Least Squares and Projections

June 27, 2021

Least Squares

We want to focus on solving $A\boldsymbol{x} = \boldsymbol{b}$ for an over-determined system so A is $m \times n$. In general, there will not be a solution to this so we ask instead, what is the "best possible" approximate solution. This of course is vague so to be less vague consider all possible values of $A\boldsymbol{x}$, this is just $\mathrm{Img}(A) = \mathrm{CS}(A)$ and we want to find the point $\hat{b}$ in $\mathrm{CS}(A)$ closest to $\boldsymbol{b}$ where closest is measured in the usual notion of distance. So we mean to find $\hat{\boldsymbol{b}} \in \mathrm{rng}(A)$ so that $||\hat{\boldsymbol{b}} - \boldsymbol{b}||_2$ is as small as possible. This is equivalent to minimizing $||A\hat{\boldsymbol{x}} - \boldsymbol{b}||_2^2$, this is where the name "least squares" comes from. It is easy to see that the desired $A\hat{\boldsymbol{x}}$ is the orthogonal projection of $\boldsymbol{b}$ onto $\mathrm{rng}(A)$.

Suppose $A\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ is such that $\boldsymbol{b} - \hat{\boldsymbol{b}} \perp \mathrm{rng}(A)$, we will see that $\|\hat{\boldsymbol{b}} - \boldsymbol{b}\|_2^2 = \|A\hat{\boldsymbol{x}} - \boldsymbol{b}\|_2^2$ is minimized for such an $\hat{\boldsymbol{b}}$.

$$\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2 = \|A\boldsymbol{x} - A\hat{\boldsymbol{x}} + A\hat{\boldsymbol{x}} - \boldsymbol{b}\|_2^2$$

Since $A\boldsymbol{x} - A\hat{\boldsymbol{x}} \in \mathrm{rng}(A)$ we have $A\boldsymbol{x} - A\hat{\boldsymbol{x}} \perp A\hat{\boldsymbol{x}} - \boldsymbol{b}$ so the Pythagorean Theorem gives

$$\begin{aligned} &= \|A\boldsymbol{x} - A\hat{\boldsymbol{x}}\|_2^2 + \|A\hat{\boldsymbol{x}} - \boldsymbol{b}\|_2^2 \\ &\geq \|A\hat{\boldsymbol{x}} - \boldsymbol{b}\|_2^2 \end{aligned}$$

Rewriting $\boldsymbol{b} - \hat{\boldsymbol{b}} \perp \mathrm{rng}(A)$ we get that for all $\boldsymbol{x}$:

$$\langle \boldsymbol{b} - A\hat{\boldsymbol{x}}, A\boldsymbol{x}\rangle = (A\boldsymbol{x})^T(\boldsymbol{b} - A\hat{\boldsymbol{x}}) = \boldsymbol{x}^T A^T \boldsymbol{b} - \boldsymbol{x}^T A^T A\hat{\boldsymbol{x}} = \boldsymbol{x}^T(A^T\boldsymbol{b} - A^T A\hat{\boldsymbol{x}}) = 0$$

Recall: For all $\boldsymbol{x}$, $\boldsymbol{x}^T C = 0 \iff C = \boldsymbol{0}$.

So

$$\text{For all } \boldsymbol{x}, \boldsymbol{x}^T(A^T\boldsymbol{b} - A^T A\hat{\boldsymbol{x}}) = \boldsymbol{0} \iff (A^T\boldsymbol{b} - A^T A\hat{\boldsymbol{x}}) = \boldsymbol{0}$$

So we are searching for $\hat{\boldsymbol{x}}$ so that $A\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$ and this amounts to finding $\hat{\boldsymbol{x}}$ so that $\boxed{A^T\boldsymbol{b} = A^T A\hat{\boldsymbol{x}}}$. This equation is called the **normal equation** and we say $\hat{\boldsymbol{x}}$ is a **least-square solution** to $A\boldsymbol{x} = \boldsymbol{b}$ iff $\hat{\boldsymbol{x}}$ satisfies the normal equation.

Notice that the following are equivalent where $\hat{\boldsymbol{b}}$ is the orthogonal projection of $\boldsymbol{b}$ onto $\mathrm{rng}(A)$

- $\hat{\boldsymbol{x}}$ is a least-squares solution to $A\boldsymbol{x} = \boldsymbol{b}$.
- $A^T A\hat{\boldsymbol{x}} = A^T\boldsymbol{b}$.
- $A\hat{\boldsymbol{x}} = \hat{\boldsymbol{b}}$.

