Numerical methods for Regression
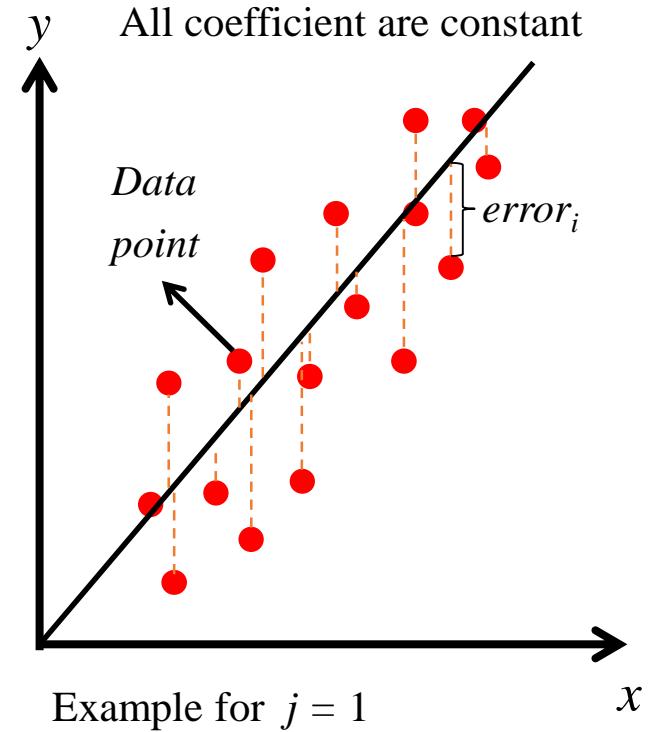
Concept for Least-Square

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

Fitting curve: $y = a_1 x + a_0$

- **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.

All coefficient are constant

Data point

$error_i$

Example for $j = 1$

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), $i = 1, 2, \ldots, n$, then, we have n group of linear equations system

$$\begin{cases} y_1 = bx_1 + a \\ \vdots \\ y_n = bx_n + a \end{cases} \xrightarrow[\text{representation}]{\text{Matrix}} \underset{n \times 1}{\overset{Y}{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}} = \underset{n \times 2}{\overset{A}{\begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}}} \underset{2 \times 1}{\overset{x}{\begin{bmatrix} b \\ a \end{bmatrix}}} \rightarrow Y = Ax$$

- 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
$$\langle Ax, y \rangle_m = \langle x, A^T y \rangle_n, \quad \langle \, , \, \rangle: \text{ inner product}$$

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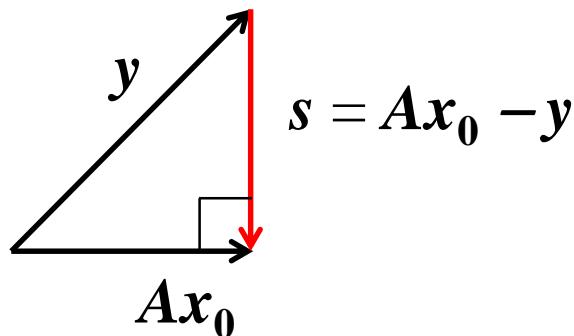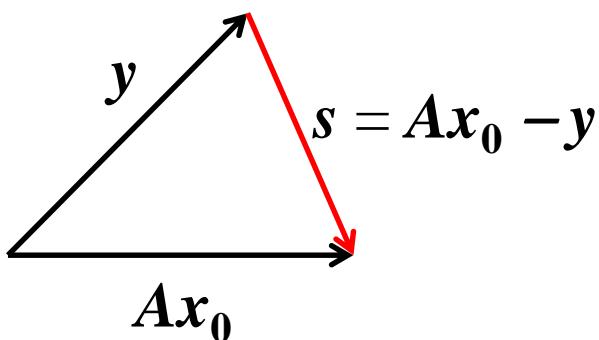
Let $A \in M_{m \times n}(F)$, $m \geq n$, $y \in F^m$, then $\exists\, x_0 \in F^n$ s.t.

