

Numerical methods for Regression

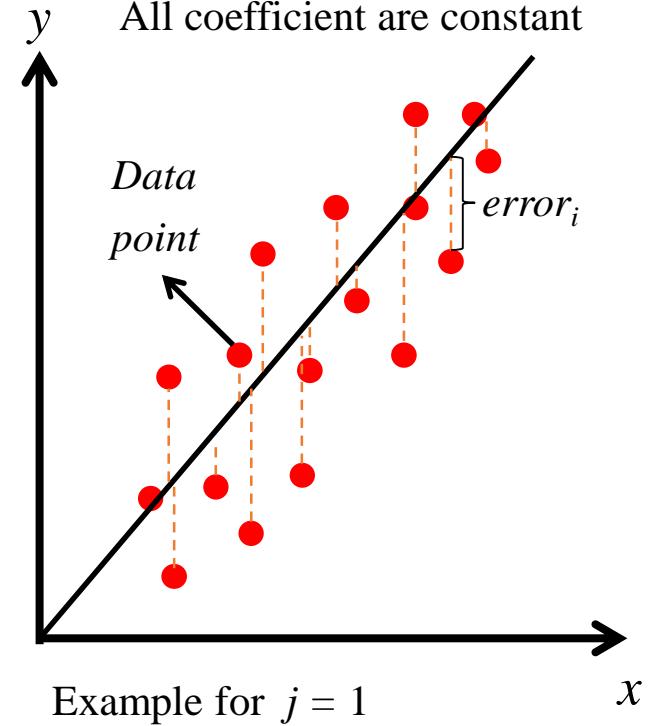
Concept for Least-Square

- **Concept:** There are many data (x_i, y_i) , after setting on the x - y plane, the distribution may be fitted by some curve $y = f(x)$ or $f(x, y) = k$, k is some constant, $\forall x, y, k \in \mathbb{R}$.
- Define $error_i = y_i - y$
- Total error $E = \sum_{i=1}^n (y_i - y)^2$
- **Want:** E is minimum value.

$$y = \sum_{j=0}^n a_j x^j \quad n = 2, 4$$

Fitting curve: $y = a_1 x + a_0$

All coefficient are constant



Derive the Mathematical Eq.

Existence and Uniqueness

- For linear regression, we use $y = bx + a$ to fit our data. If there are n group of data (x_i, y_i) , $i = 1, 2, \dots, n$, then, we have n group of linear equations system

$$\begin{cases} y_1 = bx_1 + a \\ \vdots \\ y_n = bx_n + a \end{cases} \xrightarrow{\text{Matrix representation}} \begin{matrix} \mathbf{Y} & \mathbf{A} \\ \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} & \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \\ n \times 1 & n \times 2 \end{matrix} \xrightarrow{2 \times 1} \mathbf{Y} = \mathbf{Ax}$$

- In this case, there are 2 unknown parameters, so n must be bigger than 2 so that there **may exist** a and b **uniquely**.
- Lemma 1: Let $A \in M_{m \times n}(F)$, $x \in F^n$, $y \in F^m$, $m \geq n$ then
 $\langle Ax, y \rangle_m = \langle x, A^T y \rangle_n$, \langle , \rangle : inner product
- Lemma 2: Let $A \in M_{m \times n}(F)$, $\text{rank}(A) = \text{rank}(A^T A)$
- Corollary: Let $A \in M_{m \times n}(F)$ and $\text{rank}(A) = n$, then $A^T A$ is invertible

Derive the Mathematical Eq.

Existence and Uniqueness

Theorem of Least-square approximation:

