Computional Homework3 Report

Student: 纪浩正, jihz2023@mail.sustech.edu.cn

1 Comparison of Chebyshev Interpolation and Cubic Splines for the Gaussian Function e^{-9x^2} on $[-3, 3]$

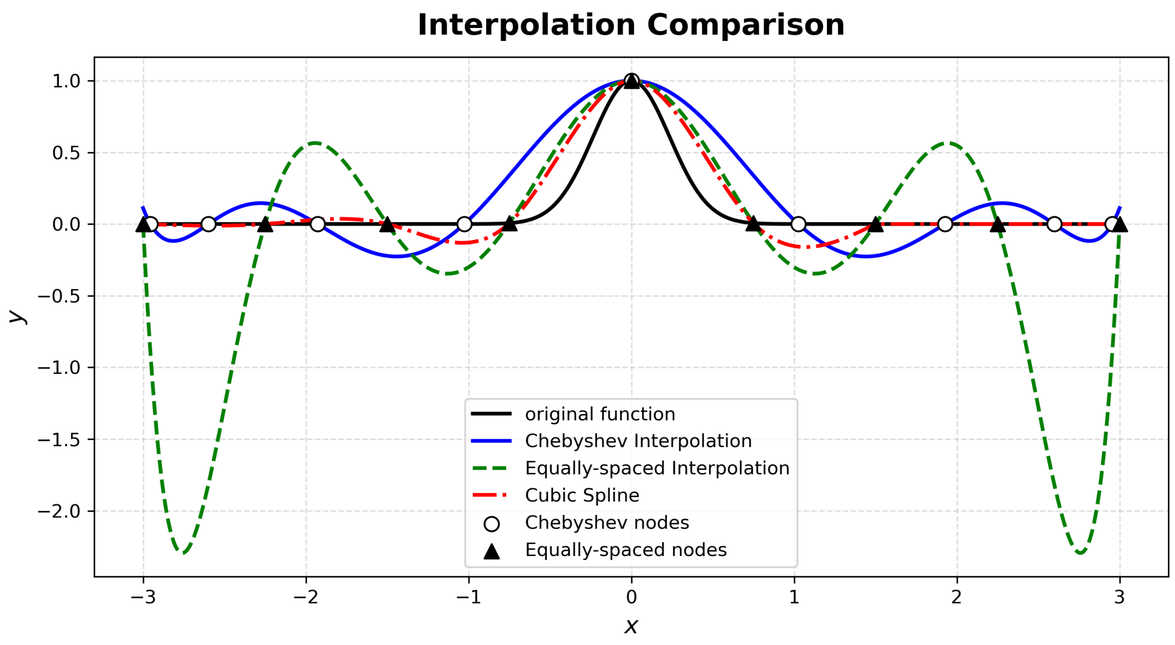

Figure 1 Interpolation Comparison

1.1 Comparative Analysis

The interpolation comparison reveals distinct characteristics of both methods when applied to the Gaussian function $f(x) = e^{-9x^2}$ on the interval $[-3, 3]$.

1.1.1 Performance Comparison

Behavior Near Boundaries

- **Chebyshev**: Node clustering near endpoints ($x = \pm 3$) effectively captures the rapid decay of the Gaussian function, avoiding Runge phenomenon
- **Cubic Splines**: Uniform node distribution with natural boundary conditions provides smooth behavior but may require more nodes for equivalent accuracy near boundaries

1.1.2 Theoretical Advantages

Chebyshev Interpolation	**Cubic Splines**
• Optimal polynomial approximation	• C^2 continuity everywhere
• Exponential convergence for analytic functions	• Local control and stability
• Minimal maximum error principle	• Natural curvature minimization
• Global polynomial representation	• Piecewise construction

1.1.3 Suitability Analysis

For the Gaussian Function e^{-9x^2}:

- **Chebyshev Excels**: The function is infinitely differentiable (analytic), allowing Chebyshev to achieve spectral accuracy with exponential convergence
- **Cubic Splines Excel**: For applications requiring guaranteed smoothness and local modifications without global impact

General Guidelines:

- **Choose Chebyshev when**: Function is smooth/analytic, global approximation is needed, highest accuracy is priority
- **Choose Cubic Splines when**: C^2 continuity is required, local control is important, or dealing with non-smooth functions

1.1.4 Conclusion

For the specific case of approximating e^{-9x^2} on $[-3, 3]$, both methods perform excellently but serve different purposes: - Chebyshev interpolation provides superior global accuracy with fewer nodes due to optimal node placement - Cubic splines offer superior smoothness guarantees and numerical stability for practical applications requiring local control

The choice depends on whether global optimality (Chebyshev) or local smoothness and stability (cubic splines) is the primary concern.

