Least Squares and Projections

February 26, 2021

Least Squares

We want to focus on solving $A\boldsymbol{x} = \boldsymbol{b}$ for an over-determined system so A is $m \times n$ with $m > n$ (more equations than variables) and where $\operatorname{rank}(A) = n$ so the columns are linearly independent. In general, there will not be a solution to this so we ask instead, what is the "best possible" solution. This of course is vague so to be less vague consider all possible values of $A\boldsymbol{x}$, this is just $\operatorname{Img}(A) = \operatorname{CS}(A)$ and we want to find the point in $\operatorname{CS}(A)$ closest to $\boldsymbol{b}$ where closest is measured in the usual notion of distance. So we mean to find $\hat{\boldsymbol{x}}$ so that $||A\hat{\boldsymbol{x}} - \boldsymbol{b}||_2$ is as small as possible. This is equivalent to minimizing $||A\hat{\boldsymbol{x}} - \boldsymbol{b}||_2^2 = \langle A\boldsymbol{x} - \boldsymbol{b}, A\boldsymbol{x} - \boldsymbol{b}\rangle$, this is where the name "least squares" comes from.

Consider

$$\begin{aligned} ||A\boldsymbol{x} - \boldsymbol{b}||_2^2 &= \langle A\boldsymbol{x} - \boldsymbol{b}, A\boldsymbol{x} - \boldsymbol{b}\rangle \\ &= \end{aligned}$$

Think geometrically, we want to find $\hat{\boldsymbol{b}}$ in $\operatorname{CS}(A)$ that is as close as possible to $\boldsymbol{b}$. It would make sense that $\hat{\boldsymbol{b}}$ would be the *orthogonal projection* of $\boldsymbol{b}$ onto $\operatorname{CS}(A)$. That is find $\hat{\boldsymbol{b}} \in \operatorname{CS}(A)$ so that $\boldsymbol{b} - \hat{\boldsymbol{b}}$ is orthogonal to $\operatorname{CS}(A)$.

Notice $\boldsymbol{b} - \hat{\boldsymbol{b}} \perp \operatorname{CS}(A)$ iff $A^T(\boldsymbol{b} - \hat{\boldsymbol{b}}) = \boldsymbol{0}$ so we want to solve

$$\begin{aligned} A^T\boldsymbol{b} &= A^T\hat{\boldsymbol{b}} && \text{since } \boldsymbol{b} - \hat{\boldsymbol{b}} \perp \operatorname{CS}(A) \\ \hat{\boldsymbol{b}} &= A\boldsymbol{u} \text{ for some } \boldsymbol{u} \in \mathbb{R}^n && \text{since } \hat{\boldsymbol{b}} \in \operatorname{CS}(A) \end{aligned}$$

This leads to

$$A^T\boldsymbol{b} = A^T\hat{\boldsymbol{b}} = A^TA\boldsymbol{u}$$

so

$$\boldsymbol{u} = (A^TA)^{-1}A^T\boldsymbol{b}$$

Exercise 1. Show that A^TA is non-singular. Recall A is $m \times n$, $n < m$, and $\text{rank}(A) = n$.

In this way we have solved the orthogonal projection problem and the least squares problem in one go. The least squares solution to $A\boldsymbol{x} = \boldsymbol{b}$ is

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

and the orthogonal projection of $\boldsymbol{b}$ onto CS(A) is

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

The $m \times m$ matrix

$$P = A(A^TA)^{-1}A^T$$

is called the *orthogonal projection matrix* of $\mathbb{R}^m$ onto CS(A) and $P\boldsymbol{b}$ is the orthogonal projection of $\boldsymbol{b}$ onto CS(A) for any $\boldsymbol{b} \in \mathbb{R}^m$.

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

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

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 $A = \begin{bmatrix} \mathbf{1} & \boldsymbol{x} & \boldsymbol{x}^2 & \cdots & \boldsymbol{x}^m \end{bmatrix}$ and $\hat{\boldsymbol{\alpha}} = (A^TA)^{-1}A^T\boldsymbol{y}$, 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

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

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 $\hat{\boldsymbol{q}}_{\boldsymbol{i}} = \boldsymbol{q}_i/||\boldsymbol{q}_i||$.

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

$$\boldsymbol{a}_i = <\hat{\boldsymbol{q}}_{\boldsymbol{1}}, \boldsymbol{a}_{\boldsymbol{i}}> \hat{\boldsymbol{q}}_{\boldsymbol{1}} + \cdots + <\hat{\boldsymbol{q}}_{\boldsymbol{i}}, \boldsymbol{a}_{\boldsymbol{i}}> \hat{\boldsymbol{q}}_{\boldsymbol{i}}.$$

This clearly shows that $A = QR$ where $Q = \begin{bmatrix} \hat{\boldsymbol{q}}_{\boldsymbol{1}} & \cdots & \hat{\boldsymbol{q}}_{\boldsymbol{n}} \end{bmatrix}$ and hence $Q^T R = Q^T Q R = IR = R$, 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
```