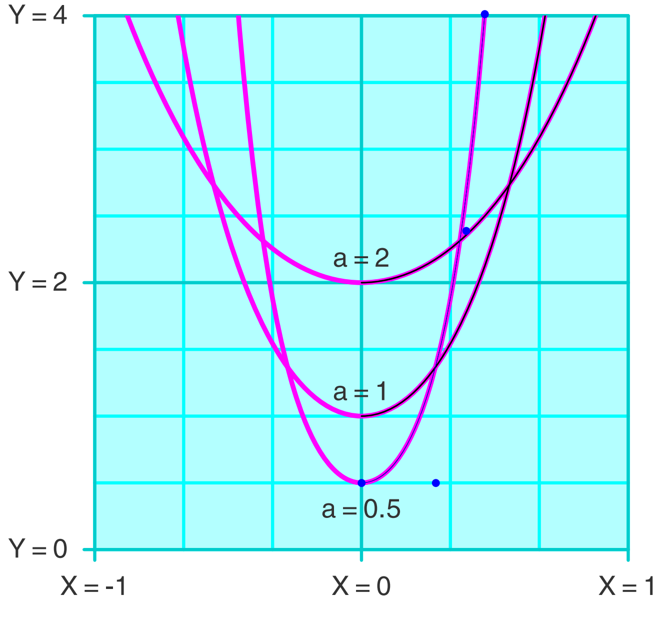
Cubic Spline Approximations To a Catenary Curve

Don Lancaster
Synergetics, Box 809, Thatcher, AZ 85552
http://www.tinaja.com   don@tinaja.com
copyright c2006 as GuruGram #69.
(928) 428-4073

One very interesting geometric form is the **catenary**. Which is the **shape that a chain will assume when hanging from its own weight**. Or inverted, is an arch form supporting only its own weight in which **all stress is along the line of the arch**. With no shear forces.

Attempting to approximate a **catenary** with one or more **Bezier cubic splines** leads to several unexpected surprises. But ends up quite easily and simply done.

Here is a group portrait of several catenaries...



There are two common expressions for the catenary, one involving hyperbolic trig and one involving series. The more used **trig form** is...

$$y = a \cosh(x/a)$$

Control parameter "**a**" gets set by how long the chain is compared to the width of its tie points. **Most useful catenaries will have $a > 0.5$**.

The less used but more informative **series form** is…

$$y = 1 + x^2/2!a + x^4/4!a^3 + x^6/6!a^5 + x^8/8!a^7 + \ldots$$

which reduces to…

$$y = 2 + x^2/4 + x^4/192 + x^6/23040 + \ldots \qquad \text{for } a=2$$

$$y = 1 + x^2/2 + x^4/24 + x^6/720 + \ldots \qquad \text{for } a=1$$

$$y = 0.5 + x^2 + x^4/3 + 2x^6/45 + x^8/315 + \ldots \qquad \text{for } a=0.5$$

While others may claim that a catenary is "**almost a parabola**", those fourth, and sixth power terms are usually significant. Even the eighth power term makes a noticeable but small difference at a=0.5 and becomes very important for smaller values of a. Or for larger values of x.

The catenary **slopes** may be of interest when cubic spline fitting…

$$dy/dx = x/2 + x^3/48 + x^5/3840 + \ldots \qquad \text{for } a = 2$$

$$dy/dx = x + x^3/6 + x^5/120 + \ldots \qquad \text{for } a = 1$$

$$dy/dx = 2x + 4x^3/3 + 4x^5/15 + 8x^7/315 + \ldots \qquad \text{for } a = 0.5$$

Just Plot It

The catenary itself is easily and simply plotted in **PostScript**. Just as we have done in our above figure. And may add only 1K or so to file length and may calculate in less than a tenth of a second. So there might not be much need to actually create a cubic spline approximation. Even though the spline fit should execute much faster, store in only a few bytes, and be device and final size independent.

PostScript has no hyperbolic cosine or **cosh** command, which we can add by this definition pair…

```
/ee 2.718281828 store

/cosh { /x1 exch store /a1 exch store
x1 a1 div dup ee exch exp exch neg ee
exch exp add a1 mul 2 div } store
```

We have used this to plot the red portions of our previous figure. Resolution is 100 steps per x unit and can be read from the **sourcecode** to this **GuruGram**.

