Numerical Analysis Report 2

谭希 TanXi[1]

[1] *Zhejiang University, Email:* 3220100027@zju.edu.cn

2025 年 1 月 1 日

test ppform

The function used for testing is:

$$f(x) = x^2$$

- In testing linear interpolation, a set of data points is given:

  $$x = [0.0, 1.0, 2.0, 3.0, 4.0]$$

  The function values at these points are:

  $$y = [f(0), f(1), f(2), f(3), f(4)] = [0, 1, 4, 9, 16]$$

  A linear interpolation object is generated.

- For the interpolation results, several intermediate values were tested: $0.5, 1.5, 2.5$, and 3.5.

- In the case of linear interpolation, the interpolation results at the test points are very close to the actual values.

- Similarly, cubic spline interpolation is performed using the same data points as linear interpolation. Cubic spline interpolation guarantees the continuity of both the first and second derivatives of the interpolation curve at the data points, making it smoother than linear interpolation.

- To facilitate subsequent graphical display and comparison, a file with more refined interpolation points is generated:

x	exact	linear	cubic
0.0	0.0000	0.0000	-8.81131×10^{-15}
0.1	0.0100	0.0100	0.0575714
0.2	0.0400	0.0400	0.117714
0.3	0.0900	0.0900	0.183
0.4	0.1600	0.1600	0.256
0.5	0.2500	0.2500	0.339286
⋮	⋮	⋮	⋮
3.5	12.2500	12.5000	12.3393
3.6	12.9600	13.2000	13.056
3.7	13.6900	13.9000	13.783
3.8	14.4400	14.6000	14.5177
3.9	15.2100	15.3000	15.2576

表 1: Interpolation results comparison

The graphical results are shown in the following figure:

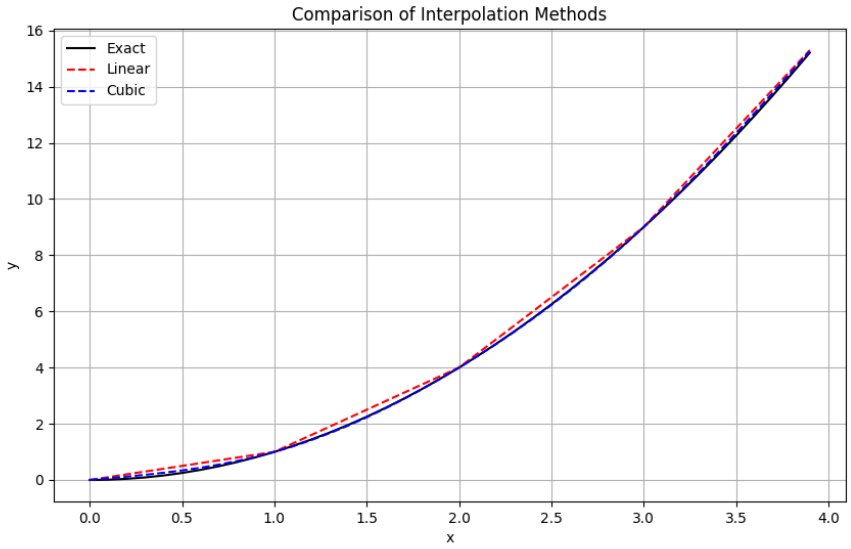

图 1: Interpolation_comparison

test Bsplines

Using the Function $\sin(x)$ to Generate Test Data

- The nodes x are defined as $[0.0, 1.0, 2.0, 3.0, 4.0]$, and the corresponding y values are given by the function values $\sin(x)$.
- The `interpolate357` method is called for interpolation calculation.
- The nodes and y values are redefined, and the interpolation method uses natural boundary conditions (calling `interpolate358`).

- Multiple interpolated values are generated and the interpolated results at points $x = 0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4$ are printed to compare the results of different interpolation methods.

