Homework No.4

Ruqi Feng

September 2022

1 Interpolation

1.1 Description

Newton interpolation of (i) 10 equal spacing points of cos(x) within [0,π], (ii) 10 equal spacing points $\frac{1}{1+25x^2}$ within [-1,1]. Compare the results with the cubic spline interpolation.

1.2 Solution

涉及的插值方法有牛插值顿法和三次样条插值。

1.2.1 牛顿法

牛顿插值法利用多项式插值，使拟合多项式过所有数据点，与拉格朗日插值的区别是，牛顿法有着在增加数据点时不需要重新计算已有的数据点的优势。输入 $n+1$ 个插值点时，牛顿法产生的多项式为

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

其中 (x_i, y_i) 是第 i 对数据点，$f[x_0] = y_0$，而

$$f[x_i,x_{i-1},...,x_j] = \frac{f[x_i,x_{i-1},...,x_{j+1}] - f[x_{i-1},x_{i-2},...,x_j]}{x_i - x_j} \tag{2}$$

可以发现，牛顿插值法利用有限差分的想法产生一个 n 次多项式，这是唯一过所有 $n+1$ 个数据点的 n 次多项式。因此，只需要对所有可能的 i 计

算出 $f[x_i, ..., x_{i-k}]$ 就可以递推得到对所有可能的 i 的 $f[x_{i+1}, ..., x_{i-k}]$。从而确定公式1中的系数，得到插值多项式。

1.2.2 三次样条插值

三次样条插值则利用三次多项式分段插值，通过要求插值函数过 n 个数据点，以及相邻分段的多项式在边界处一阶、二阶导数相等，和数据边界上二阶导数为 0 共 $4n-4$ 个方程来确定插值分段多项式的系数。

这样的插值方式等价于在不同数据点之间对节点上的二阶导数线性插值。其中二阶导数可以用节点上导数相等的条件确定。便可以得到

$$f''(x) = f''(x_{i-1})\frac{x - x_i}{x_{i-1} - x_i} + f''(x_i)\frac{x - x_{i-1}}{x_i - x_{i-1}}, \text{ when } x \in (x_{i-1}, x_i) \quad (3)$$

将公式3对 x 积分两次得到

$$\begin{aligned} f(x) =& \frac{f''(x_{i-1})}{6(x_i - x_{i-1})}(x_i - x)^3 - \frac{f''(x_i)}{6(x_i - x_{i-1})}(x_{i-1} - x)^3 \\ &+ [\frac{f(x_{i-1})}{x_i - x_{i-1}} - \frac{f''(x_{i-1})(x_i - x_{i-1})}{6}](x_i - x) \\ &- [\frac{f(x_i)}{x_i - x_{i-1}} - \frac{f''(x_i)(x_i - x_{i-1})}{6}](x_{i-1} - x) \end{aligned} \quad (4)$$

其中，$f'(x_{i-1})$ 已经在 x_i 代入 x 之后，利用 $f(x_i)$ 消去。于是未知数只剩下各个节点处的 $f''(x_i)$。为了利用一阶导的条件，对公式4求导，并利用 $x \in (x_{i-1}, x_i)$ 和 $x \in (x_i - x_{i+1})$ 在 x_i 处的一阶导数相等得到

$$\begin{aligned} &(x_i - x_{i-1})f''(x_{i-1}) + 2(x_{i+1} - x_{i-1})f''(x_i) + (x_{i+1} - x_i)f''(x_{i+1}) \\ =& \frac{6f(x_{i+1}) - 6f(x_i)}{x_{i+1} - x_i} + \frac{6f(x_{i-1}) - 6f(x_i)}{x_i - x_{i-1}} \end{aligned} \quad (5)$$

得到的方程组5再加上边界上 $f''(x_0) = f''(x_n) = 0$ 就可以求解出所有 $f''(x_i)$。代入公式4即为样条插值结果。

1.3 Input and Outputs

如图1所示，在 0 到 π 间取 10 个点牛顿插值和三次样条插值的结果都很接近精确值。为了比较牛顿插值和样条插值的结果，在图2中画出牛顿插值法和三次样条插值相对 $\cos x$ 的偏差。在图中可以看出牛顿法残差远小于样条插值残差，牛顿插值法残差绝对值约为 4.05×10^{-8}，样条残差最大值约为 3.92×10^{-4}。