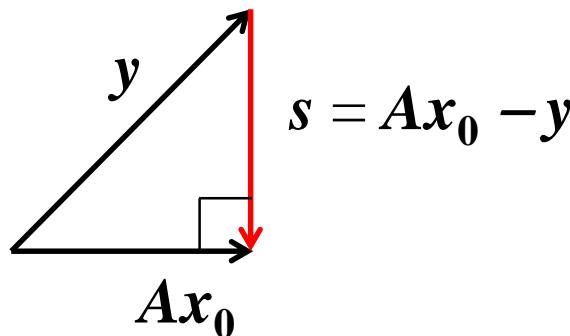
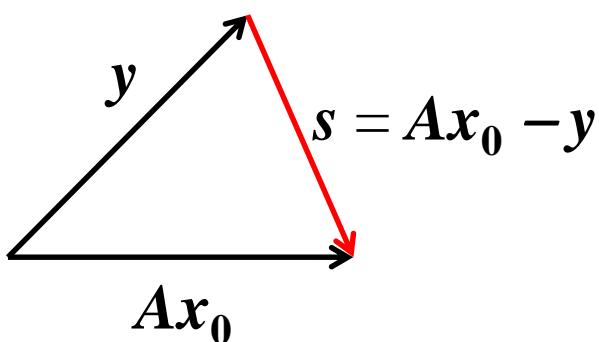
Let $A \in M_{m \times n}(F)$, $m \geq n$, $y \in F^m$, then $\exists x_0 \in F^n$ s.t.

$$(A^T A) x_0 = A^T y \text{ and } \|Ax_0 - y\| \leq \|Ax - y\|, x \in F^n$$

Furthermore, if $\text{rank}(A) = n$ (full rank), then $x_0 = (A^T A)^{-1} A^T y$

If $Ax = y$ is consistent, then $\exists!$ solution s

pf: In the view point of geometry, if there exists 2 vectors y and Ax_0 , $s = Ax_0 - y$, when $s \perp Ax_0$, then $\|s\|$ is minimum and exactly one
 $\rightarrow \langle Ax, s \rangle = \langle x, A^T s \rangle = \langle x, A^T(Ax_0 - y) \rangle = 0 \rightarrow$



For $x \neq 0$, $A^T A x_0 = A^T y$

$$x_0 = (A^T A)^{-1} A^T y$$

Derive the Mathematical Eq. Extend to monomial Regression

- Total error $E = \sum_{i=1}^n (y - y_i)^2 = \sum_{i=1}^n (y - bx_i - a)^2$, just a **parabolic eq.** with concave up, so there exists a minimum value. By calculus,

$$\begin{cases} \frac{\partial E}{\partial b} = 0 = \sum_{i=1}^n -2(x_i)(y_i - bx_i - a) \\ \frac{\partial E}{\partial a} = 0 = \sum_{i=1}^n -2(y_i - bx_i - a) \end{cases} \rightarrow \begin{cases} \sum_{i=1}^n x_i y_i = \sum_{i=1}^n (bx_i^2 + ax_i) \\ \sum_{i=1}^n y_i = \sum_{i=1}^n (bx_i + ax_i) \end{cases}$$

↓

$$\underbrace{\begin{bmatrix} x_1 & \dots & x_n \\ 1 & \dots & 1 \end{bmatrix}}_{A^T} \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_Y = \underbrace{\begin{bmatrix} x_1 & \dots & x_n \\ 1 & \dots & 1 \end{bmatrix}}_{A^T} \begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} \leftarrow \begin{bmatrix} \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n y_i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & 1 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix}$$

- Here we take the **derivative on the coefficient**, not x and y , so we can apply this method to our regression.

Derive the Mathematical Eq. Extend to monomial Regression

$$f(x, y) = a_0 + a_1x + a_2y + a_3x^2 + a_4xy + a_5y^2 + a_6x^3 + a_7x^2y + a_8xy^2 + a_9y^3 + a_{10}x^4 + a_{11}x^3y + a_{12}x^2y^2 + a_{13}xy^3 + a_{14}y^4$$

Here x, y are independent. By previous derivation, our total error is

$$E = E(a_0, a_1, a_2, \dots, a_{14})$$

so that we can apply our concept to find the minimum error for surface regression.

$$\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix} = \begin{bmatrix} 1 & x_1 & \cdots & y_1^4 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \cdots & y_n^4 \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_{14} \end{bmatrix}$$

$n \times 1$ $n \times 15$ 15×1

So we complete the derivation.

