Numerical analysis: Assignment 5

Niccolo Zuppichini

October 14, 2020

Exercise 1

Given a Bernstein polynomial $B_i^n = \binom{n}{i}(1-s)^{n-i}s^i$, then it's derivative is given by:

$$(B_i^n)'(s) = D(\binom{n}{i}(1-s)^{n-i}s^i) =$$

Because $\binom{n}{i}$ is a constant we can take it out:

$$= \binom{n}{i} D\Big((1-s)^{n-i}s^i)\Big) =$$

By taking the derivative using the chain rule, it follows:

$$= \binom{n}{i}\Big(is^{i-1}(1-s)^{n-i} + s^i(n-i)(1-s)^{n-i-1}(-1)\Big) =$$

By rearranging:

$$= \binom{n}{i}\Big(is^{i-1}(1-s)^{n-i} - s^i(n-i)(1-s)^{n-i-1}\Big) =$$

By definition of the newton binomial:

$$= \frac{n!}{i!(n-i)!}\Big(is^{i-1}(1-s)^{n-i} - s^i(n-i)(1-s)^{n-i-1}\Big) =$$

It follows:

$$= \Big(\frac{n!}{i!(n-i)!}is^{i-1}(1-s)^{n-i} - \frac{n!}{i!(n-i)!}s^i(n-i)(1-s)^{n-i-1}\Big) =$$

$$= \Big(\frac{n!}{(i-1)!(n-i)!}s^{i-1}(1-s)^{n-i} - \frac{n!}{i!(n-i-1)!}s^i(1-s)^{n-i-1}\Big) =$$

$$= \left(n \frac{(n-1)!}{(i-1)!(n-i)!} s^{i-1}(1-s)^{n-i} - n \frac{(n-1)!}{i!(n-i-1)!} s^i (1-s)^{n-i-1} \right) =$$

$$= n \left(\frac{(n-1)!}{(i-1)!(n-i)!} (1-s)^{n-i} s^{i-1} - \frac{(n-1)!}{i!(n-i-1)!} (1-s)^{n-i-1} s^i \right) =$$

Observe that the two factors inside the big brackets are given by B_{i-1}^{n-1} and B_i^{n-1}, therefore:

$$n \left(B_{i-1}^{n-1}(s) - B_i^{n-1}(s) \right) = (B_i^n)'(s)$$

Concluding the proof.

Exercise 2

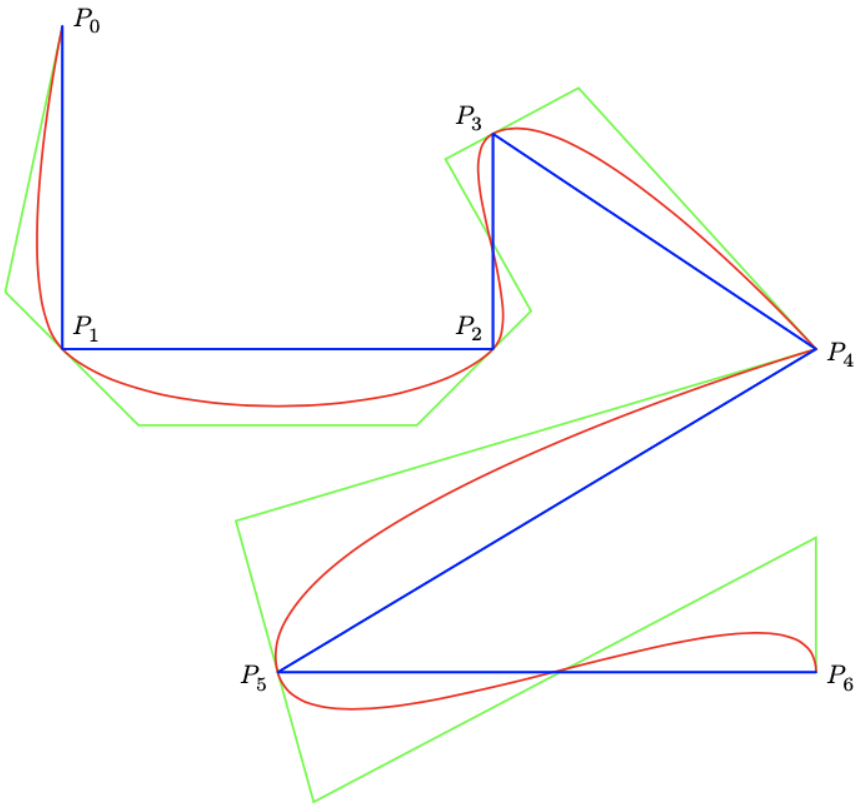


Figure 1: The piecewise spline

In my opinion, this is a combination of two splines, joining in the point P_4: one going through the points $P_0, P_1, P_2, P_3, P_,4$ and a second one through P_4, P_5, P_6. The two splines are composed of many cubic bezier curves satisfying the following conditions for each pair of points P_i, P_{i+1}:

1. The transition around P_i is smooth $\forall\, i \in [0, 6]$

2. The curvature on the boundary points is zero

The first condition is given by ensuring that $(B_i^n(t=0))' = (B_{i-1}^n(t=1))'$ and along with $(B_i^n(t=0))'' = (B_{i-1}^n(t=1))''$ it ensures smoothness around the points P_i.

