

Bezier Spline Sigmac Project Report

1

Given points $P_0 = (0, 0)$ $P_1 = (1, 1)$ $P_3 = (3, 0)$, investigate graphically the range of motion of the Bezier cubic possible depending on the choice of control point P_2 .

With the control points indicated, the starting and ending point of the curve are guaranteed to be $(0, 0)$ and $(3, 0)$, and from P_1 , we can indicate that the derivative is one at the point $(0, 0)$.

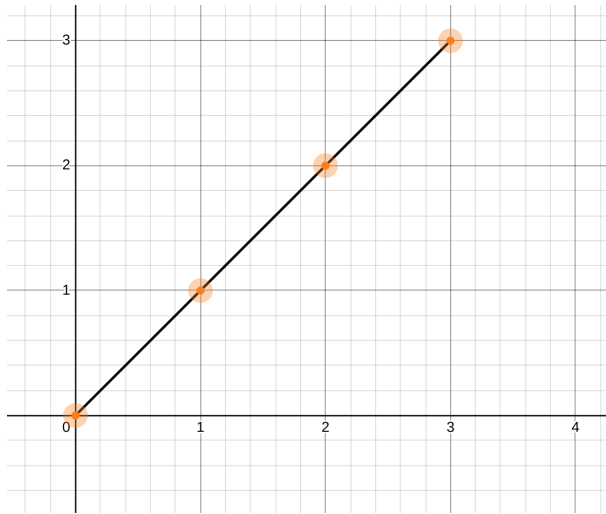
Graphically, this means that the starting and ending point will be the same regardless of P_3 , and the curve will always start by shooting up a little bit no matter what, from there though, almost anything can happen, as having control of the derivative at P_3 lets you make the function shoot as far up or down as you wish after the initial hump, meaning an almost limitless amount of cubics are made possible, shooting as high or as low as you wish, however always coming back to hit that $(3, 0)$ point.



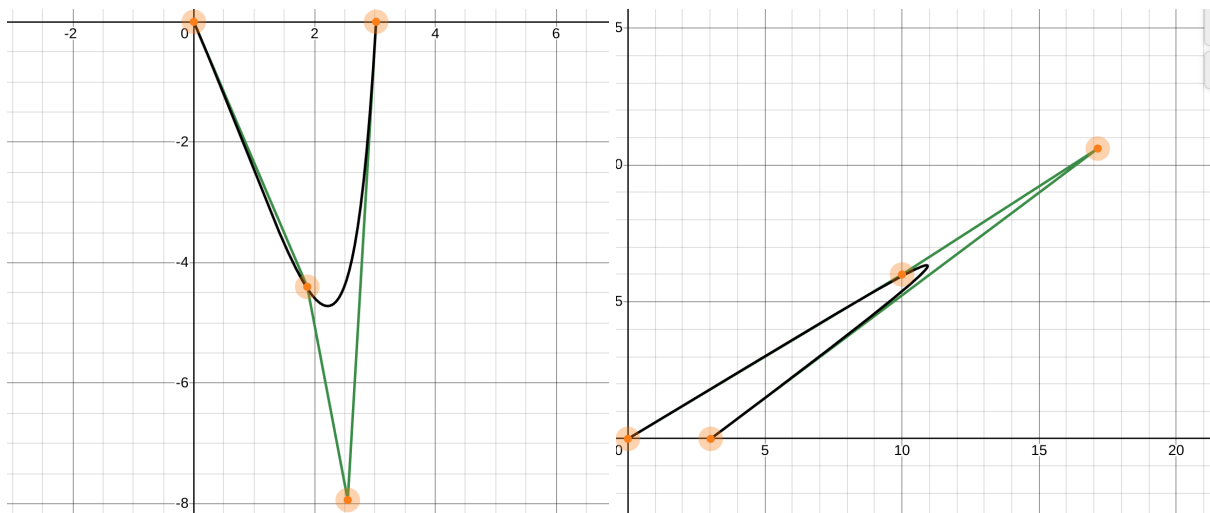
2

Under what conditions will a Bezier cubic actually pass through one or both of the control points P_1 and P_2 ?

In order for the cubic to pass all the control points, they have to be lined up in a line as shown below, this makes sense, as by doing so, you can match the derivatives and the points to the curve.

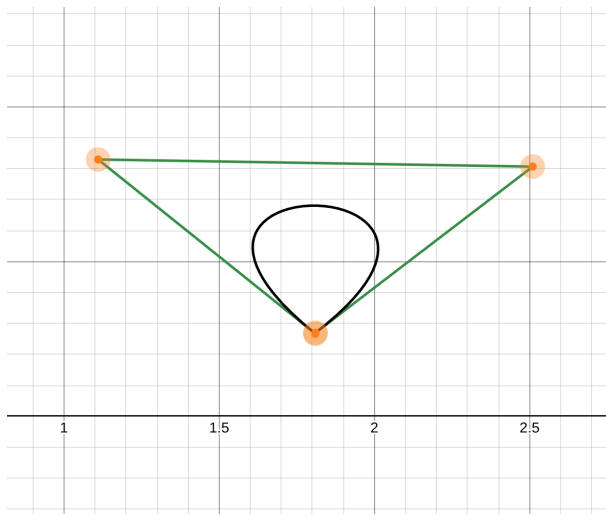


Assuming a fixed P_0 and P_3 , there seems to be an infinite amount of ways to make the polynomial pass through control point P_1 by simply adjusting P_2 , same vice versa, this also makes a lot of sense, as even when freezing three points, you can make the polynomial cross through any point you desire by manipulating the 4th, in which case you can make it just happen to pass through P_2 or P_3 .



