

Homework4

袁健 23307110123

October 18, 2025

1 Problem1: Interpolation

1.1 problem description

Newton interpolation of

(i) 10 equal spacing points of $\cos(x)$ within $[0, \pi]$,

(ii) 10 equal spacing points $\frac{1}{1+25x^2}$ within $[-1, 1]$.

Compare the results with the cubic spline interpolation.

1.2 algorithm description

(1) Newton interpolation:

Given data points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, the Newton interpolation polynomial is constructed using divided differences. The polynomial $P(x)$ is given by:

$$P(x) = f[x_0] + f[x_0, x_1](x - x_0) + \dots + f[x_0, x_1, \dots, x_n](x - x_0)(x - x_1) \dots (x - x_{n-1}) \quad (1)$$

where $f[x_i, x_{i+1}, \dots, x_j]$ are the divided differences. can be computed recursively as:

$$f[x_0] = y_0, \quad f[x_0, x_1, \dots, x_i] = \frac{f[x_1, \dots, x_i] - f[x_0, \dots, x_{i-1}]}{x_i - x_0}, \quad (i = 1, 2, \dots, n) \quad (2)$$

(2) Cubic Spline Interpolation:

Given data points $(x_0, y_0), (x_1, y_1), \dots, (x_n, y_n)$, cubic spline interpolation compute n cubic polynomials $f_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$ for each interval $[x_i, x_{i+1}]$, $i = 0, 1, 2, \dots, n-1$.

The function $f_i(x)$ must satisfy the following conditions:

1. $f_i(x_i) = y_i; f_i(x_{i+1}) = y_{i+1}$, for $i = 0, 1, \dots, n-1$ (giving $2n$ equations).
2. $f'_i(x_{i+1}) = f'_{i+1}(x_{i+1})$ for $i = 0, 1, \dots, n-2$. (giving $n-1$ equations).
3. $f''_i(x_{i+1}) = f''_{i+1}(x_{i+1})$ for $i = 0, 1, \dots, n-2$. (giving $n-1$ equations).

4. $f_0''(x_0) = f_{n-1}''(x_n) = 0$ (giving 2 equations).so there are $4n$ equations in total.

Let $h_i = x_{i+1} - x_i$. The cubic polynomial for the interval $[x_i, x_{i+1}]$ can be written as:

$$f_i(x) = \frac{f''(x_i)}{6h_i}(x_{i+1}-x)^3 + \frac{f''(x_{i+1})}{6h_i}(x-x_i)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{f''(x_{i+1})h_i}{6} \right)(x-x_i) + \left(\frac{y_i}{h_i} - \frac{f''(x_i)h_i}{6} \right)(x_{i+1}-x) \quad (3)$$

The continuity of the first derivative $f'_{i-1}(x_i) = f'_i(x_i)$ leads to a system of linear equations for the unknown second derivatives $f''(x_i)$:

$$\frac{h_{i-1}}{6}f''(x_{i-1}) + \frac{h_{i-1} + h_i}{3}f''(x_i) + \frac{h_i}{6}f''(x_{i+1}) = \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}} \quad (4)$$

for $i = 1, 2, \dots, n-1$. This gives $n-1$ equations for $n+1$ unknown second derivatives $f''(x_i)$. Plus 2 natural conditions $f_0''(x_0) = f_{n-1}''(x_n) = 0$, and it's able to be solved in a tridiagonal matrix with Thomas algorithm, which I employed in my code, this algorithm is similar to gaussian elimination, forward elimination to get upper triangular matrix plus backward solving, but only acting on 3 diagonal elements every step, with a time complexity of $O(n)$.

1.3 output

run problem1.py

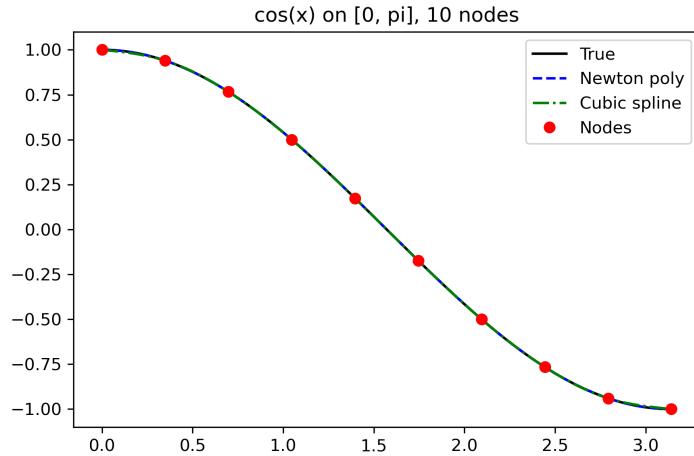


Figure 1: $\cos(x)$ interpolation

The Newton interpolation method agree well with Cubic spline method here, both in good agreement with the true value of $\cos(x)$, this is because $\cos(x)$ can be expanded in power series $\sum_{k=0}^{\infty} \frac{(-)^k}{(2k)!} x^{2k}$, and using the first low order terms can describe this function precisely(because high order terms can be small enough to be ignored at 0 or π , $\lim_{k \rightarrow \infty} \frac{\pi^k}{(k)!} = 0$). So the Newton polynomial didn't appear Runge error at boundary.

