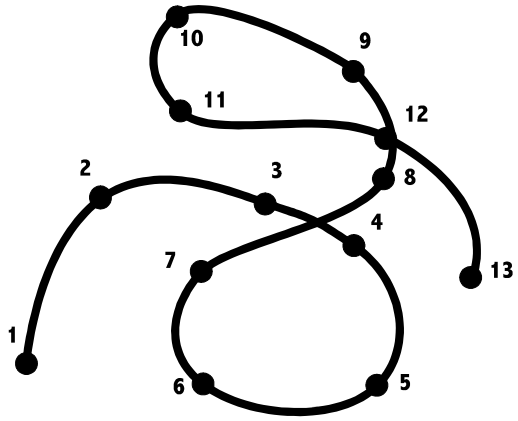
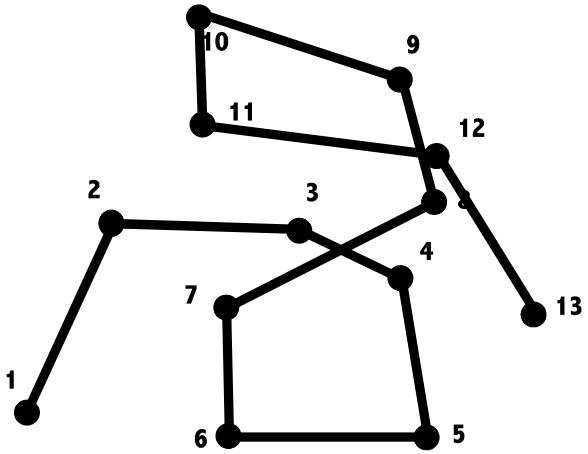
Using One-Dimensional Splines to Fit General Curves in the Plane
Due Tuesday Nov. 19 by 9:30 AM

Suppose one knows how to fit a one-dimensional spline (i.e., $y = s(x)$). Given an array



$[(x_1, y_1),(x_2, y_2),\ldots,(x_n, y_n)]$ of n points in the plane, how can this one-dimensional approach be used to fit a curves through those points in the proper order? One might be tempted to try to break the curve into pieces, fit each piece separately, and somehow glue the pieces together. The example above shows how terrible that would work. Would the first piece include points 1 through 5 or just 1 through 4? If it included 1 through 5 it could not be moving to the left as it passed through point 5 - the reversal in the horizontal direction would have to come *after* point 5. But that probably wouldn't look right. Even if the break were made between points 4 and 5, the joint between the two pieces would have an infinite slope (something rarely allowed in curve fitting). Lastly, if one simply rotated the original data, the curve would not be a simple rotation. In fact, one might have to start all over breaking the points into groups. What's called for is an approach that is more "directional independent" where there is nothing special about horizontal or vertical directions.

Here is a semi-clever idea. Begin with the simple piece-wise linear fitting of the data:



This has the general shape we want but we don't like the sharp corners. Suppose, however that we simply use this curve to define a distance parameter. We do it this way.

Let

$$s_1 = 0$$

and for i = 2, …, n

$$s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\ .$$

You may think of s as a "polygonal arclength parameter" measuring the distance along the piecewise linear curve (also known as the "polygon"). Suppose now that f is a real-valued function with the property that

$$f(s_i) = x_i \text{ for } i = 1, \ldots, n$$

and g is a real-valued function with the property that

$$g(s_i) = y_i \text{ for } i = 1, \ldots, n.$$

The pair (f, g) maps into the plane and satisfies '

$$(f, g)(s_i) = (x_i, y_i) \text{ for } i = 1, \ldots, n.$$

In other words, this is a parametric representation of a curve in two dimensions that maps the interval $[0, s_n]$ onto a curve passing through the points $\{(x_i, y_i)\}$ in exactly the right order. Furthermore, if f and g are smooth functions, perhaps so is this curve. (That's not automatic, by the way, but it's true as long as the derivatives f' and g' are never both zero at the same point.) The obvious choice is to make both f and g cubic splines in the parameter s.

The process is this:

1. Define $s_1 = 0$ and $s_i = s_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$ for i = 2, …, n.
2. Fit the cubic spline f satisfying $f(s_i) = x_i$ for i = 1, …, n. (i.e. preprocess)
3. Fit the cubic spline g satisfying $g(s_i) = y_i$ for i = 1, …, n. (i.e. preprocess)
4. For any value $s \in [0, s_n]$, the point $(f(s), g(s))$ is a point on the curve. (i.e evaluate)

Assignment:

Write a `Matlab` function `Spline2D (x, y, m)` whose inputs are column arrays `x` and `y` (of the same length) and `m`, a positive integer. Output is a pair of arrays `[xout, yout]` of length `m` that are the mappings of m equipaced points from 0 to s_n onto the curve.

Test your function with these data sets:

1. Nine points on a circle: `t = linspace (0, 2*pi, 9)'; x = cos(t); y = sin(t);`
   Use m = 101.
2. 25 points on a circle: `t = linspace (0, 2*pi, 25)'; x = cos(t); y = sin(t);`
   Use m = 501.
3. 31 points on a squiggle: `t = linspace (0, 2.5, 31)'; x = sin(t)+1/2*cos(t).*sin(5*t); y = sin(t);` Use m = 501.

Plot each with something of the form `plot (x, y, 'ro', xout, yout 'b')` to see both the original points and your curve. (If this simply whets your appetite, develop an extension to three dimensions.)