**“Nice” plotting of proteins II**

Final remark regarding efficiency: It is possible to write the Newton representation in a way that can be computed efficiently, using similar bracketing that we made for the first representation of the polynomial

$$polynomial(x) = c_1 + (x - x_1)\left(c_2 + (x - x_2)\left(c_3 + (x - x_3)...\right)\right)$$

**Is polynomial interpolation good for plotting protein curves?**

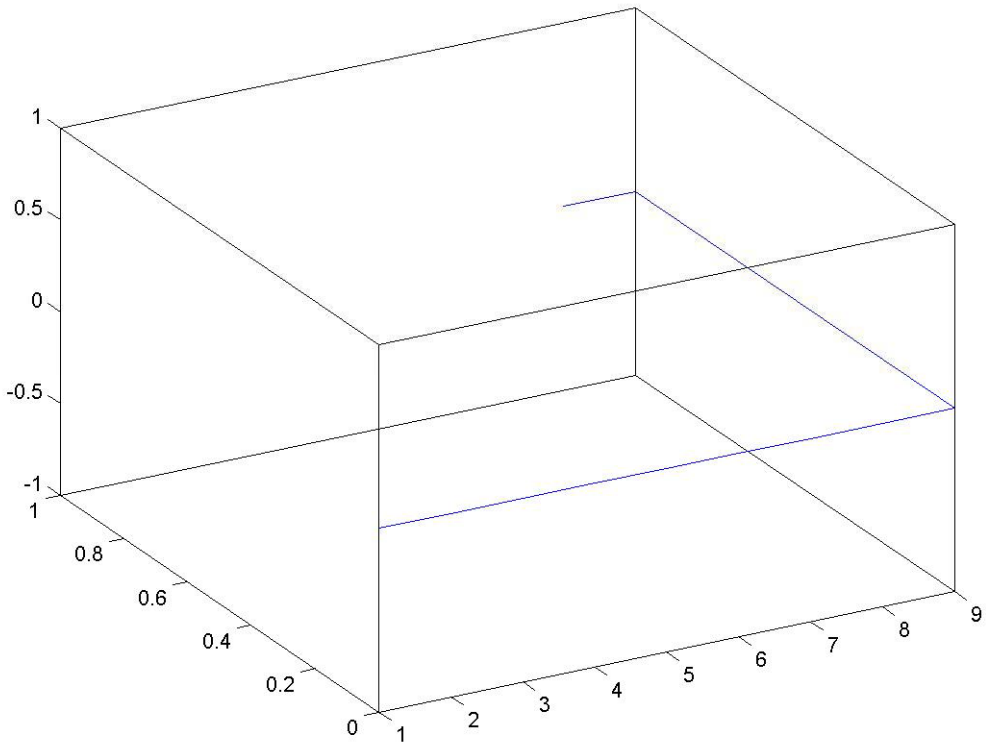
After the lengthy “introduction” to polynomials and interpolation it is about the time to ask: “Is the polynomial fit any good for protein chains”? Can we use polynomial fits to get a much smoother and better views of protein structures?

Polynomial fits are sound if higher order derivatives of the “true” function (beyond the order of the polynomial) do not contribute significantly to the description of the function. However, the reverse is also true… If high order derivatives do make a significant contribution then the fit is not sound. Consider for example a typical secondary structure element in protein - the beta pleated sheet. To a first approximation a beta pleated sheet is a simple straight line where the $C_\alpha$ atoms are equally space on it. Since the protein shape is compact the chain must come back on itself. At the end of a secondary structure element, there is usually a sharp turn modifying considerably the direction of the chain.

Let us try to create a simple model of the above qualitative description and check what is the result of a polynomial fit that we may obtain. Our chain will be embedded (for simplicity) in two dimensions. The chain is of length 10, the first nine points are along the X-axis and the last two points represent a sharp turn backward.