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

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

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

$\rightarrow <Ax, s> = <x, A^T s> = <x, A^T(Ax_0 - y)> = 0 \rightarrow$

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

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

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,

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

$$\underbrace{\begin{bmatrix} x_1 & \cdots & x_n \\ 1 & \cdots & 1 \end{bmatrix}^T}_{A^T} \underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{Y} = \underbrace{\begin{bmatrix} x_1 & \cdots & x_n \\ 1 & \cdots & 1 \end{bmatrix}^T}_{A^T} \underbrace{\begin{bmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} b \\ a \end{bmatrix}}_{x_0} \leftarrow \begin{bmatrix} \sum_{i=1}^{n} x_i y_i \\ \sum_{i=1}^{n} y_i \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i & 1 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix}$$

- 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

$$f(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + a_7 x^2 y + a_8 xy^2 + a_9 y^3$$

$$+ a_{10} x^4 + a_{11} x^3 y + a_{12} x^2 y^2 + a_{13} xy^3 + a_{14} y^4$$

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

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

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

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

$$n \times 1 \qquad\qquad n \times 15 \qquad\qquad 15 \times 1$$

So we complete the derivation.

Algorithm Selecting

- Solve $(A^TA)\, x_0 = A^Ty$ → The worst method

  Easy to write

- Solve $x_0 = (A^TA)^{-1}A^Ty$ by getting $(A^TA)^{-1}$

  Less stable

  1. Gaussian elimination → Backward unstable
  2. LU decomposition (LUD) → More stable than the former

- QR Decompose $A = QR = [Q_1\ Q_2]\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1$ → $R_1 x = Q_1^T y$
  (QRD)

  Highest CP value

  1. Modified Gram-schmidt process

  Matlab algorithm

  2. Givens Rotation

  for this problem

  3. Householder transform → **Candidate**

  x = A\Y

- SV Decomposition (SVD) $A = U \Sigma V^{\,T}$ → **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. $B = A^TA$ → This step produces more error.
3. Get B^{-1} by **LU decomposition with pivoting (PLUD)**
4. Get $x_0 = (A^TA)^{-1}A^Ty$

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

$\downarrow$ Pivoting

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

$$\begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ b_{2,1} & b_{2,2} & b_{2,3} & b_{2,4} \\ b_{3,1} & b_{3,2} & b_{3,3} & b_{3,4} \\ b_{4,1} & b_{4,2} & b_{4,3} & b_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \dfrac{b_{2,1}}{b_{1,1}} & 1 & 0 & 0 \\ \dfrac{b_{3,1}}{b_{1,1}} & 0 & 1 & 0 \\ \dfrac{b_{4,1}}{b_{1,1}} & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_{1,1} & b_{1,2} & b_{1,3} & b_{1,4} \\ 0 & b_{2,2} - \dfrac{b_{1,2} \times b_{2,1}}{b_{1,1}} & b_{2,3} - \dfrac{b_{1,3} \times b_{2,1}}{b_{1,1}} & b_{2,4} - \dfrac{b_{1,4} \times b_{2,1}}{b_{1,1}} \\ 0 & b_{3,2} - \dfrac{b_{1,2} \times b_{3,1}}{b_{1,1}} & b_{3,3} - \dfrac{b_{1,3} \times b_{3,1}}{b_{1,1}} & b_{3,3} - \dfrac{b_{1,4} \times b_{3,1}}{b_{1,1}} \\ 0 & b_{4,2} - \dfrac{b_{1,2} \times b_{4,1}}{b_{1,1}} & b_{4,3} - \dfrac{b_{1,3} \times b_{4,1}}{b_{1,1}} & b_{4,4} - \dfrac{b_{1,4} \times b_{4,1}}{b_{1,1}} \end{bmatrix}$$

$\boldsymbol{B}$

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

Do 3 times

$$= \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_{2,1} & 1 & 0 & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 & 0 \\ \ell_{4,1} & \ell_{4,2} & \ell_{4,3} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,3} \\ 0 & 0 & 0 & u_{4,4} \end{bmatrix}}_{U}$$

Algorithm
Matrix decomposition (LUD)

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

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

L : Lower triangular   U : Upper triangular

$\underbrace{}_{c}$

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

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

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

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

$$\begin{bmatrix} 400 & -201 \\ -800 & 401 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 200 \\ -200 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -100 \\ -200 \end{bmatrix}$$

$$\begin{bmatrix} 401 & -201 \\ -800 & 401 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 200 \\ -200 \end{bmatrix} \rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 40000 \\ 79800 \end{bmatrix}$$

- This issue can be by using QRD or SVD.

