计算物理作业 4

杨远青　22300190015　CompPhys 24

2024 年 10 月 22 日

班会旧语新知：“物理学系就业面广。”

1　题目 1：牛顿插值法

1.1　题目描述

Newton interpolation of:

(1) 10 equal spacing points of $\cos(x)$ within $[0, \pi]$;

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

Compare the results with the cubic spline interpolation.

1.2　程序描述

发现等距节点在一些病态函数的插值中的确会出现 Runge 现象，故对有理函数 $\frac{1}{1+25x^2}$ 额外使用 Chebyshev 节点（第一类 Chebyshev 多项式的零点）与 Leja 节点进行对比研究。前者在区间 $[-1, 1]$ 上的分布为

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

用于插值 $(n-1)$ 阶多项式，它们同时也是单位圆上的等距点在 $[-1, 1]$ 上的投影，可以使得拉格朗日余项

$$f(x) - P_{n-1}(x) = \frac{f^{(n)}(\xi)}{n!}\prod_{i=1}^{n}(x - x_i)$$

中的 $\max_{x\in[-1,1]}\left|\prod_{i=1}^{n}(x - x_i)\right|$ 最小化，故最终在 $[-1, 1]$ 上的插值余项满足

$$|f(x) - P_{n-1}(x)| \leq \frac{1}{2^{n-1}n!}\max_{\xi\in[-1,1]}\left|f^{(n)}(\xi)\right|.$$

上述理论可以拓展到任意长度的区间 $[a, b]$，仅需使用相应的 Chebyshev 节点

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

在思考拓展题的舍入误差时，联想到牛顿插值法的数值稳定性可能的确受到节点顺序的影响。一篇满是奇怪符号和不等式的论文[1] 告诉我，可以使用 Lebesgue 常数来衡量重心形式拉格朗日插值的数值稳定性：

$$\Lambda\left(\mathbf{x}\right) := max_{x\in[a,b]}\left(\sum_{i=0}^{n}\prod_{k\neq i}^{n}\left|\frac{x-x_k}{x_i-x_k}\right|\right)$$

对于牛顿插值，该文给出了三类上界，公式与推导很催眠，我选择使用两个更病态的函数进行了数值实验，详见拓展题。使用的另一套节点是 Leja 节点，首先定一个初始节点（通常为峰值点），再按照满足如下条件增加节点，使得每个节点 “尽可能离之前的点远”

$$|x_{k+1}-x_0|\cdot|x_{k+1}-x_1|\cdot\cdots\cdot|x_{k+1}-x_k| = \max_{x\in[a,b]}\prod_{i=0}^{k}|x-x_i|,\quad k=0,1,\ldots,n-1$$

一篇看起来符号少些但逻辑不是很清楚的文章[2]告诉我，使用中心序的 Leja 节点可以使得牛顿插值法的数值稳定性更好，但实验结果好像不符（倒是和三次插值配合很好），可能是代码问题，也可能是优化方法并不普适。为了更好利用 `numpy` 库的向量化操作，本程序构建完牛顿差商表计算给定点值时，使用了 Horner's Rule，即迭代计算

$$\begin{aligned}P_n(x) &= f[x_0]+f[x_1,x_0](x-x_0)+f[x_2,x_1,x_0](x-x_0)(x-x_1)+\cdots\\ &\quad+f[x_n,x_{n-1},\ldots,x_0](x-x_0)(x-x_1)\ldots(x-x_{n-1})\\ &= f[x_0]+(x-x_0)\left(f[x_1,x_0]+(x-x_1)\left(f[x_2,x_1,x_0]+\cdots+(x-x_{n-1})f[x_n,x_{n-1},\ldots,x_0]\right)\right)\end{aligned}$$

将 $n(n+1)/2$ 次乘法降至 n 次。似乎还有一种古老的 Neville's Algorithm[3]，可以在局部点加速收敛。

在三次插值部分使用了 Thomas 算法来快速求解条状矩阵以提高数值稳定性，剩下的大段代码都是 GPT 的绘图美化工程，它真的很尽职。最终插值结果详见下文结果示例。请在本程序目录使用`python -u newton_cubic.py` 运行数值实验，需安装 `numpy` 和 `matplotlib` 库。