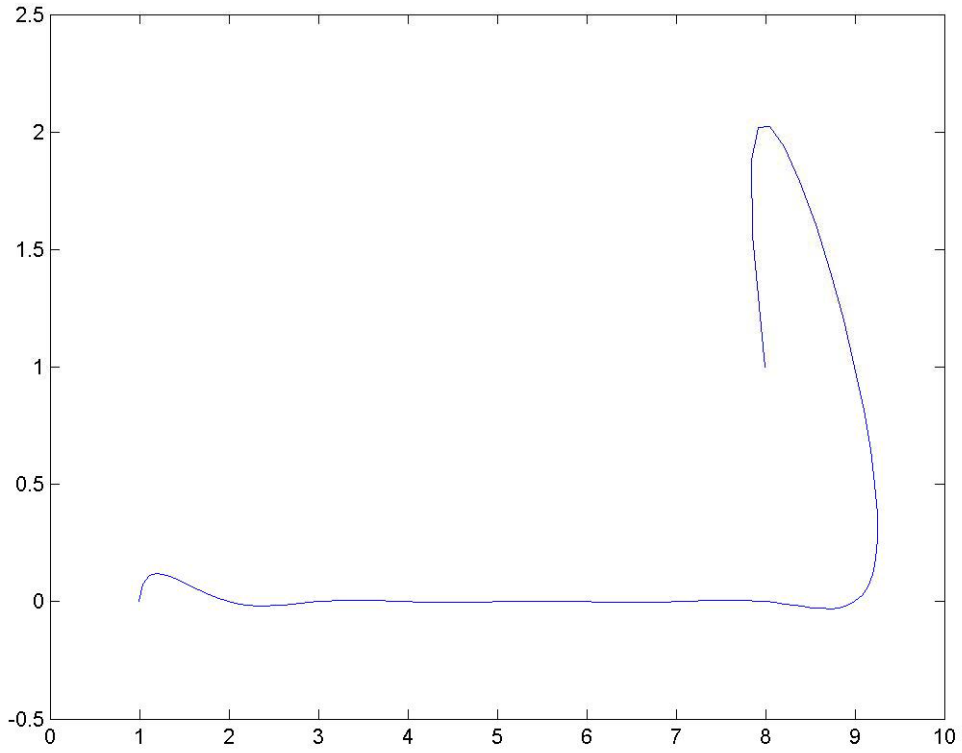
Here is a quick view of the above in MATLAB

```
>> x

x =

   1    2    3    4    5    6    7    8    9    9    8

>> y

y =

   0    0    0    0    0    0    0    0    0    1    1

>> plot(x,y)
```




Now let us try to get a polynomial fit to the above “protein model”.
Here is a MATLAB code that does it:

```
>> t = 1:11;
>> cx = polyfit(t,x,length(t)-1);
>> cy = polyfit(t,y,length(t)-1);
>> t=1:0.1:11;
>> polyx = polyval(cx,t);
>> polyy = polyval(cy,t);
>> plot(polyx,polyy)
```




Well, it is quite clear that the over-shooting of the turn is not what we had in mind about a smooth and more pleasing representation of the protein chain.

We need to find an alternative way of generating the smooth representation of the protein path.

The problem we have is that we fit a large number of points (amino acid positions) to a single polynomial. If the curve varies significantly and abruptly as the index of the amino acid (the $t$ variable), then "violent" responses, which do not coincide with our view of protein shapes, may occur.

The polynomial is smooth and differentiable to all orders. However, for graphical presentations, this may be unnecessary. We can hardly detect (by looking at a curve) that a third derivative of the curve is discontinuous. We usually are able to detect a discontinuous second derivative. We therefore set a somewhat milder threshold – a fit that is **locally** continuous up to a second derivative. This leads to the notion of piecewise continuous function and to cubic spline interpolation.

The problem can be formulated as follows:

Given a set of points $(t_1, x_1), (t_2, x_2), ..., (t_n, x_n)$ we wish to fit a piecewise continuous curve to it. This means that we will fit sequential subsets of points to cubic functions and

insist that the function, first and second derivatives will match at the boundary. The Hermite interpolation scheme is simple and therefore a good starting point:

Consider the edges of the interval that we wish to fit ( $x_L$ and $x_R$ - two points only of the complete data points). We write the polynomial $q(x)$ interpolating between the two points as:

$$q(x) = a + b\left(x - x_L\right) + c\left(x - x_L\right)^2 + d\left(x - x_L\right)^2\left(x - x_R\right)$$

The parameters $a - d$ are determined from the conditions that the function and the first derivative will be continuous at the edges of the interval (when we patch additional polynomials in between). There are four conditions that are sufficient to determine the four parameters (function and first derivative of the function at the two end points).

