NUMERICAL METHODS FOR SURFACE REGRESSION

Outline

- Methods for Surface Regression - Least-Square
- Derive the Mathematical Eq. For Surface Regression
  1. *Existence and Uniqueness*
  2. *Extend to Surface Regression*
- *Algorithm*
  1. *Selecting*
  2. *Matrix Decomposition*

Methods for Surface Regression

- For example, the 4th order surface

$$
\begin{aligned}
f(x,y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + a_7 x^2 y + a_8 xy^2 + a_9 y^3 \\
+ a_{10} x^4 + a_{11} x^3 y + a_{12} x^2 y^2 + a_{13} xy^3 + a_{14} y^4
\end{aligned}
$$

- Here *x, y* are independent variables
- This problem can be solved by 3 methods:

1. Spline

a. Thin Plate Spline approximation

b. Multilevel B-Spline approximation

2. Least-square approximation

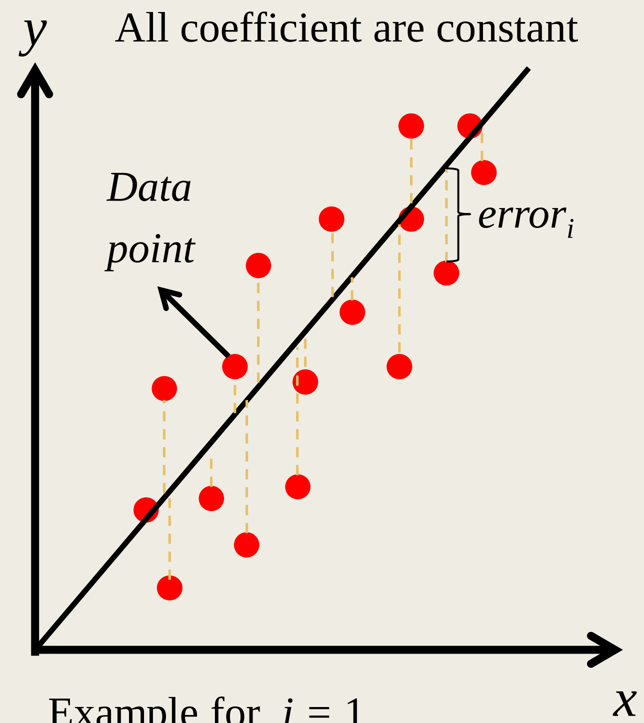
Concept for Least-Square

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

Fitting curve: $y = a_1 x + a_0$

All coefficient are constant

- **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.



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

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

- 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 $\boldsymbol{A} \in M_{m\times n}(F)$, $\boldsymbol{x} \in F^n$; $\boldsymbol{y} \in F^m$, $m \geq n$ then

$$<\boldsymbol{Ax}, \boldsymbol{y}>_m = <\boldsymbol{x}, \boldsymbol{A}^T\boldsymbol{y}>_n, \quad < , >: \text{inner product}$$

- Lemma 2: Let $\boldsymbol{A} \in M_{m\times n}(F)$, rank($\boldsymbol{A}$) = rank($\boldsymbol{A}^T\boldsymbol{A}$)

- Corollary: Let $\boldsymbol{A} \in M_{m\times n}(F)$ and rank($\boldsymbol{A}$) = n, then $\boldsymbol{A}^T\boldsymbol{A}$ is invertible

Derive the Mathematical Eq.
Existence and Uniqueness

Theorem of Least-square approximation:

