Numerical Methods in Comp. Eng. Term Project: Piecewise Interpolation for Robot Path Planning

Muhammed Abdullah Özdemir
Istanbul Technical University
Artificial Intelligence and Data Eng.
ID: 150220340
Email: ozdemirmuh22@itu.edu.tr

***Abstract*—In this project, I explored and compared four widely used interpolation methods—Newton, Lagrange, Cubic Spline, and B-Spline—with a focus on their behavior in robot path planning tasks. My goal was to understand how these methods perform in terms of accuracy, smoothness, local control, and their sensitivity to waypoint density. I used a ready-made experiment interface and tested each method across different waypoint configurations. Based on the results, I observed that Newton and Lagrange methods tend to produce globally unstable paths, especially with dense or irregular data. Cubic Spline achieved a good balance between accuracy and smoothness, while B-Spline stood out for its local control and stable behavior. This comparison helped me better understand the trade-offs between different interpolation strategies in real-world applications.**

I. Introduction

In many real-world applications, from robotics and automation to computer graphics and engineering simulations, interpolating discrete data points into continuous curves is a fundamental task.

This project was done as part of the Numerical Methods in Comp. Eng. course at Istanbul Technical University. My main goal was to understand, implement, and analyze different interpolation techniques in an educational and practical context. By comparing these methods side by side, I aimed to gain a deeper understanding of how they work mathematically, and how they perform in solving problems like path planning.

In this project, I focused on 4 commonly used interpolation techniques: Newton, Lagrange, Cubic Spline, and B-Spline. These methods were chosen because they are widely used in engineering and scientific computations, and each of them has its own strengths. While Newton and Lagrange interpolation try to pass through every given point exactly, spline-based methods like Cubic Spline and B-Spline aim to produce smoother curves and offer better control over specific parts of the path. Comparing these methods side by side made it easier to see how different mathematical approaches can lead to very different behaviors in practical applications.

The motivation behind this project is not just to implement these methods, but to evaluate how they behave under different conditions. Through a series of experiments, we investigate each method's accuracy, visual smoothness, sensitivity to changes (local control), and performance with varying waypoint densities. In doing so, I aim to identify the most suitable interpolation strategy for different real-life scenarios and gain experience in applying numerical methods to engineering problems.

II. Interpolation Methods

A. Newton Polynomial Interpolation

Newton interpolation constructs a polynomial incrementally using divided differences. Each polynomial $p_k(x)$ of degree k is built by adding a term to $p_{k-1}(x)$.

$$p_k(x) = c_0+c_1(x-x_0)+c_2(x-x_0)(x-x_1)+\cdots+c_k(x-x_0)\ldots(x-x_{k-1})$$

$$= p_k(x) = \sum_{i=0}^{k} c_i \prod_{j=0}^{i-1}(x-x_j)$$

This interpolation is particularly useful for incremental computations. When a new point is added, only one new coefficient needs to be calculated, without recalculating previous terms. This makes Newton interpolation efficient for dynamically growing datasets.

B. Lagrange Polynomial Interpolation

Lagrange interpolation expresses the polynomial as a linear combination of cardinal basis polynomials $\ell_i(x)$, where each $\ell_i(x)$ satisfies $\ell_i(x_j) = \delta_{ij}$. The interpolation formula is:

$$p(x) = \sum_{k=0}^{n} y_k \ell_k(x)$$

where

$$\ell_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}, \quad (0 \le i \le n)$$

This method is straightforward but can be inefficient for large datasets, and it is sensitive to Runge's phenomenon due to the use of a single global polynomial.Ideal for applications requiring exact waypoint adherence with small datasets, such as precision positioning systems.

