第四次作业

尹朝阳 物理学系

1 题目 1

1.1 题目描述

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 程序描述

本题目要求用牛顿插值法逼近所要求的函数，并与三次样条插值进行比较。

1.2.1 牛顿插值法

对于所给的 n 个点，牛顿插值法的实现思路如下：

$$\begin{aligned} y(x) = f(x_0) &+ f[x_1, x_0](x - x_0) \\ &+ f[x_2, x_1, x_0](x - x_0)(x - x_1) \\ &+ ... \\ &+ f[x_{n-1}, x_{x_2}, ..., x_0](x - x_0)(x - x_1)(x - x_2)...(x - x_{n-2}) \end{aligned} \tag{1}$$

其中 $f[x_i, x_{i-1}, ..., x_0]$ 为要求的系数，其满足如下递推公式：

$$f[x_i, x_{i-1}, ..., x_0] = \frac{f[x_i, x_{i-1}, ..., x_1] - f[x_{i-1}, x_{i-2}, ..., x_0]}{x_i - x_0} \tag{2}$$

对于 $0 \leq i \leq n - 1$ 均成立。

则可以构造如下图所示的左上三角矩阵：

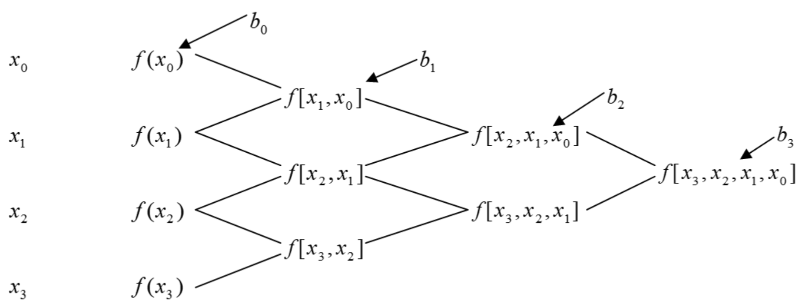


图 1: parmatrix of newton interpolation

最终从该矩阵中取出第一行，即为系数组成的行向量，最终化简可得插值方程。

1.2.2 三次样条插值法

而对于三次样条插值法，其实现思路如下：

三次样条插值法采用三次函数对于 n 个所给点所分成的 $n-1$ 段进行分段插值，其要求在每个分段内二阶导数为一条直线。再加上插值曲线需要过所给点的约束条件，可以求出在第 $i-1$ 段的函数表达式如下：

$$
\begin{aligned}
f_{i-1}(x) = &\frac{f''(x_{i-1})}{6(x_i - x_{i-1})}(x_i - x)^3 + \frac{f''(x_i)}{6(x_i - x_{i-1})}(x - x_{i-1})^3 \\
&+ [\frac{f(x_{i-1}}{x_i - x_{i-1}} - \frac{f''(x_{i-1})(x_i - x_{i-1})}{6}](x_i - x) \\
&+ [\frac{f(x_i}{x_i - x_{i-1}} - \frac{f''(x_i)(x_i - x_{i-1})}{6}](x - x_{i-1})
\end{aligned} \tag{3}
$$

为了求出每段的插值曲线 $f_{i-1}(x)$，只需要求出相应的 $f''(x_{i-1})$ 和 $f''(x_i)$ 即可，即需要求出所给集合中的每个点的二阶导数值。

对于 n 个所给点的 n 个二阶导数值，需要联立 n 个方程进行求解。首先由约束条件，对每个点 $x_i(1 \le i \le n-2)$，有 $f'_{i-1}(x_i) = f'_i(x_i)$，解得

$$
\begin{aligned}
&(x_i - x_{i-1})f''(x_{i-1}) + 2(x_{i+1} - x_{i-1})f''(x_i) + (x_{i+1} - x_i)f''(x_{i+1}) \\
= &\frac{6}{x_{i+1} - x_i}[f(x_{i+1}) - f(x_i)] + \frac{6}{x_i - x_{i-1}}[f(x_{i-1}) - f(x_i)]
\end{aligned} \tag{4}
$$