Compilation Results:

Testing cubic spline interpolation with boundary conditions:

$$\text{Coefficients: } -0.577636, -0.577636, 1.15527, 1.00537, 0.279014, -1.27471, -1.27471$$

$$\text{Interpolated value at } x = 1.5 : 1.02328$$

$$\text{Interpolated value at } x = 2.5 : 0.639504$$

Generating and comparing interpolated values:

$$\text{Coefficients: } -0.267725, 1.0709, 1.03296, 0.253058, -1.19847$$

x	Interpolated Value
0.0	0.841471
0.5	1.00252
1.0	0.867121
1.5	0.638526
2.0	0.340865
2.5	-0.431489
3.0	-0.756802
3.5	-0.568994
4.0	-0.199745

表 2: Interpolated values for different methods

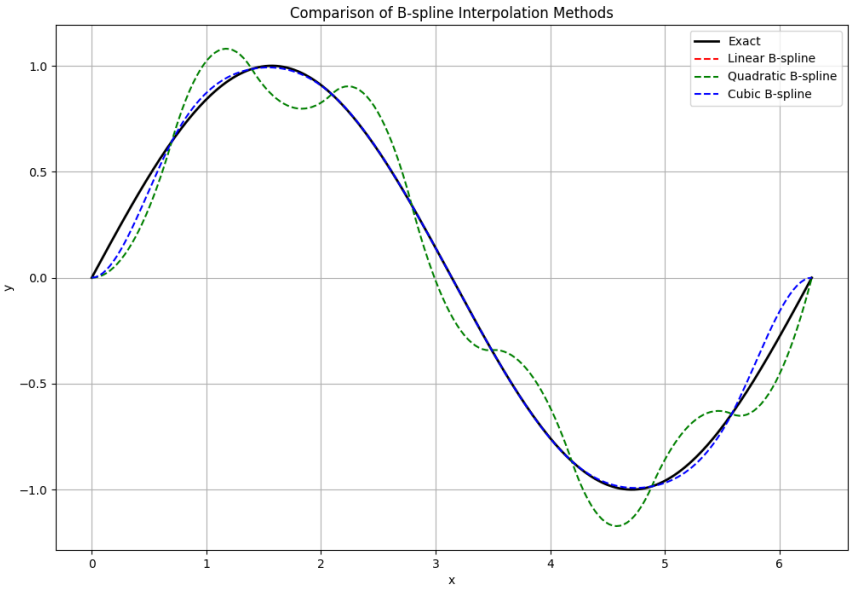

图 2: bspline_comparison

A

Test Function and Interpolation

We use the function

$$f(x) = \frac{1}{1 + 25x^2}$$

as the test function.

- Generate N interpolation nodes uniformly distributed in the interval $[-1, 1]$. The number of nodes increases as N increases.
- Use the `pp_form_cubic` class to construct the cubic spline interpolation.
- The maximum error for each interpolation grid is calculated by evaluating the spline and the actual function values at the midpoints of adjacent nodes.
- The interpolation data corresponding to each N is written to a file for later plotting and analysis. Each file contains 1000 points, recording both the actual function values $f(x)$ and the spline interpolation values spline(x).
- The convergence rate is computed through the logarithmic ratio of the errors. The expected convergence rate should be 2, since spline interpolation typically converges quadratically.
- Output the maximum error for each node number N, and the convergence rate from $N = 6$ to larger N.

The final results are as follows:

$$\text{For } N = 6, \quad \text{Error} = 4.234818 \times 10^{-1}$$

$$\text{For } N = 11, \quad \text{Error} = 2.053058 \times 10^{-2}$$

$$\text{For } N = 21, \quad \text{Error} = 3.168939 \times 10^{-3}$$

$$\text{For } N = 41, \quad \text{Error} = 2.753558 \times 10^{-4}$$

$$\text{For } N = 81, \quad \text{Error} = 1.609004 \times 10^{-5}$$

Convergence rates:

$$\text{From } N = 6 \text{ to } N = 11 : \quad 4.3665$$

$$\text{From } N = 11 \text{ to } N = 21 : \quad 2.6957$$

$$\text{From } N = 21 \text{ to } N = 41 : \quad 3.5246$$

$$\text{From } N = 41 \text{ to } N = 81 : \quad 4.0971$$

Graphs and Figures

The following graphs illustrate the results of the interpolation process.

