Cubic Bezier Fitting with Least Squares

Jialin Lu luxxxlucy@gmail.com 2025-04-30

code is available at luxxxlucy/bezier-rs

*Note that this is the first blog in the **Bezielogue** series that I plan to write about curves. More fun to come!*

TL;DR We introduce the basic task of fitting a cubic Bezier curve to an ordered list of points. We explore different approaches as follows:

1. Solving a linear least square as introduced in "Curve Fitting" and "Least Squares Bezier Fit".
2. The vanilla method only optimizes the control points while fixing the parameter values. To improve, we can optimize both by an alternating two-stage method, iteratively improving control points and parameter values. This converges to the global minimum, but slowly.
3. We further utilize the relationship between control points and parameter values by variable projection. This eliminates the alternating steps and converts the problem into a single optimization over parameter values.
4. Lastly an ad-hoc improvisation is introduced. It is simpler and get faster convergence.

Curve Fitting, Chapter 35 of The Bezier Primer

Least Squares Bezier Fit, Jim Herold 2012

Bezier Curve Fitting, Tim A Pastva 1998

Introduction

In most software applications, cubic Bezier segments are the canonical form: other types are often converted to cubic segments during preprocessing.

Given a list of points in 2D space, we want to fit a cubic Bezier curve to them. A cubic Bezier curve is defined by four control points $P = \{p_0, p_1, p_2, p_3\}$:

$$B_P(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t) t^2 p_2 + t^3 p_3 \quad (1)$$

The coefficients form the Bernstein basis functions, following Pascal's triangle:

$$(1)$$
$$(1, 1)$$
$$(1, 2, 1)$$
$$(1, 3, 3, 1)$$
$$(1, 4, 6, 4, 1)$$
$$\dots$$

As shown in the figure below, a cubic Bezier curve consists of four control points:

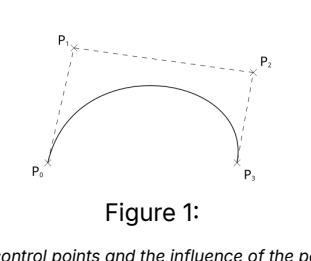

Figure 1:

A cubic Bezier curve with its control points and the influence of the parameter t (source: wikipedia)

where P_0 and P_3 are the on-curve endpoints, while P_1 and P_2 are the off-curve control points that influence the curve's shape. The parameter $t \in [0, 1]$ is the domain range that kind of represents flow of time that traces the curve, with $t = 0$ at the start and $t = 1$ at the end.

The Least Square Method

Given a list of points $D = \{d_1, d_2, ..., d_n\}$ in a 2-dimensional space, the cubic bezier curve which fits these points is the one that minimizes the following least square error:

$$\min_{p_0,p_1,p_2,p_3} \sum_{i=1}^{n} \left\| B_{p_0,p_1,p_2,p_3}(t_i) - d_i \right\|^2 \quad (2)$$

where t_i is the parameter value for the i-th data point.

Now look at the Equation 2, we found that even though p_0, p_1, p_2, p_3 are the control points we are interested in, t_0, t_1, t_2, t_3 are introduced as auxlilary variables that we need to estimate as well.

To simplify the formulation we denote $P = \{p_0, p_1, p_2, p_3\}$ and $T = \{t_0, t_1, t_2, t_3\}$ and $D = \{d_1, d_2, ..., d_n\}$ and rewrite the error function as:

$$\min_{P,T} \sum_{i=1}^{n} \left\| B_{P(t_i)} - d_i \right\|^2 \quad (3)$$

Level 1: least square, simply

Least Squares Bezier Fit Jim Herold, 2012

The Bezier Primer Chapter 35 Curve Fitting

A simple solution starts by estimating the t-values first, then solving the least squares problem for P. This works well because Equation 3 is linear with respect to P.

The algorithm follows two simple steps:

1. Estimate T using a heuristic
2. Solve for P given T

For the first step, we can use several heuristics. Let's start with the chord length heuristic, which assigns t-values to points $d_0, d_1, ..., d_n$ based on their relative positions along the polyline connecting all data points. Specifically, for each data point, we assign a parameter t proportional to its distance along this polyline.