C. Cubic Spline Interpolation

Cubic spline interpolation connects data points with piecewise cubic polynomials $S_i(x)$ defined on intervals $[t_i, t_{i+1}]$. The spline $S(x)$ is defined as:

$$S(x) = \begin{cases} S_0(x), & x \in [t_0, t_1] \\ S_1(x), & x \in [t_1, t_2] \\ \vdots \\ S_{n-1}(x), & x \in [t_{n-1}, t_n] \end{cases}$$

The spline satisfies continuity of the function and its first and second derivatives:

$$\begin{aligned} S_{i-1}(t_i) &= S_i(t_i) = y_i \\ S'_{i-1}(t_i) &= S'_i(t_i) \\ S''_{i-1}(t_i) &= S''_i(t_i) \end{aligned}$$

The cubic spline on $[t_i, t_{i+1}]$ is:

$$\begin{aligned} S_i(x) =& \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 \\ &+ \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x - t_i) \\ &+ \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(t_{i+1} - x) \end{aligned}$$

Here $h_i = t_{i+1} - t_i$ and $z_i = S''(t_i)$.

The cubic spline formula represents a balance between interpolation accuracy and smoothness:

- The first two terms contain cubic functions that ensure second-derivative continuity
- The last two terms ensure that the spline passes through the data points y_i and y_{i+1}
- The parameters z_i (second derivatives at the knots) are determined by solving a tridiagonal system of equations that enforces smooth connections between adjacent cubic pieces
- This approach guarantees continuous first and second derivatives across the entire curve, resulting in visually smooth trajectories

Commonly used in robotics and CAD applications where path smoothness and continuous derivatives are essential for natural motion.

D. B-Spline Interpolation

B-splines are piecewise polynomial functions defined over a knot sequence $\{t_i\}$ and provide local control. The zeroth-degree B-spline basis function $B_i^0(x)$ is defined as:

$$B_i^0(x) = \begin{cases} 1, & t_i \le x < t_{i+1} \\ 0, & \text{otherwise} \end{cases}$$

Higher-degree B-spline basis functions are defined recursively by the Cox-de Boor formula:

$$B_i^k(x) = \frac{x - t_i}{t_{i+k} - t_i} B_i^{k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1}^{k-1}(x)$$

B-spline curves are then formed by a linear combination of basis functions:

$$C(x) = \sum_{i=0}^{n} P_i B_i^k(x)$$

P_i : Control points

1) The Properties of B-Splines:

- **Approximation vs. Interpolation**: Unlike Newton and Lagrange methods, B-splines generally do not pass through their control points. Instead, they form a smooth curve that is influenced by these points, making them an approximation rather than a strict interpolation technique. This property is particularly visible with sparse control points.
- **Local Control**: Modifying a single control point only affects the curve locally over a finite number of intervals, providing better local control compared to global polynomials.
- **Smoothness Guarantee**: B-splines automatically ensure continuity up to a specified derivative order determined by the degree k of the basis functions.
- **Variation Diminishing**: The curve does not oscillate more than its control polygon, avoiding the unwanted oscillations that can occur with high-degree polynomial interpolation (Runge's phenomenon).

Preferred in computer graphics, animation, and high-speed robotics where smooth approximation is more important than exact waypoint following.

III. Experimental Setup

To better understand the behavior of each interpolation method, I ran several experiments using a ready-made interface developed by our instructor. My goal here wasn't to design a testing system from scratch, but to evaluate and analyze how each method performs when subjected to different path-planning conditions.

A. Waypoint Sets

I worked with several predefined waypoint sets in runner file:

- **Simple Curves (3 Points)**
- **Complex Paths (5 Points)**
- **Zigzag Path (7 Points)**
- **Sharp Turns (7 Points)**
- **Circle (8 Points)**

B. Evaluation Metrics

For the evaluation, I focused these:

- **Path Length** – How long the resulting curve is
- **Max Deviation** – The farthest distance between the curve and a given waypoint

- **Mean Deviation** – The average distance across all waypoints
- **Max Curvature** – How sharp the sharpest bend is, to understand smoothness

C. *Experiment Types*

These are the three main types of experiments I explored:

1) **Accuracy vs. Smoothness**
2) **Local Control**
3) **Waypoint Density Analysis**

IV. Results and Discussion

A. *Accuracy vs. Smoothness*

I analyzed how each method balances accuracy and smoothness on two test paths: a simple curve with just 3 points and a circle with 8 points.