在知道 $f(x_i)$ 的条件下，可得二阶导数值相邻三项的递推公式，一共有 $n-2$ 个方程。加上人为设定的 $f''(x_0) = f''(x_{n-1}) = 0$，一共 n 条方程，解出 n 个二阶导数值，代入 (3) 式，即可得到最后的分段插值函数。

本程序的代码文件如下。

第一题的所有程序均位于 Interpolation 文件夹中，其中

- NewtonInterpolation.py 实现牛顿插值法
- CubicSplineInterpolation.py 实现三次样条插值法
- question1.py 分别调用 NewtonInterpolation.py 和 CubicSplineInterpolation.py 对具体函数 $\cos(x)$ 进行插值求解，并比较两种方法距离原函数的逼近程度。
- question2.py 分别调用 NewtonInterpolation.py 和 CubicSplineInterpolation.py 对具体函数 $\frac{1}{1+25x^2}$ 进行插值求解，并比较两种方法距离原函数的逼近程度。

第一题所用到的第三方库有 numpy==1.22.3 和 matplotlib==3.5.1，编码格式为 utf-8。

1.3 伪代码

Algorithm 1.1 Newton Interpolation

Input : $points(x_i, y_i)$

Output : the newton interpolation function $f(x)$

for $i \leftarrow 0$ to $n-1$ **do**
  $parmatrix[i][0] \leftarrow y_i$ ▷ the first column $= [y_i]$
end for
for $j \leftarrow 1$ to $n-1$ **do** ▷ $f[x_i, ..., x_0] = \frac{f[x_i,...,x_1] - f[x_{i-1},...,x_0]}{x_i - x_0}$
  for $i \leftarrow 0$ to $n-1-j$ **do**
    $parmatrix[i][j] \leftarrow (parmatrix[i][j-1] - parmatrix[i+1][j-1])/(x_i - x_{i+j})$
  end for
end for
$f(x) \leftarrow \sum_{i=0}^{n-1}[parmatrix[0][i] * \prod_{j=0}^{i-1}(x - x_j)]$ ▷ fomula (1)

Algorithm 1.2 Cubic Spline Interpolation

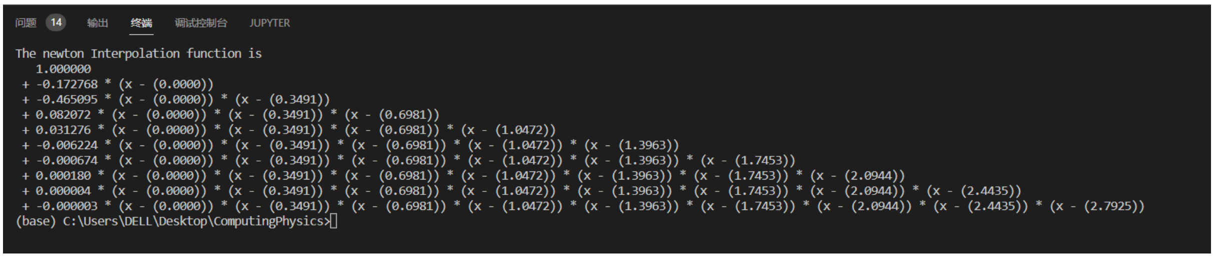
Input : $points(x_i, y_i)$

Output : the second order derivative $f''(x_i)$, $0 \leq i \leq n-1$

$A[0][0] \leftarrow 1$
$A[n-1][n-1] \leftarrow 1$
$b[0][0] \leftarrow 0$ ▷ $1 * f''(x_0) = 0$
$b[n-1][0] \leftarrow 0$ ▷ $1 * f''(x_{n-1}) = 0$
for $i \leftarrow 1$ to $n-2$ **do** ▷ fomula (4)
  $A[i][i-1] \leftarrow x_i - x_{i-1}$
  $A[i][i] \leftarrow 2(x_{i+1} - x_{i-1})$
  $A[i][i+1] \leftarrow x_{i+1} - x_i$
  $b[i][0] \leftarrow 6/(x_{i+1} - x_i) * (y_{i+1} - y_i) + 6/(x_i - x_{i-1}) * (y_{i-1} - y_i)$
end for
$x \leftarrow A^{-1}B$
$f''(x_i) \leftarrow x[i][0]$
use the fomula (3) to solve $f_i(x)$ with $\{f''(x_i)\}$

