NA Project Homework

陈澎 Chen Peng 3220103443 *

Xinji 2201, Zhejiang University

2024 年 12 月 23 日

Problem A

Compile and run the program `A.cpp`. Its output includes max errors and convergence rates of different situations and the values of the interpolation curves for plotting.

Compile and run the program `plotA.py` to plot the interpolation curves and the original function.

In addition to the original requirement of using cubic complete boundary interpolation curves, this program uses PPForm and BSpline to fit linear and cubic spline functions, and for cubic spline functions, it tests four boundary conditions: `NATURAL`, `COMPLETE`, `NOT_A_KNOT`, and `SECOND`. This better tests whether the two interpolation methods can correctly implement linear spline functions under uniform node conditions and cubic spline functions under various boundary conditions.

The program's output max errors and convergence rates are stored in the file `A_maxerror_convergence.csv` in directory `output\problemA\`, and the interpolation data is also output in the `output\problemA` directory, named in a concise and clear manner.

According to the requirements, plotA.py was designed to plot the interpolation curves and the original function, stored in the `figure\problemA` directory.

As shown in Fig.1, the interpolation curves for cubic PPForm splines under four different boundary conditions with different numbers of uniform interpolation points are displayed. (Add text later to express that spline interpolation is free of the wide oscillations in the Runge phenomenon.)

In fact, from the output interpolation data, it can be seen that PPForm and BSpline produce the same curves when using the same interpolation points and boundary conditions for cubic interpolation.

*Email: cpzju@zju.edu.cn

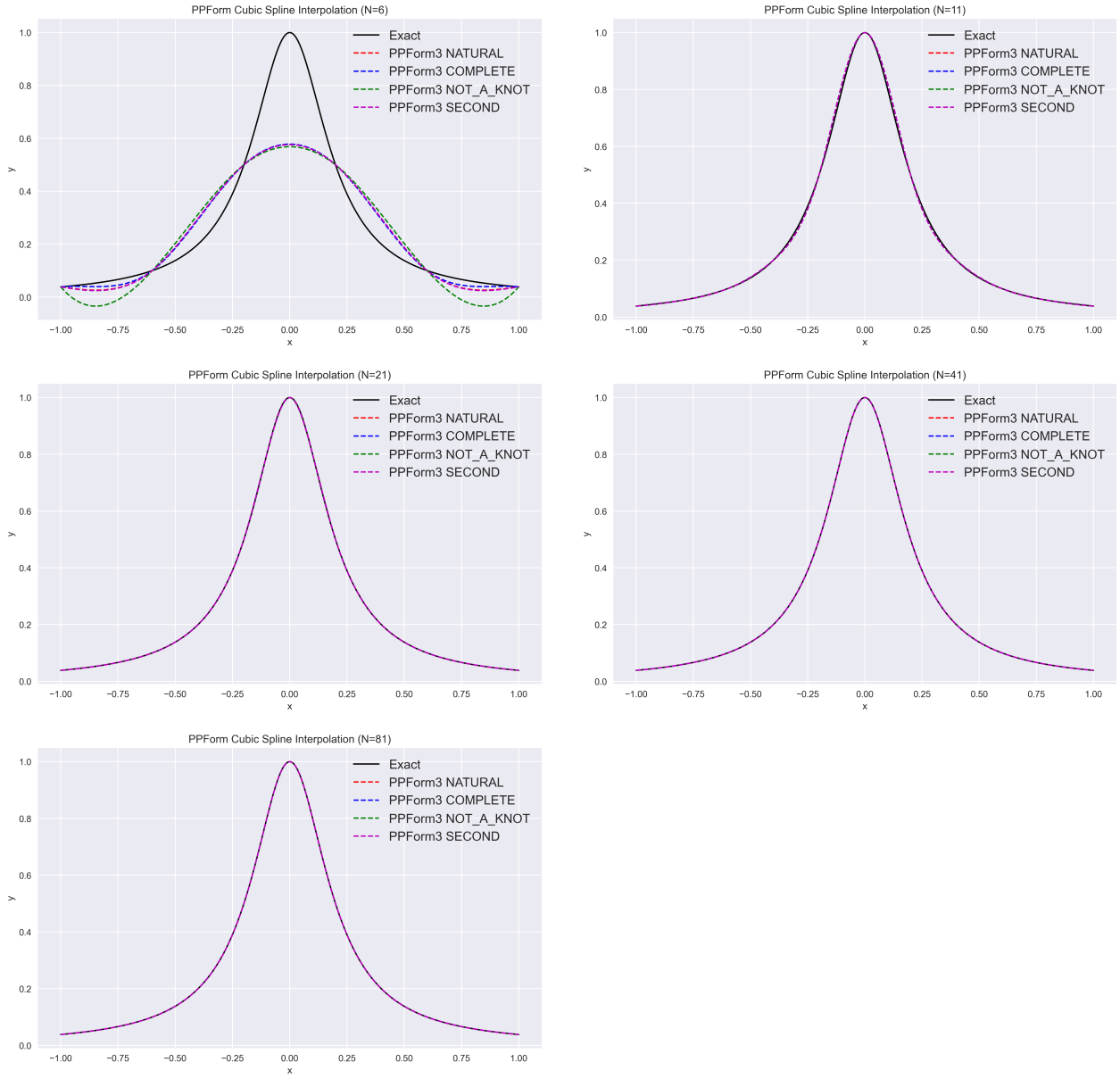

Fig. 1: Interpolation curves for different boundary conditions by cubic PPForm splines

As shown in Fig.2, the interpolation curves for linear PPForm and BSpline splines with different numbers of uniform interpolation points are displayed.

In fact, from the output interpolation data, it can be seen that both produce the same curves when using the same interpolation points and boundary conditions for linear interpolation.

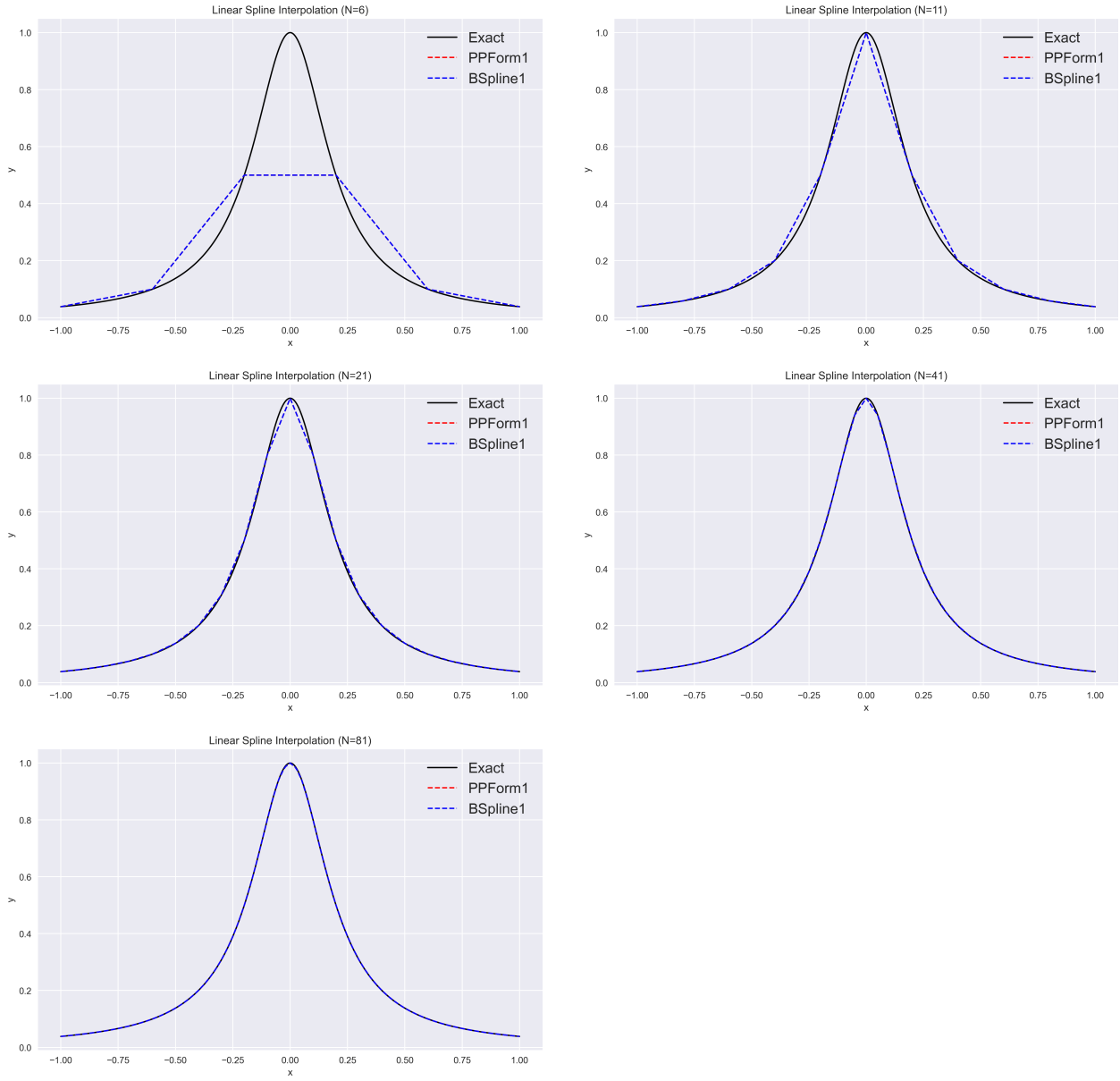

Fig. 2: Interpolation curves by linear splines

From the above tests, it can be concluded that the two interpolation methods can correctly implement linear spline functions under uniform node conditions and cubic spline functions under various boundary conditions. A separate program will be designed later to test interpolation under non-uniform node conditions and periodic boundary conditions.