Algorithm Matrix decomposition (QRD(Householder))

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

column →

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,15} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,15} \\ a_{3,1} & a_{3,2} & \cdots & a_{2,15} \\ a_{4,1} & a_{4,2} & \cdots & a_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,15} \end{bmatrix}$$

u_1

Row ↑

$\xrightarrow{\ H_1A\ }$

$$R_1 \in M_{m \times n}(F)$$

$$\begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,15} \\ 0 & r_{2,2} & \cdots & r_{2,15} \\ 0 & r_{3,2} & \cdots & r_{2,15} \\ 0 & r_{4,2} & \cdots & r_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & r_{m,2} & \cdots & r_{m,15} \end{bmatrix}$$

u_2

$\xrightarrow{\ H_2R_1\ }$

$$R_2 \in M_{m \times n}(F)$$

$$\begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,15} \\ 0 & r_{2,2}' & \cdots & r_{2,15}' \\ 0 & 0 & \cdots & r_{2,15}' \\ 0 & 0 & \cdots & r_{2,15}' \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{m,15}' \end{bmatrix}$$

$\xrightarrow{\ H_3R_2\ }$

$$\|u_1\| = k_1$$

$$v_1 = \frac{u_1 - ke_1}{\|u_1 - ke_1\|}$$

$$H_1 = I_m - 2v_1v_1^T$$
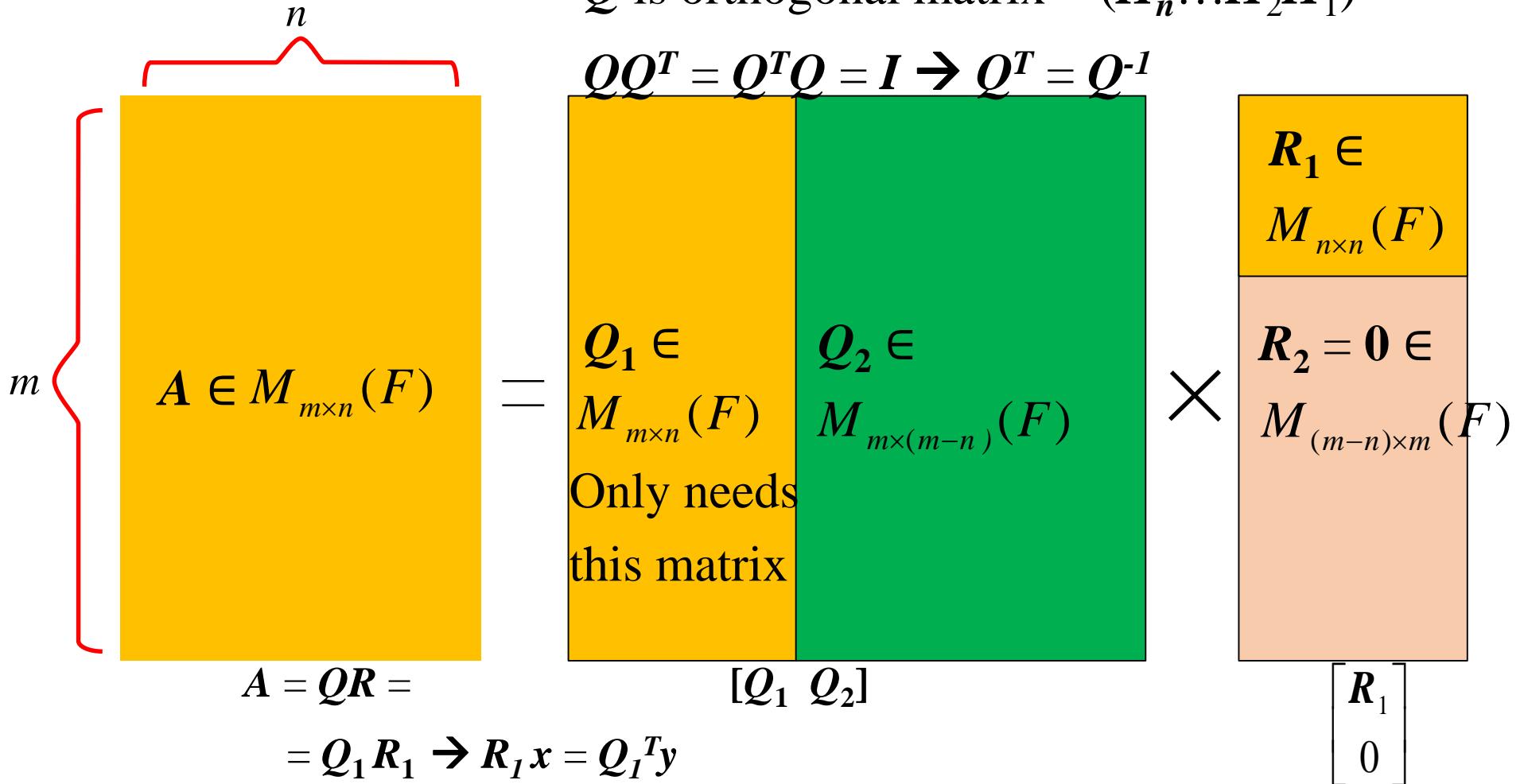
$$\|u_2\| = k_2$$