1.4 测试用例

第 (1) 问:


```
问题 14   输出   终端   调试控制台   JUPYTER

The newton Interpolation function is
   1.000000
 + -0.172768 * (x - (0.0000))
 + -0.465095 * (x - (0.0000)) * (x - (0.3491))
 + 0.082072 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981))
 + 0.031276 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981)) * (x - (1.0472))
 + -0.006224 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981)) * (x - (1.0472)) * (x - (1.3963))
 + -0.000674 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981)) * (x - (1.0472)) * (x - (1.3963)) * (x - (1.7453))
 + 0.000180 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981)) * (x - (1.0472)) * (x - (1.3963)) * (x - (1.7453)) * (x - (2.0944))
 + 0.000004 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981)) * (x - (1.0472)) * (x - (1.3963)) * (x - (1.7453)) * (x - (2.0944)) * (x - (2.4435))
 + -0.000003 * (x - (0.0000)) * (x - (0.3491)) * (x - (0.6981)) * (x - (1.0472)) * (x - (1.3963)) * (x - (1.7453)) * (x - (2.0944)) * (x - (2.4435)) * (x - (2.7925))
(base) C:\Users\DELL\Desktop\ComputingPhysics>
```

图 2: newton interpolation

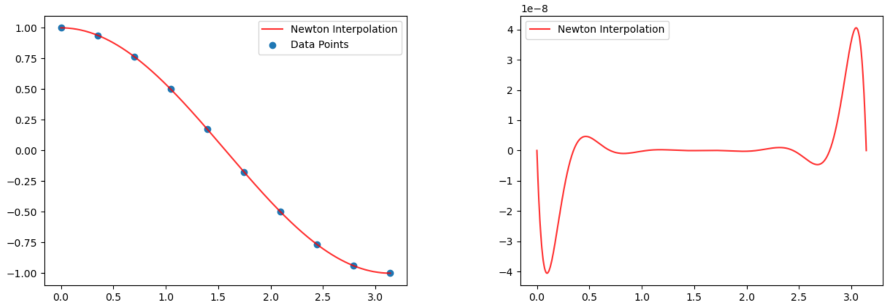


图 3: newton interpolation(1)

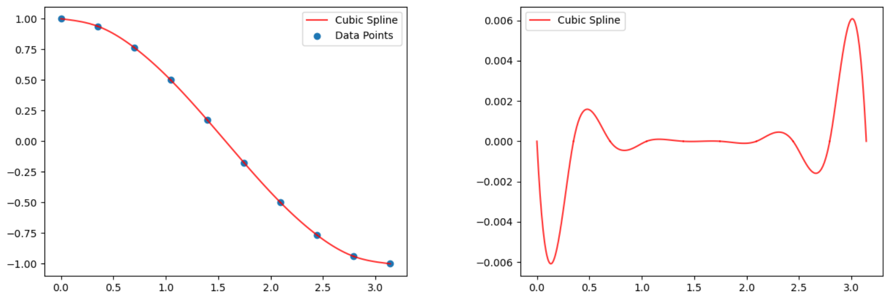
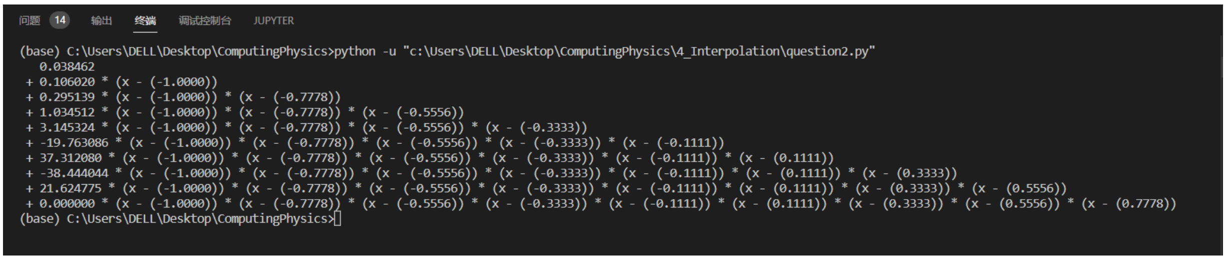