Should you wish to avoid hyperbolic cosines in another language, the power series approximations can be used instead. The thin black lines overlain on the above figure for **x>0** show how good the series approximations are. Sixth order curves are used for **a=1** and **a=2**, while an eighth order curve corrects a slight but measurable error at **a=0.5**.

You can easily and extremely magnify our figure to note these small differences.

Cubic Spline Catenary Fitting

A review of cubic spline fundamentals appears **here**. With bunches of additional support in our **Cubic Spline** library. A Bezier cubic spline can be normally coded in **PostScript** as...

```
x0 y0 moveto    x1 y1 x2 y2 x3 y3  curveto
```

Here **x0** and **y0** are the initial position of our curve, while **x3** and **y3** are its final position. **x1** and **y1** set the direction and enthusiasm (or tension) with which the spline exits its initial position. While **x2** and **x3** set the direction and enthusiasm with which the spline enters its final position.

One rude surprise quickly becomes apparent...

A full and balanced catenary consists of EVEN POWERS only!

A single spline approximation will have a ZERO cubic term!

Thus, **a single cubic spline is useless to approximate a catenary because its zero cubic term will downgrade it into a quadratic**. At least **two** splines will be needed to accurately approximate a catenary.

One for positive values and one for negative.

We have already seen several approaches to cubic spline approximations to various curves and functions in our **Cubic Spline** library. These included our original **Gonzo Curvetracing** and our later **Bezier Curve through Four Points**.

A method that would likely be better for catenaries is to set up a **fitcat** routine. One that accepts an initial and final x and an initial and final tension. Either trial and error or a least squares process can then be done to give us the best possible spline **with slope and position matches at both ends**.

Here is a hand coded routine that is specific to **a=0.5** ...

```
/fitcat.5 { /extent exch store /intent exch store
          /x3 exch store /x0 exch store

          /y0 0.5 x0 cosh store /y3 0.5 x3 cosh store

          /entang x0 findslopea.5  store
          /x1 x0 entang cos intent mul add store
          /y1 y0 entang sin intent mul add store

          /extang x3 findslopea.5  store
          /x2 x3 extang cos extent mul neg add store
          /y2 y3 extang sin extent mul neg add store

          % insert any scaling and offsetting here
          % insert optional dot plotting here

          x1 y1 x2 y2 x3 y3 curveto

          } store
```

This can be called by **0 1.386 1.10 0.10 fitcat.5** or whatever. Inputting four values of initial and final x and initial and final slope tension.

The routine directly grabs **x0** and **x3**. It then calculates **y0** and **y3** via the hyperbolic cosines of the catenary. It next calculates the input and exit **slopes** and converts them to slope **angles** using the **PostScript atan** operator. **x1** and **y1** are found by scaling the slope angle by the initial tension. **x2** and **y2** are similarly found by scaling the slope angle by the final tension. **Note that the final tension is rotated by 180 degrees**. Finally, a plain old **curveto** is done using the captured or calculated spline control points.

Optional scaling and control point dot plotting can be added as needed. The scaling used in our figure graph was **x′ = 3.3333 x + 10** and **y′ = 5y**. Control point viewing is useful when debugging.

The actual slopes are calculated by this or a similar service proc...

```
/findslopea.5 { /xx exch store
                    xx 2 mul xx dup mul xx mul 4 mul 3 div
                    add xx dup mul xx mul xx dup mul add 4
                    mul 15 div add xx dup mul xx mul dup
                    mul xx mul 8 mul 315 div add 1 atan} store
```

The blue line on the right of the **a=0.5** curve in our figure shows this solution. I've purposely kept this "not quite" optimized so you can experiment with your own tension values. As before, you can use extreme (up to 6400 percent) Acrobat magnification to observe how good your fits are.

A single pair of splines looks good enough for most common catenaries. Very low values of **a** or higher values of **x** may require multiple spline fits. The best possible multiple spline fits likely should have matching entry and exit tensions.

This example code can be further automated and generalized either through **additional consulting** or as an exercise for the serious student.

For More Help

As previously mentioned, the **sourcecode** for this **GuruGram** includes ready-to-run **PostScript** spline fitting routines.

Additional assistance is available via our **Cubic Spline** and **Consulting Infopack** library pages.