

Cubic Splines

By Dave Slomer

[*Note: Before starting any example or exercise below, press [2nd][F6] on the home screen to **Clear a-z.***]

Curve fitting, the process of finding a function that passes through (or near) a set of given data points, is a major application of mathematics. Often, algebra is all that is needed.

Example 1: Find the equation of the parabola passing through (2,3), (5,9), and (7,6).

Define $f(x)=a \cdot x^2+b \cdot x+c$. Then get 3 equations from the given points (fig. 1a) and find **a, b, and c** via the command **solve(ans(1) and ans(2) and ans(3),{a,b,c})** (fig. 1b). The command **f(x)|ans(1)** substitutes the values into the equation for **f** (graphed in fig. 1c).

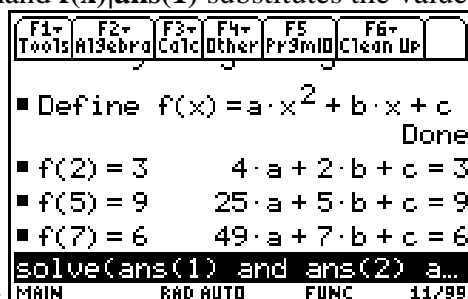


Fig. 1a

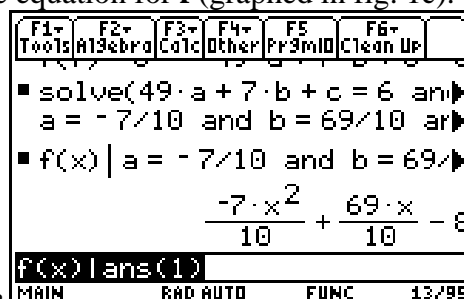


Fig. 1b

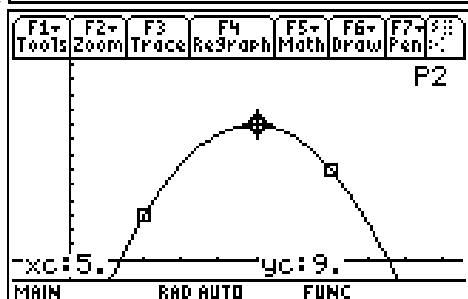


Fig. 1c

An exact fit occurs as a result of the algebra.

Data such as that in the table below (plotted in figure 2a) often lends itself to algebraic as well as statistical analysis (regression). Similar to Example 1, a system of 8 equations in 8 variables would give an exactly-fitting 7th degree polynomial, but we have no control over the resulting function (fig. 2b) and it is an algebraic nightmare. [If you try the various regression models built into the TI-89, just to see what happens, you will find that the cubic and quartic models tie for “best” fit, but are woefully inaccurate. This is not an appropriate use of regression, however.]

x	5	15	25	35	50	65	75	85
y	15	10	25	10	25	10	30	25

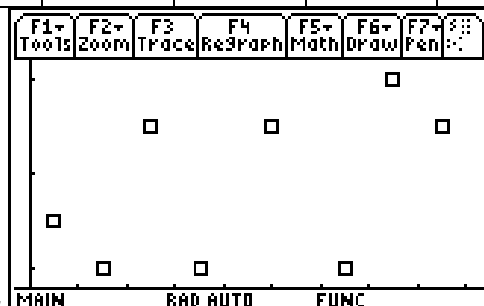


Fig. 2a

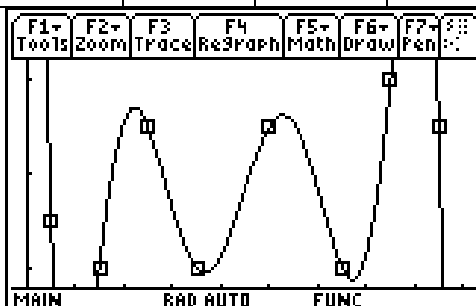


Fig. 2b

But we can control the outcome. We can produce a function that (1) hits all 8 points, (2) is differentiable everywhere, and (3) has derivative 0 at each point, as in figure 3a.