1.3 伪代码

Algorithm 1: Leja Node Generation

Input: $a(float)$, $b(float)$, $n(int)$, $m(int)$, $f(function)$

Output: $nodes(array, shape=n)$

$x(array, shape=m) \leftarrow$ discretize interval $[a,b]$ into m points $nodes(array, shape=n)$; `// Initialize array for Leja nodes`

nodes$[0] \leftarrow x[\arg\max(|f(x)|)]$; `// Select the point where` $|f(x)|$ `is maximum as the first node`

for $i \leftarrow 1$ **to** $n-1$ **do**

  $distances[j] \leftarrow \min(|x[j]-\text{nodes}[k]|)$ for $k=0,\ldots,i-1$ and $j=0,\ldots,m-1$; `// Compute minimum distance from each point to the current nodes`

  nodes$[i] \leftarrow x[\arg\max(distances)]$; `// Select the point with the maximum distance as the next node`

end

return $nodes$; `// Return the array of Leja nodes`

[1]Camargo, A.P. (2020). On the numerical stability of Newton' s formula for Lagrange interpolation. *Journal of Computational and Applied Mathematics*, 365.10.1016/j.cam.2019.112369

[2]Carnicer, J.M., Khiar, Y. & Peña, J.M. (2019). Central orderings for the Newton interpolation formula. *Bit Numer Math*, 59, 371–386.10.1007/s10543-018-00743-2

[3]详见 *Numerical Recipes* §3.1

Algorithm 2: Newton's Divided Difference Method

Input: $x(array, shape = n+1)$, $y(array, shape = n+1)$

Output: $coef(array, shape = n+1)$

$n \leftarrow \text{length}(y)$ - 1 ; // Determine the degree of the polynomial

$coef(array, shape = n+1) \leftarrow y$; // Initialize coefficients with y values

for $j \leftarrow 1$ **to** n **do**

  for $i \leftarrow n$ **to** j **do**

    $\text{coef}[i] \leftarrow \frac{\text{coef}[i]-\text{coef}[i-1]}{x[i]-x[i-j]}$; // Compute divided differences

  end

end

return $coef$; // Return the array of divided difference coefficients

Algorithm 3: Cubic Spline Interpolation

Input: $x(array, shape = n+1)$, $y(array, shape = n+1)$

Output: $a(array, shape = n)$, $b(array, shape = n)$, $c(array, shape = n+1)$, $d(array, shape = n)$

$h[i] \leftarrow x[i+1] - x[i]$ for $i = 0, \ldots, n-1$ $A[0,0] \leftarrow 1, A[n,n] \leftarrow 1$; // Set boundary conditions

for $i \leftarrow 1$ **to** $n-1$ **do**

  $A[i,i-1] \leftarrow h[i-1], A[i,i] \leftarrow 2(h[i-1]+h[i]), A[i,i+1] \leftarrow h[i]$; // Fill tridiagonal matrix

  $b[i] \leftarrow 3\left(\frac{y[i+1]-y[i]}{h[i]} - \frac{y[i]-y[i-1]}{h[i-1]}\right)$; // Compute right-hand side vector for spline system

end

$c \leftarrow \texttt{SolveTridiagonal}(A, b)$; // Solve for c coefficients using the tridiagonal matrix

for $i \leftarrow 0$ **to** $n-1$ **do**

  $b[i] \leftarrow \frac{y[i+1]-y[i]}{h[i]} - \frac{h[i]}{3}(2c[i]+c[i+1])$; $d[i] \leftarrow \frac{c[i+1]-c[i]}{3h[i]}$

end

$a \leftarrow y[0:n]$; // Assign a coefficients from y values

return a, b, c, d ; // Return the spline coefficients

Algorithm 4: Solve Tridiagonal System (Thomas Algorithm)

Input: $A(array, shape = (n+1, n+1))$, $b(array, shape = n+1)$

Output: $x(array, shape = n+1)$

$c'[0] \leftarrow \frac{A[0,1]}{A[0,0]}$, $d'[0] \leftarrow \frac{b[0]}{A[0,0]}$; // Initial forward sweep for c' and d'

for $i \leftarrow 1$ **to** $n-1$ **do**

  $denom \leftarrow A[i,i] - A[i,i-1] \cdot c'[i-1]$; // Calculate denominator for row i