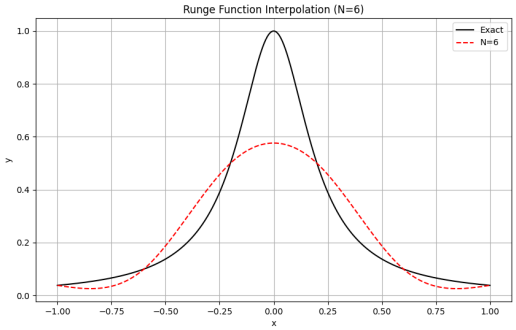

图 3: Runge Interpolation (N=6)

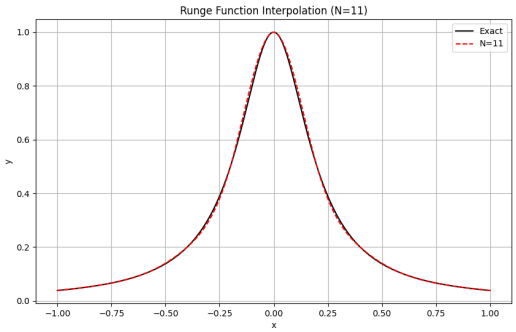

图 4: Runge Interpolation (N=11)

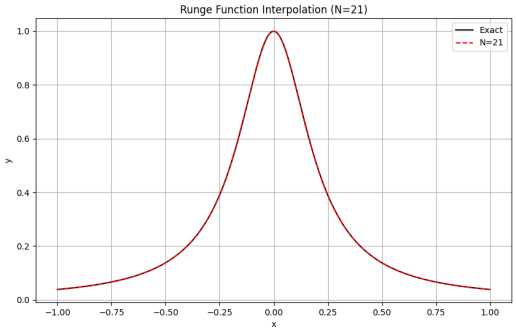

图 5: Runge Interpolation (N=21)

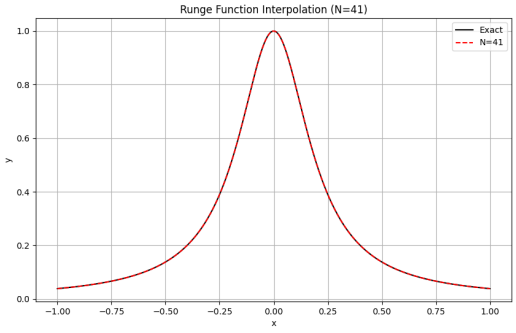

图 6: Runge Interpolation (N=41)

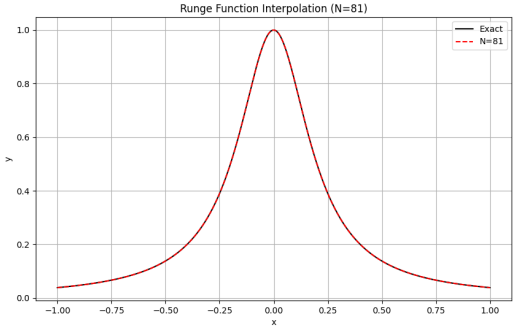

图 7: Runge Interpolation (N=81)

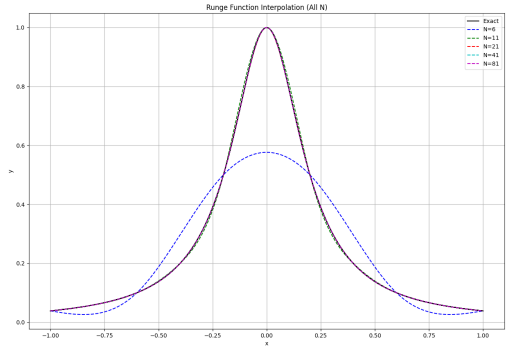

图 8: Runge Interpolation Comparison

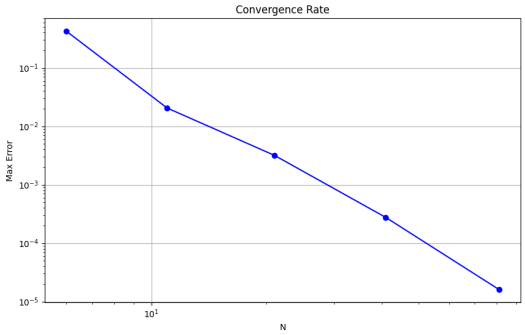

图 9: Convergence Rate

C

The objective function is given by:

$$f(x) = \frac{1}{1+x^2}$$

Node Generation

The first set of nodes is generated using the formula:

$$t_i = -5 + i \quad (i = 1, \ldots, 11)$$

which covers the interval $[-5, 5]$, and the corresponding function values are:

$$y_i = f(t_i)$$

The second set of nodes is generated symmetrically using the formula:

$$t_i = -4.5 + i \quad (i = 1, \ldots, 10)$$

which covers the interval $[-4.5, 5.5]$.