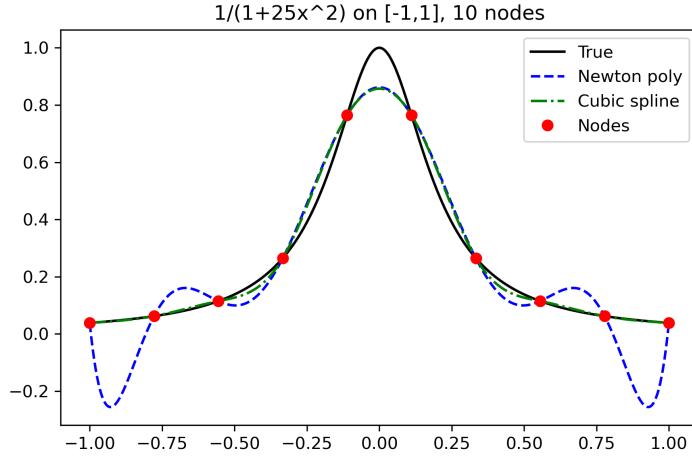


Figure 2: $\frac{1}{1+25x^2}$ interpolation

In this case the cubic spline method achieves a better interpolation than Newton method, because cubic spline guarantee the smooth in each interval while Newton method appears Runge error at two ends. This is obvious when consider $\frac{1}{1+25x^2} = \sum_{k=0}^{\infty} (-1)^k (25x^2)^k$, the high order terms cannot be ignored when x approaches ± 1 .

2 Problem2: Least-square fit

2.1 problem description

The table below gives the temperature T along a metal rod whose ends are kept at fixed constant temperatures. The temperature T is a function of the distance x along the rod.

x (cm)	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0
T (°C)	14.6	18.5	36.6	30.8	59.2	60.1	62.2	79.4	99.9

- (i) Compute a least-squares, straight-line fit to these data using $T(x) = ax + b$
- (ii) Compute a least-squares, parabolic-line fit to these data using $T(x) = ax^2 + bx + c$

2.2 algorithm description

For straight-line fit , target function is

$$T(a, b) = \sum_i [y_i - (ax_i + b)]^2 \quad (5)$$

with $\frac{\partial T}{\partial a} = 0$, $\frac{\partial T}{\partial b} = 0$, we have:

$$\begin{cases} a \sum_i x_i^2 + b \sum_i x_i = \sum_i x_i y_i \\ a \sum_i x_i + b \sum_i 1 = \sum_i y_i \end{cases} \quad (6)$$

consider it as a 2×2 matrix $Ax = b$, I use gaussian elimination method to get $x = [a, b]^T$.

For parabolic-line fit, target function is:

$$T(a, b, c) = \sum_i [y_i - (ax_i^2 + bx_i + c)]^2 \quad (7)$$

with $\frac{\partial T}{\partial a} = 0$, $\frac{\partial T}{\partial b} = 0$, $\frac{\partial T}{\partial c} = 0$, we have:

$$\begin{cases} a \sum_i x_i^4 + b \sum_i x_i^3 + c \sum_i x_i^2 = \sum_i x_i^2 y_i \\ a \sum_i x_i^3 + b \sum_i x_i^2 + c \sum_i x_i = \sum_i x_i y_i \\ a \sum_i x_i^2 + b \sum_i x_i + c \sum_i 1 = \sum_i y_i \end{cases} \quad (8)$$

Simililarly, consider it as a 3×3 matrix $Ax = b$, and use gaussian elimination method to get $x = [a, b, c]^T$.

2.3 output

run problem2.py

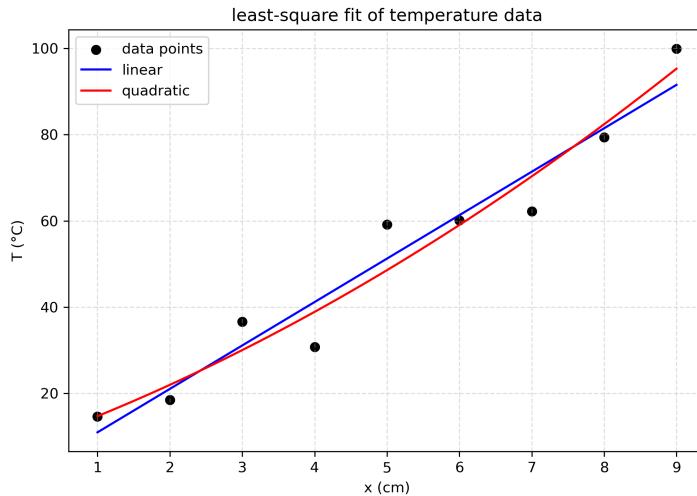


Figure 3: Least-square fit results

the fitting results show that the parabolic fit is better than linear fit, which can be seen from the Figure 4 below, linear fit's RMSE=6.5063, $R^2=0.9411$; parabolic fit's RMSE=6.066, $R^2=0.9488$.

- $T(x) = ax + b$
 $a = 10.073333, b = 0.888889$
 $RMSE = 6.506062, R^2 = 0.941113$

- $T(x) = ax^2 + bx + c$
 $a = 0.402165, b = 6.051688, c = 8.261905$
 $RMSE = 6.065795, R^2 = 0.948813$

x	y	ax+b	ax^2+bx+c
1.0	14.600	10.962	14.716
2.0	18.500	21.036	21.974
3.0	36.600	31.109	30.036
4.0	30.800	41.182	38.903
5.0	59.200	51.256	48.574
6.0	60.100	61.329	59.050
7.0	62.200	71.402	70.330
8.0	79.400	81.476	82.414
9.0	99.900	91.549	95.302

Figure 4: Residuals of least-square fit