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

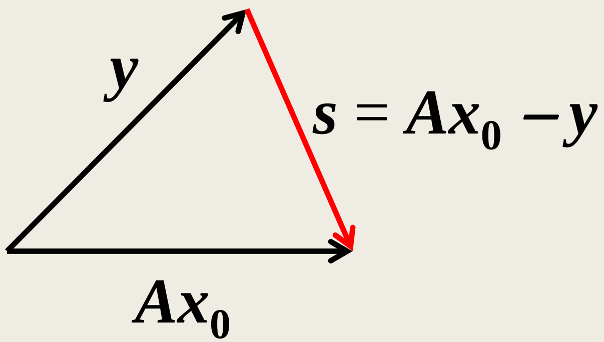
$$(\boldsymbol{A}^T\boldsymbol{A})\, \boldsymbol{x_0} = \boldsymbol{A}^T\boldsymbol{y} \text{ and } \|\boldsymbol{A}\boldsymbol{x_0} - \boldsymbol{y}\| \leq \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}\|, \boldsymbol{x} \in F^n$$

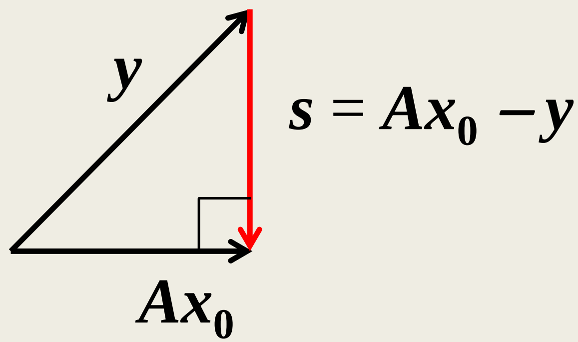
Furthermore, if rank($\boldsymbol{A}$) = n (full rank), then $\boxed{\boldsymbol{x_0} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{y}}$

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

pf : In the view point of geometry, if there exists 2 vectors $\boldsymbol{y}$ and $\boldsymbol{Ax_0}$, $\boldsymbol{s} = \boldsymbol{Ax_0} - \boldsymbol{y}$, when $\boldsymbol{s} \perp \boldsymbol{Ax_0}$, then $\|\boldsymbol{s}\|$ is minimum and exactly one

$\rightarrow \langle \boldsymbol{Ax}, \boldsymbol{s}\rangle = \langle \boldsymbol{x}, \boldsymbol{A}^T\boldsymbol{s}\rangle = \langle \boldsymbol{x}, \boldsymbol{A}^T(\boldsymbol{Ax_0} - \boldsymbol{y})\rangle = 0 \rightarrow$





For $\boldsymbol{x} \neq 0$, $\boldsymbol{A}^T\boldsymbol{Ax_0} = \boldsymbol{A}^T\boldsymbol{y}$

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

Derive the Mathematical Eq. Extend to Surface 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,

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

$$\downarrow$$

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

- 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 Surface 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.

$$\underset{n\times 1}{\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}} = \underset{n\times 15}{\begin{bmatrix} 1 & x_1 & \cdots & y_1^4 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \cdots & y_n^4 \end{bmatrix}} \underset{15\times 1}{\begin{bmatrix} a_0 \\ \vdots \\ a_{14} \end{bmatrix}}$$

So we complete the derivation.

Derive the Mathematical Eq. Extend to Surface Regression

- If some costumers want to use higher order , just extend the matrix and to calculate the new spec.
- For example, 2D, 6th order surface, there are 28 coefficients.

$$f(x,y) = a_0 + a_1 x + a_2 y + \cdots + a_{27} y^6$$

$$\underset{n\times 1}{\begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}} = \underset{n\times 28}{\begin{bmatrix} 1 & x_1 & \cdots & y_1^{\,6} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & \cdots & y_n^{\,6} \end{bmatrix}} \underset{28\times 1}{\begin{bmatrix} a_0 \\ \vdots \\ a_{27} \end{bmatrix}}$$

Algorithm Selecting

- Solve $(A^TA)\, x_0 = A^Ty$ → The worst method
- Solve $x_0 = (A^TA)^{-1}A^Ty$ by getting $(A^TA)^{-1}$
  1. *Gaussian elimination → Backward unstable*
  2. *LU decomposition (LUD) → More stable than the former*