The first derivative is given by (I took it from wikipedia [1]):

$$B_i' = 3(1-t)^2(P_1 - P_0) + 6(1-t)t(P_2 - P_1) + 3t^2(P_3 - P_2)$$

where P_1 and P_2 are the unknowns (i.e. the green handlers) and P_0 and P_3 are given (blue points). Let's call P_1 and P_2 a, b for clarity. Then:

$$B_i' = 3(1-t)^2(a - P_i) + 6(1-t)t(b - P_{i+1}) + 3t^2(P_{i+1} - b)$$

By employing the first order condition:

$$\begin{aligned} (B_{i-1}(t=1))' &= (B_i(t=0))' \\ \implies a_i + b_{i-1} &= 2P_i \end{aligned} \tag{1}$$

The second derivative (took it from wikipedia as well) is given by:

$$B_i'' = 6(1-t)(b - 2a + P_i) + 6t(P_{i+1} - 2b + a)$$

By employing the second order condition:

$$\begin{aligned} (B_{i-1}(t=1))'' &= (B_i(t=0))'' \\ \implies a_{i-1} + 2a_i &= 2b_{i-1} + b_i \end{aligned} \tag{2}$$

Then for the boundaries:

$$(B_0(t=0)'' = 0$$

$$\implies 2a_0 - b_0 = P_0$$

$$(B_{n-1}(t=1)'' = 0)$$

$$\implies a_{n-1} - 2b_{n-1} + P_n = 0$$

We end up with a system of equations:

$$\begin{cases} a_i + b_{i-1} = 2P_i \\ a_{i-1} + 2a_i = 2b_{i-1} + b_i \\ 2a_0 - b_0 = P_0 \\ a_{n-1} - 2b_{n-1} + P_n = 0 \end{cases} \quad \text{for } i \in [1, n-1]$$

[1] `https://en.wikipedia.org/wiki/Bezier_curve#Cubic_Bezier_curves`

(3)

The goal now is to express the system as a matrix-vector product. Therefore we must explicit one of unknowns (a or b) in function of the other. I choose to express b in function of a because it looked easier to do (note how now the index i goes from 0 to n-2 rather than 1 to n-1):

$$b_i = 2P_{i+1} - a_{i+1} \forall i \in [0, n-2]$$

Then:

$$\begin{cases} b_i = 2P_{i+1} - a_{i+1} \\ a_{i-1} + 2a_i = 2(2P_i - a_i) + 2P_{i+1} - a_{i+1} \\ 2a_0 - (2P_1 - a_1) = P_0 \\ a_{n-1} - 2b_{n-1} + P_n = 0 \end{cases} = \begin{cases} b_i = 2P_{i+1} - a_{i+1} \\ a_{i-1} + 4a_i + a_{i+1} = 4P_i + 2P_{i+1} \\ 2a_0 + a_1 = P_0 + 2P_1 \\ a_{n-1} - 2b_{n-1} + P_n = 0 \end{cases} \quad i \in [1, n-1]$$

We can't insert b_i into the fourth equation because it's not defined for $n-1$, but we can use the second equation from the system of equations and re-arrange it for b_{n-1}:

$$b_{n-1} = -2b_{n-2} + a_{n-2} + 2a_{n-1}$$

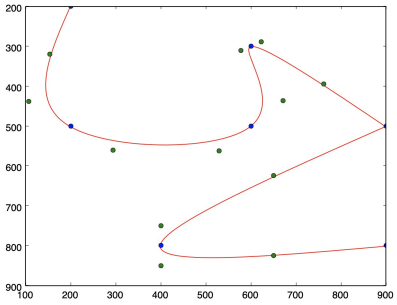
Then:

$$a_{n-1} - 2(-2b_{n-2} + a_{n-2} + 2a_{n-1}) + P_n = 4b_{n-2} - 2a_{n-2} - 3a_{n-1} + P_n =$$
$$= 4(2P_{n-1} - a_{n-1}) - 2a_{n-2} - 3a_{n-1} + P_n = 8P_{n-1} - 7a_{n-1} - 2a_{n-2} + P_n = 0$$

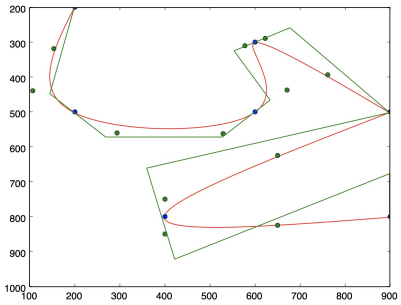
We can finally write the final system of equation:

$$\begin{cases} b_i = 2P_{i+1} - a_{i+1} \\ a_{i-1} + 4a_i + a_{i+1} = 4P_i + 2P_{i+1} \\ 2a_0 + a_1 = P_0 + 2P_1 \\ 7a_{n-1} + 2a_{n-2} = P_n + 8P_{n-1} \end{cases} \quad i \in [1, n-1] \tag{4}$$

Writing this system of equations as a linear system in the form $Ba = P$ is now trivial. B is a tridiagonal matrix that employs the relations of the variable, a is a vector containing all the unknowns (i.e. $a = [a_0 ... a_{n-1}]$ and P another vector containing the data points. I have implemented the system in python and solved it, however, I did not get the same results as adobe illustrator.



(a) The red curve is my spline, the blue points the data points and the green points the handlers A, B I've computed



(b) Same plot but shows also the same green convex hull given in the assignment

Figure 2: Result from my python implementation.

Figure 2b clearly shows that my piecewise cubic bezier doesn't agree with the green convex hull (i.e. a bezier should always be inside its convex hull) therefore it's clear that my implementation differs from the one adobe illustrator uses, however, the idea behind it (i.e. satisfy first and second-order conditions) probably agrees with mine. In conclusion, I think that adobe illustrator implementation differs on the way they treat the boundary points (i.e. P_0, P_4 and P_6) but I'm unsure about how they do it.

The python code can be found in the folder *code*.