3

the only way to create a closed curve was to set the two points right on top of each other From there, I used the control points in order to smooth it out "as best as I can", as it is quickly apparent that with the uneven distribution of points on the circle, it will be lopsided, looking more like a teardrop than a circle, the most that can be done to minimize this is making P_1 and P_2 form a triangle with P_0 and P_3 , and adjust the high until it looks as much like a circle as it can. Making a "nice" circle (Closed, curvy, and could maybe fool a toddler)



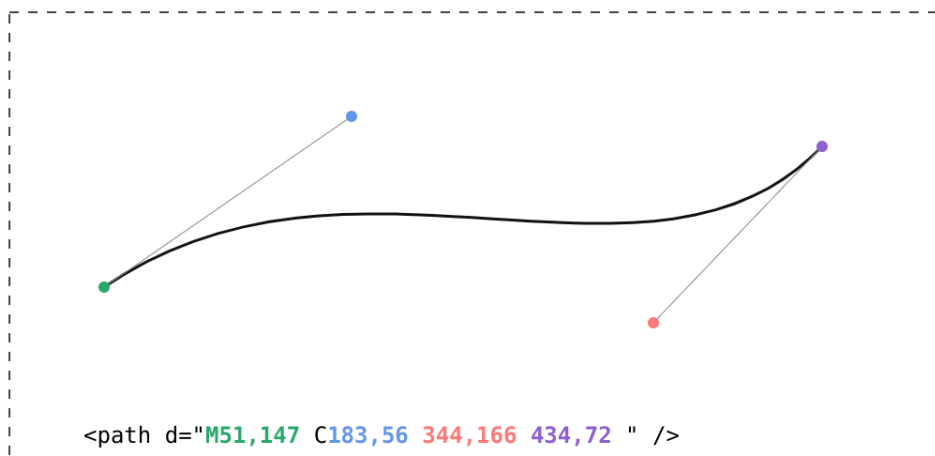
4

For this fourth one, I decided to get a bit creative on the project to see if i could find a way to make my own signature usin current technology!

To do so I decided to experiment with the idea of svg, a extremely efficient image type that takes advantage of these awesome bezier curves.

Essentially an svg is just a collection of control points, that all link up to make lines, which all together group up to make an entire drawing, allowing you to store entire images as jsut mathematical equations!

Here is a demo shown below:



This image is actually an svg! But its also a bezier curve, and as we see here, the code for the svg actually comprises simply of the 4 points that make up the control points!

Here is where things become awesome, I can actually take advantage of any svg painter to draw out a signature, and then extract the control points from the svg files to reverse engineer the bezier curves!

<https://medium.com/@bragg/cubic-bezier-curves-with-svg-paths-a326bb09616f>

This article is really cool in explaining how it all works, and how you can even modify the aspects of the curve with tags written in the svg script!

To grab the svg and control points, I simply draw out my signature as follows,



After conversion into svg, and extractions of the ponits, we are left with:

```
d="M 450 150 Q 435 240, 450 240 Q 570 180, 450 150 Q 480 240, 510 240 Q 510 180, 540 195 Q 570 240, 510 240 Q 525 180, 540 195 Q 570 255, 570 240 Q 555 210, 570 180 Q 615 195, 600 240 Q 600 180, 600 180 M 600 180 Q 645 180, 645 240 Q 660 180, 660 180 Q 660 240, 675 240 Q 690 195, 690 180 Q 690 285, 675 300 Q 675 210, 705 210 Q 720 210, 720 240 Q 720 210, 735 165 Q 750 210, 750 240 Q 750 195, 765 165 Q 780 225, 780 240 Q 780 210, 810 195 Q 780 210, 780 225 Q 795 240, 810 240 Q 810 210, 825 210 Q 855 225, 840 240 Q 780 240, 825 210 Q 855 225, 840 240 Q 840 330, 825 330 Q 795 285, 810 270 Q 870 225, 870 225 Q 855 225, 855 240 Q 900 240, 885 210 Q 855 210, 855 225 Q 900 255, 900 255 Q 930 240, 915 225 Q 900 210, 900 225 Q 870 240, 900 255 Q 945 225, 960 240 Q 960 240, 960 240"
```

Quite a lot! but it makes a lot of sence, in order to almost perfectly reproduce he picture above, we need a ton of points and derivatives.

