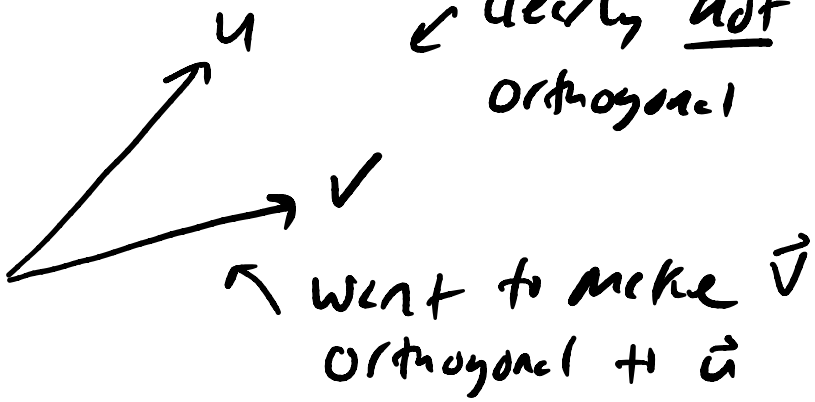
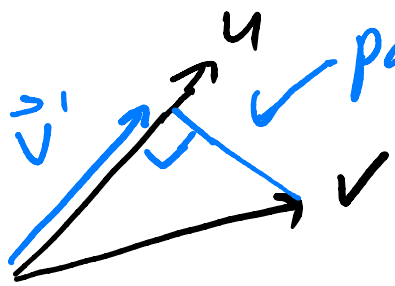
- **Gram-Schmidt algorithm (intuition):**

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

2D example:

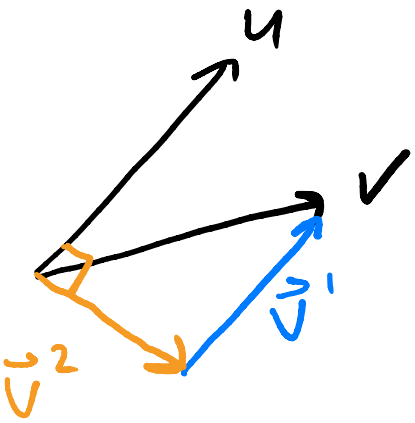


$\Rightarrow$ $\text{proj}_{\vec{u}}(\vec{v})$ = project $\vec{v}$ onto $\vec{u}$



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

$$u_1 = v_1$$

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

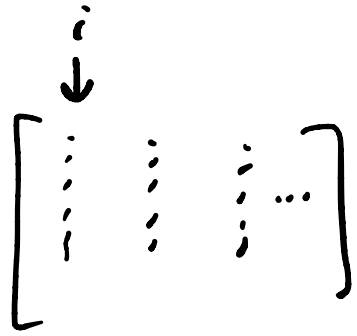
$$u_3 = v_3 - \text{proj}_{\vec{u}_1}(\vec{v}_3) - \text{proj}_{\vec{u}_2}(\vec{v}_3)$$

$$\vdots$$

• Gram-Schmidt algorithm (details):

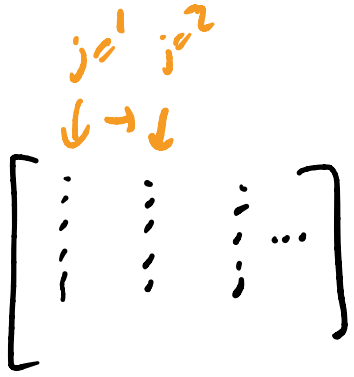
1) Initialize Q to an $(N, M+1)$ matrix.

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

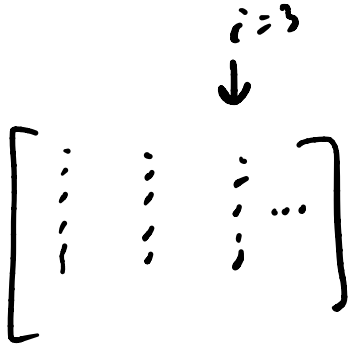


3) With a seperate "pointer" j, revisit all columns to the left of i in Q. $j < i$

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:

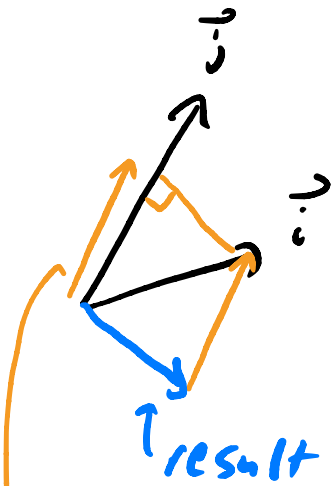
- element-wise multiplication

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

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

6) $\vec{col}(i)$, derived from A, becomes $\vec{col}(i)$ of Q.