Once we have T, we can solve for P that minimizes Equation 3. Let's expand this using the Bezier curve equation from Equation 1:

$$\min_{P,T} \sum_{i=1}^{n} \left\| \left((1-t_i)^3 p_0 + 3(1-t_i)^2 t_i p_1 + 3(1-t_i) t_i^2 p_2 + t_i^3 p_3 \right) - d_i \right\|^2$$

We can express this more compactly using matrix notation. First, note that:

$$B_{P(t)} = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t) t^2 p_2 + t^3 p_3$$
$$= \begin{pmatrix} 1 \\ t \\ t^2 \\ t^3 \end{pmatrix}^T \begin{pmatrix} 1 & & & \\ -3 & 3 & & \\ 3 & 6 & 3 & \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

We can now rewrite Equation 3 as:

$$\min_{P,T} \sum_{i=1}^{n} \| B(t_i)p - d_i \|^2 = \min_{P,T} \| \mathcal{T}_T B P - D \|_F \quad (4)$$

where $\|.\|_F$ is the Frobenius norm, $\mathcal{T}_T$ is the matrix of t-values raised to powers, B is the Bernstein matrix, P contains our control points, and D contains our data points.

$$\mathcal{T}_T = \begin{pmatrix} 1 & t_0 & t_0^2 & t_0^3 \\ 1 & t_1 & t_1^2 & t_1^3 \\ \dots & \dots & \dots & \dots \\ 1 & t_n & t_n^2 & t_n^3 \end{pmatrix}$$

The polynomials $\mathcal{T}_T$ are trivially derived from T.

$$B = \begin{pmatrix} 1 & & & \\ -3 & 3 & & \\ 3 & 6 & 3 & \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

Also known as the Bernstein matrix

$$P = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{pmatrix}$$

P represents the control points of the cubic Bezier curve, which are our variables of interest.

$$D = \begin{pmatrix} d_1 \\ d_2 \\ \dots \\ d_n \end{pmatrix}$$

The sampled data points. Note that each d_i is actually a 2D point, but we've simplified the notation here for clarity.

Solving Equation 4 given T reduces to a standard linear least squares problem. Let $L(P, T)$ be the objective in Equation 4:

$$L(P, T) = \| \mathcal{T}_T B P - D \|_F = (\mathcal{T}_T B P - D)^T (\mathcal{T}_T B P - D)$$
$$\frac{\partial (L(P,T))}{\partial (P)} = -2 \mathcal{T}_T^T (D - \mathcal{T}_T B P)$$

Since $L(P, T)$ is linear with respect to P, we can find the minimum by setting its derivative to zero. For simplicity, let's denote $A = \mathcal{T}_T B$. Then we have:

$$P^* = (A^T A)^{-1} A^T D \text{ where } A = \mathcal{T}_T B \quad (5)$$

where $(A^T A)^{-1} A^T$ is the pseudo-inverse of $A = \mathcal{T}_T B$. Computing Equation 5 is efficient and straightforward.

Now let us test this visually. We first generate a ground truth curve and then sample points from it, we then apply the least square method to fit the curve.

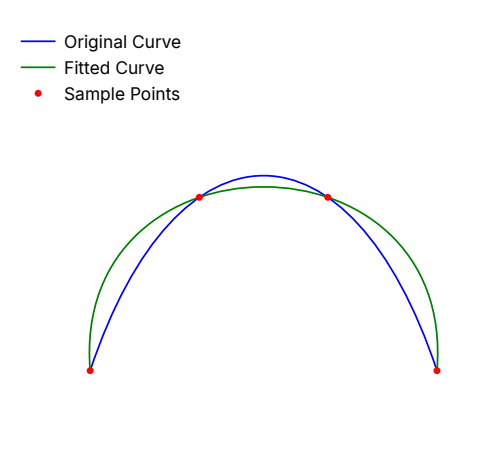

Figure 2:

Comparison of Original and Fitted Curves

Unfortunately the results seem off, suggesting that curve fitting does not recover the original curve, not even close.

Notice how the fitted curve passes through all sample points but has a significantly different shape from the original curve.