  $c'[i] \leftarrow \frac{A[i,i+1]}{denom}$, $d'[i] \leftarrow \frac{b[i]-A[i,i-1]\cdot d'[i-1]}{denom}$; // Update c' and d' for current row

end

$d'[n] \leftarrow \frac{b[n]-A[n,n-1]\cdot d'[n-1]}{A[n,n]-A[n,n-1]\cdot c'[n-1]}$; $x[n] \leftarrow d'[n]$; // Final update for $d'[n]$

for $i \leftarrow n-1$ **to** 0 **do**

  $x[i] \leftarrow d'[i] - c'[i] \cdot x[i+1]$; // Back substitution step for $x[i]$

end

return x ; // Return the solution array x

1.4 结果示例

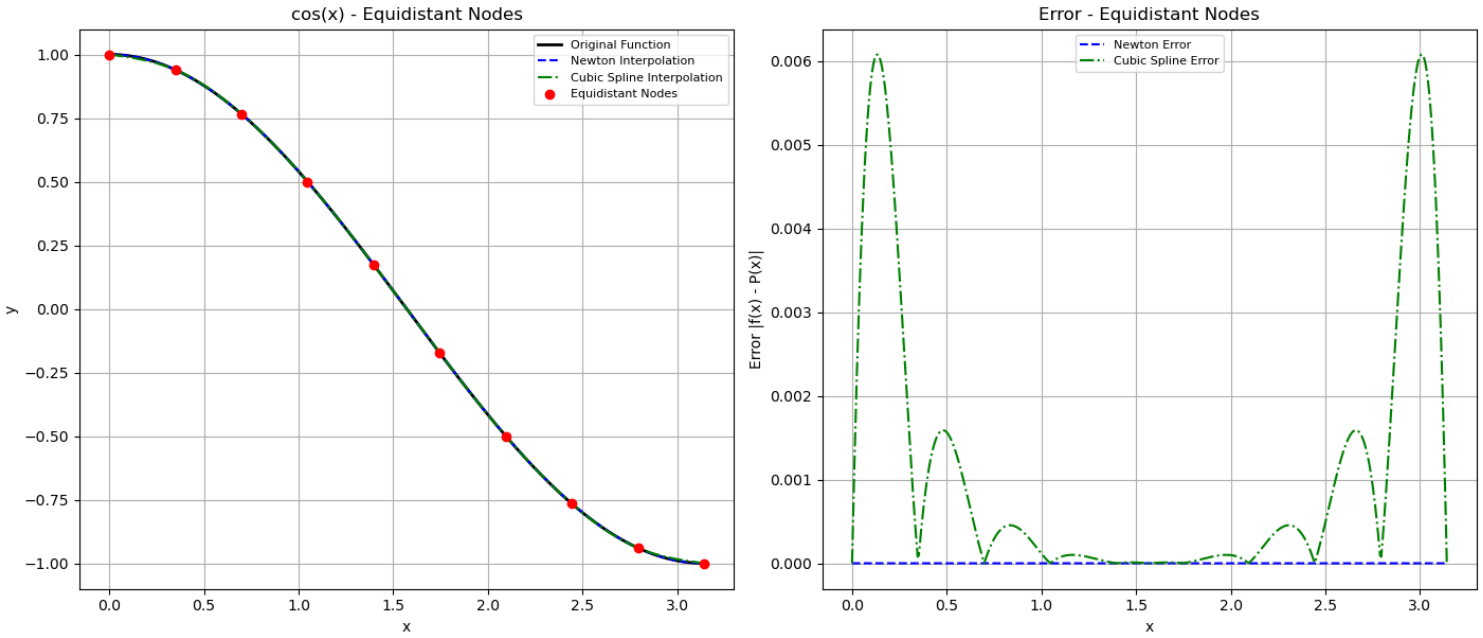


图 1: $cos(x)$ 插值效果对比

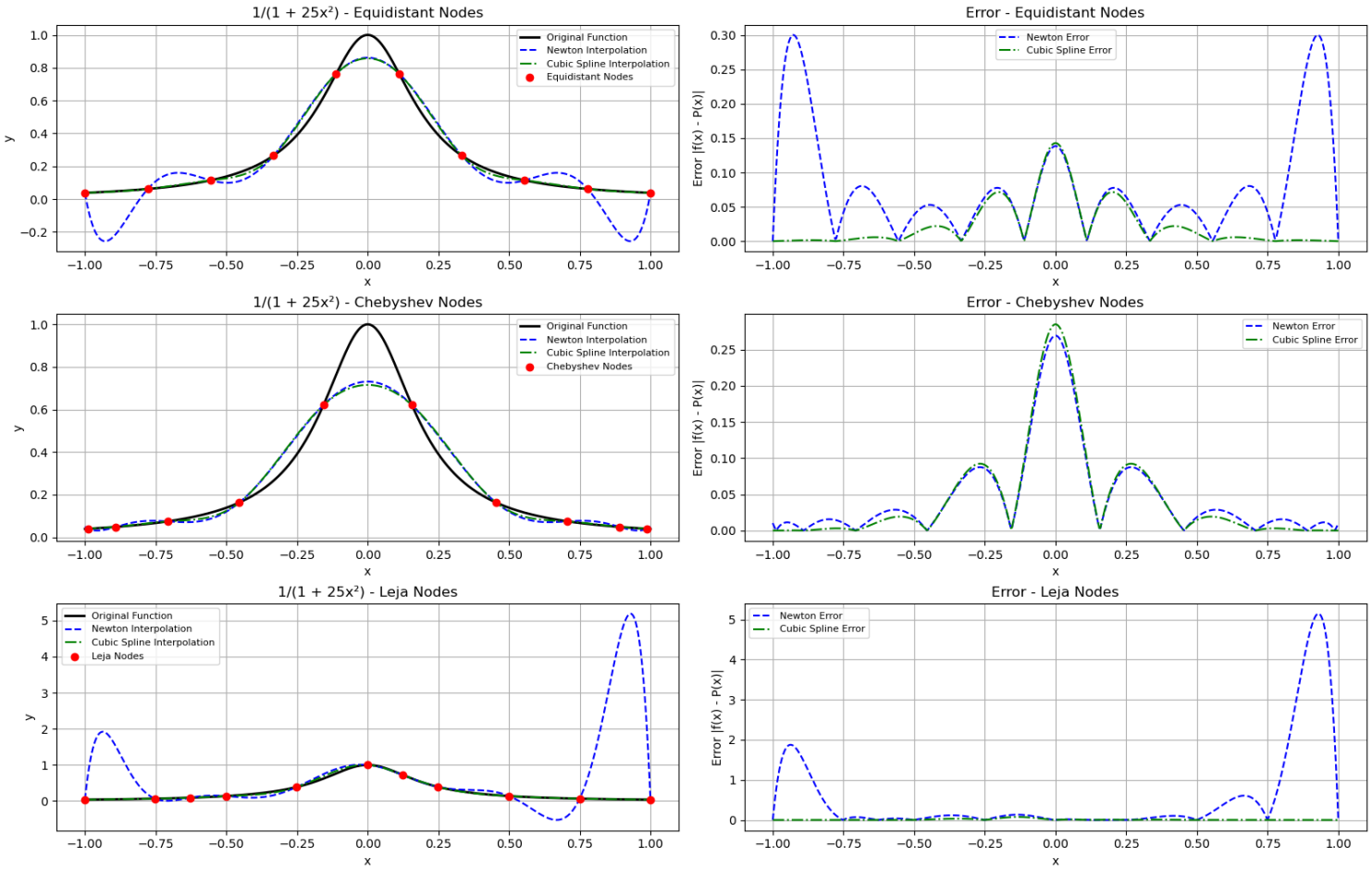


图 2: $\frac{1}{1+25x^2}$ 插值效果对比

```
(base) PS D:\BaiduSyncdisk\Work\Courses\Junior_Fall\CompPhys\A
### Experiment 1: cos(x) with Equidistant Nodes ###

cos(x) - Equidistant Nodes
Newton Interpolation Max Error: 0.000000
Cubic Spline Interpolation Max Error: 0.006077

### Experiment 2: 1/(1 + 25x²) with Different Node Types ###

### Node Type: Equidistant ###

1/(1 + 25x²) - Equidistant Nodes
Newton Interpolation Max Error: 0.300288
Cubic Spline Interpolation Max Error: 0.142804

### Node Type: Chebyshev ###

1/(1 + 25x²) - Chebyshev Nodes
Newton Interpolation Max Error: 0.269097
Cubic Spline Interpolation Max Error: 0.284735

### Node Type: Leja ###

1/(1 + 25x²) - Leja Nodes
Newton Interpolation Max Error: 5.143522
Cubic Spline Interpolation Max Error: 0.073877
```

图 3: 终端输出

2 题目 2：金属棒温度数据拟合

2.1 题目描述

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

(1) Compute a least-squares, straight-line fit to these data using $T(x) = a + bx$.

(2) Compute a least-squares, parabolic-line fit to these data using $T(x) = a + bx + cx^2$.