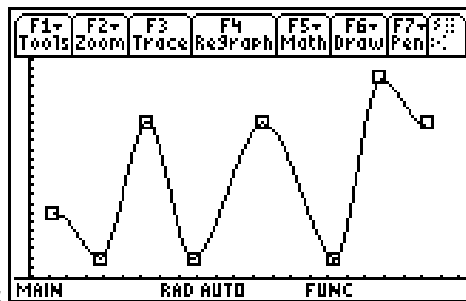


Fig. 3a

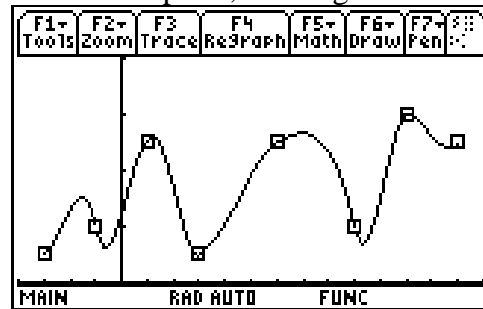


Fig. 3b

The function graphed in figure 3a is a 7-piece piecewise-defined function, each piece being a so-called **cubic spline**. Piecewise functions are not generally easy to make differentiable (making them continuous is hard enough), but any number of points can be connected by curves via cubic splines with the result always being differentiable. The idea permits a variety of ways to connect the dots (see figures 3b and 3c), including the result in figure 3d, a slightly better-behaved version of the 7th degree function (fig. 2b). Given a few “target” points, all that needs to be done is to provide the slope at each point. Needless to say, cubic splines arise from *calculus* principles. Here’s how.

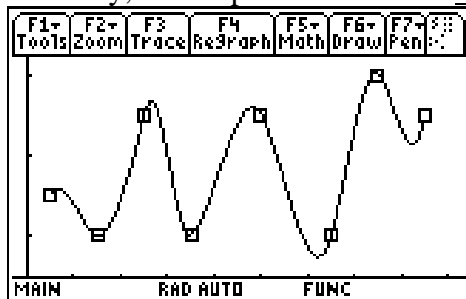


Fig. 3c

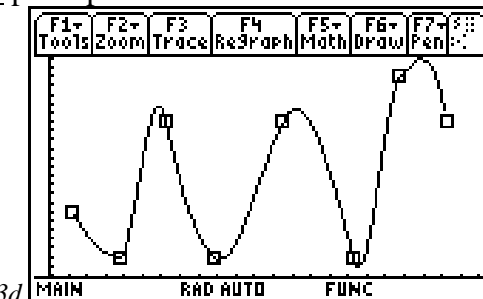


Fig. 3d

Similar to Example 1, four non-collinear points with distinct x -coordinates generally define a unique cubic [or quadratic—Why?] function. That is algebra. It is also true that two points and two slopes at those points determine one. That is calculus. So, for example, the points (2,4) and (5,1), called *nodes* in the world of cubic splines, will define a unique cubic function f provided we supply slope information for each node (that is, give values to $f'(2)$ and $f'(5)$).

Example 2: Find the cubic spline for which $f(2) = 4$, $f'(2) = 3$, $f(5) = 1$, $f'(5) = -6$.

If we define $f(x) = ax^3 + bx^2 + cx + d$, there are 4 constants to determine. We will get 4 equations from the given conditions, winding up having to solve a 4x4 linear system of equations. On the TI-89, the action looks like the screens in figures 4a through 4c.