Algorithm Selecting

- Solve $(A^T A) \mathbf{x}_0 = A^T \mathbf{y} \rightarrow$ The worst method
 - Solve $\mathbf{x}_0 = (A^T A)^{-1} A^T \mathbf{y}$ by getting $(A^T A)^{-1}$
 1. Gaussian elimination \rightarrow Backward unstable
 2. LU decomposition (LUD) \rightarrow More stable than the former
- Easy to write
Less stable

- QR Decompose $A = QR = [Q_1 \begin{bmatrix} R_1 \\ Q_0 \end{bmatrix}] = Q_1 R_1 \rightarrow R_1 \mathbf{x} = Q_1^T \mathbf{y}$
 1. Modified Gram-schmidt process **Highest CP value**
 2. Givens Rotation **Matlab algorithm**
 3. Householder transform \rightarrow **Candidate** **for this problem**
- x = A\Y**

- SV Decomposition (SVD) $A = U \Sigma V^T \rightarrow$ **Candidate**

Most stable

Most expensive

Algorithm Selecting

Criterion for selecting an algorithm

- Stability
- Time and memory cost
- Speed of convergence

In general, these 2 relation are in direct proportion.

Now I use LU Decomposition since it is easy to write.

1. Loading data and getting A and A^T
2. $B = A^T A \rightarrow$ This step produces more error.
3. Get B^{-1} by **LU decomposition with pivoting (PLUD)**
4. Get $x_0 = (A^T A)^{-1} A^T y$

$$\begin{bmatrix} 10^{-20} & 0 \\ 1 & 1 \end{bmatrix} \xrightarrow{\text{Pivoting}} \begin{bmatrix} 1 & 1 \\ 10^{-20} & 0 \end{bmatrix}$$

$$\begin{bmatrix}
b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\
b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\
b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\
b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4}
\end{bmatrix} \quad \mathbf{B} = \begin{bmatrix}
1 & 0 & 0 & 0 \\
\frac{b_{2,1}}{b_{1,1}} & 1 & 0 & 0 \\
\frac{b_{3,1}}{b_{1,1}} & 0 & 1 & 0 \\
\frac{b_{4,1}}{b_{1,1}} & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\
0 & b_{2,2} - \frac{b_{1,2} \times b_{2,1}}{b_{1,1}} & b_{2,3} - \frac{b_{1,3} \times b_{2,1}}{b_{1,1}} & b_{2,4} - \frac{b_{1,4} \times b_{2,1}}{b_{1,1}} \\
0 & b_{3,2} - \frac{b_{1,2} \times b_{3,1}}{b_{1,1}} & b_{3,3} - \frac{b_{1,3} \times b_{3,1}}{b_{1,1}} & b_{3,4} - \frac{b_{1,4} \times b_{3,1}}{b_{1,1}} \\
0 & b_{4,2} - \frac{b_{1,2} \times b_{4,1}}{b_{1,1}} & b_{4,3} - \frac{b_{1,3} \times b_{4,1}}{b_{1,1}} & b_{4,4} - \frac{b_{1,4} \times b_{4,1}}{b_{1,1}}
\end{bmatrix}$$

$$= \begin{bmatrix}
1 & 0 & 0 & 0 \\
\ell_{2,1} & 1 & 0 & 0 \\
\ell_{3,1} & 0 & 1 & 0 \\
\ell_{4,1} & 0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\
0 & b_{2,2}^{(1)} & b_{2,3}^{(1)} & b_{2,4}^{(1)} \\
0 & b_{3,2}^{(1)} & b_{3,3}^{(1)} & b_{3,4}^{(1)} \\
0 & b_{4,2}^{(1)} & b_{4,3}^{(1)} & b_{4,4}^{(1)}
\end{bmatrix}$$

Do 3 times

$$= \begin{bmatrix}
1 & 0 & 0 & 0 \\
\ell_{2,1} & 1 & 0 & 0 \\
\ell_{3,1} & \ell_{3,2} & 1 & 0 \\
\ell_{4,1} & \ell_{4,2} & \ell_{4,3} & 1
\end{bmatrix} \begin{array}{c|c}
\begin{matrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{matrix} & \text{U} \\ \hline \text{L} & \end{array}$$

Algorithm

Matrix decomposition (LUD)

- $Bx_0 = A^T y \rightarrow LUx_0 = A^T y \rightarrow L(Ux_0) = L(c) = b, Ux_0 = c, A^T y = b$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{15,1} & \ell_{15,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,15} \\ 0 & u_{2,2} & \cdots & u_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & u_{15,15} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{15} \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{15,1} & \ell_{15,2} & \ell_{15,3} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{15} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{15} \end{bmatrix}$$

L : Lower triangular

U : Upper triangular

L and U matrix can be
store in one matrix to
reduce the memory usage.