The max norm of the interpolation error vector at mid-points of the subintervals for cubic PPForm with complete boundary conditions are as shown in Table.1:

Table. 1: Max norm of the interpolation error vector at mid-points of the subintervals for cubic PPForm with complete boundary conditions

N	Method	BoundaryCondition	MaxNorm
6	PPForm3	COMPLETE	0.421705
11	PPForm3	COMPLETE	0.0205289
21	PPForm3	COMPLETE	0.00316894
41	PPForm3	COMPLETE	0.000275356
81	PPForm3	COMPLETE	1.609e-005

The error and convergence rate data for cubic PPForm with complete boundary conditions are as shown in Table.2:

Table. 2: Max errors and convergence rates for cubic PPForm with complete boundary conditions

N	Method	BoundaryCondition	MaxError	ConvergenceRate
6	PPForm3	COMPLETE	0.421705	0.000000
11	PPForm3	COMPLETE	0.0219704	-4.2626
21	PPForm3	COMPLETE	0.00318271	-1.39362
41	PPForm3	COMPLETE	0.000277684	-1.17291
81	PPForm3	COMPLETE	1.609e-005	-1.0273

The convergence rate is calculated using the formula:

$$\text{rate} = \frac{\log\left(\frac{\text{max\_error\_current}}{\text{max\_error\_previous}}\right)}{\log\left(\frac{\frac{2}{N_{\text{current}}-1}}{\frac{2}{N_{\text{previous}}-1}}\right)}$$

where max_error_current and max_error_previous are the maximum errors for the current and previous number of interpolation points N_{current} and N_{previous}, respectively.

To illustrate this relationship, we can plot N against the maximum error on a logarithmic scale. The least squares fit of the error data in logarithmic coordinates calculated by `src\problemA\plotA_leastSquare.py` is also shown. It is depicted in Fig.3.

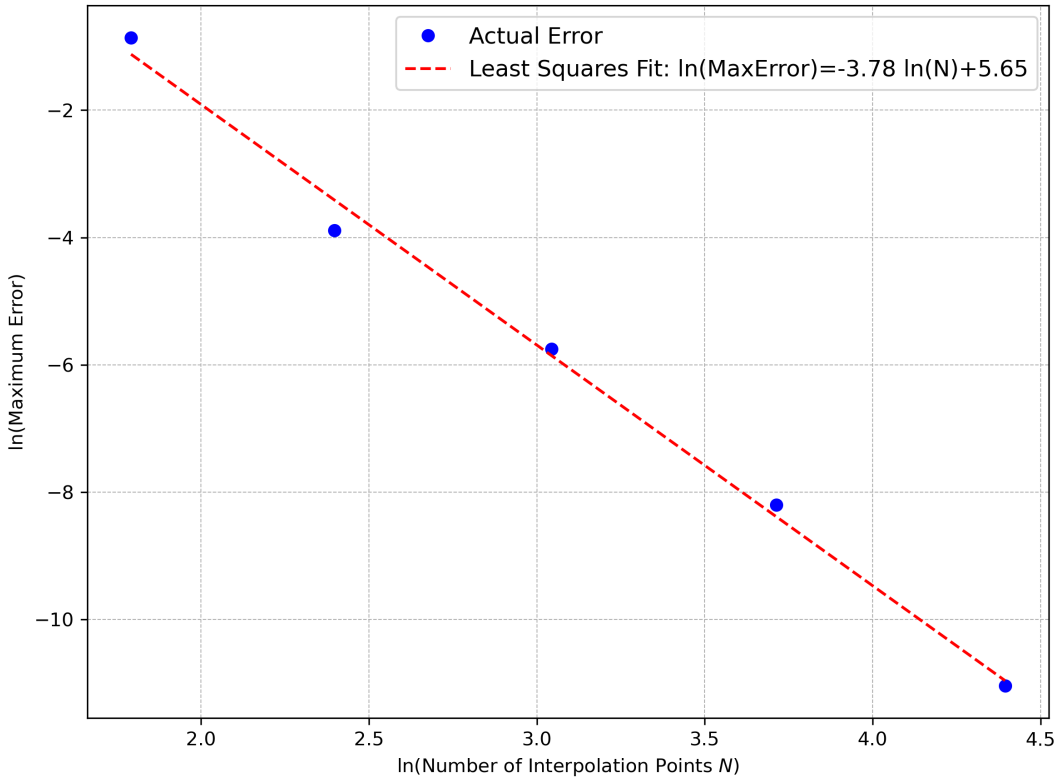

Fig. 3: Log-log plot of N vs. Maximum Error for cubic PPForm splines with complete boundary conditions

The relationship between N and the maximum error can be analyzed from several aspects:

1. **Small Dataset Behavior (N $\leq$ 11):**
   For small N, the convergence rate is notably high (around -4.36), indicating rapid error reduction. This is particularly evident in the transition from $N = 6$ to $N = 11$, where the maximum error drops dramatically from 0.421705 to 0.020529.

2. **Large Dataset Behavior (N $\geq$ 41):**
   As N increases, the convergence rate gradually stabilizes near -1, suggesting a more moderate decrease in error. For example, from $N = 41$ to $N = 81$, the convergence rate is approximately -1.02426.

The least squares fit of the error data in logarithmic coordinates yields:

$$\ln(\text{MaxError}) = -3.78 \ln(N) + 5.65$$

or equivalently in exponential form:

$$\text{MaxError} = e^{5.65} N^{-3.78} \approx 284.290 N^{-3.78}$$

This confirms that the error decreases approximately as $O(N^{-3.77})$.

Problem C

Compile and run the program `C.cpp`. Its output includes the values of the interpolation curves for plotting. The data stores in the directory `output\problemC`.

Compile and run the program `plotC.py` to plot the interpolation curves and the original function. The figure is shown in Fig.4.

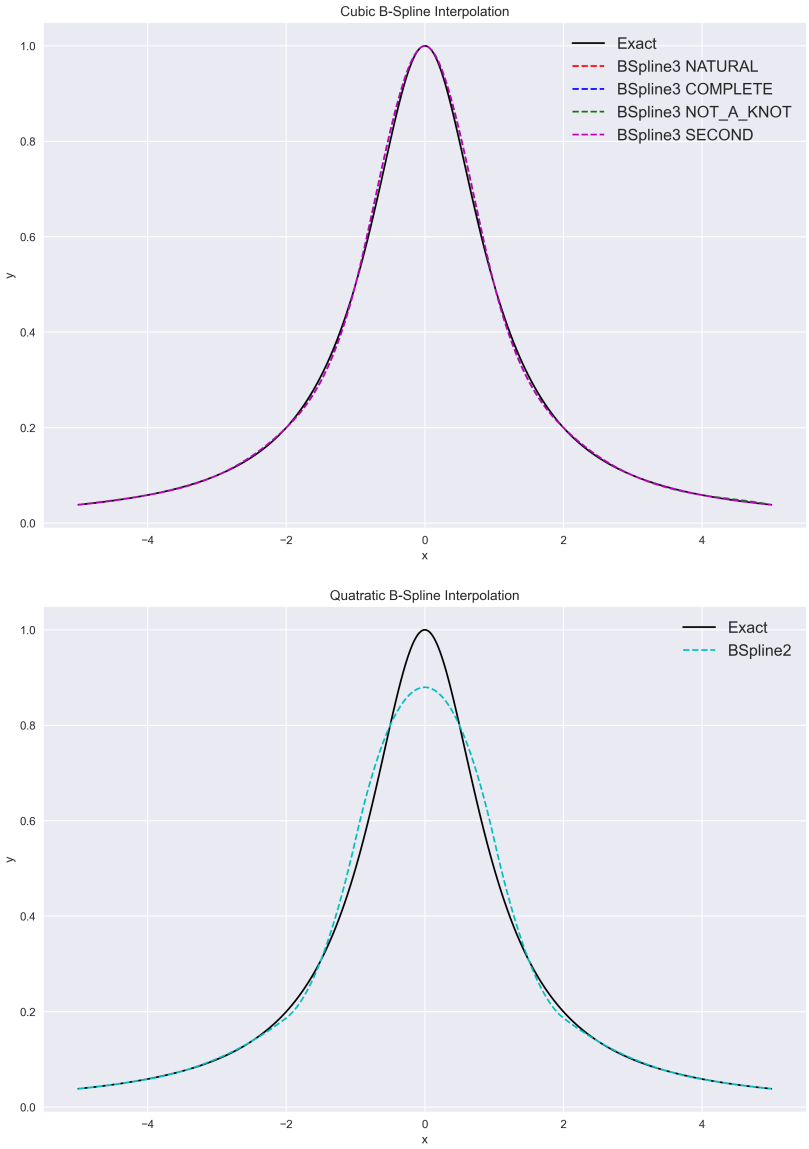

Fig. 4: Quatratic and cubic BSpline interpolation for Problem C

The program tests quatratic B-spline curve fitting for the target function and cubic B-spline curve fitting under various boundary conditions and linear B-spline curve fitting.