In fact, the fitting process does not guarantee that the fitted curve will pass through all sample points. This is just a coincidentally good case where the points were sampled at uniform t values.

We further run the algorithm with different heuristics for t-values:

1. Uniform: evenly distribute t values from 0 to 1, without considering the actual point values
2. Chord length: the default method, which approximates arc length
3. Centripetal: square root of chord length, another good approximation

The results are shown below:

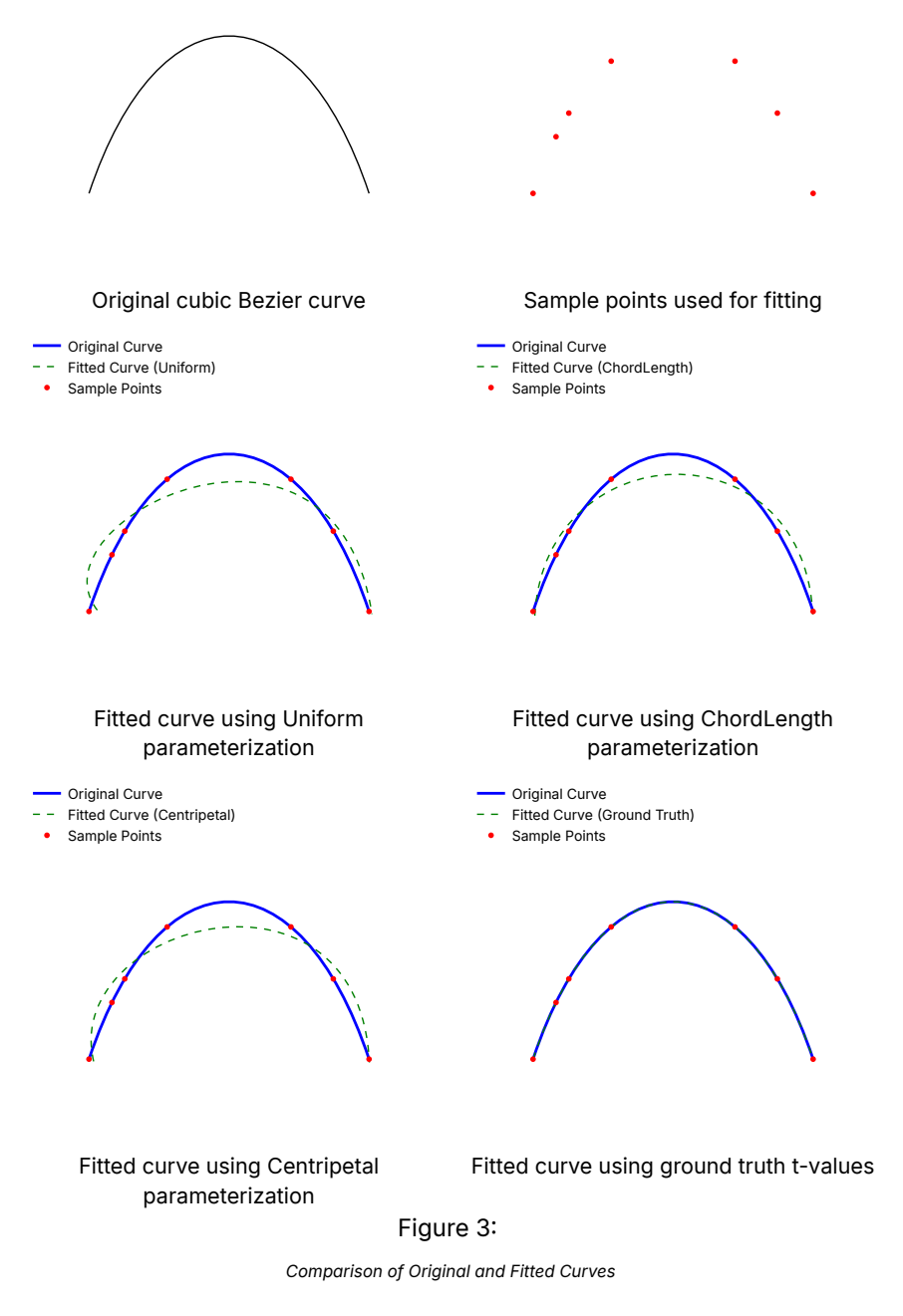

Figure 3:

Comparison of Original and Fitted Curves

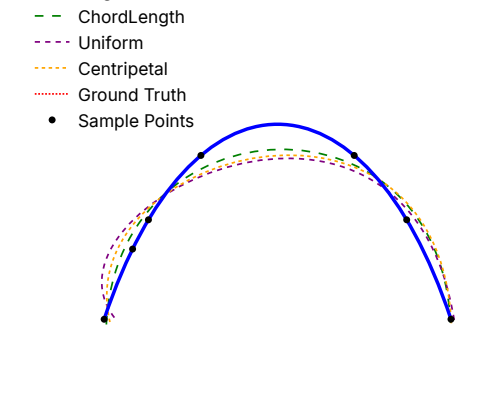

Figure 4:

Combined visualization of all methods

If we had the ground truth t-values, we for sure could recover the original curve perfectly. But heuristics would always be non-perfect. This then becomes too rigid - we need more flexibility. We should be able to adjust the t-values as well instead of relying on heuristics once and for all.

Level 2: An Alternating Method

Bezier Curve Fitting, Tim A Pastva 1998

Now we want to move a step forward. Let us also optimize T as well. We here introduce an alternating method based on Pastva Tim's thesis.

The alternating method is an iterative method that alternates between estimating T and P:

1. Get initial T
2. Find P given T
3. Update T given P
4. Repeat steps 2 and 3 until convergence

We will first use the chord length heuristic as default to get the initial T and then use that to fit P. Step 2 is essentially Equation 5, so I will not repeat here.

Step 3 updates T based on P. Let us start with something simple. Update T given P can be done simply by updating T by the nearest point on the curve.

For each $d_i \in D$, find t that minimizes $\left\| B_{P(t)} - d_i \right\|_2$. Although we can solve this by solving a (quartic) equation, for simplicity and robustness we can just opt for a binary search.

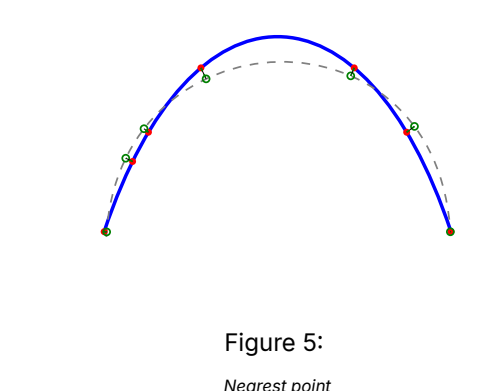

Figure 5:

Nearest point

We can see it converges (the result from 0 to 10 iteration steps is plotted below). But we can also see that the convergence is just too slow.

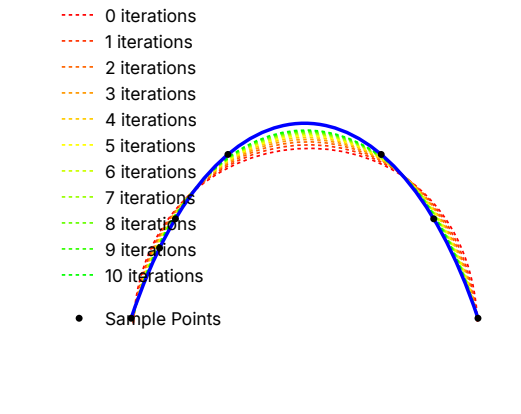

Figure 6:

Convergence of the alternating method

Pastva considers this nearest point method as the first variant. It also talks about another variant that uses Gauss-Newton method to update T directly.

Notice that Equation 4 is in fact a non-linear optimization problem w.r.t T. Gauss-Newton tries to solve this by approximating a good `gradient` for T and updating it iteratively. Specifically:

We first define the loss function aka residual vector as:

$$R = \begin{pmatrix} B_{x(t_1)} - x_1 \\ B_{y(t_1)} - y_1 \\ B_{x(t_2)} - x_2 \\ B_{y(t_2)} - y_2 \\ \vdots \\ B_{x(t_n)} - x_n \\ B_{y(t_n)} - y_n \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ \dots \\ r_n \end{pmatrix} \in \mathbb{R}^2 n$$

where each $r_i = \begin{pmatrix} B_{x(t_i)} - x_i \\ B_{y(t_i)} - y_i \end{pmatrix}$.