表 1: Temperature data along the metal rod

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

2.2 程序描述

本程序按照以下流程执行：先使用 `np.vstack` 函数垂直堆叠数组，构建设计矩阵。两个模型分别为

$$A_{\text{linear}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \end{bmatrix}^T, A_{\text{quadratic}} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \end{bmatrix}^T$$

然后，对设计矩阵进行 SVD(本程序的 *SVD 分解借助 **numpy** 库实现*)，即 $A = U\Sigma V^T$ 其中 U 和 V 是正交矩阵，Σ 是对角矩阵，包含奇异值 $\sigma_1, \sigma_2, \ldots, \sigma_p$，其中 $p = \min(m, n)$。为了计算 Moore-Penrose 伪逆 A^+，首先要对奇异值进行处理，将接近零的奇异值的倒数**简单截断**，即

$$\sigma_i^+ = \begin{cases} \frac{1}{\sigma_i}, & \text{if } \sigma_i > \tau \\ 0, & \text{otherwise} \end{cases}$$

本程序中阈值 $\tau = 1 \times 10^{-10}$。构建 Σ^+ 矩阵后，伪逆矩阵 A^+ 通过以下公式计算：$A^+ = V\Sigma^+U^T$ 利用伪逆矩阵，可以求解最小二乘问题的参数向量 $\boldsymbol{\beta}$，即 $\boldsymbol{\beta} = A^+T$. 这种正则化步骤减少了因奇异值接近零而可能导致的数值不稳定，降低了对噪声的敏感性。**岭回归**也是一种常用的正则化方法，通过在最小二乘目标函数中加入 L^2 正则化项，

$$\min_{\boldsymbol{\beta}} \|A\boldsymbol{\beta} - T\|_2^2 + \lambda\|\boldsymbol{\beta}\|_2^2$$

其中 λ 是正则化参数，其解析解为 $\boldsymbol{\beta}_{\text{ridge}} = (A^TA + \lambda I)^{-1}A^TT$，其有效地减小了参数的方差，提升了模型在新数据上的泛化能力。但本程序没有用此牛刀。在 `src` 目录使用`python -u least_squares.py` 即可运行。

2.3 伪代码

Algorithm 5: Pseudo-inverse computation and model fitting

Input: $x(\text{array}, \text{shape} = n)$, $T(\text{array}, \text{shape} = n)$

Output: $\beta(\text{array})$, $T_{\text{fit}}(\text{array})$, $RSS(float)$

$A_{\text{linear}} \leftarrow \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \end{bmatrix}^T$, $A_{\text{quadratic}} \leftarrow \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \end{bmatrix}^T$; `// Design matrices for fits`

Function `compute_pseudo_inverse(`A`):`

  $U, \Sigma, V^T \leftarrow$ `np.linalg.svd(`A`,` full_matrices=False) ; `// Compute SVD of` A

  foreach $\Sigma_i \in \Sigma$ **do**

    if $\Sigma_i > 10^{-10}$ **then**

      $\Sigma_{\text{inv}}[i] \leftarrow \frac{1}{\Sigma_i}$

    else

      $\Sigma_{\text{inv}}[i] \leftarrow 0$

    end

  end

  $\Sigma_{\text{inv}} \leftarrow \text{diag}(\Sigma_{\text{inv}})$ $A^+ \leftarrow V^T \cdot \Sigma_{\text{inv}} \cdot U^T$ **return** A^+

end

$\beta \leftarrow A^+T$; $T_{\text{fit}} \leftarrow \sum_{i=0}^{d} \beta_i x^i$; $RSS \leftarrow \sum(T - T_{\text{fit}})^2$; `// Compute residual sum of squares (RSS)`

return $\beta_{linear}, T_{fit,\ linear}, RSS_{linear}, \beta_{quadratic}, T_{fit,\ quadratic}, RSS_{quadratic}$

