Numerical methods for Regression by least-square

Concept for Least-Square

$$y = \sum_{j=0}^{n} a_j x^j \quad n \in \mathbf{Z}^+$$

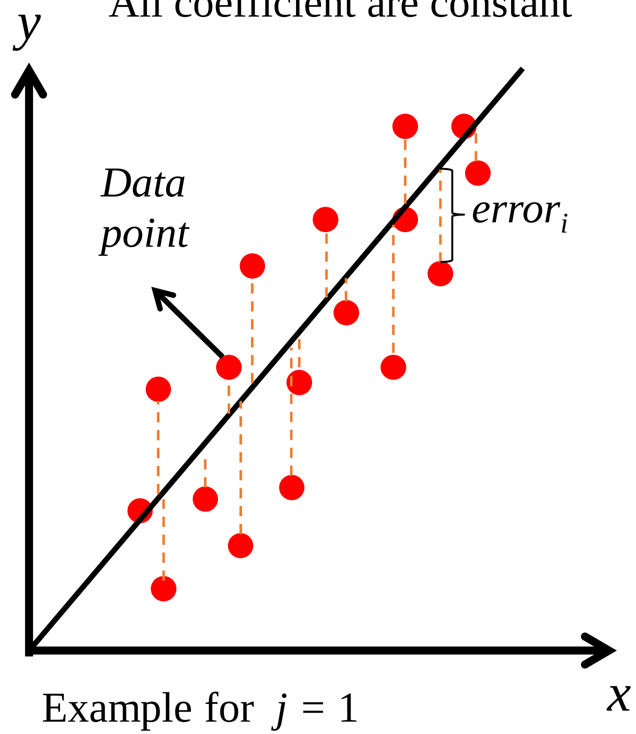
Fitting curve: $y = a_1 x + a_0$
All coefficient are constant

- **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 $\underline{y = bx + a}$ 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

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

- 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

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

- 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 $\exists$ $\boldsymbol{x_0}$ F^{n} s.t.

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

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 $\exists!$(**exist uniquely**) solution $\boldsymbol{s}$

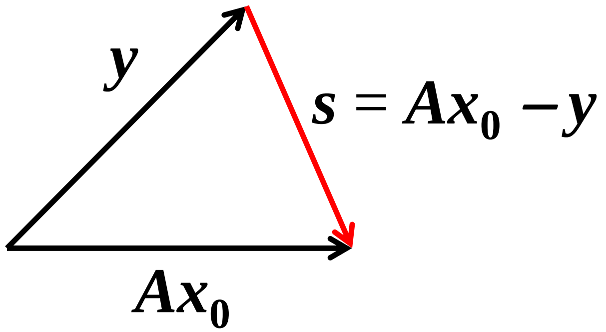
pf : In the view point of geometry, if there exists 2 vectors $\boldsymbol{y}$ and $\boldsymbol{Ax_0}$,

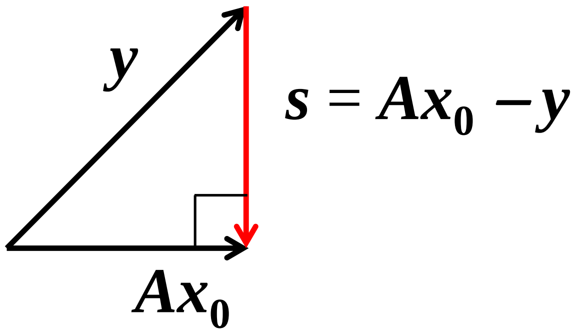
$\boldsymbol{s} = \boldsymbol{Ax_0} - \boldsymbol{y}$, when $\boldsymbol{s} \perp \boldsymbol{Ax_0}$, then $\|\boldsymbol{s}\|$ is minimum and exactly one

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

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 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,

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

$$\downarrow$$

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

- 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 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.

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

So we complete the derivation.