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,15} \\ \ell_{2,1} & u_{2,2} & \cdots & u_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{15,1} & \ell_{15,2} & \cdots & u_{15,15} \end{bmatrix}$$

Solving the linear eq.

$$\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,15} \\ 0 & u_{2,2} & \cdots & u_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & u_{15,15} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{15} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{15} \end{bmatrix}$$

Ill-condition

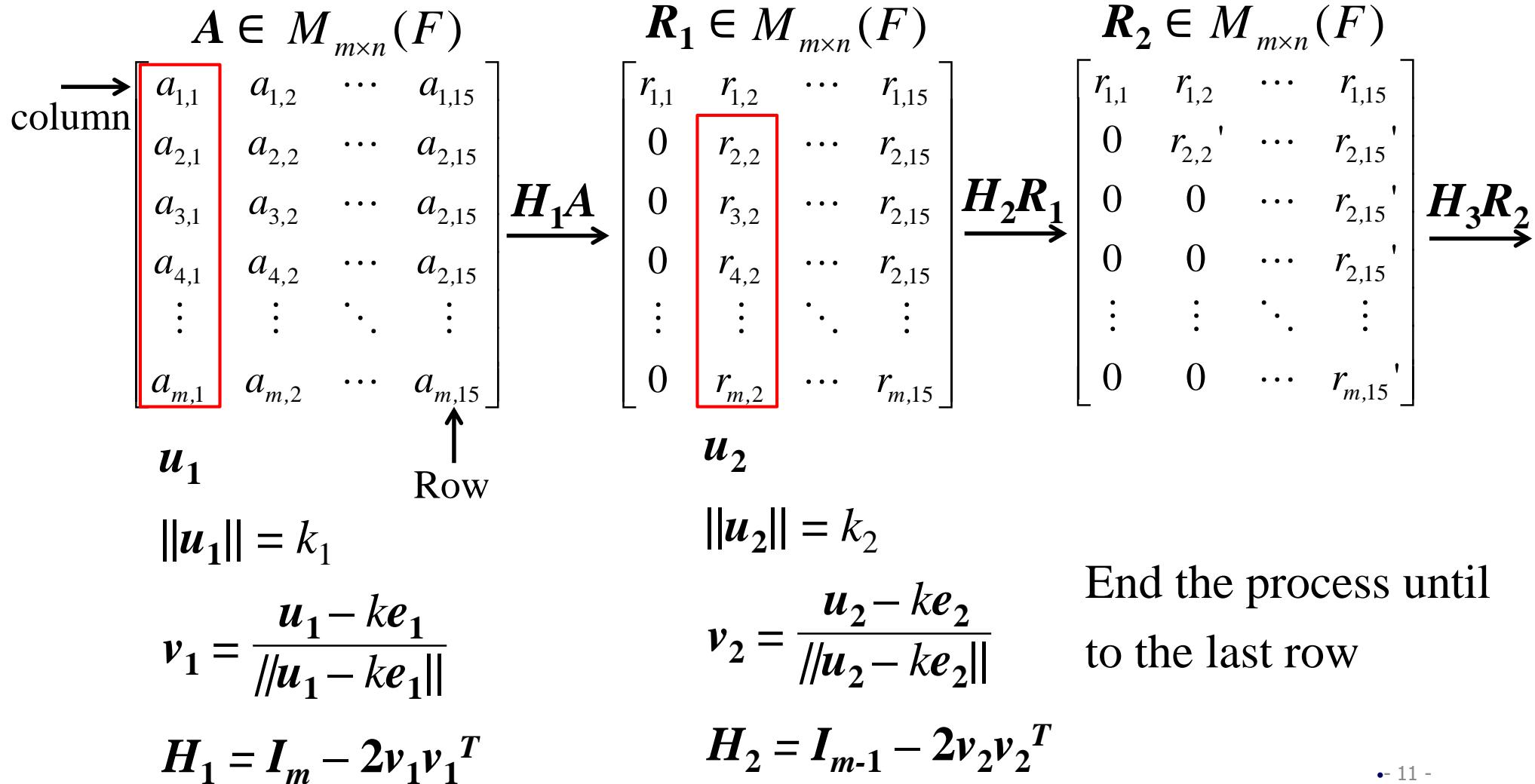
Numerical error
may cause

Mathematical
problem

$$\begin{array}{c} \text{Numerical error} \\ \text{may cause} \\ \text{Mathematical} \\ \text{problem} \end{array} \xrightarrow{\quad} \begin{array}{l} \boxed{400} \\ \boxed{-800} \end{array} \begin{array}{l} -201 \\ 401 \end{array} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 200 \\ -200 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -100 \\ -200 \end{bmatrix}$$
$$\begin{array}{l} \boxed{401} \\ \boxed{-800} \end{array} \begin{array}{l} -201 \\ 401 \end{array} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 200 \\ -200 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 40000 \\ 79800 \end{bmatrix}$$

- This issue can be solved by using QRD or SVD.

Algorithm Matrix decomposition (QRD(Householder))



Algorithm Matrix decomposition (QRD(Householder))

$$A \in M_{m \times n}(F)$$

$$A = QR = Q_1 R_1 \rightarrow R_1 x = Q_1^T y$$

Solving the linear eq.

Q is orthogonal matrix $= (H_n \dots H_2 H_1)^T$

$$QQ^T = Q^T Q = I \rightarrow Q^T = Q^{-1}$$
$$A = Q_1 R_1 \times Q_2 R_2 = [Q_1 \ Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$$

$Q_1 \in M_{m \times n}(F)$
Only needs this matrix

$Q_2 \in M_{m \times (m-n)}(F)$

$R_1 \in M_{n \times n}(F)$
 $R_2 = 0 \in M_{(m-n) \times m}(F)$

Algorithm Matrix decomposition (SVD)

- $A \in M_{m \times n}(F)$, $m \geq n$ and $\text{rank}(A) = n \rightarrow$ full rank.
 $\text{rank}(A) < n \rightarrow$ rank-deficient
 - This method is not appropriate in our situation, but it's a strong method for much more application.
 - So we decompose $A = U\Sigma V^T$ into 3 matrix as below.

