $\boldsymbol{A}$ is only lower-dimensional subspace of $\mathbb{R}^{\square}$.
    - Important reason why the singular value decomposition is so useful for revealing properties of a matrix, including its rank and null space.
  - Another reason to look forward to learning about SVD in Chapter 14!

Property of QR Decomposition

- **QR decomposition is not unique for all matrix sizes and ranks.**
  - It is possible to obtain $\boldsymbol{A} = \boldsymbol{Q}_1\boldsymbol{R}_1$ and $\boldsymbol{A} = \boldsymbol{Q}_2\boldsymbol{R}_2$ where $\boldsymbol{Q}_1 \neq \boldsymbol{Q}_2$.
- **All QR decomposition results have the same properties described in this section.**
- **QR decomposition can be made unique when given additional constraints.**
    - E.g., positive values on diagonals of $\boldsymbol{R}$.
  - But! **Not necessary** in most cases.
    - Not implemented in MATLAB.

Orthogonalization

- **Orthogonalization works column-wise from left to right.**
  - Later columns in $\boldsymbol{Q}$ are orthogonalized to earlier columns of $\boldsymbol{A}$.
- **Lower triangle of $\boldsymbol{R}$ comes from** $\boxed{}$**.**
- **Earlier columns in $\boldsymbol{Q}$ are not orthogonalized to later columns of $\boldsymbol{A}$.**
  - No expect their dot products to be zero.

- **Columns $\boldsymbol{i}$ and $\boldsymbol{j}$ of $\boldsymbol{A}$ were already orthogonal.**
  - Corresponding $(i, j)^{th}$ element in $\boldsymbol{R}$ would be zero.
- **If compute QR decomposition of orthogonal matrix,**
  - $\boldsymbol{R}$ will be $\boxed{}$ matrix.
    - Norms of each column in $\boldsymbol{A}$.
- **If $\boldsymbol{A} = \boldsymbol{Q}$, $\boldsymbol{R}$ is same as $\boldsymbol{I}$.**
  - Comes from equation solved for $\boldsymbol{R}$.

QR and Inverses

- **More numerically stable way to compute matrix inverse**
  - When using QR decomposition.
- **Writing out QR decomposition formula and inverting both sides of equation.**
  - Apply the **LIVE EVIL** rule as we learned before.
- **Inverse of $\boldsymbol{A}$**
  - Same as $\boxed{}$ of $\boldsymbol{R}$ times $\boxed{}$ of $\boldsymbol{Q}$.
  - $\boldsymbol{Q}$ is numerically stable.
    - Due to **Householder reflection algorithm**.
  - $\boldsymbol{R}$ is numerically stable.
    - Due to results from **matrix multiplication**.
- **Need to invert $\boldsymbol{R}$ explicitly.**
  - **Inverting triangular matrices** is highly numerically stable.
    - Through back substitution.

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$$

$$\boldsymbol{A}^{-1} = (\boldsymbol{Q}\boldsymbol{R})^{-1}$$

$$\boldsymbol{A}^{-1} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{-1}$$

$$\boldsymbol{A}^{-1} = \boldsymbol{R}^{-1}\boldsymbol{Q}^{T}$$

Compute matrix inverse using QR decomposition

Key Point of QR Decomposition

- **Provide more numerically stable way to invert matrices.**
  - Compared to algorithm presented in previous lecture.
- **On the other hand, some matrices are still very difficult to invert.**
  - Theoretically invertible but are close to singular.
- **QR decomposition doesn’t guarantee high-quality inverse.**
  - Rotten apple dipped in honey is still rotten…!

Summary

Summary

- **Orthogonal matrix**
  - All columns are pair-wise orthogonal and norm equals to 1.
  - Key to several matrix decompositions.
    - QR, eigen, singular value decomposition.
  - Important in geometry and computer graphics.
    - E.g. pure rotation matrices.

- **Can transform a nonorthogonal matrix into an orthogonal matrix.**
  - Via Gram-Schmidt procedure.
  - Involves applying orthogonal vector decomposition.
    - To isolate the component of each column.
    - Each column is orthogonal to all previous columns, previous meaning left to right.

- **QR decomposition is the result of Gram-Schmidt.**
  - Technically, it is implemented by more stable algorithm.
  - But GS is still the right way to understand it.

Code exercises

Characteristic of matrix Q

- A square Q has the following equalities:

$$Q^T Q = QQ^T = Q^{-1}Q = QQ^{-1} = I$$

- Demonstrate this in code by computing Q from a random-numbers matrix, then compute Q^T and Q^{-1}. Then show that all four expressions produce the identity matrix.
- https://kr.mathworks.com/help/matlab/ref/qr.html

```
% Clear workspace, command window, and close all figures
clc; clear; close all;

% Generate a 5x5 random matrix and compute the QR decomposition
random_matrix = randn(5, 5);

%%%%%%% TODO %%%%%%%
% Generate Q matrix
[Q, R] = ;

% Get Transpose of Q & Inverse of Q
Qt = ; % Transpose of Q
Qi = ; % Inverse of Q

% QtQ
disp("QtQ")
disp(round(, 8));

% QQt
disp("QQt")
disp(round(, 8));

% QiQ
disp("QiQ")
disp(round(, 8));

% QQi
disp("QQi")
disp(round(, 8));
%%%%%%% TODO %%%%%%%
```

Sample code

Full, Economy Sized matrix Q and Its Inverse

- This exercise will highlight one feature of the R matrix that is relevant for under-standing how to use QR to implement least squares (lecture 12): when A is tall and full column-rank, the first N rows of R are upper-triangular, whereas rows $N+1$ through M are zeros. Confirm this in MATLAB using a random 10×4 matrix. Make sure to use the complete (full) QR decomposition, not the economy (compact) decomposition.

- Of course, R is noninvertible because it is nonsquare. But (1) the submatrix comprising the first N row is square and full-rank (when A us full column-rank) and thus has a full inverse, and (2) the tall R has a pseudoinverse. Compute both inverses, and confirm that the full inverse of the first N rows of R equals the first N columns of the pseudoinverse of the tall R.

```
% Create a random 10x4 matrix
A = randn(10, 4);

% Compute the complete QR decomposition
% economy sized R
[~, R] = ;
% full sized R
[~, fullR] = ;

% Examine R (rounded to 3 decimal places)
disp('R:');
disp(round(R, 3));
disp('fullR:');
disp(round(fullR, 3));
```

```
% Invertible submatrix (first 4x4 part of R)
Rsub = ;


% Inverses
% calculate full inverse of Rsub
Rsub_inv = ;
% calculate left inverse of R
Rleftinv = ;


% Display both inverses
disp('Full inverse of R submatrix:');
disp(round(Rsub_inv, 3));


disp('Left inverse of R:');
disp(round(Rleftinv, 3));
```

Sample code

THANK YOU
FOR YOUR ATTENTION