What also makes svg awesome is that it is both a way to display and write signatures! If you take this code and replace the points bit with whatever points yuo would like, it will draw the curve associated with your control points.

```
<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" width="960" height="360">
  <path d="PLACE CONTROL POINTS HERE" stroke="rgba(41, 173, 255, 1)"
fill="none" stroke-width="2" stroke-linejoin="round"/>
</svg>
```

I had a lot of fun with this project, as there was both the theoretical and applied side to making everything work. In the theory department, I did a lot of research, starting with the splines chapter that really interested me, I put a lot of time into also playing with desmos to grasp the full ability and limitation of bezier curves (Which when combined with each other have almost no limit to what they can draw, to the extent that we even made an image type that uses them to display pictures). After that I branched into many different directions until I found an article explaining how svg's where made using

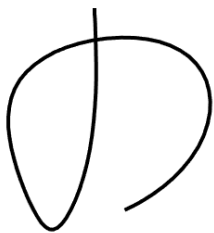
these same curves! With this idea in mind I proceeded to make everyones signatures by reverse engineering the points from the svg file.

The easiest part was actually drawing out my signature and converting it into an svg file. The hard part was discovering the SVG method, I was trying my best to use a desmos like system to manually enter all the equations and create a signature, but found that method to be far too cumbersome, and after wasting a lot of time on it, moved on to researching varius ways to create the signature, when i landed on the svg method I ended up using.

The super duper hardest parth owever was most definitely controlling how the svg was made, when I said "any editor" worked, that was a bit of an exaggeration, as many of them just wanted to give a list of lines or points! In the end, it took me dozens of tries but I found one that got me most of the way there (<https://virtual-graph-paper.com/>), and via some modification I turned the last few stubborn lines into bezier curves, making it one beautiful line of control points!

SECRET PART 6!

But wait, what about AI! This part really got me thinking, what if I could jsut take any drawing, convert it to svg with an image to svg converter, which utalizes AI to simplify an image to its key components, doing this with an online screenshot I got this result:



I then plugged it into a converter to get this svg:

```
d="M 501.32 169.95 L 501.32 170.29 L 501.32 170.86 L 501.49 171.65 L 501.73 172.54 L 501.89 173.47 L 502.10 174.73 L 502.43 176.69 L 502.76 178.67 L 503.29 182.09 L 503.90 185.96 L 504.58 190.44 L 505.26 195.29 L 505.95 200.61 L 506.63 206.30 L 507.31 212.29 L 507.84 217.98 L 508.30 223.75 L 508.68 229.37 L 508.98 234.91 L 509.21 240.14 L 509.29 245.08 L 509.21 249.56 L 509.06 253.65 L 508.68 257.14 L 508.22 260.03 L 507.69 262.46.... (it continues for about 8 pages)"
```

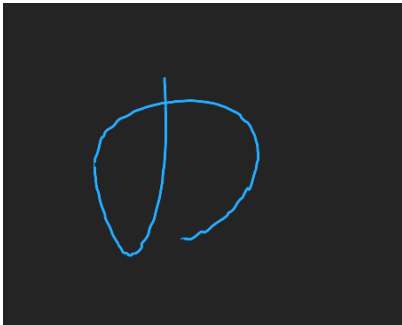
But wait, these are not bezier curves! they are just lines, what a scam! This is when I thought up of generative AI, and actually went to chatgpt, asking it to convert these lines into curves, just to see what it did.

```
turn this into a path with bezier cubics

ChatGPT
Sure, here's your SVG path data converted to use cubic Bezier curves:

<path stroke="green" stroke-width="1px" d="M 602.00 199.00
C 602.00 199.17 602.00 199.33 602.00 199.50
C 602.00 199.67 602.00 199.83 602.00 200.00
C 602.00 200.33 602.00 200.67 602.00 201.00
M 602.00 201.67 602.00 202.33 602.00 202.67
C 602.05 203.48 602.12 204.74 602.17 204.00
C 602.22 203.12 602.25 204.47 602.29 203.88
C 602.33 207.35 602.35 208.67 602.38 207.88
C 602.41 207.47 602.42 208.12 602.44 210.22
M 602.50 212.39 602.52 218.97 602.58 214.22
C 602.61 215.14 602.64 218.97 602.67 219.00
C 602.72 219.62 602.75 223.70 602.78 224.67
C 602.82 225.74 602.86 226.32 602.89 230.78
C 602.92 232.24 602.92 236.61 602.89 237.33
M 602.89 238.00 602.77 242.20 602.67 244.44
C 602.60 245.87 602.41 250.12 602.22 251.89
C 602.08 253.23 601.92 258.70 601.56 259.78
C 601.20 260.86 600.72 265.11 600.67 267.56
C 599.81 269.62 599.70 272.32 599.44 272.44
```

And to my almost null expectations, it actually worked! It still took like four pages, but it was actually able to recreate my image using bezier curves, resulting in this slightly jagged but still recognizable signature:



Now for the unfortunate truth, as you may have noticed, I only actually did a D, and not my entire signature, that's because even just one letter had almost 8 pages worth of lines to make it very high resolution! When plugging any more into chatgpt it actually is simply too much for it to process, and even after dozens of regenerations on the AI, it still was not able to convert anything more than 1 letter at a time to bezier curves :(.