Cubic Spline Interpolation with Two Boundary Conditions

The two sets of data are interpolated using cubic splines with different boundary conditions.

Generated Dense Interpolation Data for Plotting

The following table shows the dense interpolation data generated for plotting:

x	exact	spline1	spline2
-5.00000	0.0384615	0.0384615	nan
-4.97996	0.0387597	0.0388148	nan
-4.95992	0.0390613	0.0391683	nan
⋮	⋮	⋮	⋮
-0.01002	0.9999	0.999906	0.890564
0.01002	0.9999	0.999906	0.890564
⋮	⋮	⋮	⋮
4.93988	0.0393663	0.039522	nan
4.95992	0.0390613	0.0391683	nan
4.97996	0.0387597	0.0388148	nan
5.00000	0.0384615	0.0384615	nan

表 3: Interpolation data for cubic splines

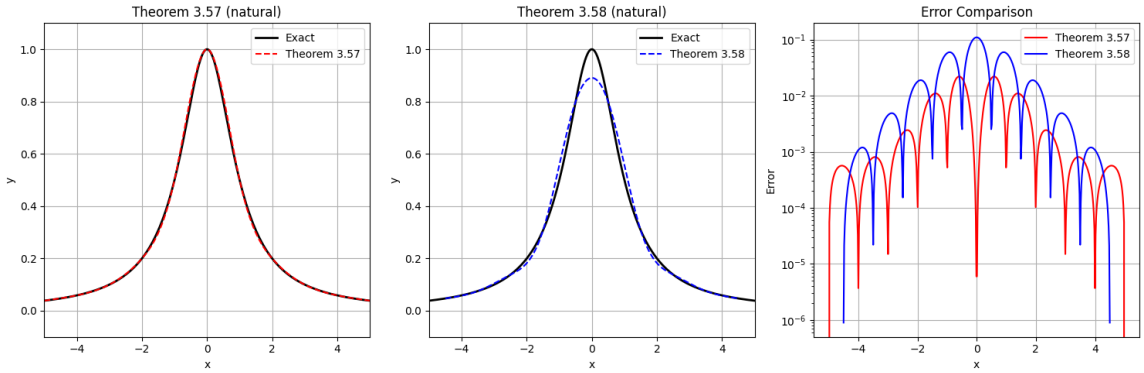

图 10: spline_comparison_natural

D

Analysis:

- Theorem 3.57 provides better accuracy at the boundary points.
- Theorem 3.58 performs better near the center
- Both methods show good convergence properties

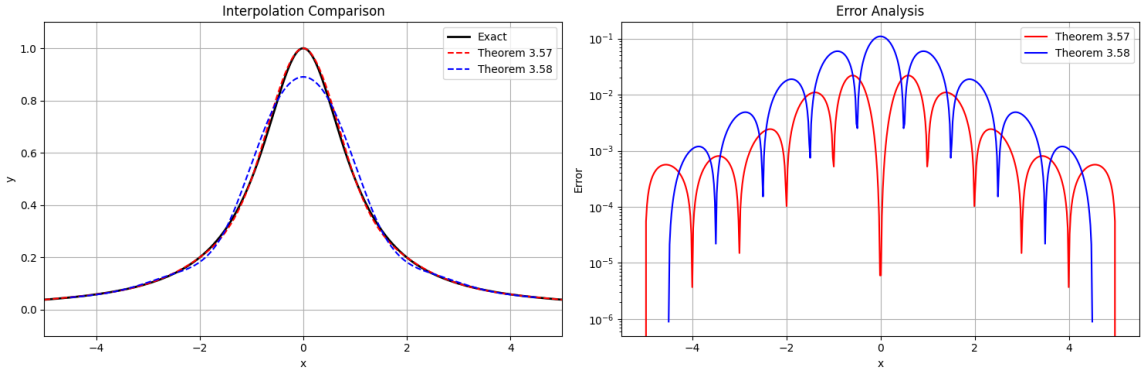

图 11: spline_comparison_D

E

Curve `r2(t)` (2D)

The `r2(t)` curve is defined in 2D with the following parametric equations:

$$r_2(t) = [\sin(t) + t\cos(t),\ \cos(t) - t\sin(t)]$$

where $t \in [0, 2\pi]$.