$A \in M_{m \times n}(F) =$

$$U \in M_{m \times m}(F)$$

$\Sigma \in M_{m \times n}(F)$

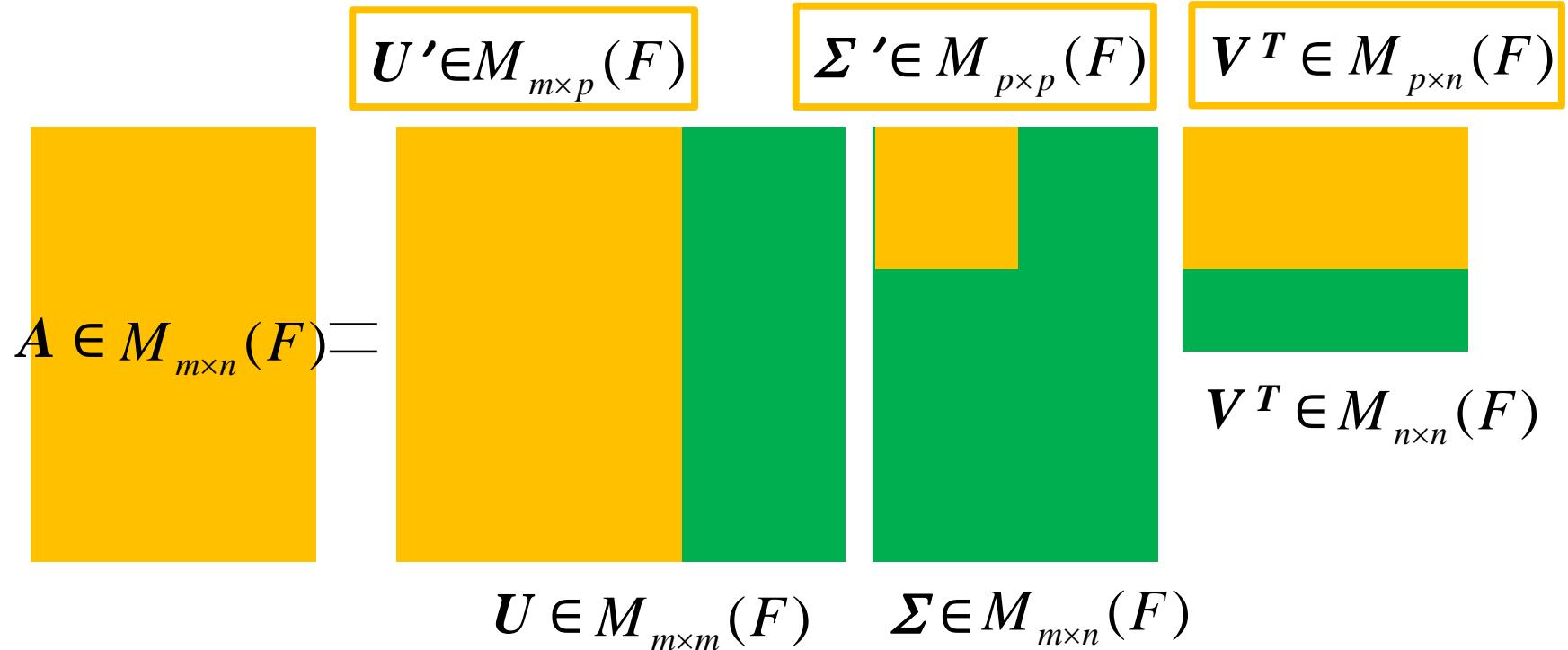
$V^T \in M_{n \times n}(F)$

Orthogonal

Diagonal

Algorithm

Matrix decomposition (SVD)

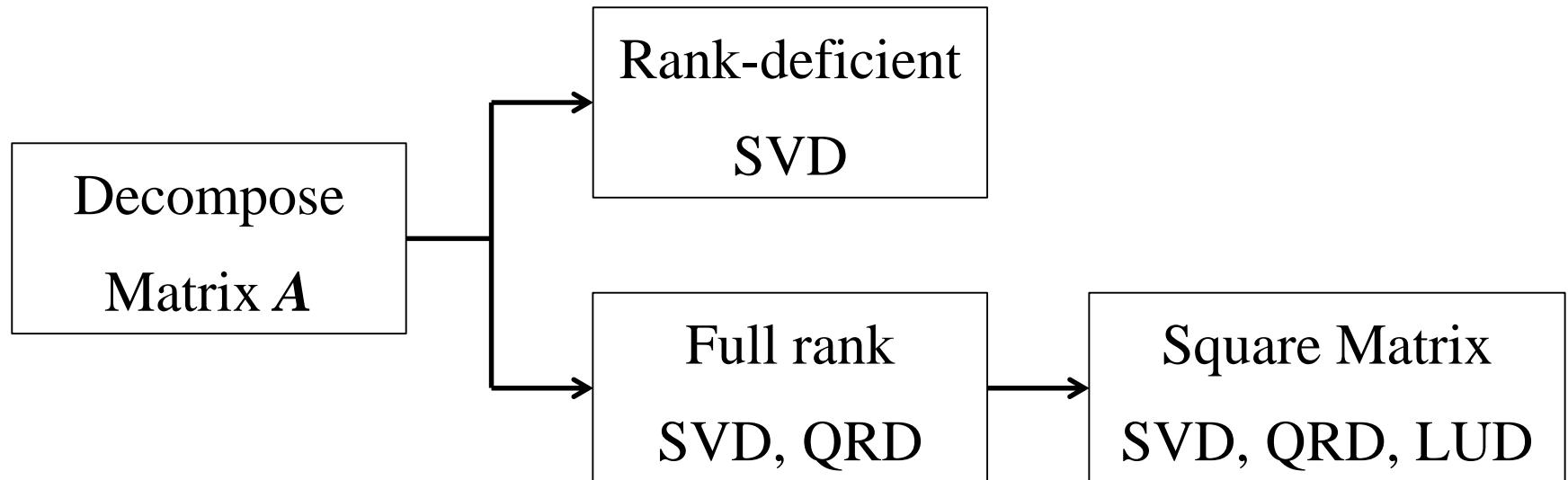


Only needs orange parts, it can reduce memory and time cost, but still costs much memory and time.

$$\forall A \in M_{m \times n}(F), \exists! U, S \text{ and } V \text{ s.t. } U\Sigma V^T$$

Algorithm

Matrix decomposition



If matrix A is always full rank (should be check **mathematically**), then **QRD** is the most appropriate algorithm for this problem.