The implementation of the quatratic B-spline curve follows the relevant theorems in the textbook. In the actual implementation, it is ensured that the quatratic spline passes through all control points and that the first derivative is continuous at the internal nodes, except for the first two and the last two nodes.

The linear B-spline curve uses the same interpolation points as the cubic B-spline, mainly to test whether the B-spline linear interpolation can be correctly implemented. The figure is shown in Fig.5.

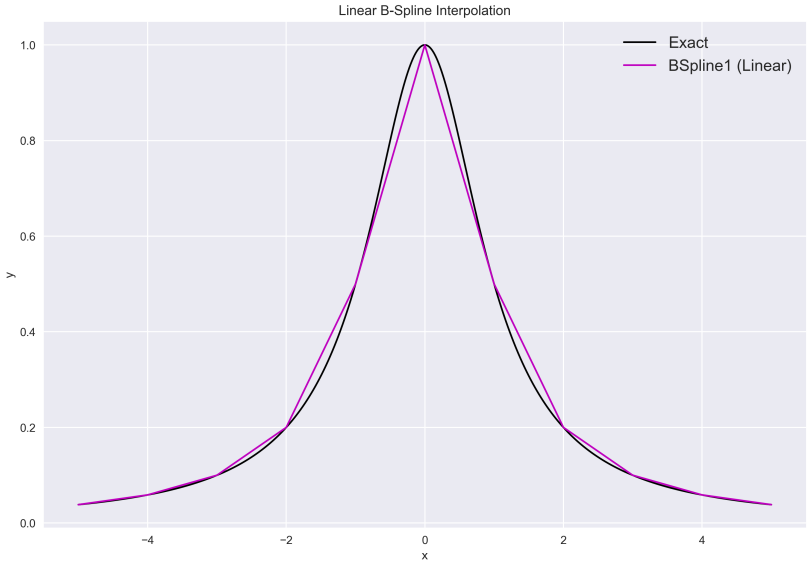

Fig. 5: Linear BSpline interpolation for Problem C

As can be seen from the figure, the fitting effect of the quatratic B-spline curve is worse than that of the cubic B-spline curve. This is because the locality of the quatratic spline is stronger than that of the cubic spline, which weakens the fitting effect. The linear B-spline curve fitting effect is also worse than the cubic B-spline curve, which is consistent with the theoretical analysis.

Problem D

Compile and run the program `D.cpp`. Its output includes the error data of specific points and max errors data for different B-spline methods and the values of the interpolation curves for plotting. The data stores in the directory `output\problemD`.

In the bspline package, the relevant functions for the 2nd-order cardinal B-spline have been implemented separately. Since the 3rd-order cardinal B-spline required by the problem can be implemented by the existing 3rd-order B-spline with arbitrary nodes, the 3rd-order cardinal B-spline has not been implemented separately in the program.

I added first-order B-splines to test whether the first-order B-spline interpolation algorithm is correct, using the same interpolation points as the third-order B-splines.

The error data of specific points for different B-spline methods is shown in Table.3.

Table. 3: Error Data for Different B-spline Methods

x	BSpline1	BSpline2	BSpline3 NATURAL	BSpline3 COMPLETE	BSpline3 NOT A KNOT	BSpline3 SECOND
-3.5000	3.9401e-003	6.6509e-017	7.8997e-004	6.6957e-004	9.9183e-004	6.8675e-004
-3.0000	5.5498e-018	1.4184e-003	1.5821e-016	4.7183e-017	6.3839e-017	3.3305e-017
-0.5000	5.0000e-002	4.4398e-017	2.0531e-002	2.0529e-002	2.0533e-002	2.0529e-002
0.0000	0.0000e+000	1.2024e-001	2.2204e-016	0.0000e+000	2.2204e-016	2.2204e-016
0.5000	5.0000e-002	4.4398e-017	2.0531e-002	2.0529e-002	2.0533e-002	2.0529e-002
3.0000	5.5498e-018	1.4184e-003	1.9428e-017	4.7183e-017	6.3839e-017	2.2206e-017
3.5000	3.9401e-003	1.6758e-017	7.8997e-004	6.6957e-004	9.9183e-004	6.8675e-004

It was observed that some of the errors close to machine precision. At -3.0 and 3.0, the error between the first-order B-spline and the actual function is close to machine precision $2.2e^{-16}$. This is because these points are control points, and the curve must pass through these points. At -3.5, -0.5, 0.5, and 3.5, the error between the second-order B-spline and the actual function is close to machine precision $2.2e^{-16}$, for the same reason.

Additionally, it may also be due to the inevitable floating-point errors in the computer's calculation when computing the actual function values, the fitted curve values and the interpolation points.

From the max errors data shown in Table.4 and the images of the interpolation curve shown in Fig.4 and Fig.5, it can be seen that under the same node conditions, the cubic B-spline interpolation is more accurate than the linear B-spline interpolation. Comparing the quadratic cardinal B-spline with the cubic cardinal B-spline, the cubic cardinal B-spline provides a better fitting effect.

Table. 4: Max Errors for Different B-spline Methods

Method	BSpline1	BSpline2	BSpline3 NATURAL	BSpline3 COMPLETE	BSpline3 NOT A KNOT	BSpline3 SECOND
MaxError	6.7431e-002	1.2024e-001	2.1974e-002	2.1972e-002	2.1977e-002	2.1972e-002

Problem E

Compile and run the program `E.cpp`. Its output includes max errors data for B-spline in `maxerror.csv` and PPForm methods of different boundary conditions and the values of the fitted and actual curves for plotting. It also outputs a file `compare.csv` that compares whether the curves obtained with the same interpolation points and boundary conditions under equidistant node conditions are consistent between PPForm and B-spline methods. The data is stored in the directory `output\problemE`.

Compile and run the program `plotE.py` to plot the interpolation curves and the original function. The figures are stored in the directory `figure\problemE`.

Compile and run the program `plotE_error.py` to plot the least squares fit of the maximum errors of different curves under different methods with equidistant nodes. The cumulative chordal length method's error is not calculated because there is no good method to compute it. The error of 2D curves is calculated using Euclidean distance, and the error on the 3D unit sphere is calculated using spherical angular distance.

Reviewing the problem, the required curves are as follows: The heart-shaped curve r_1 is given by

$$r_1(t) = (x(t), y(t)) = \left(\sqrt{3}\sin t, \frac{2\sqrt{3}}{3}\cos t + \frac{2}{3}\sqrt{\sqrt{3}|\sin t|} \right),\ t \in [0, 2\pi].$$

The spiral curve r_2 is given by

$$r_2(t) = (x(t), y(t)) = (\sin t + t\cos t, \cos t - t\sin t),\ t \in [0, 6\pi].$$

The curve r_3 is given by

$$r_3(t) = (x(t), y(t), z(t)) = (\sin(u(t))\cos(v(t)), \sin u(t)\sin v(t), \cos u(t)),\ t \in [0, 2\pi]$$

where $u(t) = \cos t,\ v(t) = \sin t$.

According to the coordinate data files of the fitted curves, as shown in Table.5, the curves obtained by PPForm and B-spline methods with the same interpolation points and boundary conditions are consistent within the allowable error range.

Table. 5: Comparison of PPForm and B-Spline Methods

Curve	N	Boundary	Is The Same
r1	10	NOT_A_KNOT	True
r1	10	PERIODIC	True
r1	40	NOT_A_KNOT	True
r1	40	PERIODIC	True
r1	160	NOT_A_KNOT	True
r1	160	PERIODIC	True
r2	10	COMPLETE	True
r2	10	NOT_A_KNOT	True
r2	10	SECOND	True
r2	40	COMPLETE	True
r2	40	NOT_A_KNOT	True
r2	40	SECOND	True
r2	160	COMPLETE	True
r2	160	NOT_A_KNOT	True
r2	160	SECOND	True
r3	10	NOT_A_KNOT	True
r3	10	PERIODIC	True
r3	40	NOT_A_KNOT	True
r3	40	PERIODIC	True
r3	160	NOT_A_KNOT	True
r3	160	PERIODIC	True

Therefore, the following discussion only considers the fitted curves under the BSpline method.

The least squares fit of the maximum errors of different curves under different methods with equidistant nodes is shown in Fig.6.