We then have the Jacobian matrix of partial derivatives:

$$J = \frac{\partial (R)}{\partial (t)} = \begin{pmatrix} \frac{\partial (B_{x(t_1)})}{\partial (t_1)} & 0 & \dots & 0 \\ \frac{\partial (B_{y(t_1)})}{\partial (t_1)} & 0 & \dots & 0 \\ 0 & \frac{\partial (B_{x(t_2)})}{\partial (t_2)} & \dots & 0 \\ 0 & \frac{\partial (B_{y(t_2)})}{\partial (t_2)} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \frac{\partial (B_{x(t_n)})}{\partial (t_n)} \\ 0 & 0 & \dots & \frac{\partial (B_{y(t_n)})}{\partial (t_n)} \end{pmatrix} \in \mathbb{R}^{2n \times n}$$

note that

$$\begin{pmatrix} \frac{\partial (B_{x(t)})}{\partial (t)} \\ \frac{\partial (B_{y(t)})}{\partial (t)} \end{pmatrix} = \frac{\partial (B(t))}{\partial (t)} = [0, 1, 2t, 3t^2] B P = [0, 1, 2t, 3t^2] B \begin{pmatrix} P_x \\ P_y \end{pmatrix}$$

The Jacobian matrix is in fact block diagonal if we consider a $\begin{pmatrix} \frac{\partial (B_{x(t_i)})}{\partial (t_i)} \\ \frac{\partial (B_{y(t_i)})}{\partial (t_i)} \end{pmatrix}$ for each t_i as a unit.

Given the residual vector and the jacobian, Gauss-Newton method tries to find the step vector that minimizes the residual by a linear approximation of R as $R + J\Delta t$.

Namely, under the first order Taylor expansion, $R(t) \approx R + J\Delta t$

$$\min_t \|R\|_F \cong \min_{\Delta t} \|R + J\Delta t\|_F$$

Finding the best step vector corresponds to solving the derivative to 0,

$$\frac{\partial (\|R + J\Delta t\|_F)}{\partial (\Delta t)} = J^T R + J^T J \Delta t = 0$$

and thus

$$\Delta t = -(J^T J)^{-1} J^T R$$

After we get the step vector Δt, we update each t_i:

$$t_i^{k+1} = t_i^k + \Delta_t = t_i^k - \left[(J^T J)^{-1} J^T R \right]_i$$

Needless to say that we need to ensure it stays in $[0, 1]$ $t_i \leftarrow \text{clamp}(t_i, 0, 1)$

While this approach appears promising, my experimental results show that Gauss-Newton's performance is comparable to the nearest point method. Across different ground truth curves and sampling patterns, the convergence rate improvement is negligible. Given its simplicity and stability, I think the nearest point method is preferred.

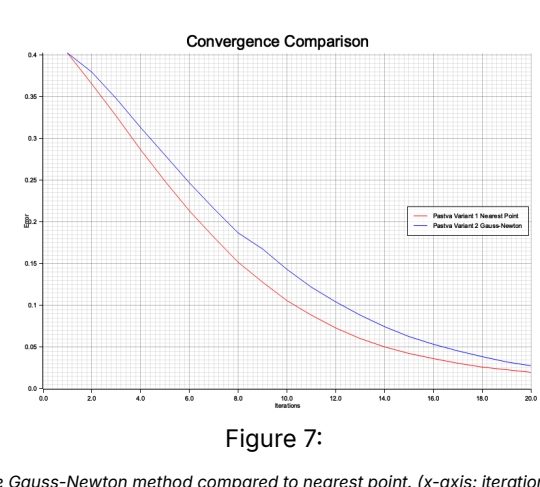

Figure 7:

Convergence of the Gauss-Newton method compared to nearest point. (x-axis: iteration steps, y-axis: error) It seems the improvement is slightly small even negligible. We denote the nearest point as Pastva variant 1 and Gauss-Newton as Pastva variant 2.

