AP5

Murmani Akhaladze
TarnNished

1. Introduction

For this AP I decided to work with three Greek letters: α, θ and ϕ. I chose them because they have interesting shapes that are a bit more fun to reconstruct than just simple Latin characters or digits. The main idea of the task was to take some points that roughly outline each letter, and then try to rebuild the full smooth curve using parametric splines.

The assignment also asks to compare different spline types and see how much the shape changes depending on how many points (nodes) we use or how much smoothing we apply. So basically, this AP is half programming and half a small experiment on how splines behave on real shapes.

2. Model & Approach

The model behind the whole thing is pretty straightforward. Each letter is represented by a bunch of sample points (x_i, y_i) that follow its outline. To turn these points into a parametric curve, I compute a parameter t_i for every point based on the cumulative distance along the shape. This makes t go from 0 to 1 and increase more in places where the letter curves a lot.

After that, I fit two kinds of splines:

- **Natural cubic spline**: very smooth, goes exactly through all the nodes, but can wiggle if data is uneven.
- **Cubic B-spline**: usually smoother and more stable; if smoothing parameter $s > 0$, it doesn't have to pass through all points.

All code was written in Python using NumPy, SciPy and Matplotlib. I wrote helper functions for parametrization, natural cubic fitting, B-spline fitting, error measurement, and plotting. This made it easier to experiment quickly with different parameters.

3. Experiments

I performed three small experiments, because the task required us to check how spline choice and nodes actually affect the fitting quality.

1. Basic reconstruction

First I plotted the original points together with both reconstructed curves. Natural cubic splines always interpolated the nodes perfectly, but sometimes they created tiny overshoots near sharp edges (especially for θ). B-splines (with $s = 0$) were smoother but still captured the general shape very well.

2. Changing number of nodes

Then I tried keeping every 2nd and every 3rd point to see how the quality drops. With fewer points:

- natural spline started bending in weird ways (it tries too hard to interpolate),
- B-spline behaved better and preserved the shape more gracefully.

So basically, node placement matters, and B-splines handle low-quality or sparse data better.

3. Changing smoothing parameter

Finally, I tested B-spline smoothing $s = 0, 0.1, 0.5$. With small smoothing $s = 0.1$, the curve becomes a bit nicer and removes small noise. With larger smoothing $s = 0.5$, the letter becomes too "soft" and loses some details (like the inside bar of θ). So smoothing is good, but not too much.

4. Conclusions

Overall, the project was actually more interesting than it looked at first. Even with simple letters like these, natural splines and B-splines behave pretty differently. Natural cubic spline is great when you want exact interpolation, but it can be unstable if the nodes are not chosen nicely. B-splines are more robust and give smoother shapes, especially when the original points are uneven.

From the experiments:

- B-splines usually give a nicer and more stable reconstruction.
- Using fewer nodes makes natural splines worse much faster.
- Smoothing helps, but too much smoothing basically destroys the letter.

So, if I had to choose a method for general curve drawing or letter reconstruction, I would definitely go with cubic B-splines. They feel more controllable and less "dramatic" than natural cubic splines.