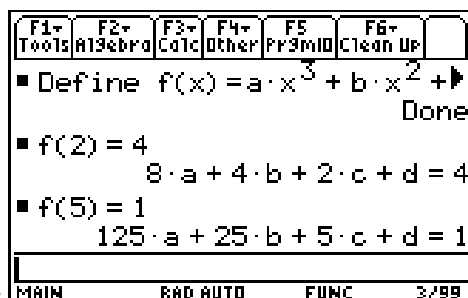


Fig. 4a

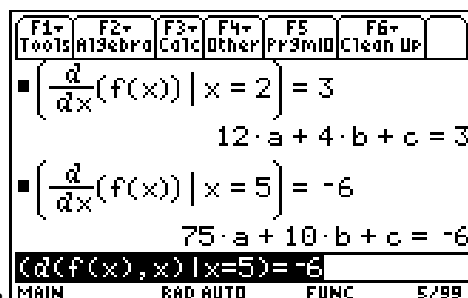


Fig. 4b

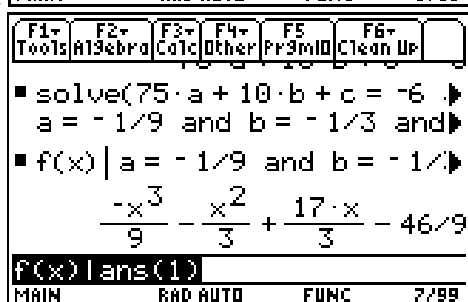


Fig. 4c

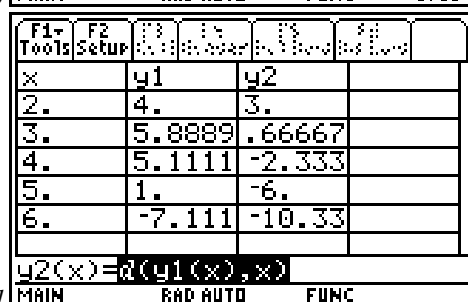


Fig. 4d

The command **solve(ans(1) and ans(2) and ans(3) and ans(4),{a,b,c,d})** would then solve the system (as shown in figure 4c and tabulated, including the derivatives, in figure 4d, as a reality check), but there is a much neater, slicker way.

Exercise 1: Re-do Example 1, writing f as a *polynomial expansion about* $x = 2$ (which is an x -coordinate of one of the given nodes): $f(x) = a(x-2)^3 + b(x-2)^2 + c(x-2) + d$.

- What does the given condition $f(2) = 4$ tell us that d must equal?
- Find $f'(x)$ (but don't simplify). What does $f'(2) = 3$ tell us that c must equal?
- So, we know that values of c and d . The 2nd node information ($f(5) = 1, f'(5) = -6$), gives a 2×2 system from which you can find a and b and, thus, the cubic spline.
- Graph the cubic spline. Are all 4 conditions ($f(2) = 4, f'(2) = 3, f(5) = 1, f'(5) = -6$) satisfied? Trace. Does it look right?

Exercise 2: The points (2,3), (5,9), and (7,6) define a unique parabolic function, as shown in Example 1. Create a piecewise function consisting of two cubic splines that make a more interesting figure (such as the one in figure 5) that is differentiable throughout [2,7].

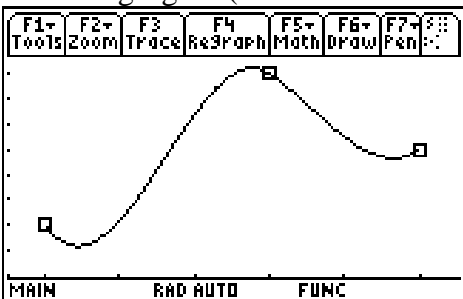


Fig. 5

To make the definition, use the TI-89 **when** command, like so:

when(x≥2 and x<5,first spline,when(x≥5 and x≤7,second spline,undef))→y1(x)

This says, literally:

If x is between 2 and 5, then

use the cubic spline result obtained from the 1st and 2nd nodes;

otherwise,

if x is between 5 and 7, then

use the cubic spline result obtained from the 2nd and 3rd nodes

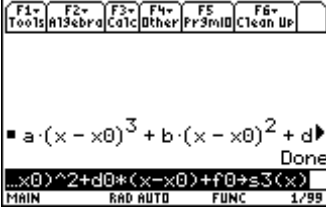
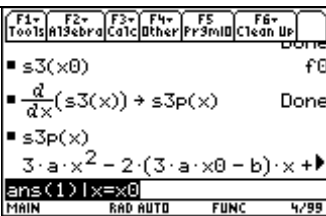
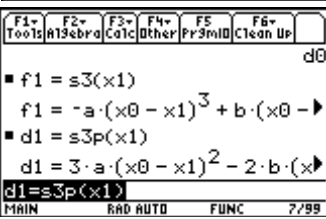
otherwise, the function is **undefined**.

- You could clean up the command above to have no **and**'s and one more **when**.
- You could make the graph more complex by adding more nodes and more splines. You would then have to add more **when**'s properly.

Exercise 3: Generalize.