图 4: Cubic Spline interpolation

对比图 3、图 4 的右图，可以发现，相较于三次样条插值法，牛顿插值法实现 $\cos(x)$ 的插值函数更加接近原函数。因为牛顿插值法幂次数更高，更加逼近 $\cos(x)$ 的 Taylor 展开形式。

第 (2) 问：



```
问题 14   输出   终端   调试控制台   JUPYTER

(base) C:\Users\DELL\Desktop\ComputingPhysics>python -u "c:\Users\DELL\Desktop\ComputingPhysics\4_Interpolation\question2.py"
   0.038462
 + 0.106020 * (x - (-1.0000))
 + 0.295139 * (x - (-1.0000)) * (x - (-0.7778))
 + 1.034512 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556))
 + 3.145324 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556)) * (x - (-0.3333))
 + -19.763086 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556)) * (x - (-0.3333)) * (x - (-0.1111))
 + 37.312080 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556)) * (x - (-0.3333)) * (x - (-0.1111)) * (x - (0.1111))
 + -38.444044 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556)) * (x - (-0.3333)) * (x - (-0.1111)) * (x - (0.1111)) * (x - (0.3333))
 + 21.624775 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556)) * (x - (-0.3333)) * (x - (-0.1111)) * (x - (0.1111)) * (x - (0.3333)) * (x - (0.5556))
 + 0.000000 * (x - (-1.0000)) * (x - (-0.7778)) * (x - (-0.5556)) * (x - (-0.3333)) * (x - (-0.1111)) * (x - (0.1111)) * (x - (0.3333)) * (x - (0.5556)) * (x - (0.7778))
(base) C:\Users\DELL\Desktop\ComputingPhysics>
```

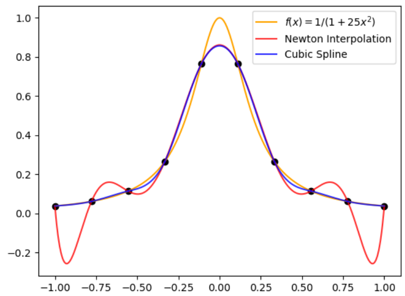


图 5: interpolation result

从图 5(2) 可以直观看出，三次样条插值拟合得相对较好，牛顿插值法出现了过拟合现象。在 $x = 0$ 附近，两个拟合函数效果都不是太好。

2 题目 2

2.1 题目描述

(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 $T(x) = a + bx + cx^2$

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

2.2 程序描述

题目要求用最小二乘法拟合所给点集，其中两问分别要求用线性拟合和二次曲线拟合的方式。

最小二乘法的本质为求通过求解评价函数 $\chi^2 = \sum\limits_{i=0}^{n-1}[p(x_i) - y_i]^2$ 的最小值，解出拟合函数 $p(x)$ 的相应系数。

对于拟合函数 $p(x)$ 为多项式函数的情况 $p(x) = \sum\limits_{k=0}^{n-1} a_k * x^k$，评价函数的最小值可以通过求导数求解。对 $0 \le k \le n-1$，有

$$\frac{\partial \chi^2}{\partial a_k} = 0$$

则得出 n 阶线性方程组 $Ax = b$。

对于 $Ax = b$，通过 SVD 分解，将矩阵 A 分解为 $A = USV^T$。由 $\frac{\partial ||Ax-b||^2}{\partial x} = 0$ 最终可以解出系数矩阵 $x = VS^{-1}U^Tb$。