Algorithm Selecting

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

- QR Decompose $\boldsymbol{A} = \boldsymbol{QR} = [\boldsymbol{Q_1}\ \ \boldsymbol{Q_2}]\begin{bmatrix} \boldsymbol{R}_1 \\ 0 \end{bmatrix} = \boldsymbol{Q_1 R_1}$ → $\boldsymbol{R_1 x} = \boldsymbol{Q_1{}^T y}$

  (QRD)
    1. Modified Gram-schmidt process
    2. Givens Rotation → for sparse matrix
    3. Householder transform → for dense matrix

Highest CP value
Matlab algorithm for this problem
x = A\Y

- SV Decomposition (SVD) $\boldsymbol{A} = \boldsymbol{U\Sigma V}^{\boldsymbol{T}}$ → **Candidate**

  Most stable
  Most expensive

Algorithm Selecting

Criterion for selecting an algorithm

- Stability
- **Time and memory cost**
- **Speed of convergence**

In general, these 2 relations are in direct proportion..

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

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

Algorithm
Matrix decomposition (LUD)

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

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

Do 3 times

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

Algorithm
Matrix decomposition (LUD)

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

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

L : Lower triangular U : Upper triangular

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 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))

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

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

$\boldsymbol{u_1}$

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

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

$$\boldsymbol{H_1} = \boldsymbol{I_m} - 2\boldsymbol{v_1}\boldsymbol{v_1}^T$$

$\boldsymbol{u_2}$

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

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

$$\boldsymbol{H_2} = \boldsymbol{I_{m\text{-}1}} - 2\boldsymbol{v_2}\boldsymbol{v_2}^T$$

End the process until to the last row

Algorithm Matrix decomposition (QRD(Householder))

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

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

$$\underbrace{\boldsymbol{A} \in M_{m\times n}(F)}_{m \times n} = \left[\; \boldsymbol{Q_1} \in M_{m\times n}(F) \;\middle|\; \boldsymbol{Q_2} \in M_{m\times (m-n)}(F) \;\right] \times \begin{bmatrix} \boldsymbol{R_1} \in M_{n\times n}(F) \\ \boldsymbol{R_2} = \boldsymbol{0} \in M_{(m-n)\times n}(F) \end{bmatrix}$$

Only needs this matrix

$\boldsymbol{A} = \boldsymbol{QR} = [\boldsymbol{Q_1} \;\; \boldsymbol{Q_2}] \begin{bmatrix} \boldsymbol{R}_1 \\ 0 \end{bmatrix}$

$= \boldsymbol{Q_1 R_1} \rightarrow \boldsymbol{R_1 x} = \boldsymbol{Q_1^T y}$

Solving the linear eq.