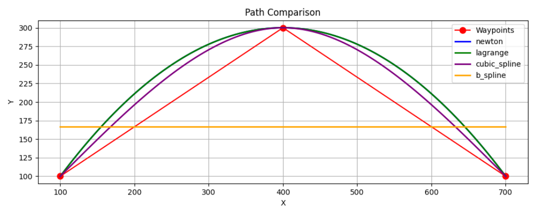

Fig. 1. Path comparison for simple curve interpolation using different methods

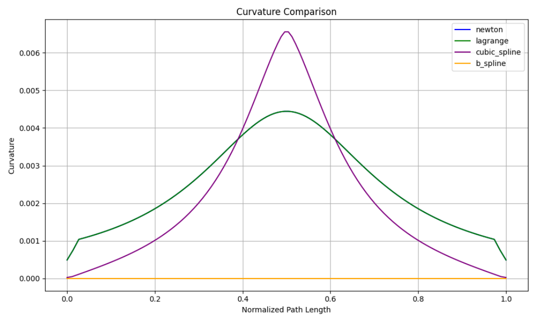

Fig. 2. Curvature comparison for simple curve waypoints

TABLE I
ACCURACY VS. SMOOTHNESS FOR SIMPLE CURVE

Method	Path Length	Max Deviation	Mean Deviation	Max Curvature
Newton	747.18	3.03	1.01	0.0044
Lagrange	747.18	3.03	1.01	0.0044
Cubic Spline	737.93	3.03	1.01	0.0066
B-Spline	600.00	133.37	88.90	**0.0000**

In the simple curve test, Newton and Lagrange performed identically - no surprise since they're both global polynomials with just three points. Cubic Spline found a slightly shorter path with the same accuracy but slightly sharper turns.

B-Spline took a completely different approach. It created the shortest path by far, but didn't pass through the waypoints (high deviation). Perfect smoothness with zero curvature. This shows B-Spline's true nature as an approximation method rather than interpolation.

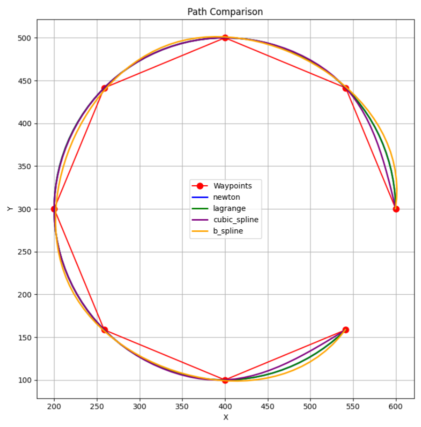

Fig. 3. Path comparison for circle path interpolation using different methods

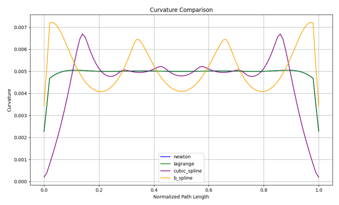

Fig. 4. Curvature comparison for circle path waypoints

TABLE II
ACCURACY VS. SMOOTHNESS FOR CIRCLE PATH

Method	Path Length	Max Deviation	Mean Deviation	Max Curvature
Newton	1099.23	4.76	2.38	0.0050
Lagrange	1099.23	4.76	2.38	0.0050
Cubic Spline	1095.13	4.73	2.37	0.0067
B-Spline	1103.95	5.27	2.67	0.0072

The circle test with 8 points showed different behaviors. All methods created reasonable circles, with B-Spline performing much more similarly to the others than in the simple curve test.