$$v_2 = \frac{u_2 - ke_2}{\|u_2 - ke_2\|}$$

$$H_2 = I_{m-1} - 2v_2v_2^T$$

End the process until to the last row

Algorithm Matrix decomposition (QRD(Householder))

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

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

n

m

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

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

$\times$

$R_1 \in M_{n \times n}(F)$

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

$A = QR =$

$[Q_1 \; Q_2]$

$\begin{bmatrix} R_1 \\ 0 \end{bmatrix}$

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

Solving the linear eq.

Algorithm
Matrix decomposition (SVD)

- $A \in M_{m \times n}(F)$, $m \geq n$ and rank(A) $= n$ → full rank.

  rank(A) $< n$ → 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.

$$A \in M_{m \times n}(F) = \quad U \in M_{m \times m}(F) \quad \Sigma \in M_{m \times n}(F) \quad V^T \in M_{n \times n}(F)$$

Orthogonal Diagonal Orthogonal

Algorithm
Matrix decomposition (SVD)

$$\boxed{U' \in M_{m \times p}(F)} \qquad \boxed{\Sigma' \in M_{p \times p}(F)} \qquad \boxed{V^T \in M_{p \times n}(F)}$$
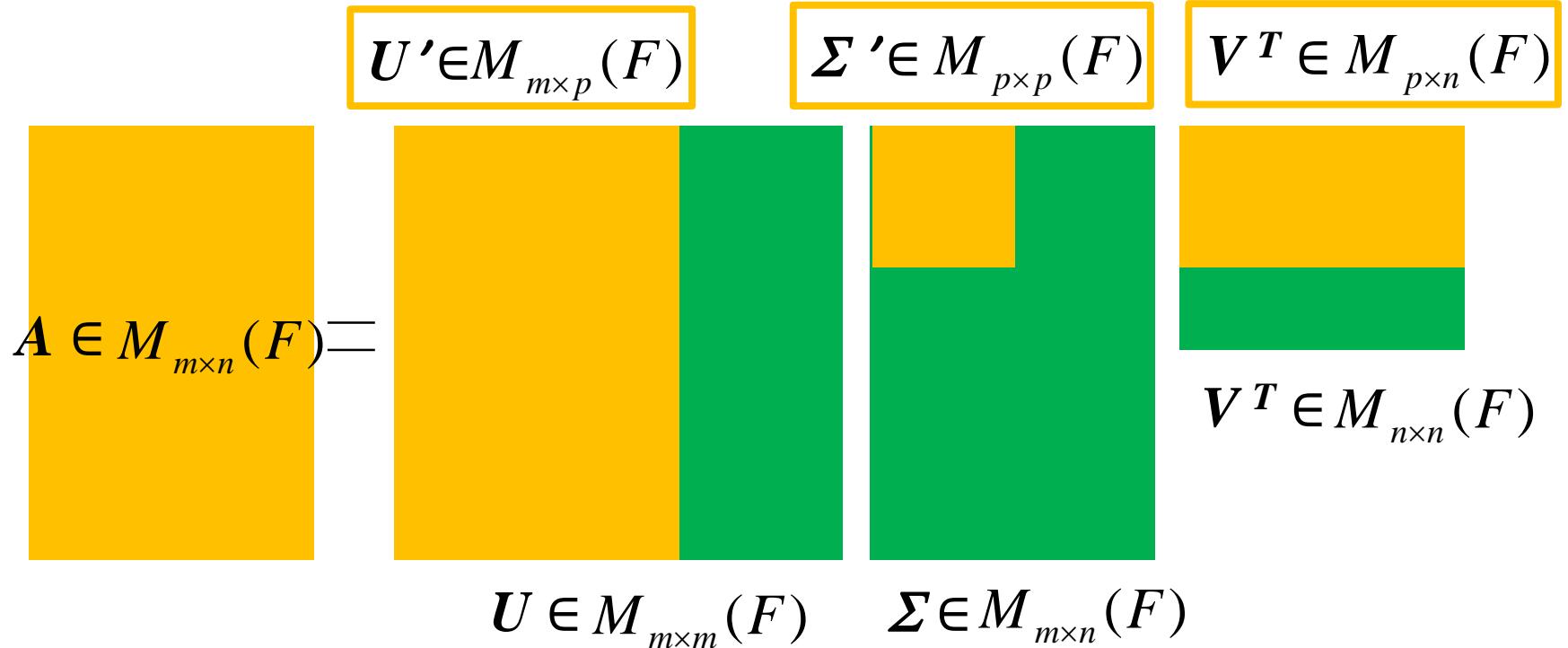
$$A \in M_{m \times n}(F) =$$