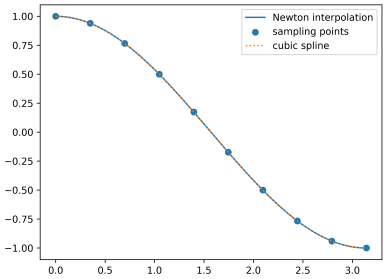

图 1: 余弦函数的插值结果

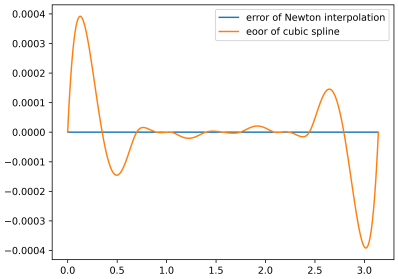

图 2: 余弦函数的插值残差

接着对函数 $\frac{1}{1+25x^2}$ 进行插值，结果和残差分别如图3和4所示。此时牛顿插值的结果很糟糕，也远差于样条插值的结果，这是龙格现象导致的。在边缘处牛顿插值的结果剧烈震荡，偏离正确值。牛顿插值的最大误差为 0.30 出现在边缘，而样条插值的最大误差约 0.14，很可能是由于 $x = 0$ 附近采样不足导致。从这两个例子可以看出，需要对不同的函数分别选择合适的插值方法才能得到较好的结果。

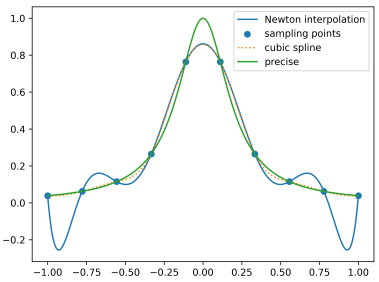

图 3: 函数 $\frac{1}{1+25x^2}$ 的插值结果

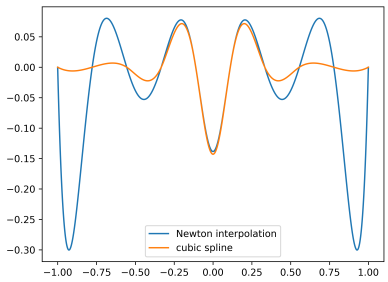

图 4: 函数 $\frac{1}{1+25x^2}$ 的插值残差

程序使用了 numpy, matplotlib 和 scipy 库。在安装了依赖库的环境中运行 `interpolation.py` 即可。

1.4 Pseudocode

牛顿插值法使用递归，虽然时间复杂度较大，但在题目的问题规模下性能足够。

Algorithm 1 Newton Interpolation

Input: data to interpolate, vectors x and y

Output: interpolation function f

function GETF(x, y) ▷ find f[xi, ..., xj] recursively
    if len(x)==1 **then**
        return y
    else
        $dy \leftarrow$GETF(x[1:], y[1:])-GETF(x[1:], y[1:])
        $dx \leftarrow x$[-1] - x[0]
        r**return** $\frac{dy}{dx}$
    end if
end function
function F(x_{plot})
    $f \leftarrow 0$
    for i in range(len(x), 0, -1) **do**
        $f \leftarrow f$+GETF(x[: i], y[: i])
        if $i > 1$ **then**
            $f \leftarrow f \times (x_{plot} - x[i-2])$
        end if
    end for
    return f
end function
return F