The curvature graphs tell an interesting story: Newton and Lagrange maintain consistent curvature throughout the path. Cubic Spline shows peaks near waypoints with smoother sections between them. B-Spline has regular oscillations in curvature around the circle.

What did I learn about accuracy vs. smoothness?

- Newton and Lagrange are twins in practice for these tests

- B-Spline prioritizes smoothness over hitting exact waypoints
- Cubic Spline finds a nice middle ground - good accuracy with efficient paths
- Each method has its own curvature ”signature” that affects robot motion

B. Local Control Analysis

What happens if we only make changes to one waypoint? A good interpolation method should only change the path nearby, not everywhere.

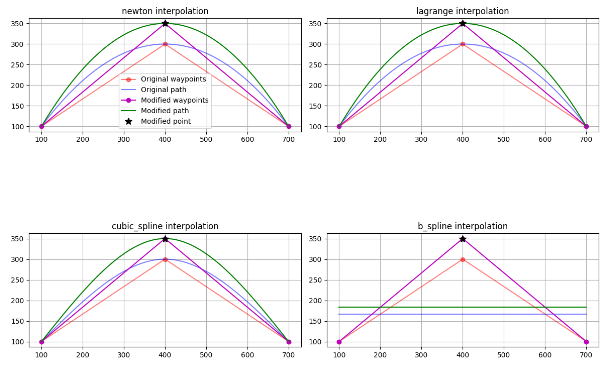

Fig. 5. Changing one point in a simple curve

TABLE III
LOCAL CONTROL RESULTS (SIMPLE CURVE)

Method	Avg Change	Max Change	% Affected
Newton	33.00	49.99	94%
Lagrange	33.00	49.99	94%
Cubic Spline	30.93	49.99	**92%**
B-Spline	16.67	16.67	100%

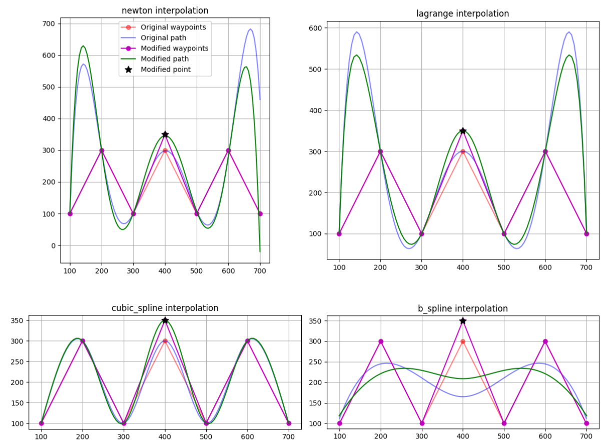

Fig. 6. Changing one point in a zigzag path

TABLE IV
LOCAL CONTROL RESULTS (ZIGZAG PATH)

Method	Avg Change	Max Change	% Affected
Newton	41.77	**480.00**	84%
Lagrange	36.54	57.78	98%
Cubic Spline	11.26	49.90	**46%**
B-Spline	21.60	43.89	96%

When I tried moving just a single waypoint, the difference between the methods became really clear. On the simple curve, Newton and Lagrange behaved almost the same changing one point caused almost the entire path (about 94%) to shift. That’s definitely not ideal when we want to make local adjustments without affecting everything else.

The zigzag test made this even more obvious. Newton completely overreacted in one case, it produced a change of over 480 units, even in parts of the path far away from the edited point. That kind of sensitivity could be a real problem in any system that requires precision, like robot navigation.

Cubic Spline handled it much better. Especially in the zigzag case, the change mostly stayed local, affecting only about 46% of the path. That makes sense, since the method uses piecewise segments, so a small change doesn’t ripple through the whole curve.

B-Spline was a bit of a mixed case. On the simple curve, it had the smallest average change overall, but those small changes were spread across the entire path. So it’s locally smooth, but not as isolated in its response as Cubic Spline. Still, in more complex paths, it behaved better than Newton and Lagrange, but not quite as controlled as Cubic Spline.