特别地，对于线性拟合的情况 $p(x) = a_0 + a_1 * x$，容易解出

$$\begin{aligned} a_0 &= \frac{c_0c_3 - c_1c_2}{c_0^2 - nc_1} \\ a_1 &= \frac{c_0c_2 - nc_3}{c_0^2 - nc_1} \end{aligned} \tag{5}$$

其中 $c_0 = \sum\limits_{i=0}^{n-1} x_i$，$c_1 = \sum\limits_{i=0}^{n-1} x_i^2$，$c_0 = \sum\limits_{i=0}^{n-1} y_i$，$c_0 = \sum\limits_{i=0}^{n-1} x_i * y_i$，第二题第一问中，就直接用这种方法解出线性拟合函数。

本程序的代码文件包括 LeastSquaresMethod/least_squares_method_q1.py 和 LeastSquaresMethod/least_squares_method_q2.py，分别为第一问和第二问的程序。程序导入的第三方库包括 numpy==1.22.3 和 matplotlib==3.5.1，编码格式为 utf-8。

2.3 伪代码

Algorithm 2.1 least squares linear fitting method

Output : the parameters a_0, a_1

$c_0 = \sum_{i=0}^{n-1} x_i$

$c_1 = \sum_{i=0}^{n-1} x_i^2$

$c_2 = \sum_{i=0}^{n-1} y_i$

$c_3 = \sum_{i=0}^{n-1} x_i * y_i$

$a_0 = (c_0 * c_3 - c_1 * c_2)/(c_0 * c_0 - n * c_1)$

$a_1 = (c_0 * c_2 - n * c_3)/(c_0 * c_0 - n * c_1)$

Algorithm 2.2 least squares parabolic-line fitting method

Output : the parameters a_0, a_1, a_2

for $j \leftarrow 0$ to 2 **do**
    for $k \leftarrow 0$ to 2 **do**
        $A[0][0] = n$
        $A[j][k] \leftarrow \sum_{i=0}^{n-1} x_i^{j+k}$
    end for
end for
for $j \leftarrow 0$ to 2 **do**
    $b[j][0] \leftarrow \sum_{i=0}^{n-1} (y_i * x_i^j)$
end for
$u, s, v \leftarrow SVD(A)$
$x \leftarrow vs^{-1}u^T b$
$a_0 \leftarrow x[0][0]$, $a_1 \leftarrow x[1][0]$, $a_2 \leftarrow x[2][0]$

2.4 测试用例

第 (1) 问:

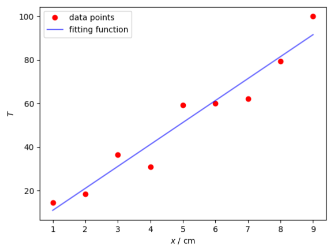
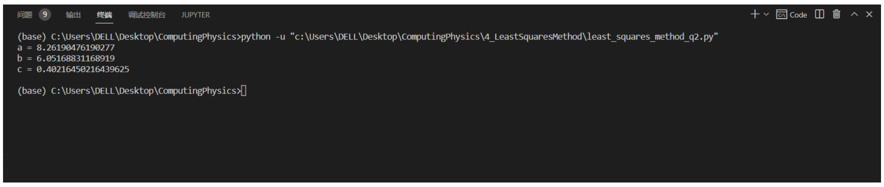


图 6: least squares linear fitting method

最终的拟合函数为 $p(x) = 0.8889 + 10.0733x$。

第 (2) 问:



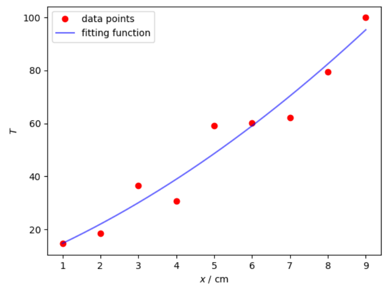


图 7: least squares linear fitting method

最终的拟合函数为 $p(x) = 8.2619 + 6.0517x + 0.4022x^2$。