Level 3: Variable Projection

The alternating method converges quite slowly. One hypothesis is that this is because each step is really small and thus the improvement of each iteration becomes negligible, more so if it reaches vicinity of the solution.

Maybe a non-alternating method should be better? Now I consider a further advanced method called variable projection.

Variable Projection for Nonlinear Least Squares Problems, Dianne O'Leary and Bert Rust 2007

Total least squares fitting of Bézier and B-spline curves to ordered data, Borges and Pastva 2002

The Variable Projection Method - Nonlinear Least Squares Fitting, 2020 Geo-Ant

The idea of variable projection is that if we look closely enough, we will find that even though we have P and T in the objective function, P is actually determined by T.

Observe that the objective function in Equation 3 is linear to P, so we can just solve for P given T and then plug it into Equation 3. Given:

$$P_T^* = (A^T A)^{-1} A^T D = \left((\mathcal{T}_T B)^T (\mathcal{T}_T B) \right)^{-1} (\mathcal{T}_T B)^T D$$

We insert P^* back into the objective function of Equation 4 and denote $A_T = \mathcal{T}_T B$. The objective is:

$$\begin{aligned} \min_{P,T} & \sum_{i=1}^{n} \|B(t_i)p - d_i\|^2 \\ &= \min_{P,T} \|\mathcal{T}_T B P - D\|_F \\ &= \min_{P,T} \|A_T P - D\|_F \\ &= \min_{T} \|A_T P_T^* - D\|_F \end{aligned} \quad (6)$$

expand $A_T = \mathcal{T}_T B$ would lead to

$$\min_T \left\| \mathcal{T}_T B \left(\left((\mathcal{T}_T B)^T (\mathcal{T}_T B) \right)^{-1} (\mathcal{T}_T B)^T D \right) - D \right\|_F$$

but I think this is just too verbose and not easy to understand. I will keep using the compact notation $A_T = \mathcal{T}_T B$.

T becomes the only variable that we will optimize over. Similarly, let us optimize this non-linear problem over T with Gauss-Newton.

The residual R is defined as $R = A_T P_T^* - D$ and the Jacobian matrix of partial derivatives is:

$$J = \frac{\partial (R)}{\partial (t)} = \begin{pmatrix} \frac{\partial (A_T P_T^*)}{\partial (t_1)} & 0 & \dots & 0 \\ \frac{\partial (A_T P_T^*)}{\partial (t_1)} & 0 & \dots & 0 \\ 0 & \frac{\partial (A_T P_T^*)}{\partial (t_2)} & \dots & 0 \\ 0 & \frac{\partial (A_T P_T^*)}{\partial (t_2)} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \frac{\partial (A_T P_T^*)}{\partial (t_n)} \\ 0 & 0 & \dots & \frac{\partial (A_T P_T^*)}{\partial (t_n)} \end{pmatrix} \in \mathbb{R}^{2n \times n}$$

For each entry in J, we have

$$\frac{\partial (A_T P_T^*)}{\partial (t)}$$
$$= \frac{\partial (A_T P_T^*)}{\partial (t)}$$
$$= \frac{\partial \left(A_T (A_T^T A_T)^{-1} A_T^T \right)}{\partial (t)}$$
$$= \frac{\partial (A_T)}{\partial (t)} (A_T^T A_T)^{-1} A_T^T D + A_T \frac{\partial \left((A_T^T A_T)^{-1} \right)}{\partial (t)} A_T^T D + A_T (A_T^T A_T)^{-1} \frac{\partial (A_T^T)}{\partial (t)} D \quad (7)$$

This all seems good, but after I implemented and made some experiments with it I found that this is numerically very unstable. Especially when we want to get the second term, in particular:

$$\frac{\partial \left((A_T^T A_T)^{-1} \right)}{\partial (t)} = -(A_T^T A_T)^{-1} \frac{\partial (A_T^T A_T)}{\partial (t)} (A_T^T A_T)^{-1}$$
$$= -(A_T^T A_T)^{-1} \left(\frac{\partial (A_T^T)}{\partial (t)} A_T + A_T^T \frac{\partial (A_T)}{\partial (t)} \right) (A_T^T A_T)^{-1}$$