2.4 结果示例

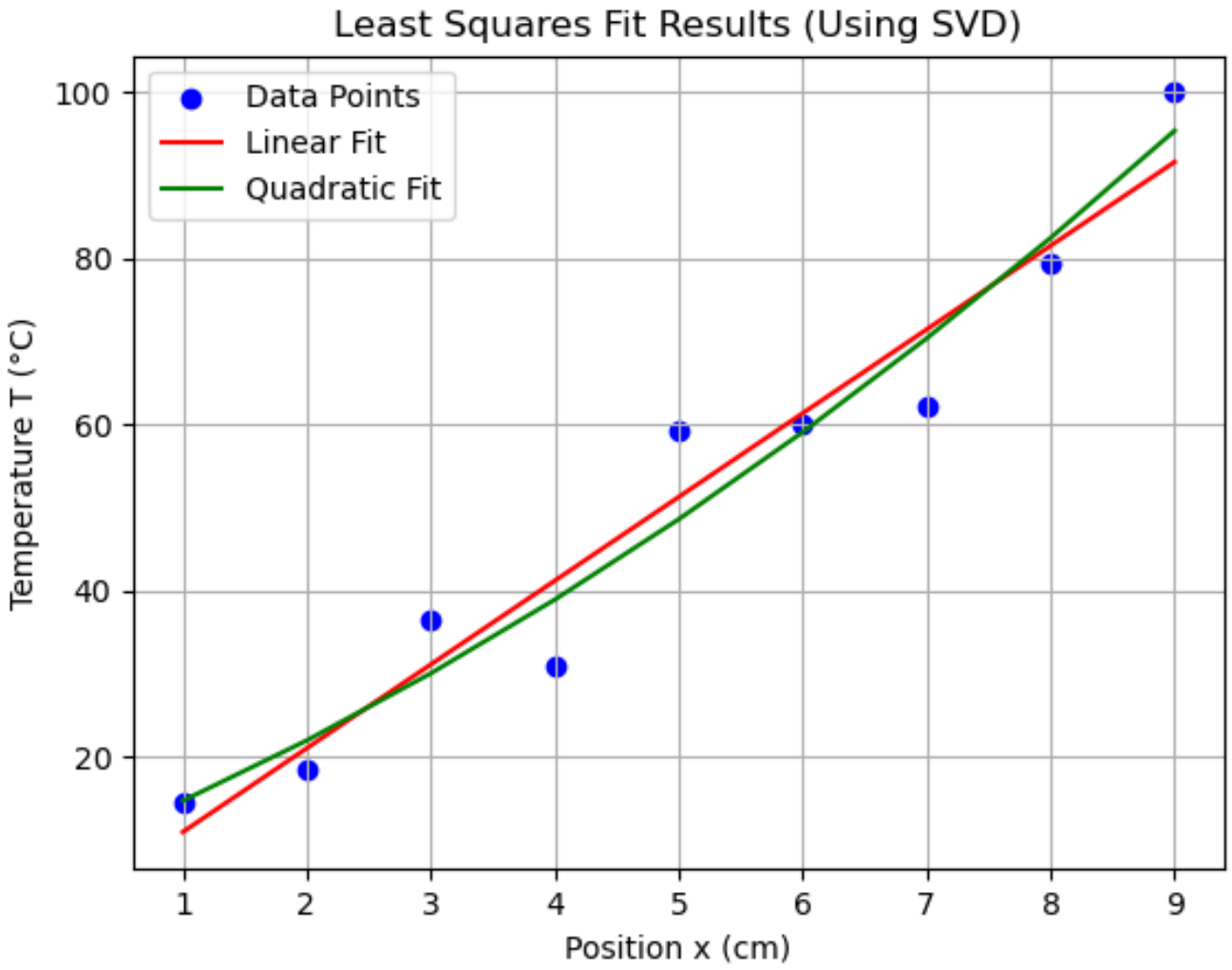


图 4: 拟合结果

```
(base) PS D:\BaiduSyncdisk\Work\Courses\Junior_Fall\CompPhys\
Linear fit result: T(x) = 0.8889 + 10.0733x
Quadratic fit result: T(x) = 8.2619 + 6.0517x + 0.4022x²
Residual Sum of Squares (RSS) for linear fit: 380.9596
Residual Sum of Squares (RSS) for quadratic fit: 331.1448
R² for linear fit: 0.9411
R² for quadratic fit: 0.9488
```