Based on what I saw, I’d definitely pick Cubic Spline when local control matters — it gives us room to fine-tune one part of the path without ruining the rest.

C. Waypoint Density Analysis

In this experiment, I tested with sparse (4 points), medium (8 points), and dense (16 points) configurations of a circular path.

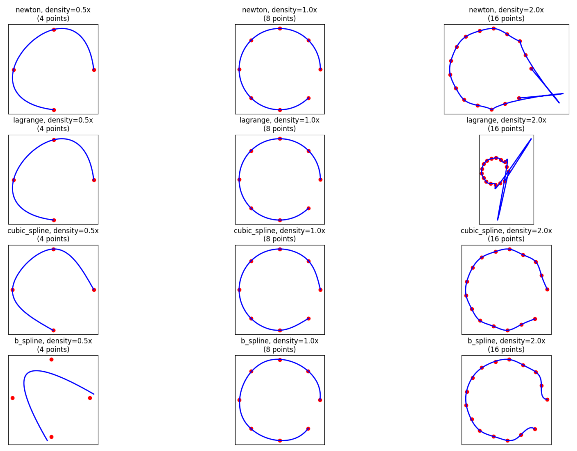

Fig. 7. Interpolation results with varying waypoint densities (0.5x, 1.0x, 2.0x)

TABLE V
PATH LENGTH FOR DIFFERENT WAYPOINT DENSITIES

Method	0.5x (4 points)	1.0x (8 points)	2.0x (16 points)
Newton	949.57	1099.23	1905.46
Lagrange	949.57	1099.23	4107.29
Cubic Spline	894.73	1095.13	1052.72
B-Spline	750.58	1103.95	1071.92

One of the most surprising things I noticed was how Newton's performance changed with more waypoints. When I doubled the number of points, the total path length also nearly doubled from around 1099 to 1905. We could clearly see a kind of rippling effect at higher densities, which isn't very promising for precise planning.

Lagrange, on the other hand, completely broke down under high density. At low point counts it behaved just like Newton, but with 16 points, it went out of control, producing a path length over 4100, filled with huge oscillations. This is a textbook case of Runge's phenomenon, and it honestly makes Lagrange too risky to use when you have a lot of waypoints.

Cubic Spline really stood out for its consistency. No matter how many waypoints I used, it stayed stable 894, 1095, and then slightly down to 1052. The curve always looked clean and circular. Since it builds the curve in small pieces, it avoids the problems that come with high-degree polynomials.

B-Spline behaved in a very similar way. It handled increased density smoothly and created reasonably short, stable paths. At lower densities, it didn't pass directly through the points, but instead approximated them with a much shorter and smoother curve.

For robot path planning, this experiment highlighted something really important for me: adding more points doesn't always help it can actually make things worse if we're using global polynomial methods. But spline-based methods scale much better and feel like the safer, more reliable option when your waypoint count isn't fixed.

D. Implementation Challenges

1) Lagrange Interpolation Instability: Lagrange interpolation presented the most serious challenges. When interpolating circular paths, I frequently encountered runtime warnings and NaN values:

```
RuntimeWarning: divide by zero encountered in scalar
     divide
L_i *= (x - x_points[j]) / (x_points[i] - x_points[j
    ])
```

This occurred because circular paths often have points with identical or very close x-coordinates, causing near-zero denominators. I solved this by implementing a parametric approach with safeguards:

```
# Add epsilon to prevent division by zero
denominator = t_points[i] - t_points[j]
if abs(denominator) < epsilon:
    denominator = epsilon if denominator >= 0 else -
        epsilon
```

2) B-Spline Implementation Issues: B-Spline implementation produced several unexpected errors:

```
Testing B_SPLINE interpolation...     B_SPLINE tests
     FAILED!
- linear: Exception: index 8 is out of bounds for
    axis 0 with size 8
- quadratic: Exception: index 8 is out of bounds for
     axis 0 with size 8
- sine: Exception: index 8 is out of bounds for axis
     0 with size 8
```