The set of linear equations that we need to solve is

$$a = y_L$$

$$b = y'_L$$

$$a + b \cdot \Delta x + c \cdot \Delta x^2 = y_R$$

$$b + 2 \cdot c \cdot \Delta x + d\Delta x^2 = y'_R$$

$$\left(\Delta x = x_R - x_L\right)$$

The above procedure makes the first derivatives continuous at the matching points. The second derivatives **are not** considered. We therefore immediately upgrade the level of our interpolation to cubic spline, adding a condition also on the second derivative

We write the cubic polynomial at two indices $i$ and $i+1$ building on the hermite polynomial results but adding a new parameter $s$

$$q_i(x) = y_i + s_i\left(x - x_i\right) + \frac{y'_{i+1} - s_i}{\Delta x_i}\left(x - x_i\right)^2 + \frac{s_i + s_{i+1} - 2y'_i}{\left(\Delta x_i\right)^2}\left(x - x_i\right)^2\left(x - x_{i+1}\right)$$

$$q_{i+1}(x) = y_{i+1} + s_{i+1}\left(x - x_{i+1}\right) + \frac{y'_{i+1} - s_{i+1}}{\Delta x_{i+1}}\left(x - x_{i+1}\right)^2 + \frac{s_{i+1} + s_{i+2} - 2y'_{i+1}}{\left(\Delta x_{i+1}\right)^2}\left(x - x_{i+1}\right)^2\left(x - x_{i+2}\right)$$

where $\Delta x_i = x_{i+1} - x_i$ and $y'_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$

Note that regardless of the value of $s$, at the matching point $x_{i+1}$ the function and the derivative of the two cubics are the same. For example, the functions are evaluated below in details

$$q_i(x_{i+1}) = y_i + s_i(x_{i+1} - x_i) + \frac{y'_{i+1} - s_i}{\Delta x_i}(x_{i+1} - x_i)^2$$

$$= y_i + s_i(x_{i+1} - x_i) + \left(\frac{y_{i+1} - y_i}{x_{i+1} - x_i} - s_i\right)(x_{i+1} - x_i) = y_{i+1}$$

$$q_{i+1}(x_{i+1}) = y_{i+1}$$

As an exercise you may wish to show the continuity of the first derivative

We are therefore left with the requirement of continuous second derivatives and undetermined parameters $\{s_i\}$ that are used to obtain this result. We have

$$q_i''(x_{i+1}) = \frac{2}{\Delta x_i}\left(2s_{i+1} + s_i - 3y_i'\right)$$

$$q_{i+1}''(x_{i+1}) = \frac{2}{\Delta x_{i+1}}\left(3y_{i+1}' - 2s_{i+1} - s_{i+2}\right)$$

that must be the same.
Fixing the value of $s_1 = a$ and $s_n = b$ we can solve a set of linear equations for the n-2 variables. These equations provides a piecewise continuous representation of the curve with continuous function, and first and second derivatives at each of the boundaries. It is therefore expected to work a lot better in the interpolation of protein chains.

Note however that the choice for $s_1$ and $s_n$ is not unique and there are three different ways of "initiating" and "terminating" a spline. The procedure we described above

(i) ($s_1 = a$ and $s_n = b$ ) is called "the complete spline". The other options are:

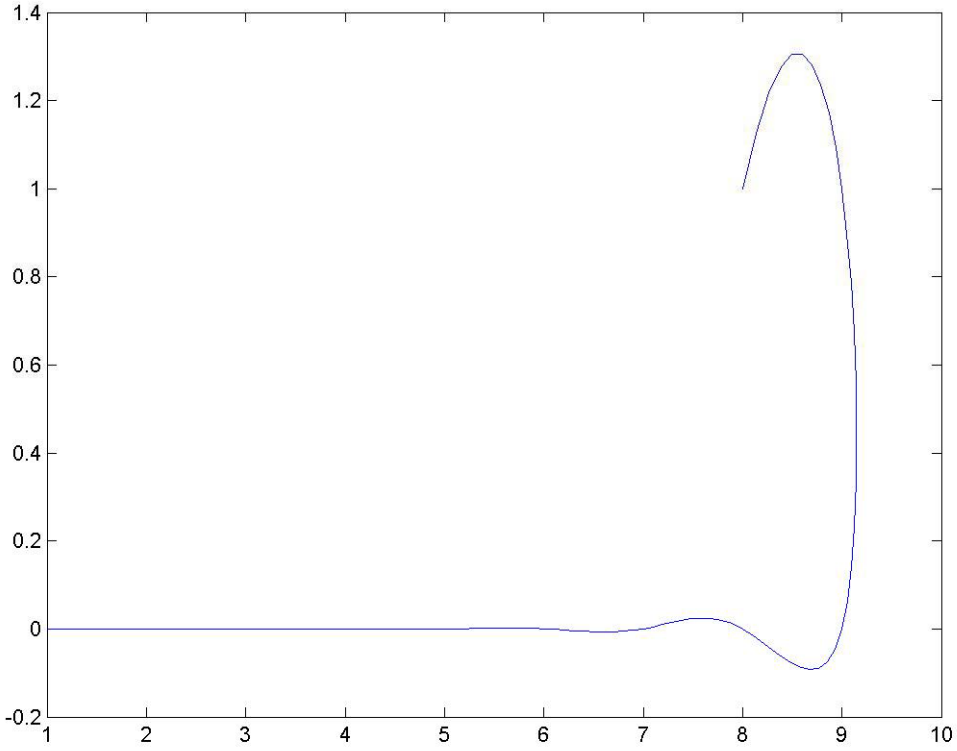
(ii) set the values of the second derivatives at the edges to fixed numbers $a = q''(x_1)$ and $b = q''(x_n)$. If both $a$ and $b$ are set to zero, the fit is called "the natural spline".