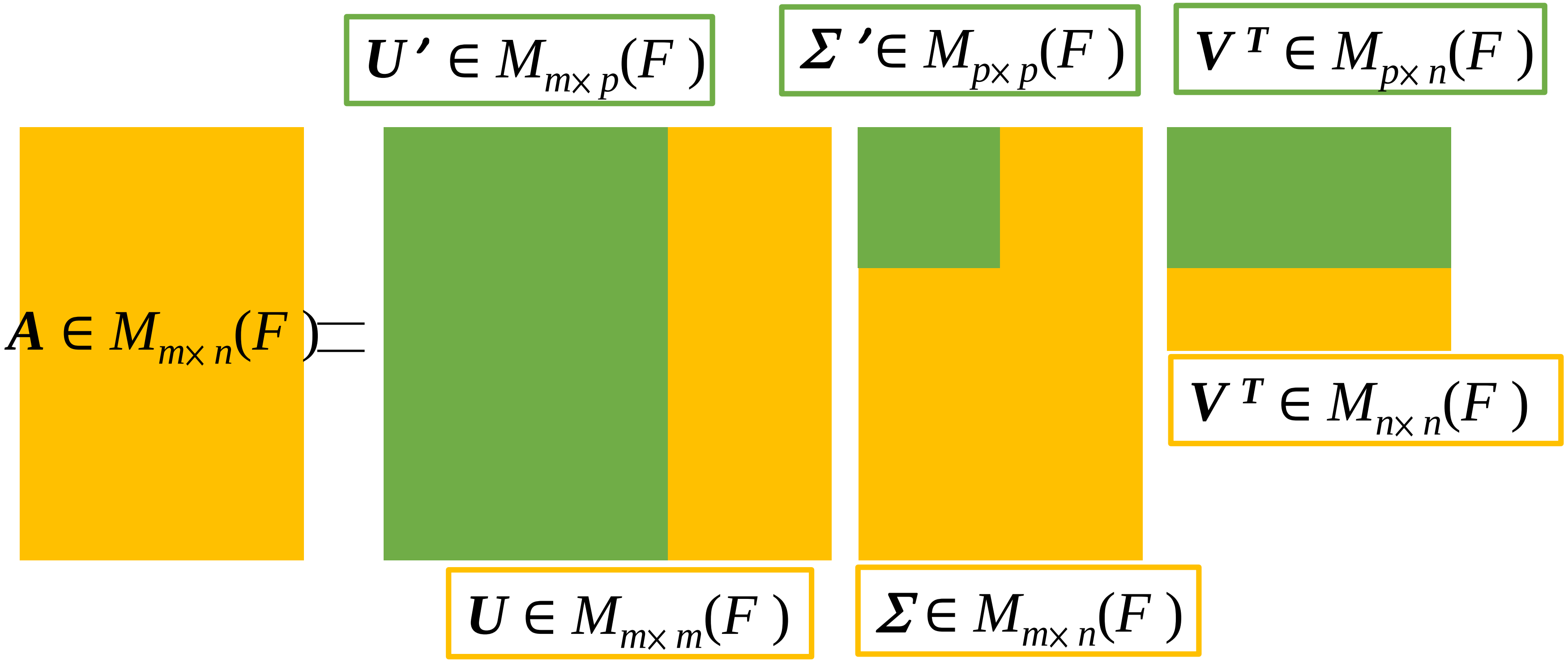
Algorithm
Matrix decomposition (SVD)

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

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

Orthogonal (for $\boldsymbol{U}$) Diagonal (for $\boldsymbol{\Sigma}$) Orthogonal (for $\boldsymbol{V^{T}}$)

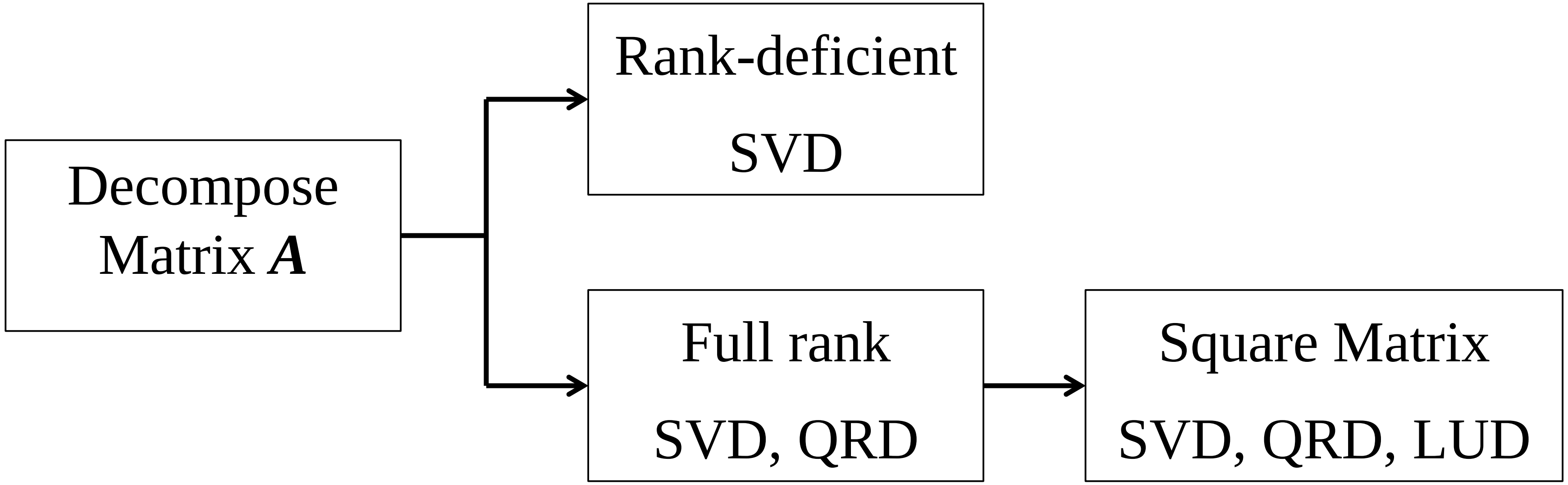
Algorithm
Matrix decomposition (SVD)



