Jim Herold, Computer Scientist

Least Squares Bezier Fit

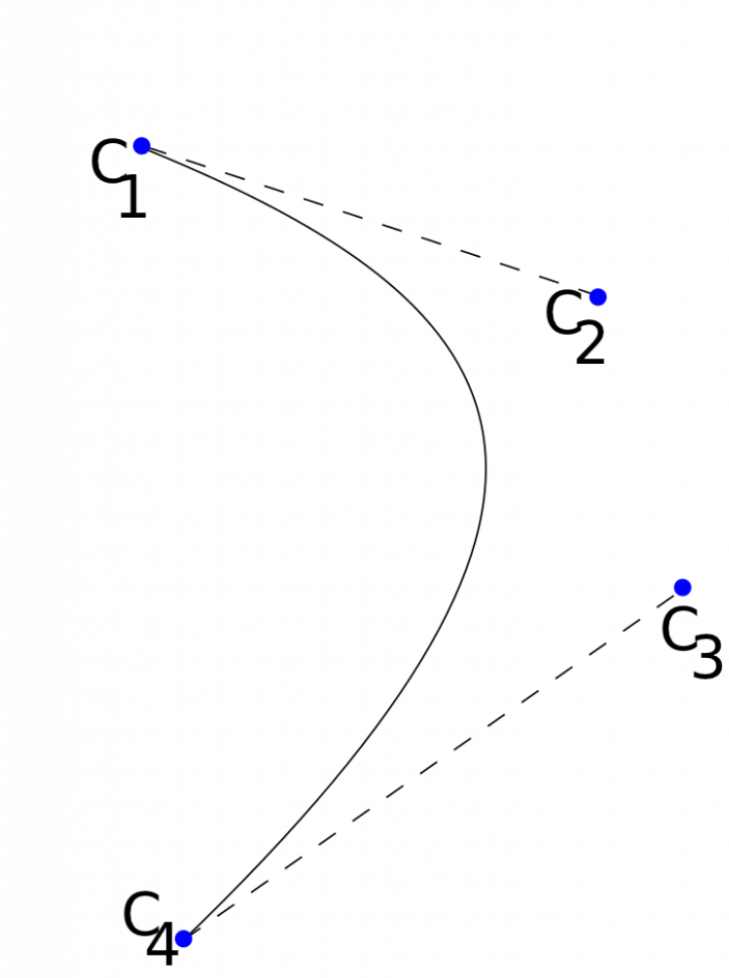
Posted on April 20, 2012 by jhero

I recently had to solve a fun math problem; I am working with digital ink data, that is time-stamped x-y coordinates of handwritten pen strokes and wanted to the best-fit Bezier curve for a segment of a stroke. In my googling I only came across approximations, e.g., gradient descent, simulated annealing, etc. These types of solutions are typically non-deterministic, slow, and do not guarantee the best solution. For that reason, I worked out a best-fit solution to this problem, very similar to linear regression.

Lets start by defining a cubic Bezier curve,

$$B(t) = c_1(1-t)^3 + 3c_2t(1-t)^2 + 3c_3t^2(1-t) + c_4t^3$$

$c_1, c_2, c_3,$ and c_4 are two dimensional Euclidean points called the control points and $t \in [0,1]$ and are demonstrated in the following curve:



A simple cubic Bezier curve and its four control points.

Simple/Intuitive explanation: At $t = 0$, B(0) = c_1 and the curve follows a trajectory towards the control point c_2. As t increases though, the "influence" will shift, such that the curve follows a trajectory towards c_4 coming from the control point c_3 and finally, at t = 1, B(1) = c_4.

To facilitate the least-squares derivation, We express the Bezier function in terms of matrix operations. Namely, given the matrices:

$$\boldsymbol{M} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{T} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{c_1} \\ \boldsymbol{c_2} \\ \boldsymbol{c_3} \\ \boldsymbol{c_4} \end{bmatrix}$$

We can now write the function for a Bezier curve as:

$$B(t) = \boldsymbol{TMC}$$

Next, lets define the input data we are trying to fit. Namely, we have a series of, N, Cartesian points comprising a segment of a handwritten stroke:

$$\boldsymbol{P} = \begin{bmatrix} P_1 \\ P_2 \\ \dots \\ P_n \end{bmatrix}$$

Such that:

$$P_i = \{x_i, y_i\}$$

If we consider the x values separately from the y values, we effectively have two vectors:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{bmatrix}$$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

We need a way to synchronize the points in the stroke segment with points in the Bezier curve when computing the error of fit. Namely, we need an index into the Bezier curve for each point along the handwritten segment. To do this, we use the path length of the segment, defined at the i^{th} point in the stroke as:

$$d_i = \sum_{j=2}^{n} |P_j - P_{j-1}|$$

$$\boldsymbol{T} = \left[\begin{array}{cccc} t_1 & t_2 & \dots & t_n \end{array} \right]$$

such that the i^{th} element of $\mathbb{T}$, T_i is equal to the percentage of path length at point i in the stroke segment:

$$t_i = \frac{|d_i - d_{i-1}|}{\sum_{j=2}^{n} |d_j - d_{j-1}|}$$

I am going to show how to fit the y-values first and the same process may simply and easily be repeated for the x-values.

As in any least squares problem, we need an equation that computes the error of fit for a given Bezier curve, defined by its control points c to the given y-values, $E(C)$. I use the residual sum of squares, summing the squared distance from each handwritten point to its Bezier curve approximation:

$$E(\boldsymbol{C_y}) = \sum_{i=1}^{n} (y_i - B(t_i))^2$$

If we define the matrix:

$$\mathbb{T} = \left[\begin{array}{cccc} t_1^3 & t_1^2 & t_1 & 1 \\ t_2^3 & t_2^2 & t_2 & 1 \\ \dots & & & \\ t_n^3 & t_n^2 & t_n & 1 \end{array} \right]$$

Then we can rewrite the error equation and find its maximum:

$$E(\boldsymbol{C_y}) = (\boldsymbol{y} - \mathbb{T}\boldsymbol{M}\boldsymbol{C_y})^T (\boldsymbol{y} - \mathbb{T}\boldsymbol{M}\boldsymbol{C_y})$$

Taking the derivative and setting it equal to zero to find the maxima:

$$\frac{\partial E}{\partial C} = 0 = -2\mathbb{T}^T (\boldsymbol{y} - \mathbb{T}\boldsymbol{M}\boldsymbol{C_y})$$

Finally, solving for $\boldsymbol{C_y}$

$$\boldsymbol{C_y} = M^{-1} (\mathbb{T}^T \mathbb{T})^{-1} \mathbb{T}^T \boldsymbol{Y}$$

And that's Howie do it. If you want to know the y-components of the best fit Bezier curve for our n points, simply apply the transformations shown in the last equations. Similarly, to find the x-components, replace the y-vector with x:

$$\boldsymbol{C_x} = M^{-1} (\mathbb{T}^T \mathbb{T})^{-1} \mathbb{T}^T \boldsymbol{X}$$

I have been receiving several requests for code and have thus created this Source Forge project that hosts a JAVA implementation of this code. It requires the UJMP package to work. Please send any feedback you have.

This entry was posted in maths, programming. Bookmark the permalink.