and so the second term of Equation 7 when expanded becomes

$$-A_T (A_T^T A_T)^{-1} \frac{\partial (A_T^T A_T)}{\partial (t)} (A_T^T A_T)^{-1} A_T^T D$$
$$= -\mathcal{T}_T B \left((\mathcal{T}_T B)^T \mathcal{T}_T B \right)^{-1} \left(\partial \frac{B^T \mathcal{T}_T^T}{\partial (t)} \mathcal{T}_T B + B^T \mathcal{T}_T^T \frac{\partial (\mathcal{T}_T B)}{\partial (t)} \right) (\mathcal{T}_T B)^{-1} \mathcal{T}_T B D$$

This unfortunately requires a very careful implementation and my attempts result in a very numerically unstable state.

If we really want to make it work, I would probably need to revise a careful implementation and add a lot of check routines as well as doing line search to determine a step size. This becomes ridiculously complicated and not really practical.

Let me be honest, it is probably because of my poor math skills, I have never been good at it. So either the implementation is not careful enough, or maybe my derivation is not correct.

Level 4: A Weak Variable Projection Method

Okay, so I am stuck. I spent some time and effort on variable projection, hoping that this could do some help, but this seems to be an unfruitful path.

How about now we look in retrospect? We do know:

1. Chord length heuristic is actually a good heuristic, so good that assuming we have the right ordered data sample points, we are already in the good basin.
2. An alternating approach is good actually, but converges super slow.
3. Even in the alternating approach, using the simpler nearest point is equally good, and considering robustness, better than Gauss-Newton.
4. Advanced method like variable projection is just a juggling of unnecessary complexity and we have not been able to reach a good solution.

Okay, so what now? Getting a correct Δt for minimizing Equation 6 is hard, but we do have a good Δt for Equation 4. It surely is suboptimal, but it is good enough. We just need some tweaks to make it work to converge faster.

We can make up for the suboptimal step update with a line search that is based on a zero-order evaluation of Equation 6:

1. First, we obtain the search direction Δt from the Gauss-Newton method on Equation 4:

$$\Delta t = -(J^T J)^{-1} J^T R$$

where R is the residual from Equation 4.

2. We perform a line search along this direction to find an optimal step size α, evaluated using Equation 6 instead of Equation 4:

$$\min_{\alpha} \|A_{T+\alpha\Delta t} P_{T+\alpha\Delta t}^* - D\|_F$$

We determine α using line search (golden section search).

3. Finally, we enhance robustness by sampling randomly around $T + \alpha\Delta t$ and selecting one that minimizes Equation 6.

The key insight is that while computing the true gradient of the total loss function is numerically unstable, we can:

- Use the simpler loss function to obtain a search direction
- Validate steps using the total loss function through line search
- Improve robustness through random sampling

The following plot can be generated by running this command in the repository:

```
cargo run --bin bezier-least-square-fit-convergence-comparison-plot
```

Results:

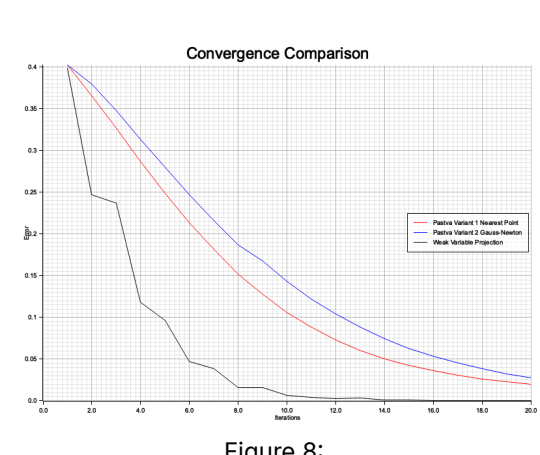

Figure 8:

Convergence of new weak variable projection method compared to other ones. The method shows better convergence than both Pastva variants. (x-axis: iteration steps, y-axis: error)

Of course it would be good, that is more heuristic and so much more steps of compute!

Lessons Learned

Recently I have seen a re-occurring theme happening in different places and it is that sometimes we should prefer simple methods just for the sake of simplicity, and simplicity sometimes correlates with robustness.