It is trivially clear that if $\boldsymbol{z} \in \mathrm{NS}(A)$ and $\hat{\boldsymbol{x}}$ is a least squares solution to $A\boldsymbol{x} = \boldsymbol{b}$, then $A(\hat{\boldsymbol{x}} + \boldsymbol{z}) = A\hat{\boldsymbol{x}} + A\boldsymbol{z} = \hat{\boldsymbol{b}} + \boldsymbol{0} = \hat{\boldsymbol{b}}$. Conversely, if $A\hat{\boldsymbol{x}} = A\boldsymbol{y} = \hat{\boldsymbol{b}}$, then $A\hat{\boldsymbol{x}} - A\boldsymbol{y} = A(\hat{\boldsymbol{x}} - \boldsymbol{y}) = \hat{\boldsymbol{b}} - \hat{\boldsymbol{b}} = \boldsymbol{0}$. Thus we have that the set of all least-square solutions to $A\boldsymbol{x} = \boldsymbol{b}$ is $\hat{\boldsymbol{x}} + \mathrm{NS}(A)$, where $\hat{\boldsymbol{x}}$ is any single least-square solution. In general, this is an infinite set unless $\mathrm{NS}(A) = \{\boldsymbol{0}\}$. The next section deals with this special case.

Special case: $\mathbf{NS}(A) = \{\boldsymbol{0}\}$

Recall that $\mathrm{NS}(A) = \mathrm{NS}(A^T A)$ so if $\mathrm{NS}(A) = \{\boldsymbol{0}\}$, then $A^T A$ is invertible and so we can use the normal equation to solve for $\hat{\boldsymbol{x}}$. In this case we get a unique $\hat{x}$, namely, $\hat{\boldsymbol{x}} = (A^T A)^{-1} A^T \boldsymbol{b}$.

General case: $\mathbf{NS}(A) \neq \boldsymbol{0}$

In this case $A^T A$ is not invertible, but there is always a matrix called the **pseudo inverse** and denoted A^+ with the property that if $\hat{\boldsymbol{x}} = A^+ \boldsymbol{b}$, then $A^T A \hat{\boldsymbol{x}} = (A^T A) A^+ \boldsymbol{b} = A^T (A A^+) \boldsymbol{b} = A^T \boldsymbol{b}$. So $A^+ \boldsymbol{b}$ is always a least-squares solution to $A\boldsymbol{x} = \boldsymbol{b}$.

In MATLAB the operation

```
%% A\b = (A^+)b = pinv(b)
x = A\b
Ainv = pinv(A)
x = Ainv*b % same result as above
```

Fitting polynomials to data.

Given a bunch of data in $\mathbb{R}^2$ of the form (x_i, i_i) for $i = 1, \ldots, N$ we can try to fit a polynomial of order m (usually much smaller than N) to the data as follows. We'd like to find $\alpha_0, \alpha_1, \ldots, \alpha_m$ so that $y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \cdots + \alpha_m x_i^m$ for all $i = 1, \ldots, N$. This can be written in matrix form as: Find a vector $\boldsymbol{\alpha}$ so that

$$\begin{bmatrix} \boldsymbol{1} & \boldsymbol{x} & \boldsymbol{x}^2 & \cdots & \boldsymbol{x}^m \end{bmatrix} \boldsymbol{\alpha} = \boldsymbol{y}$$

Here $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ and $\boldsymbol{x}^k = (x_1^k, \ldots, x_N^k)^T$. Typically if the $X_i's$ are somewhat random (maybe even distinct) and $m < N$, the coefficient matrix will have rank $m+1$ and thus we can apply the technique from above to find the least squares solution to this. (Typically there will not be an actual solution!)

Let $A = \begin{bmatrix} \boldsymbol{1} & \boldsymbol{x} & \boldsymbol{x}^2 & \cdots & \boldsymbol{x}^m \end{bmatrix}$ and $\hat{\boldsymbol{\alpha}} = (A^T A)^{-1} A^T \boldsymbol{y}$, then the m^{th} degree polynomial that best fits the data is $\hat{\boldsymbol{\alpha}}^T \boldsymbol{x} = \sum_{i=1}^N \hat{\alpha}_i x^i$ (here $\boldsymbol{x} = (1, x, x^2, x^3, \ldots, x^m)$).

