Linear regression via QR decomposition

$$\vec{C} = (A^T A)^{-1} A^T \vec{y}$$

- Normal equations are general and in theory will always work. But in practice ...
  - Matrix inverse computationally intensive to solve (large datasets)
  - Matrix inverse easily corrupted by numerical errors.

Example:

$$\begin{bmatrix} 51+\Delta & 50 \\ 50 & 50 \end{bmatrix} \xrightarrow[\Delta=0]{\text{inverse}} \begin{bmatrix} 1 & -1 \\ -1 & 1.02 \end{bmatrix}$$

$$\xrightarrow[\Delta=0.5]{\text{inverse}} \begin{bmatrix} 2/3 & -2/3 \\ -2/3 & 0.687 \end{bmatrix}$$

Numbers changed 50% with small error!

- Factoring independent variable matrix A into $A = QR$ can allow us to solve for $\vec{c}$ <u>without any matrix inversion!!</u>

★ Super numerically stable

[Not easily affected by numerical error]

$(N, M+1)$ $(M+1, M+1)$

$(N, M+1)$

$\vec{A} = Q\ R$

Orthogonal matrix

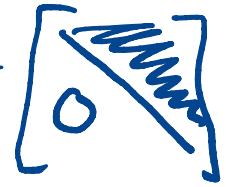
[Columns orthogonal unit vectors]

length = 1

Upper triangular matrix

$$\begin{bmatrix} 1 & 2 & 3 & \cdots \\ 0 & 1 & 4 & \cdots \\ 0 & 0 & 1 & \cdots \\ 0 & 0 & \cdots & 1 \\ 0 & 0 & \cdots & \end{bmatrix}$$

ie



Orthogonal Matrices have the property:

$$Q^T Q = I \qquad Q^T = Q^{-1}$$

- How to get Q & R?

Q: Harder part. Get via Gram-Schmidt algorithm.

R: Easy. Once we have Q, get R via

$$A = QR$$

$$Q^{-1}A = Q^{-1}QR$$

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

flip ← but $Q^{-1} = Q^T$!

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

$$\boxed{R = Q^T A}$$

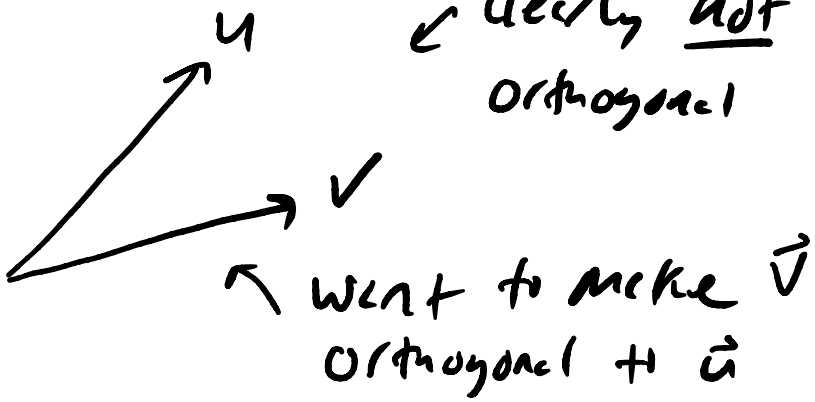
$(M+1, N)$ $(N, M+1)$

$(M+1, M+1)$

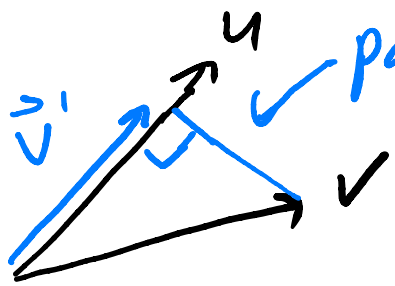
- **Gram-Schmidt algorithm (intuition):**

Idea: make column vectors orthogonal from one another by subtracting out "overlapping" components via projection:

2D example:

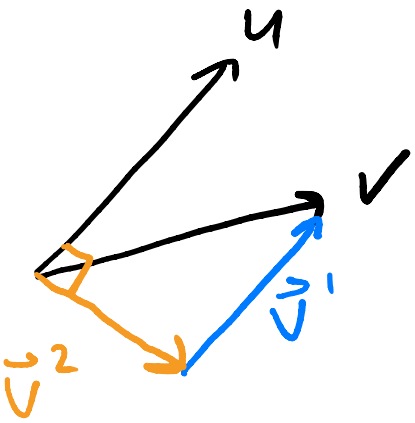


$\Rightarrow$ $\text{proj}_{\vec{u}}(\vec{v})$ = project $\vec{v}$ onto $\vec{u}$