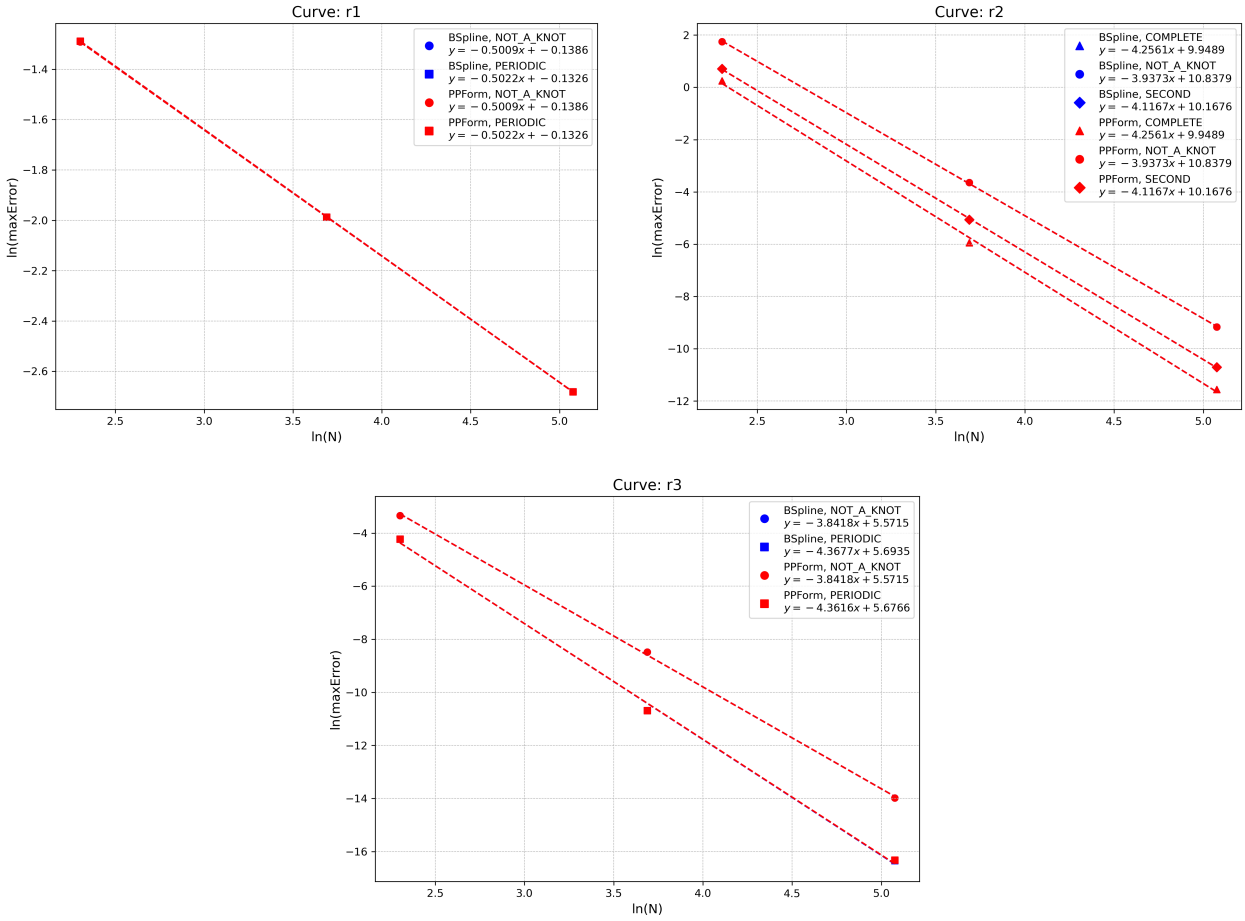

Fig. 6: Least squares fit of the maximum errors of different curves under different methods with equidistant nodes

It can be concluded that for the curve r_2, the Complete boundary condition, which provides the velocity information at both ends of the curve, gives the best fitting effect, with the smallest error and the steepest slope. For the curve r_1, the Not-a-Knot boundary condition performs well in terms of smoothness and accuracy. The Periodic boundary condition also provides a similar effect to the Not-a-Knot boundary condition, ensuring smoothness and continuity. For the curve r_3, the periodic boundary condition provides the best fitting effect. This is because the curve is inherently periodic, and using periodic boundary conditions ensures that the curve is smooth and continuous at the boundaries, avoiding any abrupt changes or discontinuities.

Special attention should be paid to the fitting of curve r_1, as there are two sharp extrema points, resulting in larger errors near these points. In practical curve fitting, segmented fitting around these points can be used to improve the fitting effect.

The max errors analysis of different boundary conditions for B-spline interpolation reveals the following insights:

Clamped or Complete Boundary

- **Suitable for:** Scenarios where boundary derivatives (e.g., slope or velocity) are known, such as when the slope of the interpolating curve is determined by physical conditions.
- **Advantages:** Allows precise control over boundary behavior, suitable for well-constrained situations.
- **Disadvantages:** If the specified derivative values are inaccurate, the overall interpolation quality may

degrade.

Second Derivative Boundary

- **Suitable for:** Scenarios where boundary curvature is known, such as in certain mechanical modeling applications.
- **Advantages:** Suitable for scenarios sensitive to boundary curvature, allowing more accurate simulation of boundary behavior.
- **Disadvantages:** Requires accurate second derivative values.

Not-a-Knot Boundary

- **Suitable for:** Data without distinct boundary features, requiring higher-order continuity without imposing excessive constraints.
- **Advantages:** Produces curves with better global continuity, automatically smoothing the curve.
- **Disadvantages:** Boundary behavior may not always meet expectations.

Periodic Boundary:

- **Suitable for:** Inherently periodic data, such as 24-hour temperature distributions in time series.
- **Advantages:** Naturally closed curves without abrupt boundary behavior.
- **Disadvantages:** Forcing non-periodic data into a periodic boundary may result in unnatural fitting.

This explains why, in the above curve fitting, different boundary conditions can be theoretically selected based on the nature of the curve and initial conditions to achieve the best fitting effect.

Finally, I will show the B-spline fitting curves of the three curves under the best boundary conditions with different data points, to compare and analyze the fitting effects of cumulative chordal length and equidistant nodes.

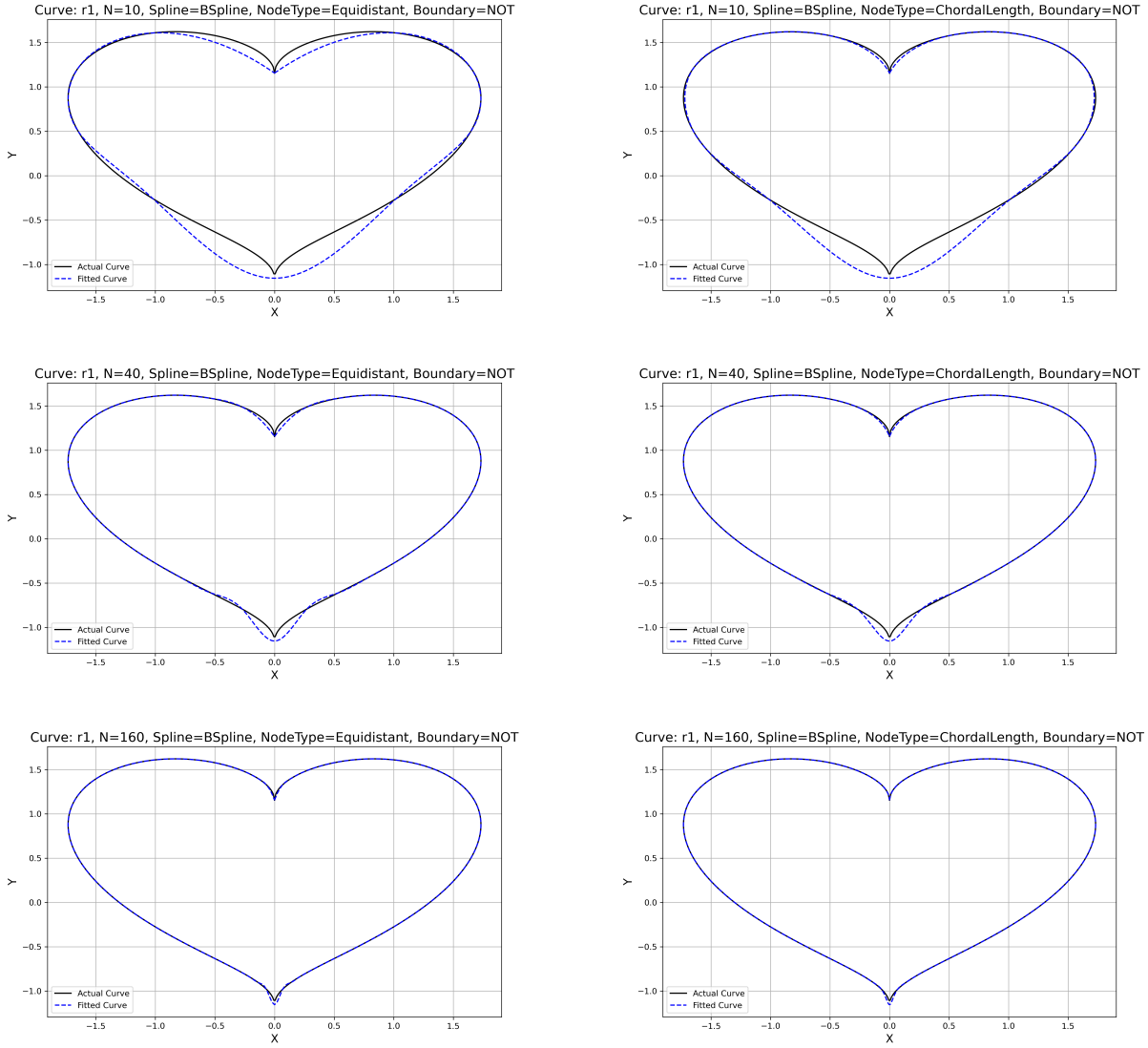

Fig. 7: Comparison of Equidistant and Chordal Length for curve r_1 with NOT boundary condition

The fitting effect of curve r_1 is shown in Fig.7. The left column is for equidistant nodes, and the right column is for cumulative chordal length. Each row corresponds to a different number of data points. It can be seen that the cumulative chordal length method performs better in areas with sharp changes, such as the tip of the heart, while the equidistant nodes perform better in smoother areas.