Let two nodes and their slopes be given:

- at (x_0, f_0) , let the slope be d_0
- at (x_1, f_1) , let the slope be d_1

<p>Call the cubic function s3 and, learning from Exercise 1, “force” $s_3(x_0) = f_0$ and $s_3'(x_0) = d_0$ by the command Define $s_3(x) = a \cdot (x - x_0)^3 + b \cdot (x - x_0)^2 + d_0 \cdot (x - x_0) + f_0$ (fig. 6a).</p>	 <p>Fig. 6a</p>
<p>Make sure that $s_3(x_0) = f_0$. Then store the derivative of s3 it into s3p. Then type s3p(x) to recall it so you can check to make sure that $s_3p(x_0) = d_0$ (fig. 6b).</p>	 <p>Fig. 6b</p>
<p>The question is, “What must a and b equal in order for the second node and its derivative to fit into the cubic?” To make the node fit, we need $s_3(x_1) = f_1$; to make the derivative at that node fit, we need $s_3p(x_1) = d_1$. See figure 6c.</p>	 <p>Fig. 6c</p>

Because of the extreme algebraic nastiness of it all, no further screen shots can hold even the commands, let alone the results, so just carefully type the commands below.

- The command **solve(ans(1) and ans(2),{a,b})** will solve the previous two results (the equations in figure 6c) for **a** and **b**. The result is a gigantic mess that you’ll have to scroll across the screen for a long time to read [not that you need to], but it says, in essence, “**a**=...first big mess... **and b**=...second big mess...”
- The command **s3(x)|ans(1)** will substitute the big messes that **a** and **b** equal into the **s3** equation. This is the general cubic spline formula. It contains one variable, **x**, and six *parameters*, **x0**, **f0**, **d0**, **x1**, **f1**, and **d1**. To use it, you would store values into those six parameters and give the command **s3(x)**. To use it again, you’d have to store 6 more values to the parameters, repeating the process. There’s a better, shorter way.

- The command **ans(1)→s36(x,x0,f0,d0,x1,f1,d1)** copies the mess that **s3** equals into a new function that will give you the spline through nodes **(x0,f0)** and **(x1,f1)** with slopes **d0** and **d1**. To use it, you would give a command like **s36(x,2,3,4,5,6,1)**, getting the spline through (2,3) with slope 4 and through (5,6) with slope 1.

Exercise 4: Use the **s36** function from Exercise 3 to re-do Exercise 2 (or use your own better idea). The commands for re-creating the two splines of Exercise 2 will be **s36(x,2,3,...,5,9...)** and **s36(x, 5,9...,7,6...)**, with each ... being replaced by whatever slopes you used in Exercise 2 (or make up some new ones; have fun). A game plan might be to store the first spline into **y1**; the second, into **y2**. Then you could store **when(x≥2 and x<5,y1(x),when(x≥5 and x≤7,y2(x),undef))** into **y99(x)** and plot only **y99** by turning both **y1** and **y2** off via **[F4]** on the **[Y=]** screen.

Exercise 5: Using the **s36** spline-maker function of Exercise 3 and the game plan from Exercise 4, design something (a logo, the outline of a car or plane, etc.). You might want to plan it out on graph paper first, using as few nodes as possible (but as many as it takes), with well-chosen slopes (also determined from the graph paper).

As an example, the design in figure 6 was made with four commands of the form **s36(x,15,5,...,30,25,...)**. The slopes used at (15,5) were 0, .1, .5 and 1, while at (30,25) the slopes were 0, -1, -5, and -10. After getting simplified, specific results for each spline, functions (commands) of the form **when(x>15 and x<30,...,undef)** were stored to **y1**, **y2**, **y3**, and **y4**, one for each of the four splines produced by the **s36** commands.

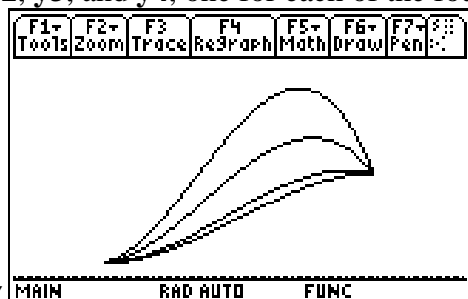


Fig. 7

The importance of cubic splines is that with just a few “control nodes” and well-chosen slopes at them, you can make some really interesting designs. If you don’t get what you want, you modify the slopes or add more nodes. The idea is used in Computer Assisted Drawing packages to enable quicker design, since relatively few points are required. The ideas are also applied in making scaleable characters for some computer printer fonts.