Easy to write

Less stable

- QR Decompose $A = QR = [Q_1 \;\; Q_2]\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1 \rightarrow R_1 x = Q_1^T y$ (QRD)
  1. *Modified Gram-schmidt process*
  2. *Givens Rotation*
  3. *Householder transform* → ***Candidate***

Highest CP value

Matlab algorithm for this problem

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 $\boldsymbol{A}$ and $\boldsymbol{A}^T$
2. $\boldsymbol{B} = \boldsymbol{A}^T\boldsymbol{A}$ → This step produces more error.
3. Get $\boldsymbol{B}^{-1}$ by **LU decomposition with pivoting (PLUD)**
4. Get $\boldsymbol{x}_0 = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{y}$

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

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

$$
\mathbf{\mathit{B}}
$$

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

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

Algorithm
Matrix decomposition (LUD)

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

$$\underbrace{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{15,1} & \ell_{15,2} & \cdots & 1 \end{bmatrix}}_{\text{L : Lower triangular}} \underbrace{\underbrace{\begin{bmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,15} \\ 0 & u_{2,2} & \cdots & u_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & u_{15,15} \end{bmatrix}}_{\text{U : Upper triangular}} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{15} \end{bmatrix}}_{\boldsymbol{c}} = \boxed{\begin{bmatrix} 1 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{15,1} & \ell_{15,2} & \ell_{15,3} & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{15} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{15} \end{bmatrix}}$$

Solving the linear eq.

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

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

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

Ill-condition

Numerical error may cause

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

Mathematical problem

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

- This issue can be by using QRD or SVD.

Algorithm Matrix decomposition (QRD(Householder))

$$\boldsymbol{A} \in M_{m\times n}(F) \qquad \boldsymbol{R_1} \in M_{m\times n}(F) \qquad \boldsymbol{R_2} \in M_{m\times n}(F)$$

column

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,15} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,15} \\ a_{3,1} & a_{3,2} & \cdots & a_{2,15} \\ a_{4,1} & a_{4,2} & \cdots & a_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,15} \end{bmatrix} \xrightarrow{\boldsymbol{H_1A}} \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,15} \\ 0 & r_{2,2} & \cdots & r_{2,15} \\ 0 & r_{3,2} & \cdots & r_{2,15} \\ 0 & r_{4,2} & \cdots & r_{2,15} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & r_{m,2} & \cdots & r_{m,15} \end{bmatrix} \xrightarrow{\boldsymbol{H_2R_1}} \begin{bmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,15} \\ 0 & r_{2,2}' & \cdots & r_{2,15}' \\ 0 & 0 & \cdots & r_{2,15}' \\ 0 & 0 & \cdots & r_{2,15}' \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & r_{m,15}' \end{bmatrix} \xrightarrow{\boldsymbol{H_3R_2}}$$

Row

$\boldsymbol{u_1}$

$\|\boldsymbol{u_1}\| = k_1$

$$\boldsymbol{v_1} = \frac{\boldsymbol{u_1} - k\boldsymbol{e_1}}{\|\boldsymbol{u_1} - k\boldsymbol{e_1}\|}$$

$$\boldsymbol{H_1} = \boldsymbol{I_m} - \boldsymbol{2v_1v_1}^T$$

$\boldsymbol{u_2}$

$\|\boldsymbol{u_2}\| = k_2$

$$\boldsymbol{v_2} = \frac{\boldsymbol{u_2} - k\boldsymbol{e_2}}{\|\boldsymbol{u_2} - k\boldsymbol{e_2}\|}$$

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

End the process until to the last row

Algorithm Matrix decomposition (QRD(Householder))

$\boldsymbol{Q}$ is orthogonal matrix $= (\boldsymbol{H}_n \ldots \boldsymbol{H}_2 \boldsymbol{H}_1)^T$

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

$$\overbrace{\boldsymbol{A} \in M_{m\times n}(F)}^{n}\Big\}{m} = \left[\; \boldsymbol{Q}_1 \in M_{m\times n}(F) \;\middle|\; \boldsymbol{Q}_2 \in M_{m\times(m-n)}(F) \;\right] \times \begin{bmatrix} \boldsymbol{R}_1 \in M_{n\times n}(F) \\ \boldsymbol{R}_2 = \boldsymbol{0} \in M_{(m-n)\times m}(F) \end{bmatrix}$$