(iii) Ensuring third derivative continuity at $x_2$ and $x_{n-1}$

In Matlab all the machinery of determining spline coefficients is already built in. The same beta turn can be fitted to a cubic spline (with default parameters from Matlab) and plotted in 4 lines:

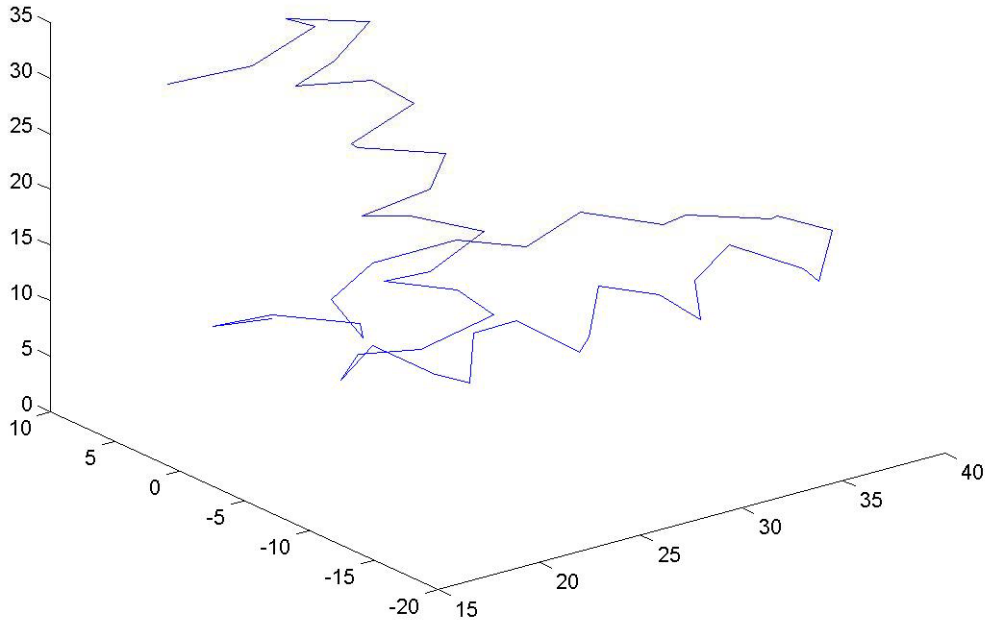
```
>> tt=1:0.1:11;
>> xx=spline(t,x,tt);
>> yy=spline(t,y,tt);
>> plot(xx,yy)
```

The result is not truly optimal but clearly better than the polynomial fit.




Let us work on example for a “real” protein plot.
We start by reading in the PDB entry 1par using the function pickCA(‘filename’) (check the course web page for source code)

```
coor = pickCA(‘1par.pdb’);
x = coor(:,1);
y = coor(:,2);
z = coor(:,3);
% straight forward plot, just connect CA
plot3(x,y,z)
```


35
30
25
20
15
10
5
0
10
5
0
-5
-10
-15
-20
15
20
25
30
35
40

```
% here start the fancy staff
t=1:52;
tt=1:0.1:52;
xx=spline(t,x,tt);
yy=spline(t,y,tt);
zz=spline(t,z,tt);
plot3(xx,yy,zz)
```