Least Squares Approximation

Gerald Farin
Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
email farin@asu.edu

October 20, 1999

1 The normal equations

In many applications, one is given more data points than can be interpolated by a polynomial curve. In such cases, an *approximating* curve will be needed. Such a curve does not pass through the data points exactly; rather it passes near them, still capturing the shape inherent to the given points. The technique best known for finding such curves is known as *least squares approximation*.

To make matters more precise, assume we are given $P+1$ data points $\mathbf{p}_0, \ldots, \mathbf{p}_P$, each $\mathbf{p}_i$ being associated with a parameter value t_i. We wish to find a polynomial curve $\mathbf{x}(t)$ of a given degree n such that the distances $\|\mathbf{p}_i - \mathbf{x}(t_i)\|$ are small. Ideally, we would have $\mathbf{p}_i = \mathbf{x}(t_i); i = 0, \ldots, P$. If our polynomial curve $\mathbf{x}(t)$ is of the form

$$\mathbf{x}(t) = \mathbf{c}_0 C_0^n(t) + \ldots + \mathbf{c}_n C_n^n(t)$$

for some set of basis functions $C_i^n(t)$, then we would have

$$\begin{aligned} \mathbf{c}_0 C_0^n(t_0) + \ldots + \mathbf{c}_n C_n^n(t_0) &= \mathbf{p}_0 \\ &\vdots \\ \mathbf{c}_0 C_0^n(t_P) + \ldots + \mathbf{c}_n C_n^n(t_P) &= \mathbf{p}_P. \end{aligned}$$

This may be condensed into matrix form:

$$\begin{bmatrix} C_0^n(t_0) & \dots & C_n^n(t_0) \\ & \vdots & \\ & \vdots & \\ C_0^n(t_P) & \dots & C_n^n(t_P) \end{bmatrix} \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} = \begin{bmatrix} \mathbf{p}_0 \\ \vdots \\ \vdots \\ \mathbf{p}_P \end{bmatrix}.$$

Or, even shorter:

$$M\mathbf{C} = \mathbf{P}. \tag{1}$$

Since we assume the number P of data points is larger than the degree n of the curve, this linear system is clearly overdetermined. We attack it by simply multiplying both sides by M^{T}:

$$M^{\mathrm{T}} M\mathbf{C} = M^{\mathrm{T}}\mathbf{P}. \tag{2}$$

This is a linear system with $n+1$ equations in $n+1$ unknowns, with a square and symmetric coefficient matrix $M^{\mathrm{T}}M$. Its soulution is staightforward, provided that $M^{\mathrm{T}}M$ is always invertible. This is in fact the case: since the C_i^n are assumed to be linearly independent, the $n+1$ columns of M are linearly independent, thus ensuring full rank of $M^{\mathrm{T}}M$.

2 The optimization approach

But is the solution meaningful? After all, we employed a rather crude trick in going from (1) to (2). It turns out that our solution is not only meaningful – in fact it is *optimal*.

In order to justify this claim (and also making it more precise), we give a second derivation of (2).

Consider the following expression:

$$f(\mathbf{c}_0, \dots, \mathbf{c}_n) = \sum_{i=0}^{P} \|\mathbf{p}_i - \mathbf{x}(t_i)\|^2. \tag{3}$$

If the curve $\mathbf{x}(t)$ does not deviate from the data points $\mathbf{p}_i$ by much, then f will attain a small value. Ideally, if $\mathbf{x}$ were to pass through all $\mathbf{p}_i$ exactly, we would have $f = 0$.

If we substitute the full definition of $\mathbf{x}(t)$ into (3), we obtain

$$f(\mathbf{c}_0, \dots, \mathbf{c}_n) = \sum_{i=0}^{P} \|\mathbf{p}_i - \sum_{j=0}^{n} \mathbf{c}_j C_j^n(t_i)\|^2. \tag{4}$$

We wish to find a set of $\mathbf{c}_i$ such that the value of f becomes minimal. Since f is a multivariate function of all components of the $\mathbf{c}_i$, such a minimum is achieved if f's partials with respect to all these components vanish:

$$\frac{\partial f}{\partial \mathbf{c}_k^d} = 0; \quad k = 0, \ldots, n; \quad d = 1, 2 \;\text{ or }\; d = 1, 2, 3,$$

where the superscript d labels the individual components of the $\mathbf{c}_k$. Computing the required derivatives yields

$$\sum_{i=0}^{P} \left[\mathbf{p}_i^d - \sum_{j=0}^{n} \mathbf{c}_j^d C_j^n(t_i)\right] C_k^n(t_i) = 0; \quad k = 0, \ldots, n; \qquad d = 1, 2 \;\text{ or }\; d = 1, 2, 3.$$

Upon rearranging, we see that this is identical to (2)!

3 Smoothing functionals

The solution to the least squares problem aims only at minimizing the error function f in (3). It does not "care" about the shape of the resulting curve. It may wiggle more than one would like, yet it is as close to the data points as possible. Sometimes wiggles are undesired, even if closeness of approximation is lost. Typically this is the case for noisy data: if the data points exibit extraneaus wiggles, there is no point in trying to reproduce these. The following shows a way to improve the shape of the approximating curve, albeit at the cost of deviating more from the given data points.

We now assume that the basis functions $C_i^n(t)$ are in fact the Bernstein polynomials $B_i^n(t)$ and the unkown coefficients are contained in a vector $\mathbf{B}$. Instead of addressing the shape of the *curve*, we will simply look at shape of the *control polygon*. Clearly, if the polygon behaves nicely, then so does the curve.

How do we measure the shape of a polygon? Many such measures are conceivable; we pick one of the simplest ones to start with. If all second differences $\Delta^2\mathbf{b}_i$ are small, then the polygon does not wiggle much. In addition to the equations (1) we might thus aim at also achieving all of the following:

$$\begin{aligned} \mathbf{b}_0 - 2\mathbf{b}_1 + \mathbf{b}_2 &= \mathbf{0} \\ &\vdots \\ \mathbf{b}_{n-2} - 2\mathbf{b}_{n-1} + \mathbf{b}_n &= \mathbf{0}. \end{aligned}$$

This we abbreviate as

$$S\mathbf{B} = \mathbf{0}. \tag{5}$$

If we simply append these equations to the overdetermined system (1), we obtain one that is even more overdetermined:

$$\begin{bmatrix} M \\ S \end{bmatrix} \mathbf{B} = \begin{bmatrix} \mathbf{P} \\ \mathbf{0} \end{bmatrix}. \tag{6}$$

It is solved in the same way as (1), i.e. by forming the symmetric linear system of normal equations. The system is solvable because the coefficient matrix of (6) still has $n+1$ linearly independent columns, as inherited from the initial matrix M.

In equation (eq:leastext), we have little control over the effect of the added equations dealing with the shape of the curve. We gain such control by weighting the two different components:

$$\begin{bmatrix} (1-\alpha)M \\ \alpha S \end{bmatrix} \mathbf{B} = \begin{bmatrix} (1-\alpha)\mathbf{P} \\ \mathbf{0} \end{bmatrix}. \tag{7}$$

For $\alpha \in [0, 1)$, this allows a weighting between the initial data fitting equations (6) and the shape optimizing equations (5). For $\alpha = 0$, we retrieve (6); for values of α near 0, we give more weight to the fitting equations. For values of α near 1, we give up much of the fitting.