Only needs this matrix

$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} = [\boldsymbol{Q}_1 \;\; \boldsymbol{Q}_2] \begin{bmatrix} \boldsymbol{R}_1 \\ 0 \end{bmatrix}$

$= \boldsymbol{Q}_1 \boldsymbol{R}_1 \rightarrow \boldsymbol{R}_1 \boldsymbol{x} = \boldsymbol{Q}_1^T \boldsymbol{y}$

Solving the linear eq.

Algorithm
Matrix decomposition (SVD)

- $\boldsymbol{A} \in M_{m\times n}(F)$, $m \geq n$ 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 $\boldsymbol{A} = \boldsymbol{U\Sigma V}^{\boldsymbol{T}}$ into 3 matrix as below.

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

Orthogonal Diagonal Orthogonal

Algorithm
Matrix decomposition (SVD)

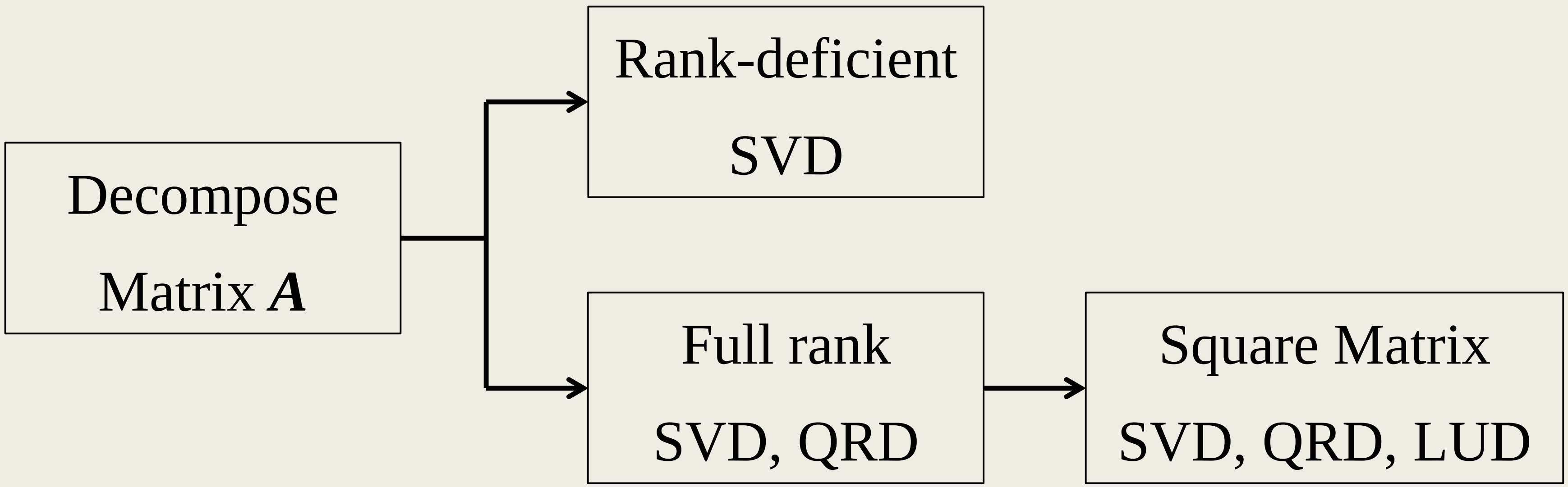
$\boldsymbol{U}' \in M_{m\times p}(F)$ $\boldsymbol{\Sigma}' \in M_{p\times p}(F)$ $\boldsymbol{V}^{\boldsymbol{T}} \in M_{p\times n}(F)$

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

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

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

Algorithm
Matrix decomposition



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