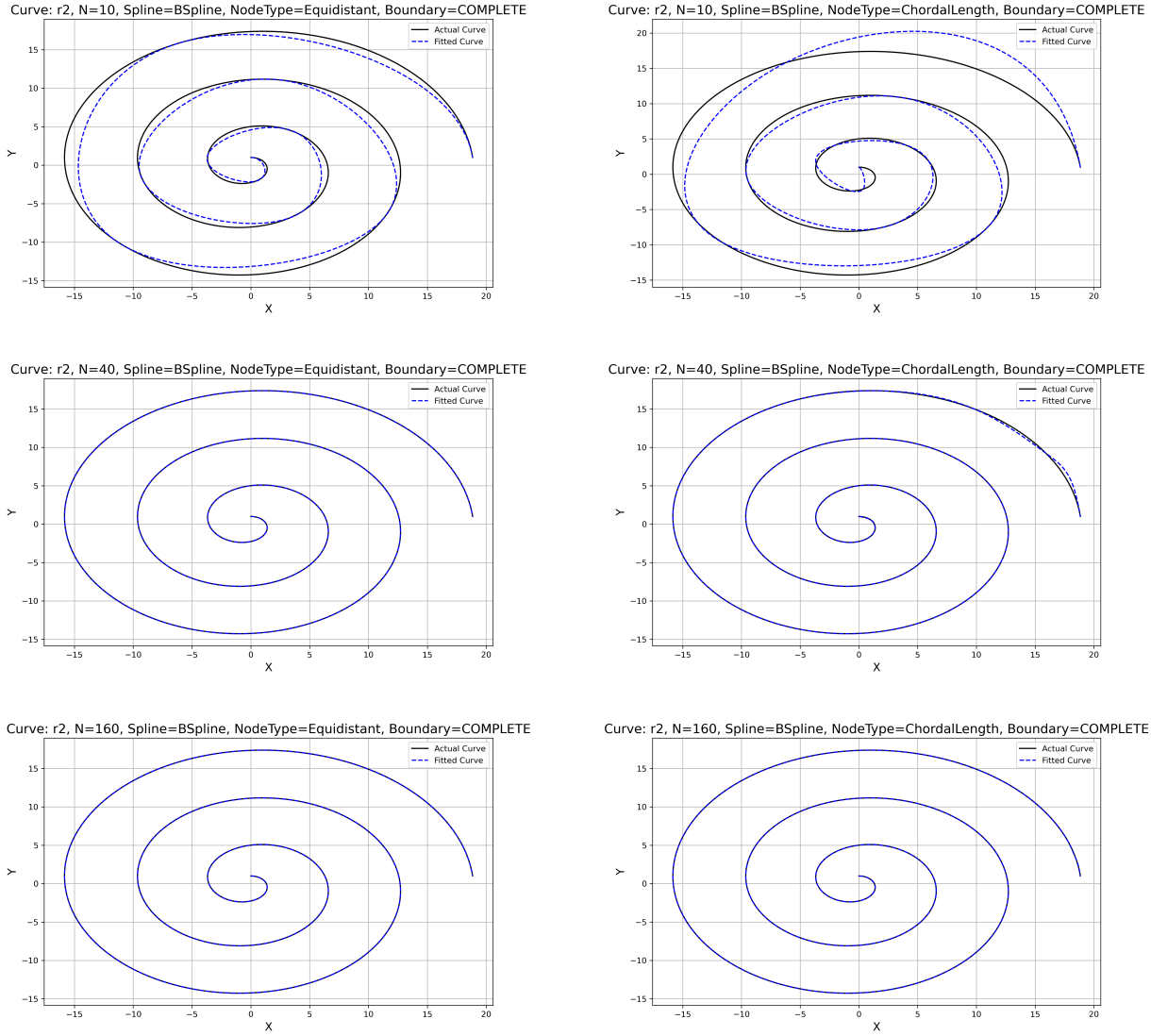

Fig. 8: Comparison of Equidistant and Chordal Length for curve r_2 with COMPLETE boundary condition

The fitting effect of curve r_2 is shown in Fig.8. This spiral curve is relatively smooth, so a small number of equidistant nodes can achieve a good fitting effect. However, for the cumulative chordal length method, due to the approximation errors in calculating the chord length, the fitting effect at the ends is poor when the number of data points is small. Therefore, more data points are needed to ensure a good fitting effect.

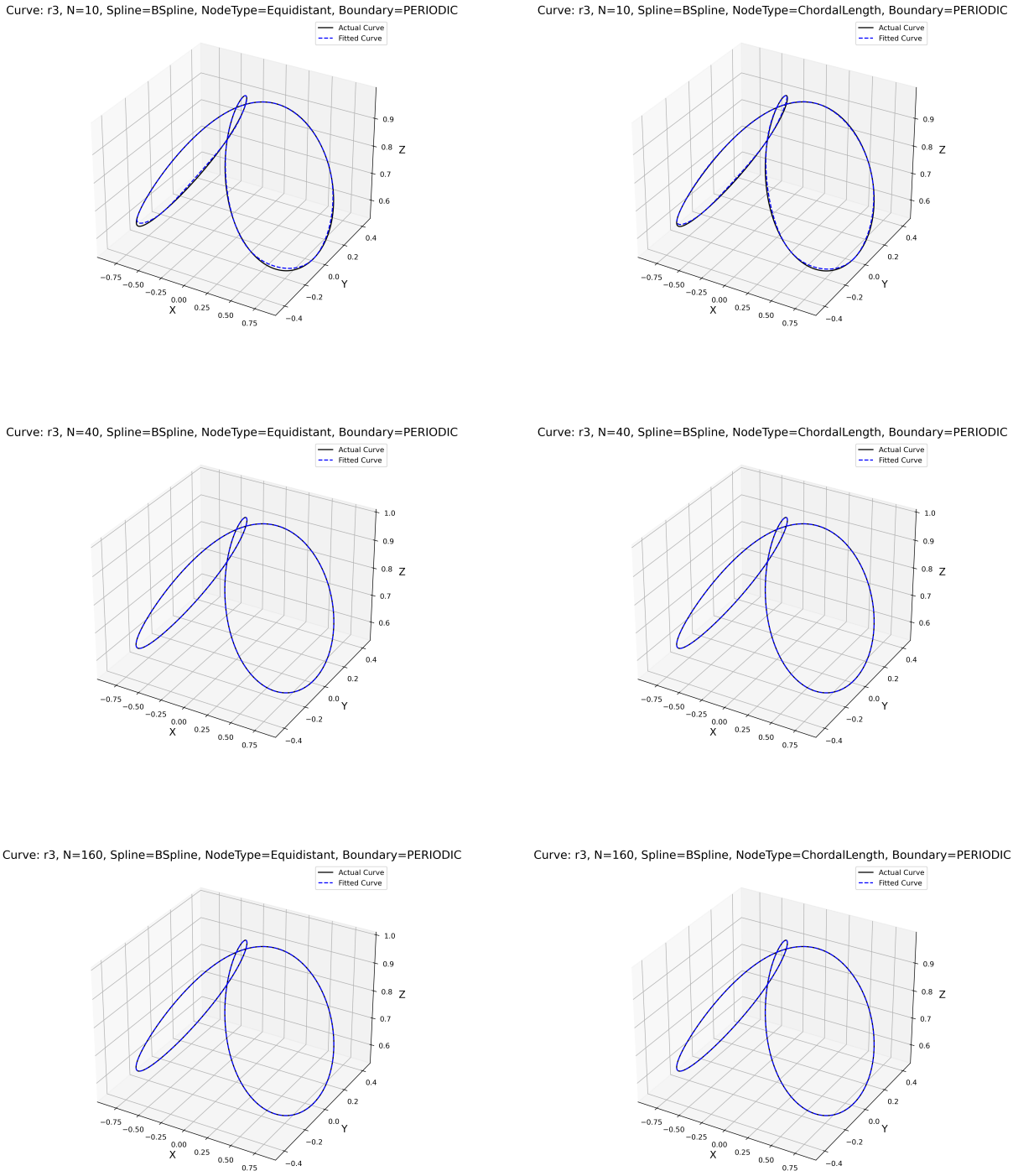

Fig. 9: Comparison of Equidistant and Chordal Length for curve r_3 with PERIODIC boundary condition

The fitting effect of curve r_3 is shown in Fig.9. This is also a smooth curve without rapid changes, so both equidistant nodes and cumulative chordal length methods can achieve good fitting results.

Combining theory and the practical analysis of the cumulative chordal length method and equidistant nodes method as above, the following conclusions can be drawn:

Cumulative Chordal Length Method

- **Advantages:**

  - Better adapts to the geometric shape of the curve.
  - Nodes are densely distributed in regions with rapid changes (e.g., curve turns or high curvature areas), improving fitting accuracy.
  - Avoids the "oscillation" phenomenon (Runge phenomenon), especially in polynomial interpolation.