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

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

Algorithm
Matrix decomposition



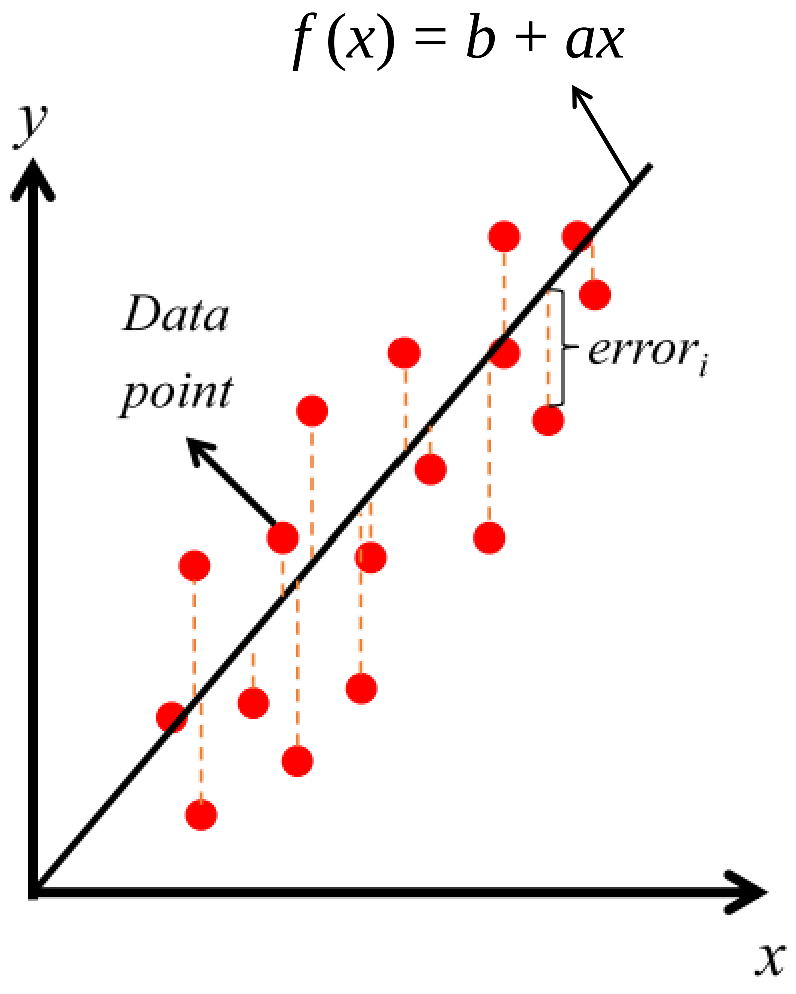
If matrix ***A*** is always full rank(should be check **mathematically**), then **QRD** is the most appropriate algorithm for this problem.
If not, then SVD