Then to make $\vec{u}, \vec{v}$ orthogonal:

$$\vec{v}^2 = \vec{v} - \underbrace{\text{proj}_{\vec{u}}(\vec{v})}_{\vec{v}'}$$



General case: To make column vectors

$$[\vec{v}_1, \vec{v}_2, \vec{v}_3, \ldots, \vec{v}_m]$$

orthogonal $-[\vec{u}_1, \vec{u}_2, \vec{u}_3, \ldots, \vec{u}_m]$

progressively subtract out projections onto previous column vectors:

orthogonal ↘ ↙ original vectors

$$u_1 = v_1$$

$$u_2 = v_2 - \text{proj}_{\vec{u}_1}(\vec{v}_2)$$

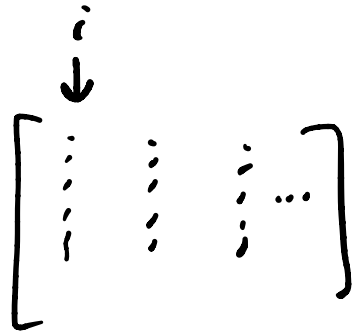
$$u_3 = v_3 - \text{proj}_{\vec{u}_1}(\vec{v}_3) - \text{proj}_{\vec{u}_2}(\vec{v}_3)$$

$$\vdots$$

• Gram-Schmidt algorithm (details):

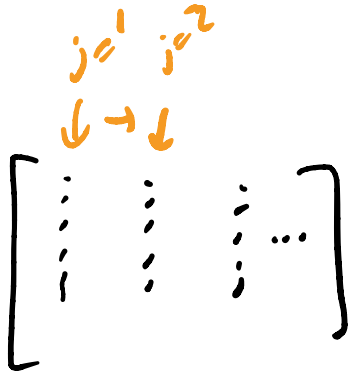
1) Initialize Q to an $(N, M+1)$ matrix.

2) Visit columns in A from left-to-right. Call the index of current column i.

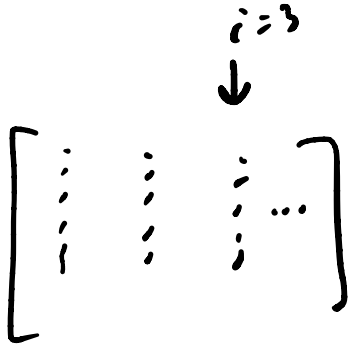


3) With a seperate "pointer" j, revisit all columns to the left of i in Q. $j < i$

Q:



A:



purpose of j: visit vectors already mutually orthogonal.

4) For all valid j, add to a copy of column i of A the following:

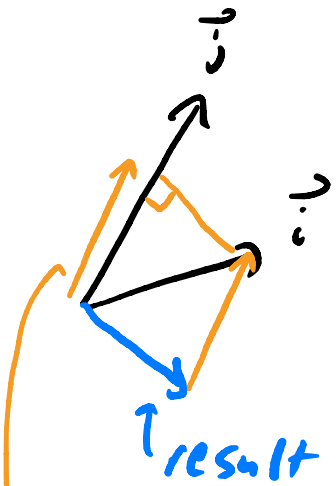
- element-wise multiplication

$$-\langle \vec{col}(i), \vec{col}(j) \rangle \cdot \vec{col}(j)$$

$\vec{col}(i)$: $(N,1)$; $\vec{col}(j)$: $(N,1)$; $\vec{col}(j)$: $(N,1)$ vector

$\langle \vec{col}(i), \vec{col}(j) \rangle$: dot product

dot product → scaler



result = $(N,1)$ "subtract off scaled version of vector j"

ie. $$\vec{col}(i) = \vec{col}(i) - \sum_{j=1}^{i-1} \langle \vec{col}(i), \vec{col}(j) \rangle \cdot \vec{col}(j)$$

temp variable
— Don't modify A

5) After subtracting off all columns j from the current i column, normalize $\vec{col}(i)$ by its length / norm:

$$\vec{col}(i) = \frac{\vec{col}(i)}{\|\vec{col}(i)\|_2} = \frac{\vec{col}(i)}{\sqrt{x_{i,1}^2 + x_{i,2}^2 + \dots}}$$

components of column i.