- **Disadvantages:**
  - Node distribution requires calculating cumulative chord length, increasing computational complexity.
  - Suitable for complex curves but may lead to overly dense distribution for very smooth curves.

Equidistant Nodes Method

- **Advantages:**
  - Simple node distribution, easy to implement.
  - Provides uniform coverage for smooth curves or regions with small changes.
  - High computational efficiency, especially suitable for uniformly distributed data.

- **Disadvantages:**
  - For curves with rapid changes, fitting errors may be significant as rapidly changing regions are not sufficiently sampled.
  - Prone to the "oscillation" phenomenon, especially at the ends of the curve.

Problem F

Compile and run the program `F.cpp`. Its output includes the values of the interpolation curves for plotting. The data stores in the directory `output\problemF`. It will also run the program `plotF_trunc.py` and `plotF_divide.py` to plot the truncated power function and divided truncated power function. The figures are stored in the directory `figure\problemF`.

First, the calculation and plotting of the first-order truncated power function were tested, as shown in Fig.10.

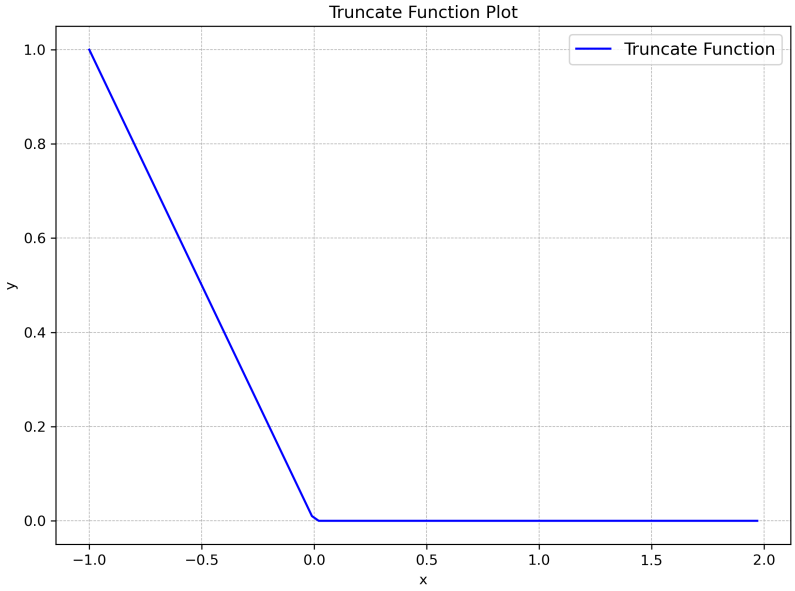

Fig. 10: Calculation and plotting of the first-order truncated power function

Next, the calculation and plotting of the second-order truncated power function were tested, as shown in Fig.11.

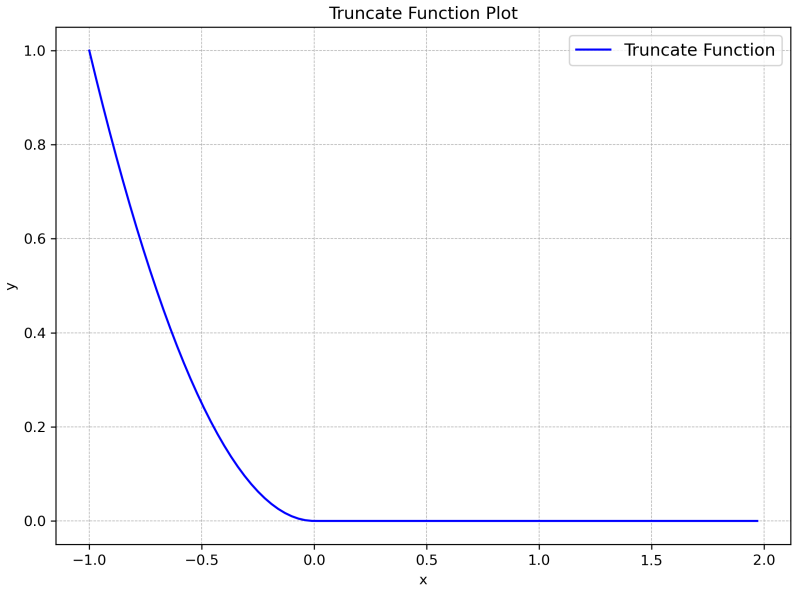

Fig. 11: Calculation and plotting of the second-order truncated power function

Additionally, the difference calculation and plotting of the first-order truncated power function were tested, as shown in Fig.12.

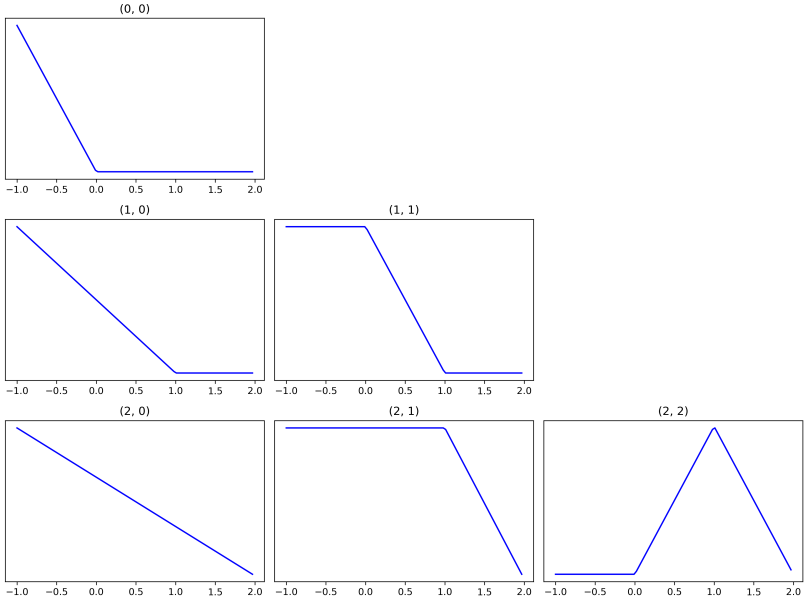

Fig. 12: Difference matrix calculation of the first-order truncated power function

Finally, the difference calculation and plotting of the second-order truncated power function were tested, as shown in Fig.13.

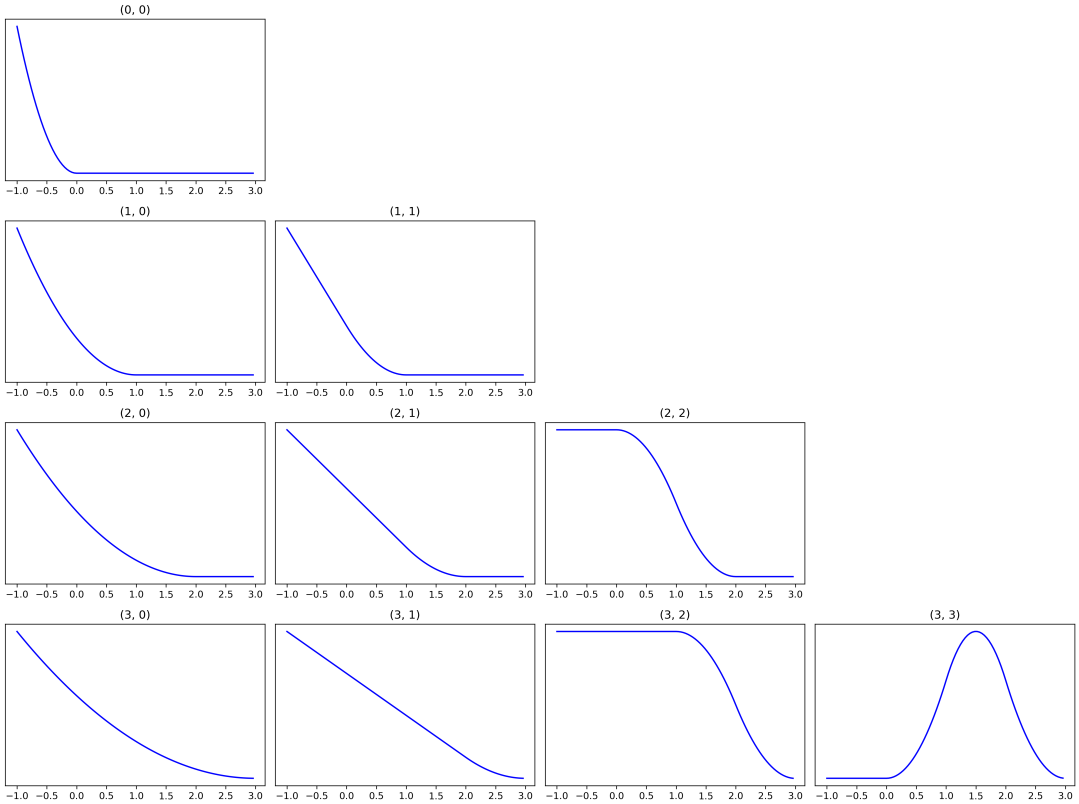

Fig. 13: Difference calculation of the second-order truncated power function

Requirement 5

Compile and run the program `require5.cpp`. Its output includes the coordinate data of the fitted curves to be plotted. The data is stored in the directory `output\requirement5`.

Compile and run the program `plotRequire5.py` to plot the interpolation curves. The figures are stored in the directory `figure\requirement5`.