Special case: linear regression

- There are n data points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, $n \geq 2$ and $n \in \mathbf{Z}^+$.
- $\exists!\ y = f(x) = b + ax$ such that the square error e is the least

$$E = \sum_{i=1}^{n} (y - y_i)^2 = \sum_{i=1}^{n} (ax_i + b - y_i)^2$$

- It's a parabolic eq. with concave up, so there exists a minimum value.
- Here we want to find $\boldsymbol{a}_{min}$ and $\boldsymbol{b}_{min}$ such that E is minimum



An O(n) method for linear regression

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

$$\sum_{i=1}^{n} \left(ax_i + b\right) = \sum_{i=0}^{n} y_i \rightarrow a\sum_{i=0}^{n} x_i + \sum_{i=0}^{n} b = \sum_{i=0}^{n} y_i \rightarrow a\mu_x + b = \mu_y \rightarrow b = \mu_y - a\mu_x$$

$$\sum_{i=1}^{n} \left(ax_i^{\,2} + bx_i\right) = \sum_{i=0}^{n} x_i y_i \rightarrow a\sum_{i=0}^{n} x_i^{\,2} + b\sum_{i=0}^{n} x_i = \sum_{i=0}^{n} x_i y_i \rightarrow a\sum_{i=0}^{n} x_i^{\,2} + n\mu_x\mu_y - an\mu_x^{\,2} = \sum_{i=0}^{n} x_i y_i$$

$$\rightarrow a\sum_{i=0}^{n} x_i^{\,2} - an\mu_x^{\,2} = \sum_{i=0}^{n} x_i y_i - n\mu_x\mu_y$$

$$\rightarrow a\left(\sum_{i=0}^{n} x_i^{\,2} - n\mu_x^{\,2}\right) = \sum_{i=0}^{n} x_i y_i - n\mu_x\mu_y$$

An O(n) method for linear regression

$$\left(\sum_{i=0}^{n} x_i^{\,2} - n\mu_x^{\,2}\right) = n\sigma_x^{\,2} = \sum_{i=0}^{n}\left(x_i - \mu_x\right)^2$$

$$\sum_{i=0}^{n}\left(x_i - \mu_x\right)\left(y_i - \mu_y\right) = n\mathrm{Cov}(x, y) = \sum_{i=0}^{n}\left(x_i y_i - x_i\mu_y - \mu_x y_i + \mu_x\mu_y\right)$$

$$= \sum_{i=0}^{n} x_i y_i - \mu_y \sum_{i=0}^{n} x_i - \mu_x \sum_{i=0}^{n} y_i + n\mu_x\mu_y$$

$$= \sum_{i=0}^{n} x_i y_i - n\mu_y\mu_x - n\mu_x\mu_y + n\mu_x\mu_y$$

$$= \sum_{i=0}^{n} x_i y_i - n\mu_y\mu_x$$

$$\boxed{\begin{aligned} a &= \frac{\sum_{i=0}^{n} x_i y_i - n\mu_x\mu_y}{\sum_{i=0}^{n} x_i^{\,2} - n\mu_x^{\,2}} \\ &= \frac{\sum_{i=0}^{n}\left(x_i - \mu_x\right)\left(y_i - \mu_y\right)}{\sum_{i=0}^{n}\left(x_i - \mu_x\right)^2} \\ &= \frac{\mathrm{Cov}(x, y)}{\sigma_x^{\,2}} \end{aligned}}$$

An O(n) method for linear regression

$$y = ax + b$$
$$= ax + (\mu_y - a\mu_x)$$
$$= \mu_y + a(x - \mu_x)$$

$$y - \mu_y = a(x - \mu_x)$$

- The regression line pass through the average of *x* and *y*

```
for(i=0; i<n; i++)
{
    xavg += x[i]
    yavg += y[i]
    xiyi += x[i]*y[i]
    xixi += x[i]*x[i]
}
a = (xiyi – n* xavg* yavg)/(xixi – n* xavg* xavg)
y = a*(x – xavg) – yavg
```