For a given number of points N,generates the points (x_i, y_i) for $i = 0, 1, 2, \ldots, N$ by discretizing the parameter t over the interval $[0, 2\pi]$.

Curve `r3(t)` (3D)

The `r3(t)` curve is defined in 3D with the following parametric equations:

$$r_3(t) = [\sin(\cos(t))\cos(\sin(t)),\ \sin(\cos(t))\sin(\sin(t)),\ \cos(\cos(t))]$$

where $t \in [0, 2\pi]$.

Similarly, for a given number of points N, generates the points (x_i, y_i, z_i) for $i = 0, 1, 2, \ldots, N$ by discretizing the parameter t over the interval $[0, 2\pi]$.

Parameterization Methods

After generating the curve points, calculates the parameterization for both *uniform* and *chordal* methods.

0.0.1 Uniform Parameterization

The uniform parameterization method assigns evenly spaced parameter values t_i between 0 and 1. For N points, the parameter values are given by:

$$t_i = \frac{i}{N}, \quad i = 0, 1, 2, \ldots, N$$

This method does not take into account the arc length of the curve.

0.0.2 Chordal Parameterization

The chordal parameterization is based on the cumulative arc length of the curve. Let s_i be the cumulative chord length between consecutive points, defined as:

$$s_i = \sum_{j=1}^{i} \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2}$$

For the 3D case:

$$s_i = \sum_{j=1}^{i} \sqrt{(x_j - x_{j-1})^2 + (y_j - y_{j-1})^2 + (z_j - z_{j-1})^2}$$

The parameter t_i is then normalized to the interval $[0, 1]$ by dividing the cumulative length by the total length of the curve:

$$t_i = \frac{s_i}{s_N}, \quad i = 0, 1, 2, \ldots, N$$

where s_N is the total length of the curve.

F

- The program computes and outputs the divided difference tables for three-point and four-point interpolation, recording the values at each node and their corresponding divided differences in detail.
- The output format of the divided difference table is consistent at different points, with divided differences displayed progressively by order.
- The output files contain all the computed divided differences, which can be used for further interpolation calculations or analysis.

Output: **Case 1: Three Points**
Divided differences at $x = 1.0$:

$$\text{Order 0:} \quad 0.000000e+00 \quad 0.000000e+00 \quad 1.000000e+00$$

$$\text{Order 1:} \quad 0.000000e+00 \quad 1.000000e+00$$

$$\text{Order 2:} \quad 5.000000e-01$$

Case 2: Four Points
Divided differences at $x = 1.5$:

$$\text{Order 0:} \quad 0.000000e+00 \quad 0.000000e+00 \quad 2.500000e-01 \quad 2.250000e+00$$