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

$$V^T \in M_{n \times n}(F)$$
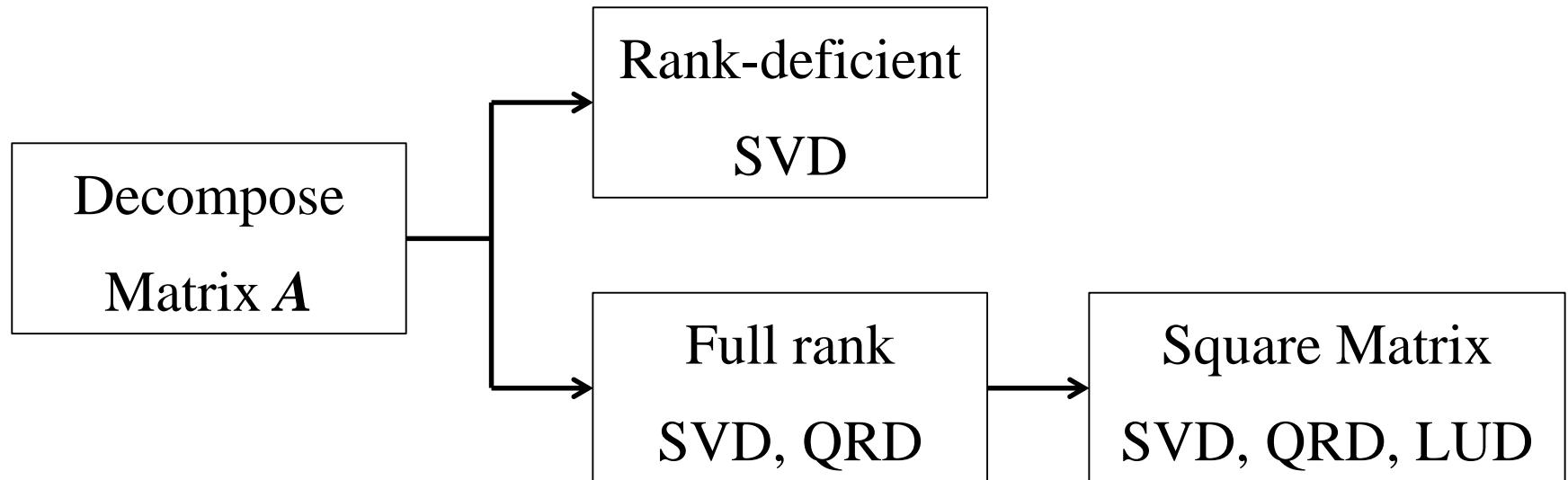
Only needs orange parts, it can reduce memory and time cost, but still costs much memory and time.

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

Algorithm
Matrix decomposition

	Rank-deficient
Decompose	SVD
Matrix A	Full rank → Square Matrix
	SVD, QRD → SVD, QRD, LUD

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