Exact Data Point Fitting
Using Power Curves

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552

http://www.tinaja.com
don@tinaja.com
(928) 428-4073

What is the "best" way to draw a smooth curve through a set of data points? For most PostScript and computer graphics uses, cubic splines end up best, owing to their ease of control. We've seen lots of details on these in our cubic spline library. And especially our 4-point Bezier Point Fitter, its underlying math analysis BEZ4PTS.PDF and our Bezier curve through fuzzy data FUZZYBEZ.PSL.

But an ordinary power or Taylor Series series can sometimes be used to exactly fit any given set of data points. Which can end up a cleaner and simpler solution for certain apps. The good news is that...

> You can always fit n data points by using a n-1 power series.

The bad news is that if the points are noisy or otherwise not well behaved, the intermediate curve values may end up wildly different than expected.

The theory is simple enough: you can fit a straight line through any two points. To fit a third point off the curve, add a piece of a second order parabola. To fit a fourth point, use a parabola and then adjust the parabola with a cubic.

I've added a new CURVEFT3.PSL utility to my PostScript and Math Stuff library pages. This fits up to ten data points with an appropriate power series. To use one of the routines, you enter your data, resave as an ordinary textfile under a new name, and then send to Acrobat Disillter. A plot is returned as a .PDF file, and the magic coefficients are found in a companion log file.

Lets look at a five data point fit as an example. By the above rule, we will need a fourth order equation...

$$a(x)^4 + b(x)^3 + c(x)^2 + d(x) + e = y$$

In interests of sanity, we will usually scale our values so that the first data point is at 0,0 and our final data point 1,1. This forces e=0 and possibly gives us other simplifications. Now, all we have to do is find the values for a, b, c, and d, and we are home free.

Let's use data points of 0.0,0.0 and 0.2,0.1 and 0.4,0.3 and 0.7,0.8 and 1.0,1.0 as an example. The game plan is to create four equations in four unknowns by inserting known data point values of (x) and (y) into the above fourth order power series equation. Since there are possible subtle advantages to using your highest data points first, we will do so. And come up with...

$$
\begin{aligned}
1.0000a + 1.000b + 1.000c + 1.000d &= 1.000 \\
0.2401a + 0.343b + 0.490c + 0.700d &= 0.800 \\
0.0256a + 0.064b + 0.160c + 0.400d &= 0.300 \\
0.0016a + 0.008b + 0.008c + 0.200d &= 0.100
\end{aligned}
$$

Let's change the notation by putting it in a simpler matrix form...

$$
\begin{bmatrix}
1.0000 & 1.000 & 1.000 & 1.000 \\
0.2401 & 0.343 & 0.490 & 0.700 \\
0.0256 & 0.064 & 0.160 & 0.400 \\
0.0016 & 0.008 & 0.008 & 0.200
\end{bmatrix}
\begin{bmatrix}
1.000 \\ 0.800 \\ 0.300 \\ 0.100
\end{bmatrix}
$$

There's all sorts of ways to solve n linear equations in n unknowns. Including simple substitution, determinants, and other data reduction schemes. One useful method is known as Gauss Jordan Elimination . This works by first doing a lot of repetitive "Gauss" front end work to change the matrix into this form...

$$
\begin{bmatrix}
1 & j01 & j02 & j03 \\
0 & 1 & j12 & j13 \\
0 & 0 & 1 & j23 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
k00 \\ k01 \\ k02 \\ k03
\end{bmatrix}
$$

The j and k values are what you happen to get when you complete the "Gauss" transformations. From here, you then do more repetitive "Jordan" front end work to get this final reduced form...

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
a \\ b \\ c \\ d
\end{bmatrix}
$$

After doing the front end dogwork, you can now view the a, b, c, and d results by inspection! Simply by looking at the right matrix.

The key two rules used to cause this reduction are…

> Any matrix row can have all values multiplied or divided by any nonzero constant without changing the results.

And…

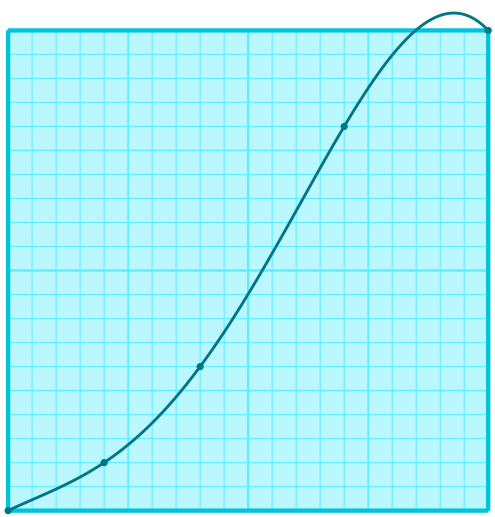
> Any matrix row can have all its values subtracted from any other row without changing the results.

This is just the same as saying you can nonzero multiply or divide everything in an equation by a constant without changing the results. And you can subtract two equations from each other term-for-term without changing the results.

To do the reduction, you scale to make j00 unity. Trivial in this case since it already is. Then you force j10 to zero by subtracting j10 times the top row from the second row. Then you force j11 to unity by dividing its row by j11. The process repeats till the principle diagonal is unity and everything below and left is precisely zero.

You then work upthe right side to do the Jordan part. By making similar zero subtractions.

Doing the actual work reveals that a=-3.86905, b=5.14881, c=-0.755954, and d=0.476191. And that our curve fitting attempt looks like this»



For More Help

Example plots are included in the utility for up to a ten data point fit. The power curve fitting scheme is extendable and useful up to twenty points or so. But at some point PostScript's 32-bit math precision will start to create problems, so you'll want to switch to JavaScript or some other 64-bit solution. It may often be better to deal with lots of points as subgroups instead. Or to add "throwaway" points at both ends where the wilder gyrations are more likely to occur.

Note that data points are best entered in monotonic increasing order, and that values very near zero or one are best avoided. This works around some of the Gauss-Jordan limitations that may crop up in more general problems.

I started exploring this method as a means of generating gamma table lookups. But I found certain data values (as in our example) may lie outside the unity box. I instead ended up with an alternate method for gamma curve generation. Which is in our Dodges & Burns tutorial in our GuruGram library. The new method keeps you inside the unity box, but may slightly miss some data points. What we have looked at here remains highly useful for other data point curve fitting apps.

Additional background along with related utilities and tutorials appears on our Math StuffGuruGram, PostScript, Cubic Spline, and Fonts & Bitmaps libraries.

Consulting assistance on any and all of these and related topics can be found at http://www.tinaja.com/info01.asp. As can other math solutions.

Additional GuruGrams await your ongoing support as a Synergetics Partner.