The case of best fitting line is just where $m = 1$.

This is super easy to implement in Octave/MATLAB!

```
N = 600;
M = 17;

% Generate N uniformly distributed x values
% between -4 and 4.
x = 8*rand(N,1) - 4;

X = sort(x);

```

```
% Apply some function to the x values
Y = sin(4*X) + cos(3*X) - X/8; % green
% Add some noise (our simulated data)
y = Y + 2*rand(N,1)-1; % blue

% Build the matrix [1 x x^2 ... x^M]
A = zeros(N,M);

for k = 0:M
    A(:,k+1) = X.^k;
end

% find the coefficients of our M-degree poly
alpha = (A'*A)^-1*A'*y;

% Generate values based on our polynomial (red)
haty = A*alpha;

plot(X,y,"b.",X,haty,'r-',X,Y,'g-')

axis([-4 4 -1 1])
axis('square')
```

QR decomposition and least squares.

Any $m \times n$ matrix A of rank n where $n < m$, can be written as QR where Q is orthogonal $m \times n$ and R is upper triangular $n \times n$ (invertible). Recall when finding the least square solution to $A\boldsymbol{x} = \boldsymbol{b}$ we had

$$\hat{\boldsymbol{x}} = (A^T A)^{-1} A^T \boldsymbol{b}.$$

This is the same as solving

$$A^T A\boldsymbol{x} = A^T \boldsymbol{b}$$

which is equivalent to

$$A^T A = R^T Q^T Q R\boldsymbol{x} = R^T I_n R\boldsymbol{x} = R^T R\boldsymbol{x} = R^T Q^T \boldsymbol{b}.$$

and this reduces to

$$R\boldsymbol{x} = Q^T \boldsymbol{b}$$

by multiplying both sides by $(R^T)^{-1}$ which is trivial to solve by back substitution, since R is upper triangular.

Getting the QR decomposition really just follows from the Gramm-Schmidt procedure applied to the columns of A (which are assumed to be linearly independent.) Recall in GS we simply subtract the orthogonal projection of $\boldsymbol{a}_i$ onto span$\{\boldsymbol{a_1, \cdots, a_{i-1}}\}$ from $\boldsymbol{a}_i$ itself, that is:

$$\boldsymbol{q}_i = \boldsymbol{a}_i - A_{i-1}(A_{i-1}A_{i-1}^T)^{-1}A_{i-1}^T\boldsymbol{a}_i,$$

where $A_j = \begin{bmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_j \end{bmatrix}$

To make the $\boldsymbol{q}_i$'s unit vectors simply normalize them setting $\boldsymbol{\hat{q}_i} = \boldsymbol{q}_i/||\boldsymbol{q}_i||$.

Recall, that span$\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$ = span$\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\}$ by construction so

$$\boldsymbol{a}_i = <\boldsymbol{\hat{q}_1, a_i}> \boldsymbol{\hat{q}_1} + \cdots + <\boldsymbol{\hat{q}_i, a_i}> \boldsymbol{\hat{q}_i}.$$

This clearly shows that $A = QR$ where $Q = \begin{bmatrix} \hat{\boldsymbol{q}}_1 & \cdots & \hat{\boldsymbol{q}}_n \end{bmatrix}$ and hence $Q^T R = Q^T QR = IR = R$, since Q is unitary, that is $R = Q^T A$.

This yields very simple MATLAB code:

```
function [Q,R] = QR(A)

    % Usage: [Q,R] = QR(A)
    % Assumption: A is a rank m, m x n matrix
    % Returns: [Q,R], Q is unitary m x n, R is upper triangular n x n

    [m, n] = size(A);

    Q = zeros(m,n);
    R = zeros(n,n);

    % Just normalize the first vector
    q = A(:,1);
    q = q/(q'*q)^.5;
    Q(:,1) = q;

    % Run GS
    for i = 2:n
        B = A(:,1:i-1);
        q = A(:,i);
        q = q - B*(B'*B)^-1*B'*q;
        q = q/(q'*q)^.5;
        Q(:,i) = q;
    end

    R = Q'*A

end
```