1. Cubic Splines, Part I

Our crude graphics program draws curves by connecting points with straight line segments, which is fine, if the length of your segments is comparable to the size of your pixels. As we saw before, however, it is more efficient and adaptable, if we store a description of a curve rather than a bitmap, or in this case, a collection of points to join with segments.

Imagine drawing a picture made up of curves on a computer. It would be inconvenient to figure out equations for the curves. In Microsoft Paint, for example, you can use your mouse to draw curves, and the computer saves something that is roughly equivalent to a bitmap. If you’ve done this, you have probably found that it is not terribly easy to get a good result. Even if you do, bitmaps are not efficient, and if you want to change the shape of a curve, your only choice is to erase the curve and redraw it.

In a more advanced graphics program, like CorelDraw for example, you can click on a series of points and indicate a tangent vector at each. The computer then draws a smooth curve through your points that is consistent with your tangent vectors. We know that we can do this with a Hermite polynomial, but as we saw, this can be somewhat cumbersome, especially if there are a lot of points.

The standard compromise is to use a cubic Hermite polynomial between consecutive points. Since the tangent vectors are the same where the cubic “segments” come together, the result is a reasonably smooth curve (the curve will be continuous, and the derivative will be continuous, but the second derivative will not be continuous in general, so these curves are only $C^1$). The curve is called a cubic spline.

2. The Basic Cubic Segment Formula

We’ll do a cubic spline as a parameterized curve, and as a result, we can always use a parameter that runs from $t = 0$ to $t = 1$ for each segment. Let’s do the math.

For the Hermite polynomial, we will assume $t_0 = 0$ and $t_1 = 1$, $f(t_0) = y_0$, $f(t_1) = y_1$, $f'(t_0) = y'_0$, and $f'(t_1) = y'_1$.

First of all, the $L$-functions.

\[
L_{10}(t) = \frac{t - 1}{0 - 1} = -t + 1
\]

(1)

\[
L_{11}(t) = \frac{t - 0}{1 - 0} = t
\]

The derivatives are

\[
L'_{10}(t) = -1
\]

(2)

\[
L'_{11}(t) = 1
\]

The $H$-functions are

\[
H_{10}(t) = [1 - 2(t - 0)(-1)](-t + 1)^2 = 2t^3 - 3t^2 + 1
\]

(3)

\[
H_{11}(t) = [1 - 2(t - 1)(1)](t)^2 = -2t^3 + 3t^2
\]

The $\hat{H}$-functions are

\[
\hat{H}_{10}(t) = (t - 0)(-t + 1)^2 = t^3 - 2t^2 + t
\]

(4)

\[
\hat{H}_{11}(t) = (t - 1)(t)^2 = t^3 - t^2
\]

This should all look familiar. The Hermite polynomial is

\[
P(t) = y_0(2t^3 - 3t^2 + 1) + y_1(-2t^3 + 3t^2) + y'_0(t^3 - 2t^2 + t) + y'_1(t^3 - t^2)
\]

(5)

\[
= (2y_0 - 2y_1 + y'_0 + y'_1)t^3 + (-3y_0 + 3y_1 - 2y'_0 - y'_1)t^2 + y'_0t + y_0
\]
Here's a program that draws the graph of $P$ from $t = 0$ to $t = 1$.

```
program cubic1;
uses graph,math;
var
  gd,gm,xc,yc,i,n : integer;
  x,y,step,t,y0,y1,yp0,yp1 : real;
  PathToDriver : string;
function P(t,y0,y1,yp0,yp1 : real) : real;
begin
  P := (2*y0-2*y1+yp0+yp1)*intpower(t,3)
         + (-3*y0+3*y1-2*yp0-yp1)*intpower(t,2)
         + yp0*t + y0;
end;
begin
  writeln('Enter function value for first point.');
  readln(y0);
  writeln('Enter derivative value for first point.');
  readln(yp0);
  writeln('Enter function value for second point.');
  readln(y1);
  writeln('Enter derivative value for second point.');
  readln(yp1);
  writeln('Enter number of plotted points desired.');
  readln(n);
  step := 1/n;
  gd := Detect;
  gm := 0;
  PathToDriver := 'c:4400pc2k';
  initgraph(gd,gm,PathToDriver);
  t := 0;
  x := t;
  y := P(t,y0,y1,yp0,yp1);
  xc := round(300+200*x);
  yc := round(350-200*y);
  moveto(xc,yc);
  for i:=1 to n do
    begin
      t := t+step;
      x := t;
      y := P(t,y0,y1,yp0,yp1);
      xc := round(300+200*x);
      yc := round(350-200*y);
      lineto(xc,yc);
    end;
  readln;
  closegraph;
end.
```
Note that the program actually plots \( n + 1 \) points, and that the number of times the for-loop runs can be controlled by a variable.

1. How far left and right can the graph get (in number of pixels from the left of the screen)?

2. What is the largest \( y \)-value that stays on the screen?

3. What is the smallest (most negative) \( y \)-value that stays on the screen?

4. Shown below are some basic cubic shapes. Can you get these with the program?

3. Parametric Cubics

The program given above graphs the graph of a cubic function. As such, you could not get curve like the one shown below.

This picture, by the way, is a simulation of what you would see in CorelDraw. By pulling on the blue arrows, you can change the shape of the curve by altering the direction of the arrows (the line of tangency) and also by changing the length of the arrows.

The math is essentially the same as what we’ve just done, except we compute a Hermite polynomial for both coordinates, not just the \( y \)-coordinate. Remember back to Calculus III. We can have a vector valued function

\[
\mathbf{r}(t) = (x(t), y(t)),
\]

where each value of \( t \) determines a point on the plane \( \mathbf{r}(t) \), and all together, we have a curve. The derivative of \( \mathbf{r} \) is also known as the velocity vector;

\[
\mathbf{v}(t) = \mathbf{r}'(t) = (x'(t), y'(t)).
\]

This name comes from the fact that if you were moving along the curve so that you were at position \( \mathbf{r}(t) \) at time \( t \), then \( \mathbf{v}(t) \) is the velocity. That is, \( \mathbf{v}(t) \) is tangent to the curve, so it indicates the direction of travel, but its length indicates speed. The velocity vector, therefore, gives us more information than the slope alone.

If we have the position of two points and a velocity vector at each point, we have essentially the following information: \((x(0), y(0)), (x'(0), y'(0)), (x(1), y(1)), \) and \((x'(1), y'(1))\). We can do what we did above, only twice.

4. Project 03

I want you to write a program similar to cubic1, but for a parametric cubic curve. It should ask for and read in the two endpoints, the two velocity/tangent vectors, and the number of segments to be drawn (more segments means a smoother curve). It should then draw the cubic curve. It should be able to handle any curve within an \( x = -2..2 \) by \( y = -2..2 \) box.