1.2 Chebyshev Interpolation

The error bound for polynomial interpolation is given by:

$$|f(x) - P_n(x)| \leq \frac{M_{n+1}}{(n+1)!} \max_{x \in [a,b]} \left| \prod_{i=0}^{n} (x - x_i) \right|$$

where $M_{n+1} = \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)|$ and $\{x_0, x_1, \dots, x_n\}$ are the interpolation nodes.

To minimize the interpolation error, we can only control the choice of interpolation nodes $\{x_0, x_1, \dots, x_n\}$. By applying a linear transformation to map the interval $[a, b]$ to the standard interval $[-1, 1]$, we can derive a universal approach.

Using the linear transformation:

$$\tilde{x} = 2\frac{x-a}{b-a} - 1$$

```
## convert chebyshev point into interpolation point
def CR(m):
   return np.cos((2 * m + 1) * np.pi / (2 * N))
```

the error bound becomes:

$$|f(x) - P_n(x)| \leq \frac{M_{n+1}}{(n+1)!} \left(\frac{b-a}{2}\right)^{n+1} \max_{\tilde{x} \in [-1,1]} \left| \prod_{i=0}^{n} (\tilde{x} - \tilde{x}_i) \right|$$

To minimize this error bound, we need to minimize:

$$\max_{\tilde{x} \in [-1,1]} \left| \prod_{i=0}^{n} (\tilde{x} - \tilde{x}_i) \right|$$

The solution to this optimization problem is provided by Chebyshev polynomials. The optimal nodes are the Chebyshev points:

$$\tilde{x}_i = \cos\left(\frac{2i+1}{2(n+1)}\pi\right), \quad i = 0, 1, \dots, n$$

These nodes minimize the maximum absolute value of the nodal polynomial on $[-1, 1]$, leading to:

$$\max_{\tilde{x} \in [-1,1]} \left| \prod_{i=0}^{n} (\tilde{x} - \tilde{x}_i) \right| = 2^{-n}$$

1.2.1 Finding the Optimal Interpolation Points

The optimization problem we need to solve is:

$$\min_{\{x_0,x_1,\ldots,x_n\}} \max_{x\in[-1,1]} \left| \prod_{i=0}^{n} (x - x_i) \right|$$

This is equivalent to finding the monic polynomial of degree $n+1$ that deviates least from zero on $[-1, 1]$.

Chebyshev Polynomials The Chebyshev polynomial of the first kind of degree n is defined as:

$$T_n(x) = \cos(n \arccos(x)), \quad x \in [-1, 1]$$

Key properties of Chebyshev polynomials:

- $T_n(x)$ oscillates between -1 and 1 on $[-1, 1]$
- $T_n(x)$ has n zeros in $(-1, 1)$ at $x_k = \cos\left(\frac{(2k+1)\pi}{2n}\right)$ for $k = 0, 1, \ldots, n-1$
- The leading coefficient of $T_n(x)$ is 2^{n-1} for $n \geq 1$

Detailed Construction of Chebyshev Polynomials *Trigonometric Identity and Recurrence Relation:* Starting with the trigonometric identity:

$$\cos((n+1)\theta) + \cos((n-1)\theta) = 2\cos(n\theta)\cos(\theta)$$

Let $\theta = \cos^{-1} x$, so $\cos(\theta) = x$. Substituting:

$$\cos\big((n+1)\cos^{-1} x\big) + \cos\big((n-1)\cos^{-1} x\big) = 2\cos\big(n\cos^{-1} x\big)\cos\big(\cos^{-1} x\big)$$

Using the definition $T_n(x) = \cos\big(n\cos^{-1} x\big)$:

$$T_{n+1}(x) + T_{n-1}(x) = 2T_n(x) \cdot x$$

Rearranging gives the recurrence relation:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$$

Verification that Chebyshev Functions are Polynomials: Starting with $T_0(x) = 1$ and $T_1(x) = x$, we can generate:

$$T_2(x) = 2x \cdot x - 1 = 2x^2 - 1 \tag{1}$$
$$T_3(x) = 2x(2x^2 - 1) - x = 4x^3 - 3x \tag{2}$$
$$T_4(x) = 2x(4x^3 - 3x) - (2x^2 - 1) = 8x^4 - 8x^2 + 1 \tag{3}$$

By induction, each $T_n(x)$ is indeed a polynomial of degree n with leading coefficient 2^{n-1} for $n \geq 1$.