图 5: 终端输出

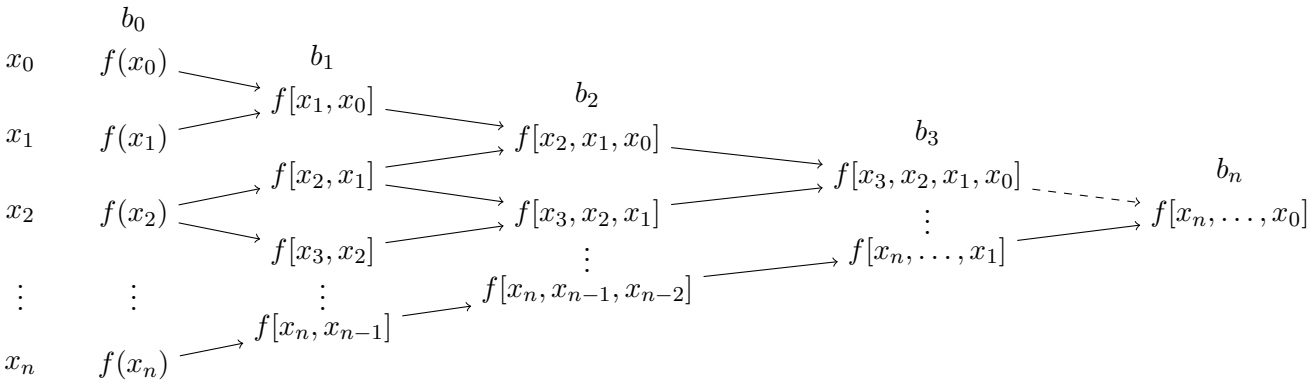
3 补充题：牛顿插值法唯一性探讨

3.1 题目描述

Given $n+1$ points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, the n-th order interpolation polynomial using Newton's method is:

$$P_n(x) = f[x_0] + f[x_1, x_0](x - x_0) + f[x_2, x_1, x_0](x - x_0)(x - x_1) + \cdots + f[x_n, x_{n-1}, \ldots, x_0](x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

where $f[x_i, x_{i-1}, \ldots, x_0]$ represents the divided differences. Taking the coefficients from the lower edge of the difference table (i.e., $f[x_n], f[x_n, x_{n-1}], \ldots, f[x_n, \ldots, x_0]$), will this provide higher accuracy for values of x near x_n?



3.2 程序描述

原本对于不考虑舍入误差的理想形式，Newton 插值与 Lagrange 插值的结果必定是唯一且等价的，也即通过给定 $(n+1)$ 个节点的 n 阶多项式唯一，这一点可以从 Vandermonde 行列式

$$\det \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} = \prod_{0 \le i < j \le n} (x_j - x_i)$$

非奇异中看出。但实际结果中，我们必须考虑舍入误差，这使得 Newton 插值的结果可能会依赖于节点的顺序。尤其是在我们使用 Horner's Rule(1.2) 的时候，显然在节点附近的插值数值稳定性依赖于节点的顺序。本程序使用1.1节中的牛顿插值代码，对两个病态函数 e^x 与 $\sin(100x)$ 展开了数值实验。

实验中，我们分别在区间 $[-10, 10]$ 和 $[-0.8, 0.5]$ 上等距取 25 个节点，对两个函数进行顺序与逆序的牛顿插值，并在子图 1 中展示插值效果，在子图 2 中对比两者相较于真实函数的相对误差，在子图 3 中展示逆序插值与顺序插值相对误差的差值。在 e^x 结果中使用了对数坐标轴以更好展示插值效果。