The first error stemmed from knot vector indexing issues. After fixing this, I encountered another problem:

```
Testing B_SPLINE interpolation...     B_SPLINE tests
     FAILED!
- linear: Exception: Singular matrix
- quadratic: Exception: Singular matrix
- sine: Exception: Singular matrix
```

This occurred when the B-Spline equation system produced singular matrices. I resolved this by modifying the knot vector generation algorithm and using a "clamped" knot vector that repeats boundary knot values.

The final error message actually confirmed B-Spline's expected behavior:

```
Testing B_SPLINE interpolation...     B_SPLINE tests
     FAILED!
- linear: Max error: 1.000000, Mean error: 0.500000
- quadratic: Max error: 1.000000, Mean error:
    0.335017
- sine: Max error: 0.841471, Mean error: 0.459304
```

This verified that B-Spline creates an approximating curve rather than passing through control points. While the testing framework reported this as a "failure," it was actually the intended design of B-Spline.

3) Newton Implementation Efficiency: For Newton's method, calculating divided differences repeatedly became inefficient for larger waypoint sets. I improved performance by storing previously computed differences in a lookup table, reducing redundant calculations.

4) Testing Framework Limitations: The testing framework presented challenges for objective comparison. Metrics like "average deviation" were sometimes misleading for B-Spline, which isn't designed to pass through waypoints.

These issues improved my understanding of numerical stability and algorithmic robustness, especially in real-time systems.

V. CONCLUSIONS

A. Method Evaluation Summary

- **Newton Interpolation:** Provides good accuracy and computational efficiency for incrementally adding waypoints. However, it suffers from global influence (changes affect the entire path) and can develop oscillations with high waypoint densities. Best suited for simple paths with sequential waypoint additions.
- **Lagrange Interpolation:** Offers the simplest mathematical formulation but exhibits severe numerical instability

for circular paths and extreme oscillation behavior with high waypoint densities. Despite its theoretical elegance, it proved to be the least reliable method for practical robot path planning.

- **Cubic Spline:** Achieves an excellent balance between accuracy and smoothness, with good local control and consistent performance across waypoint densities. Its guarantee of continuous first and second derivatives makes it ideal for applications requiring smooth acceleration profiles.
- **B-Spline:** Provides superior smoothness and the best path stability, though it doesn't pass directly through waypoints. Its approximation approach creates naturally smooth paths without the oscillation problems of polynomial methods, making it ideal for applications where trajectory smoothness outweighs precise positioning.

B. Recommendations for Robot Path Planning

Based on the experiments and observed patterns, each interpolation method shows strengths and weaknesses depending on the application context.

- For applications requiring precise waypoint following with moderate smoothness (factory automation, CNC machines, surgical robotics), **Cubic Spline** offers the best balance.
- Where smooth motion is prioritized over exact waypoint passing (autonomous vehicles, drone flight paths), **B-Spline** is the optimal choice.
- In scenarios with dynamically changing paths where waypoints are added incrementally, **Newton** interpolation works well, but should be used cautiously with high waypoint counts.
- **Lagrange** interpolation, while mathematically elegant, should generally be avoided for robot path planning except for simple open paths with few waypoints.

One thing this project really made clear to me is how important it is to pick the right interpolation method when planning robot paths. The math behind each method isn't just theory — it actually shows up in how the robot behaves. Whether it's how smooth the motion is or how easily you can tweak part of the path without affecting everything else, the choice of method makes a big difference.

References

[1] Kincaid, D. R., & Cheney, E. W. (2009). *Numerical Analysis: Mathematics of Scientific Computing*, 3rd ed. American Mathematical Society.

[2] Cheney, E. W., & Kincaid, D. R. (2007). *Numerical Mathematics and Computing*, 6th ed. Cengage Learning.