三次样条插值的实现基本和前文中的计算顺序一致。

Algorithm 2 Cubic Spline Interpolation

Input: data to interpolate, vectors x and y

Output: interpolation function f

for i in range(len(x)) **do**
  if i == 0 or i == n **then**
    A[i, i] $\leftarrow 1$
    b[i] $\leftarrow 0$
  else
    A[i, i-1: i+1] $\leftarrow [x_i - x_{i-1}, 2(x_{i+1} - x_{i-1}), x_{i+1} - x_i]$
    b[i] $\leftarrow \frac{6(y_{i+1}-y_i)}{x_{i+1}-x_i} + \frac{6(y_{i-1}-y_i)}{x_i-x_{i-1}}$
  end if
end for
$f''(x_i) \leftarrow \mathrm{A}^{-1}\mathrm{b}$
function F(x_{plot})
  $f \leftarrow 0$
  for i in range(len(x)) **do**
    if $x_i \leq x_{plot} \leq x_{i+1}$ **then**
      return Eq. 4 with $f''(x_i)$ inserted back
    end if
  end for
  return f
end function
return F

2 Least Square Fitting

2.1 Description

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

2.2 Solution

考虑一个拟合问题，有 j 个自变量 $\vec{X_1}, \vec{X_2}, \vec{X_3}...$，$n$ 个数据点。寻找 $\vec{X_i}$ 的线性组合使得残差平方和最小，等价于寻找 $\vec{x}$ 使得

$$\min \|A\vec{x} - \vec{Y}\|^2 \tag{6}$$

其中 A 是 $\vec{X_i}$ 按列排成的矩阵。求导可知这个二次型极小值的条件是

$$\vec{x} = (A^T A)^{-1} A^T \vec{Y} \tag{7}$$

公式7看似是方程组 $A\vec{x} = \vec{Y}$ 左乘 A^T 后得到的解，但事实上 $A\vec{x} = \vec{Y}$ 并没有解。之所以出现所谓" 伪逆"，是因为 A^T 扩大了 $A - I\vec{Y}$ 的零空间。但求 "解" 方程组 (当然，对于有解的方程组，解正好能让上面的二次型取极小–它恰好是 0) 能得到二次型的极小值是一件有趣的事。于是只需要求出公式7右侧的矩阵即可求得拟合的各个系数。对线性拟合来说，系数有 2 个，对应的 $A = (X\ 1)$，因为 1 是常数项系数的" 自变量"。对二次拟合来说系数有 3 个，$A = (X^2\ X\ 1)$。

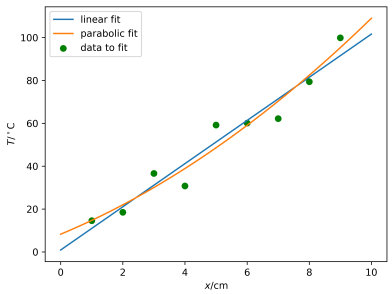

图 5: 线性拟合和二次拟合的结果

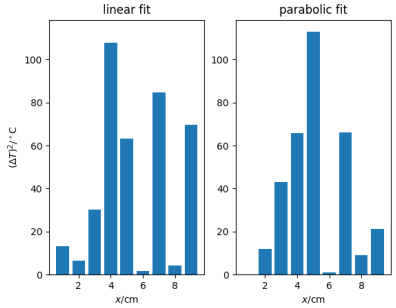

图 6: 线性拟合和二次的拟合残差

2.3 Input and Outputs

线性拟合和二次拟合的系数如表所示，拟合结果画在图5中。可以从残差平方图6中看出，在数据边缘二次拟合的效果略好于线性拟合。

程序使用了 numpy 和 matplotlib 库。在安装了依赖库的环境下运行 `linear_fit.py` 即可。

	c	b	a
linear fit	0	10.07333333333	0.888888888888
parabolic fit	0.4021645021644	6.051688311688	8.26190476190

2.4 Pseudocode

程序中线性拟合和二次拟合所用的伪代码如下：

Algorithm 3 linear fit

Input: $\vec{x} = (x_1, x_2...x_n)^T$, $\vec{y} = (y_1, y_2...y_n)^T$

Output: coefficients a, b

1: A $\leftarrow (\vec{x}\ 1)$

2: $(b, a)^T = (A^T A)^{-1} A^T y$

Algorithm 4 parabolic fit

Input: $\vec{x} = (x_1, x_2...x_n)^T$, $\vec{y} = (y_1, y_2...y_n)^T$

Output: coefficients a, b

1: A $\leftarrow (\vec{x^2}\ \vec{x}\ 1)$

2: $(c, b, a)^T = (A^T A)^{-1} A^T y$