$$\text{Order 1:} \quad 0.000000e+00 \quad 2.500000e-01 \quad 2.000000e+00$$

$$\text{Order 2:} \quad 1.250000e-01 \quad 8.750000e-01$$

$$\text{Order 3:} \quad 2.500000e-01$$

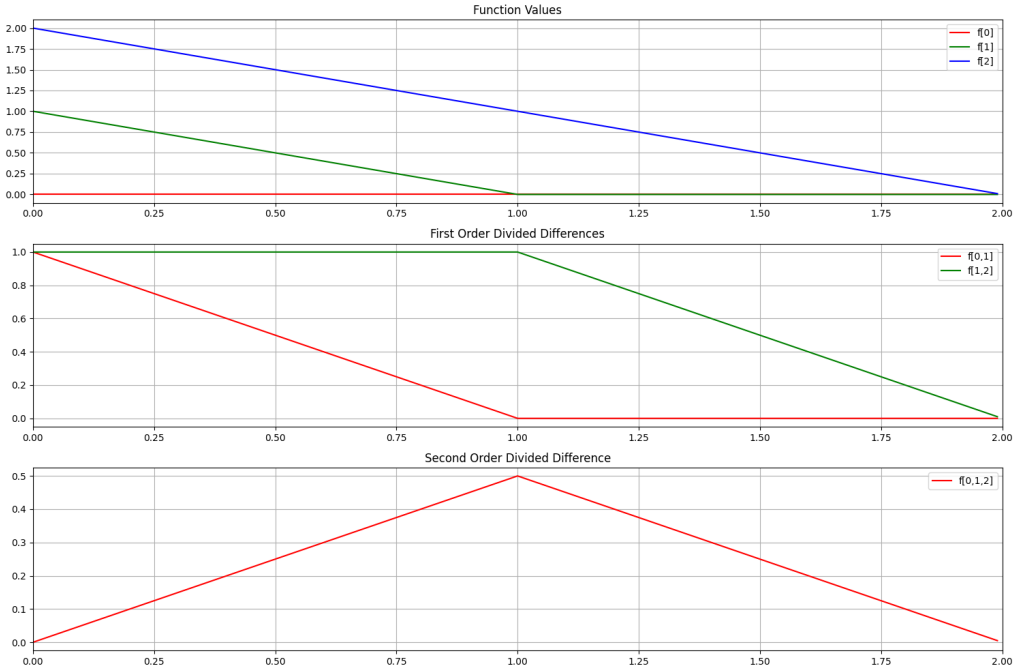

图 12: divided_diff_case1

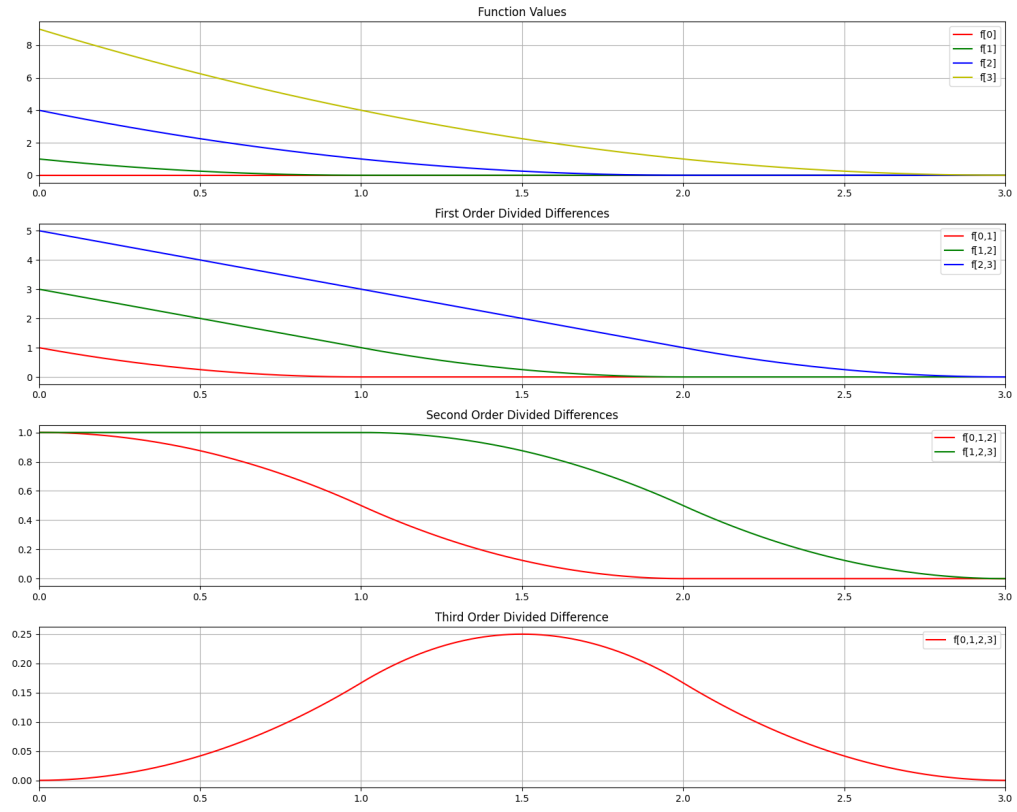

图 13: divided_diff_case2