7) Repeat for all columns $i \rightarrow$ Done!

8) Compute $R = Q^T A$

- Solve least squares problem with QR decomposition

$$A\vec{c} = \vec{y}$$

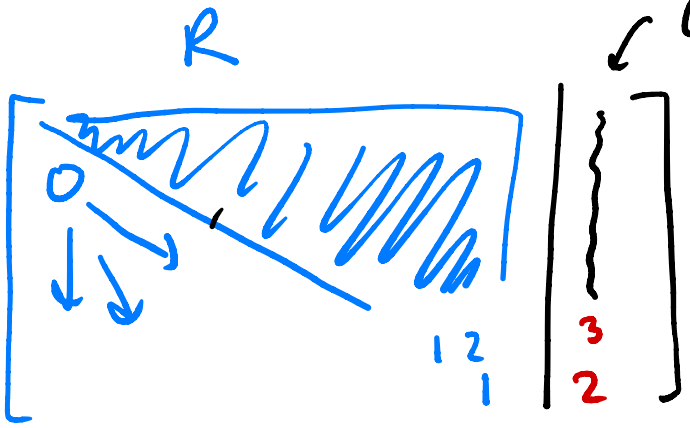
$$QR\vec{c} = \vec{y}$$

$$R\vec{c} = Q^T\vec{y}$$

$R\vec{c}$: Solve for $\vec{c}$ efficiently without inversion by back substitution b/c R upper triangular matrix.

$Q^T\vec{y}$: Shape = (M+1, 1)

Form:



ie.

$$\ddots \quad = \vdots$$

$$= \vdots$$

$$c_{n-1} + 2c_n = 3$$

$$c_n = 2$$

- In project, can use Scipy function to do back substution for you
  - scipy.linalg.solve_triangular

R C $\longrightarrow$ $(n+1, 1)$

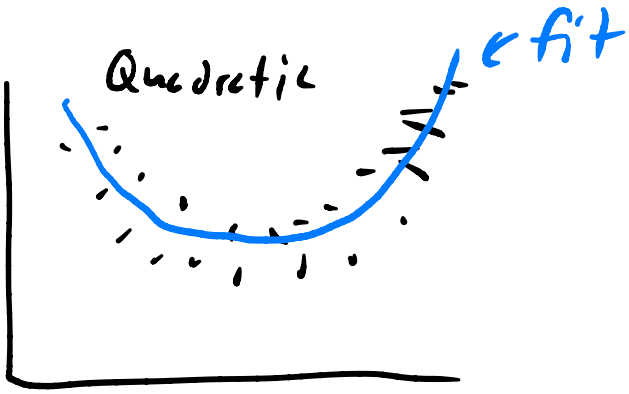
$(n+1, n+1)$ $(n+1, 1)$

$n+1$ equations

$n+1$ unknowns

Polynomial regression

- In project, you will explore fitting a polynomial regression model to one independent/dependent variable pair:



This will allow you to better fit <u>nonlinear data</u> — data that does not exhibit a linear relationship between independent and dependent variable.

Example: $\vec{x}_1 = \begin{bmatrix} 2 \\ 4 \\ 1 \\ 3 \\ 5 \end{bmatrix}$ $\vec{Y} = \begin{bmatrix} 15 \\ 5 \\ 2 \\ 11 \\ 4 \end{bmatrix}$

fit the data to the cubic polynomial model:

$$Y = C_0 + C_1 X_1 + C_2 X_1^2 + C_3 X_1^3$$

$$\underbrace{\begin{bmatrix} 15 \\ 5 \\ 2 \\ 11 \\ 4 \end{bmatrix}}_{\vec{Y}} = \underbrace{\begin{bmatrix} 1 & 2 & 4 & 8 \\ 1 & 4 & 16 & 64 \\ 1 & 1 & 1 & 1 \\ 1 & 3 & 9 & 81 \\ 1 & 5 & 25 & 125 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{bmatrix}}_{\vec{C}}$$

Let's use polynomial regression on actual data!
[Boston housing data]

problem with polynomial regression: overfitting to data

[show slides]