Using a third-order B-spline, the input data is:

```
std::vector<double> knots = {0, 1, 2, 3, 4, 5, 6};
std::vector<double> coefficients = {0, 1, 0, -1, 0, 1, 0, -1, 0};
```

Using a fifth-order B-spline, the input data is:

```
std::vector<double> knots = {0, 1, 2, 3, 4};
std::vector<double> coefficients = {0, 2, 3, 2, 0, -1, 2, 3, 2};
```

As shown in Fig.14, the third-order and fifth-order B-spline curves are plotted respectively.

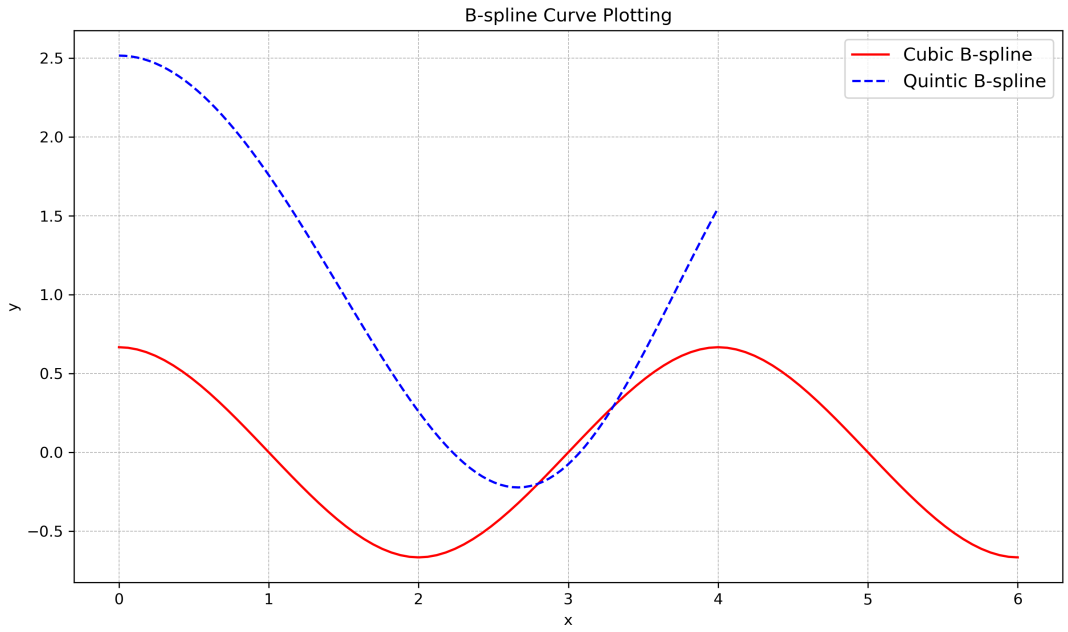

Fig. 14: B-spline curves for Requirement 5

Therefore, given the dimension and coefficients and knot sequences, `Bspline.hpp` can implement the plotting of splines of any order and any nodes.

Test

Compile and run the program `test.cpp`. Its output includes the max errors and convergence rate of different situations and the values of the interpolation curves for plotting. The data stores in the directory `output\test`. Run the program `plotTest.py` to plot the interpolation curves and the original function. The figures are stored in the directory `figure/test`.

The program is designed to test first-order B-spline and PPForm interpolation under non-uniform nodes, as well as third-order B-spline and PPForm interpolation under five different boundary conditions, along with

their interpolation effects. The target function is $\sin x$, and the interval is $[0, 2\pi]$. The program actually uses a quadratic distribution to generate non-uniform nodes.

As shown in Fig.15, below are the interpolation results of linear PPForm and BSpline interpolation for different values of N.

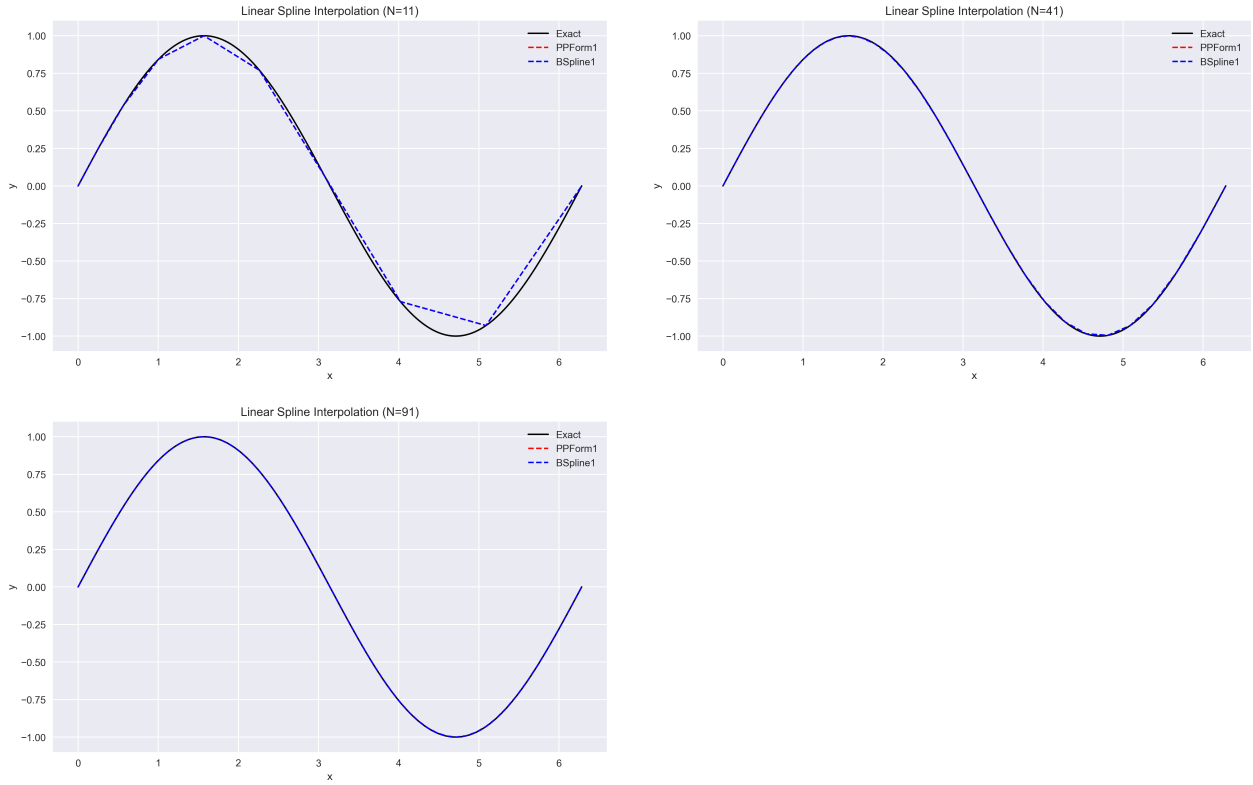

Fig. 15: Linear PPForm and BSpline interpolation for different N values

As shown in Fig.16, below are the interpolation results of third-order PPForm and BSpline interpolation for different values of N.