3.3 实验结果

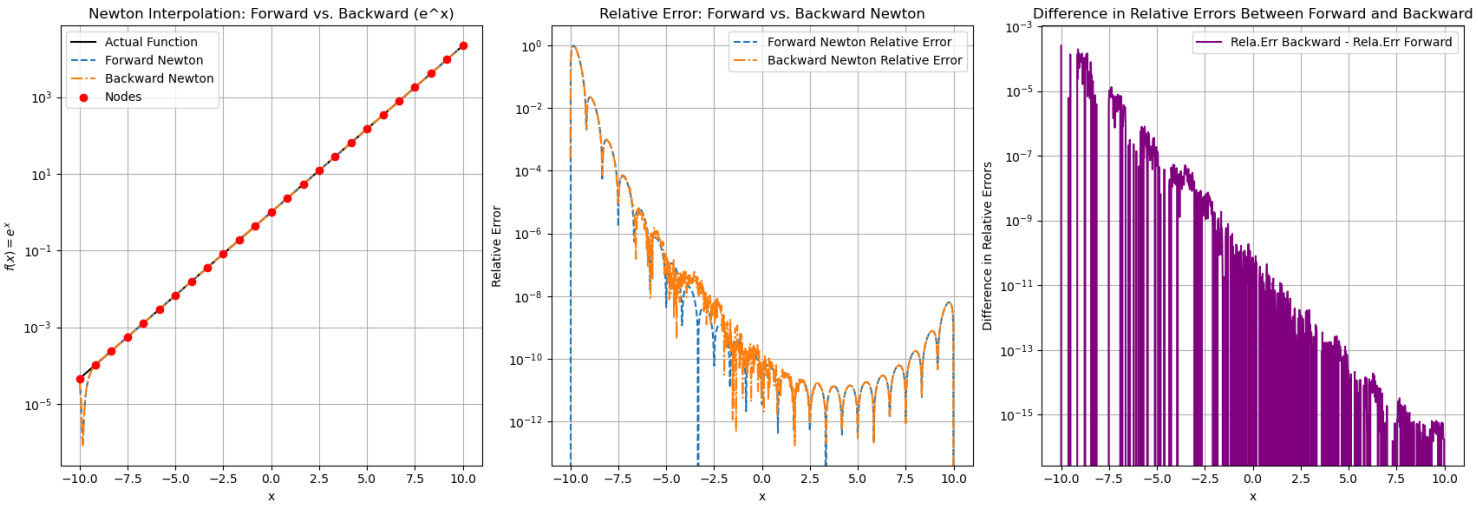


图 6: e^x 实验结果

令人惊奇的是，对于病态函数 e^x，顺序插值的效果始终优于逆序插值（表现为子图 2 中蓝色虚线始终位于橙色虚线下方，子图 3 中紫色曲线值始终非负）猜测可能是由于逆序差值的系数在计算初期便受到了较大的舍入误差的影响，而这种影响在 Horner's Rule 中被逐级放大，最终在传播后半段（子图左侧）逐渐显现。

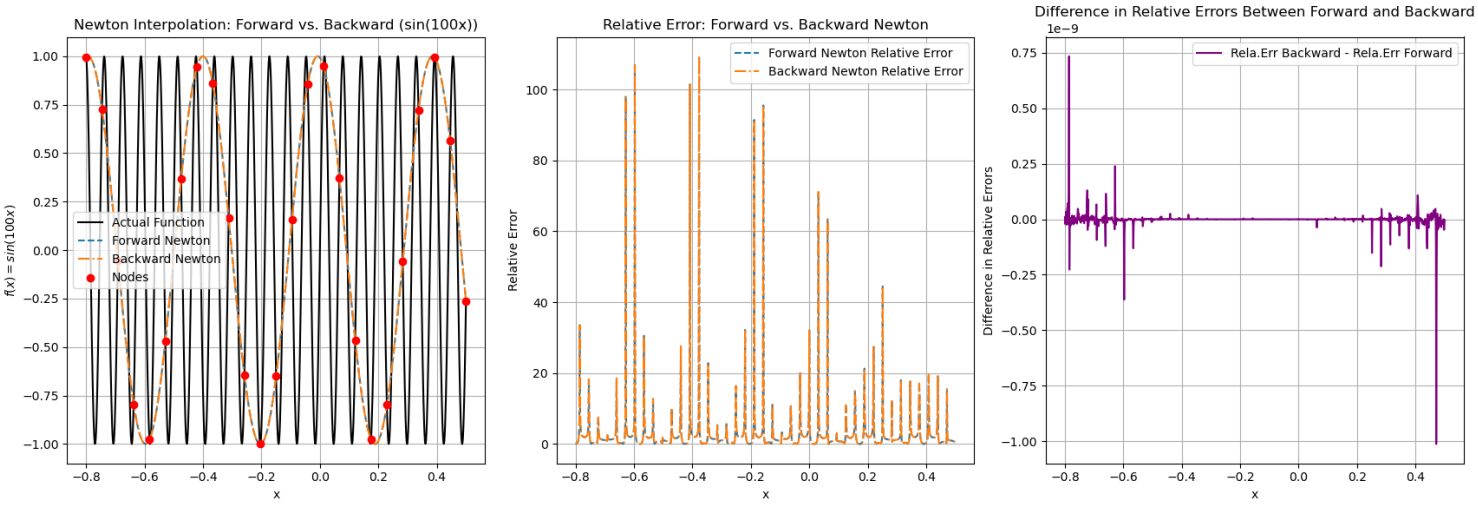


图 7: $\sin(100x)$ 实验结果

对于高频震荡的病态函数 $\sin(100x)$，逆序差值与顺序差值这对卧龙凤雏，均展现了高达一倍以上的相对误差，相较于 e^x 的结果，更为病态，也符合预期。令人惊喜的是，由于 $\sin(100x)$ 本身的对称性与导数有界性，两种差值顺序的优劣对比在区间两端出现了符合预期的差别：在正向传播的末端（子图 3 右侧），顺序插值的相对误差超过了逆序差值，反之在另一侧，顺序差值结果更优，这符合我们最开始的朴素想法：随着节点的进行，舍入误差的影响逐渐增大，致使区间两侧效果不一，这也提示我们在边界附近应当采集更密的节点，如1.1节中的 Chebyshev 与 Leja 节点。