Zeros of Chebyshev Polynomials: The zeros of $T_n(x) = \cos\big(n \cos^{-1} x\big)$ occur when:

$$n \cos^{-1} x = \frac{(2k+1)\pi}{2}, \quad k = 0, 1, \ldots, n-1$$

Solving for x:

$$x_k = \cos\left(\frac{(2k+1)\pi}{2n}\right), \quad k = 0, 1, \ldots, n-1$$

The Optimal Solution Consider the Chebyshev polynomial $T_{n+1}(x)$, which has degree $n+1$ and leading coefficient 2^n. The monic version is:

$$\frac{T_{n+1}(x)}{2^n} = x^{n+1} + \text{lower order terms}$$

The roots of $T_{n+1}(x)$ are:

$$x_i = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right), \quad i = 1, \ldots, n+1$$

```
## calculate the chebyshev point
def CR(m):
   return np.cos((2 * m + 1) * np.pi / (2 * N))
```

Relationship Between Nodal Polynomial and Chebyshev Polynomials: Since $T_{n+1}(x)$ has degree $n+1$ and leading coefficient 2^n, we can write:

$$T_{n+1}(x) = 2^n \prod_{i=1}^{n+1} (x - x_i)$$

Therefore:

$$\prod_{i=1}^{n+1}(x - x_i) = \frac{T_{n+1}(x)}{2^n}$$

Since $T_{n+1}(x) = \cos\big((n+1)\cos^{-1} x\big)$ oscillates between -1 and 1 on $[-1, 1]$, we have:

$$\max_{x\in[-1,1]} \left|\frac{T_{n+1}(x)}{2^n}\right| = \frac{1}{2^n}$$

The Equioscillation Property The optimal polynomial $\frac{T_{n+1}(x)}{2^n}$ achieves its maximum absolute value $1/2^n$ at exactly $n+1$ points in $[-1, 1]$:

$$x = \cos\left(\frac{k\pi}{n+1}\right), \quad k = 1, \ldots, n+1$$

with alternating signs $(-1)^k \cdot \frac{1}{2^n}$.

1.2.2 Final Result

Therefore, the optimal interpolation nodes are the Chebyshev points:

$$\tilde{x_i} = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right), \quad i = 1, \ldots, n+1$$

and the minimum possible value is:

$$\min_{\{x_1,\ldots,x_{n+1}\}} \max_{x\in[-1,1]} \left|\prod_{i=1}^{n+1}(x - x_i)\right| = \frac{1}{2^n}$$

Summary of the Connection

$$\boxed{\begin{aligned} \text{Optimal nodes:} \quad & x_i = \cos\left(\frac{(2i+1)\pi}{2(n+1)}\right) \\ \text{Nodal polynomial:} \quad & \prod_{i=0}^{n}(x - x_i) = \frac{T_{n+1}(x)}{2^n} \\ \text{Minimum monic polynomial:} \quad & \max_{x\in[-1,1]} \left|\prod_{i=1}^{n+1}(x - x_i)\right| = \frac{1}{2^n} \\ \text{Final error bound:} \quad & |f(x) - P_n(x)| \leq \frac{M_{n+1}}{(n+1)!} \cdot \frac{(b-a)^{n+1}}{2^{2n+1}} \end{aligned}}$$

1.2.3 Why Chebyshev Interpolation Achieves Superior Performance

From the final Chebyshev error bound, we can analyze why this method is so effective:

Error Bound Component Analysis The error bound consists of three factors:

- $\frac{M_{n+1}}{(n+1)!}$: Determined by the function's smoothness, cannot be changed by node selection
- $(b-a)^{n+1}$: Determined by the interval length, typically fixed
- $\frac{1}{2^{2n+1}}$: This is the key advantage of Chebyshev interpolation!

Fundamental Reason for Chebyshev Superiority In the interpolation error formula, the only term we can optimize is:

$$\max_{x\in[a,b]} \left| \prod_{i=0}^{n} (x - x_i) \right|$$

Chebyshev nodes achieve optimality through:

Mathematical Optimality:

$$\min_{\{x_0,\ldots,x_n\}} \max_{x\in[-1,1]} \left| \prod_{i=0}^{n} (x - x_i) \right| = \frac{1}{2^n}$$

This minimum value is **uniquely** achieved by Chebyshev nodes. Any other node selection produces a larger maximum value.