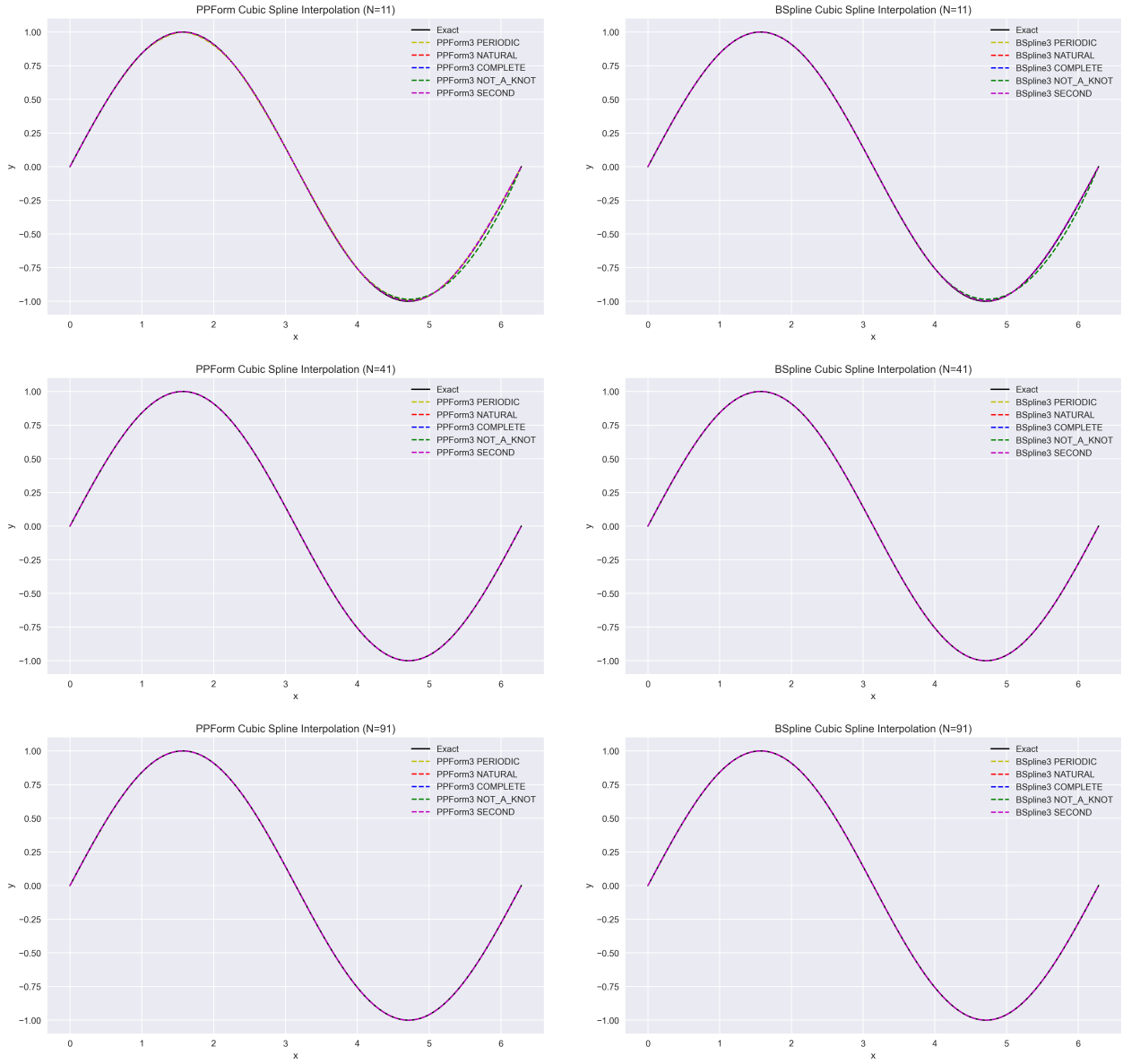

Fig. 16: Comparison of quadratic PPForm and BSpline interpolation for different N values

The error and convergence rate data for different methods and boundary conditions are shown in Table.6: It can be seen that under non-uniform node conditions, the fitting effect remains excellent, successfully achieving the fit.

Table. 6: Max errors and convergence rates for different situations

N	Method	BoundaryCondition	MaxError	ConvergenceRate
11	PPForm1	NA	0.13757	0
11	BSpline1	NA	0.13757	0
11	PPForm3	PERIODIC	0.0119971	0
11	PPForm3	NATURAL	0.00530427	0
11	PPForm3	COMPLETE	0.00479301	0
11	PPForm3	NOT_A_KNOT	0.0457951	0
11	PPForm3	SECOND	0.00530427	0
11	BSpline3	PERIODIC	0.00475759	0
11	BSpline3	NATURAL	0.00530427	0
11	BSpline3	COMPLETE	0.00479301	0
11	BSpline3	NOT_A_KNOT	0.0457951	0
11	BSpline3	SECOND	0.00530427	0
41	PPForm1	NA	0.00942869	-1.93348
41	BSpline1	NA	0.00942869	-1.93348
41	PPForm3	PERIODIC	0.000274479	-2.72492
41	PPForm3	NATURAL	1.56755e-005	-4.20125
41	PPForm3	COMPLETE	1.57222e-005	-4.126
41	PPForm3	NOT_A_KNOT	7.95351e-005	-4.58469
41	PPForm3	SECOND	1.56755e-005	-4.20125
41	BSpline3	PERIODIC	1.57215e-005	-4.12067
41	BSpline3	NATURAL	1.56755e-005	-4.20125
41	BSpline3	COMPLETE	1.57222e-005	-4.126
41	BSpline3	NOT_A_KNOT	7.95351e-005	-4.58469
41	BSpline3	SECOND	1.56755e-005	-4.20125
91	PPForm1	NA	0.00186516	-0.737482
91	BSpline1	NA	0.00186516	-0.737482
91	PPForm3	PERIODIC	2.37987e-005	-1.11288
91	PPForm3	NATURAL	6.0513e-007	-1.48115
91	PPForm3	COMPLETE	6.0513e-007	-1.4825
91	PPForm3	NOT_A_KNOT	1.47664e-006	-1.8143
91	PPForm3	SECOND	6.0513e-007	-1.48115
91	BSpline3	PERIODIC	6.0513e-007	-1.48248
91	BSpline3	NATURAL	6.0513e-007	-1.48115
91	BSpline3	COMPLETE	6.0513e-007	-1.4825
91	BSpline3	NOT_A_KNOT	1.47664e-006	-1.8143
91	BSpline3	SECOND	6.0513e-007	-1.48115

Intersection

Compile and run the program `intersect_test.cpp`. Its output includes the intersection information. The data stores in the directory `output\intersect\intersect_curve.csv`. Run the program `plotIntersect.py`

to plot the intersection curve r_4 with its self-intersection points calculated by python. The figures are stored in the directory `figure/intersect`.

The program tests 4 curves including r_1, r_2, r_3 in problem E and a special curve r_4 for intersection. The formula for r_4 is

$$\begin{cases} x = \frac{3\sin t}{2} + a\sin\frac{3t}{2}, \\ y = \frac{3\cos t}{2} - a\cos\frac{3t}{2}, \end{cases}$$

where a is a constant. a is 0.5 in the program and the interval is $[0, 4\pi]$.

The figure of r_4 is shown in Fig.17.

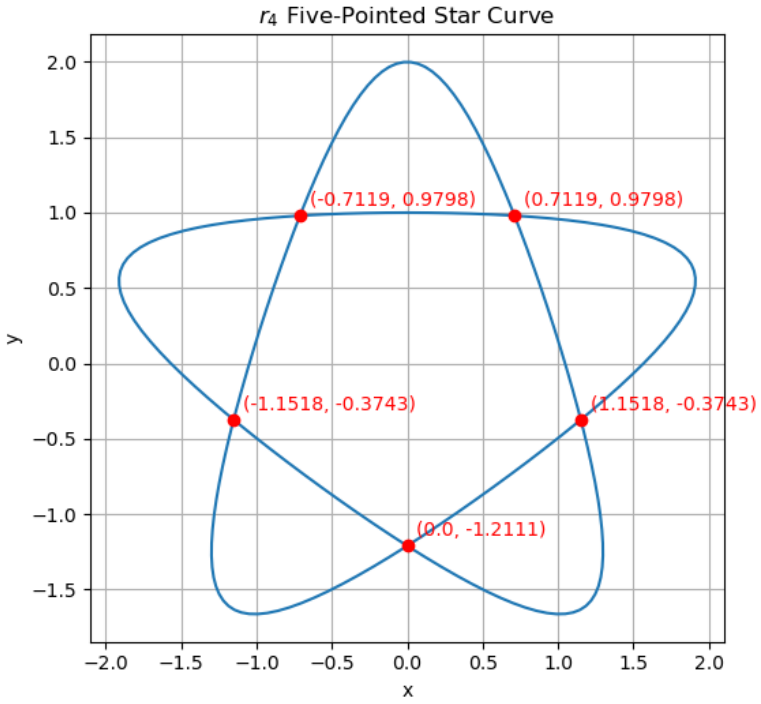

Fig. 17: The figure of five pointed star curve r_4

The result is as follows:

```
Curve r1:
---- Self-Intersection Summary ----
Curve is closed.
The curve is closed and does not self-intersect.
-----------------------------------
Curve r2:
---- Self-Intersection Summary ----
Curve is open.
The curve does not self-intersect.
-----------------------------------
Curve r3:
---- Self-Intersection Summary ----
Curve is closed.
The curve has self-intersections.
```

```
Number of Intersection Points: 1
Details of Intersections:
  Intersection 1:
    Segment A: (0.000430257 0.000670086     1) --> (-0.00497295 -0.00774419 0.999958)
    Segment B: (-0.00129077 0.00201025 0.999997) --> ( 0.00411236 -0.00640421 0.999971)
    Intersection Point: (-1.15087e-009 5.52261e-008 0.999996)
-----------------------------------
Curve r4:
---- Self-Intersection Summary ----
Curve is closed.
The curve has self-intersections.
Number of Intersection Points: 5
Details of Intersections:
  Intersection 1:
    Segment A: (0.702739 0.980356) --> ( 0.72358 0.979079)
    Segment B: (0.707697 0.990374) --> (0.715351 0.970945)
    Intersection Point: (0.711864 0.979797)
  Intersection 2:
    Segment A: ( 1.16613 -0.362438) --> ( 1.15005 -0.375713)
    Segment B: ( 1.14963 -0.365758) --> ( 1.15485 -0.385972)
    Intersection Point: ( 1.15182 -0.374253)
  Intersection 3:
    Segment A: (0.0134886 -1.20249) --> (-0.00414068 -1.21374)
    Segment B: (0.0153422 -1.22086) --> (-0.00225124 -1.20967)
    Intersection Point: (-1.53292e-006 -1.2111)
  Intersection 4:
    Segment A: ( -1.15296 -0.378645) --> ( -1.14771 -0.358402)
    Segment B: ( -1.13976 -0.384182) --> ( -1.1559 -0.370892)
    Intersection Point: ( -1.15182 -0.374249)
  Intersection 5:
    Segment A: (-0.712574 0.978009) --> (-0.704917 0.997398)
    Segment B: (-0.716028 0.979547) --> (-0.695151 0.980808)
    Intersection Point: (-0.711868 0.979799)
-----------------------------------
```

The results are consistent with the self-intersection conditions of the four curves. The calculation of the intersection points is also correct and within the computational error range.