Geometric Intuition:

- Chebyshev nodes cluster more densely near interval endpoints
- This prevents the **Runge phenomenon** (oscillations near endpoints)

```python
## make chebyshev interpolation
cheby_point = np.array([CR(m) for m in range(N)])
x_cheby = cr2x(cheby_point, x0, x8)
y_c = f(x_cheby)
diff_cheby = diff_table(x_cheby, f(x_cheby))
cheby = make_newton_inter(diff_cheby, x_cheby)
y_cheby = cheby(x_line)
```

1.3 Cubic Spline Interpolation

Cubic spline interpolation provides a smooth interpolation method that maintains C^2 continuity while avoiding oscillatory behavior.

1.3.1 Algorithm Implementation

The implementation consists of several key phases:

Phase 1: Initialization and Setup

```
n = N - 1
xi = np.linspace(x0, x8, N)
yi = f(xi)
a = np.zeros(N)
b = np.zeros(N)
A = np.zeros(N)
B = np.zeros(N)
M = np.zeros(N)
def h(ii):
    return xi[ii + 1] - xi[ii]
```

Phase 2: Setup Tridiagonal System The tridiagonal system for natural cubic splines is:

$$(1 - a_i)M_{i-1} + 2M_i + a_i M_{i+1} = b_i$$

```
for ii in range(1, n):
    a[ii] = h(ii) / (h(ii) + h(ii - 1))
    b[ii] = (
        6
        / (h(ii) + h(ii - 1))
        * ((yi[ii + 1] - yi[ii]) / h(ii) - (yi[ii] - yi[ii - 1]) / h(ii - 1))
    )
```

Where: $a[i] = \frac{h_i}{h_{i-1}+h_i}$, $b[i] = \frac{6}{h_{i-1}+h_i}\left[\frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}\right]$

Thomas Algorithm Solution The original tridiagonal system is:

$$(1 - a_i)M_{i-1} + 2M_i + a_i M_{i+1} = b_i$$

Forward elimination transforms this to:

$$M_i = A_i M_{i+1} + B_i$$

Initial step ($i = 1$):

$$A_1 = \frac{-a_1}{2}, \quad B_1 = \frac{b_1}{2}$$

General step ($i = 2, 3, \ldots, n-1$):

$$A_i = \frac{-a_i}{2 + (1 - a_i)A_{i-1}}$$

$$B_i = \frac{b_i - (1 - a_i)B_{i-1}}{2 + (1 - a_i)A_{i-1}}$$

```
A[1] = -a[1] / 2
B[1] = b[1] / 2
for ii in range(2, n - 2):
    A[ii] = -a[ii] / (2 + (1 - a[ii]) * A[ii - 1])
    B[ii] = (b[ii] - (1 - a[ii]) * B[ii - 1]) / (2 + (1 - a[ii]) * A[ii - 1])
```

Phase 4: Back Substitution Compute the second derivatives M_i:

Final equation ($i = n - 1$):

$$M_{n-1} = \frac{b_{n-1} - (1 - a_{n-1})B_{n-2}}{2 + (1 - a_{n-1})A_{n-2}}$$

Back substitution ($i = n - 2, n - 3, \ldots, 1$):

$$M_i = A_i M_{i+1} + B_i$$

```
M[n - 1] = (b[n - 1] - (1 - a[n - 1]) * B[n - 2]) / (2 + (1 - a[n - 1]) * A[n - 2])
for ii in range(n - 2, 0, -1):
    M[ii] = A[ii] * M[ii + 1] + B[ii]
```

Phase 5: Spline Evaluation Function Construct the piecewise cubic polynomial using the computed second derivatives:

For $x \in [x_i, x_{i+1}]$, the cubic spline is:

$$S(x) = \frac{M_i(x_{i+1} - x)^3}{6h_i} + \frac{M_{i+1}(x - x_i)^3}{6h_i} + \left(y_i - \frac{M_i h_i^2}{6}\right)\frac{x_{i+1} - x}{h_i} + \left(y_{i+1} - \frac{M_{i+1}h_i^2}{6}\right)\frac{x - x_i}{h_i}$$

where $h_i = x_{i+1} - x_i$.

```
def cubic_spline(x):
    ii = np.searchsorted(xi, x) - 1
    y_out = (
        M[ii] * (xi[ii + 1] - x) ** 3 / (6 * h(ii))
        + M[ii + 1] * (x - xi[ii]) ** 3 / (6 * h(ii))
        + (yi[ii] - M[ii] * h(ii) ** 2 / 6) * (xi[ii + 1] - x) / h(ii)
        + (yi[ii + 1] - M[ii + 1] * h(ii) ** 2 / 6) * (x - xi[ii]) / h(ii)
    )
    return y_out
```