Programming Report of Chapter 2

张志心 混合 2106

日期：2023 年 10 月 8 日

1 设计思路

1.1 多项式函数类

要使用多项式函数类，请调用相关库 `#include "function.hpp"`。

多项式函数类 `Poly_n<Type>` 继承于函数类 `Function<Type, Type>`。默认情况为 double → double 的多项式函数。多项式函数类包含的成员变量有两个，`int16_t n;` 和 `vector<Type> coef;` 分别表示多项式的次数加一的值（为了方便遍历多项式的系数）和系数。

```
template<class Type=double>class Poly_n: public Function<Type, Type>
```

除了默认构造以及拷贝构造方法之外，还定义两种多项式函数的构造方法，具体如下，分别用于临时指定次数的多项式以及特定多项式的构造。

```
explicit Poly_n(int16_t n) : n(n), coef(n) {}
explicit Poly_n(const vector<Type>& a) : n(a.size()), coef(a) {}
```

多项式函数类包含的特有方法有：多项式加、减、数乘运算，多项式乘法（采用朴素 $O(n^2)$ 复杂度做法）、获取导函数等。此外，为了方便使用多项式函数的表达式进行计算机绘图，还定义了如下三个函数，用于将多项式的表达式以字符串格式保存为可被 tikz、latex、python 识别的格式。

```
const std::string to_tikz() const;
const std::string to_latex() const;
const std::string to_python() const;
```

1.2 多项式插值

要使用插值的相关类或函数，请调用库 `#include "Interpolation.hpp"`，多项式插值类以及相关接口函数均已封装在命名空间 `namespace polynomial_interpolation` 中。

多项式插值类 `Polynomial_Interpolation<Type>` 为基本模版类，主要成员包括输入的点与点值序列（`vector<Type> x, y;`），差商表（`vector<Type> d_table;`）以及插值多项式的次数（`int16_t n;`）。请使用点与点值序列（用 `std::vector<Type>` 存储）来构造多项式插值类，构造方法如下：

```
Polynomial_Interpolation (const vector<Type> &x, const vector<Type> &y)
                         : x(x), y(y), n(x.size()-1) {}
```

使用 `Poly_n<Type> Polynomial_Interpolation::solve()` 来求解多项式插值函数。求解过程为，先计算差商表，然后根据下式计算插值函数。

$$p_n(x) = \sum_{k=0}^{n} [x_0, \cdots, x_k] f \pi_k(x), \qquad \pi_1(x) = 1, \pi_k(x) = \prod_{j=0}^{k-1} (x - x_j), k > 1$$

注意，由于插商表实际有用的位置只有对角线元素，且实际计算的时候可以通过数组滚动达到空间复杂度为 $O(n)$，因此项目中的差商表为一维线性表（`std::vector<Type>` 类型）。

计算差商表的方法由于插值类型的不同，有不同的定义，因此基类中 `void Polynomial_Interpolation::make_d_table()` 仅仅为一个纯虚函数，而在其子类中再进行明确定义。

1.2.1 Newton 插值

Newton 插值法为多项式插值类的子类，

```
template<class Type=NUM>
class Newton_Interpolation : public Polynomial_Interpolation<Type>;
```

调用示例子如下：

```
Newton_Interpolation sol(x, y);
Poly_n<NUM> res = sol.solve();
```

在计算差商表的时候，按照列的顺序进行计算，并且使用了滚动数组的方式，使得空间复杂度为 $O(n)$。差商表的递推公式为

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

```
void make_d_table() {
    d_table = y; // the first column is equal to the value array : y
    for(int i = 1; i <= n; ++i) {
        for(int j = n; j>=i ; --j) {
            if(x[j-i]==x[j]) d_table[j] = d_table[j]/i;
            else d_table[j]=(d_table[j]-d_table[j-1])/(x[j]-x[j-i]);
        }
    }
}
```

为牛顿插值添加了两类接口函数，用于求解均匀插值问题和切比雪夫插值问题。

```
template<class T=NUM>
const Poly_n<T> Grid_Interpolation(const T& l, const T& r,
                        const int16_t& n, const Function<T>& f) {
    vector<T> x(n+1), y(n+1);
    for(int i = 0; i <= n; ++i) x[i]=l+i*(r-l)/n, y[i]=f(x[i]);
    Newton_Interpolation<T> solver(x, y);
    return solver.solve();
}
template<class T=NUM>
const Poly_n<T> Chebyshev_Interpolation (const T& l, const T& r,
                        const int16_t& n, const Function<T>& f) {
    static const T Pi = acos(-1.); // 3.14159265358979323846 maybe more precise
    vector <T> x(n+1), y(n+1);
    for (int i = 0; i <= n; ++ i) x[i] = (l+r)/2+(r-l)/2*cos(Pi*(2*i+1)/(2*n+2)), y[i] = f(x
        [i]);
    Newton_Interpolation<T> solver(x, y);
    return solver.solve();
}
```

1.2.2 Hermite 插值

Hermite 插值法为多项式插值类的子类，

```
template<class Type=NUM>
class Hermite_Interpolation : public Polynomial_Interpolation<Type>;
```

Hermite 插值与 Newton 插值的构造方法相同，为了保证一致性，输入的点与点值序列 x, y 应当长度相同，对于 $f^(k)(x_i) = y_i^{(k)}, k = 0, \cdots, K$ 的限制应当以 $\{(x_i, y_i^{(k)})\}_{k=0}^{K}$ 的形式对应且连续的放入 x, y 中。

调用示例子如下：

```
Hermite_Interpolation sol(x, y);
Poly_n<NUM> res = sol.solve();
```

差商表的计算，要判断当前要求的 $[x_i, \cdots, x_j]f$ 中 $x_i, \cdots, x_j$ 是否都相同。如果是，则值为 $\frac{f^{(j-i)}(x_i)}{(j-i)!}$，否则等同于 Newton 插值法的计算公式。

```
void make_d_table() {
    vector<int16_t> start(n+1);
    d_table.resize(n+1);
    d_table[0] = y[0]; start[0] = 0;
    for(int i = 1; i <= n; ++i) {
        if(x[i] == x[i-1]) d_table[i] = d_table[i-1], start[i]=start[i-1];
        else d_table[i] = y[i], start[i] = i;
    }
    Type fac = 1.;
    for(int i = 1; i <= n; ++i, fac *= i) {
        for(int j = n; j>=i ; --j) {
            if(x[j-i]==x[j]) d_table[j] = y[start[j]+i]/fac;
            else d_table[j]=(d_table[j]-d_table[j-1])/(x[j]-x[j-i]);
        }
    }
}
```

2 求解结果

以下程序使用 `std::ostream << (const Poly_n<double> &p)const;` 输出多项式，其格式为：

$$\{\mathrm{n} = N, \mathrm{coef} = \{e_0, e_1, \cdots, e_{N-1}\}\},$$

其中 e_i 表示多项式 x^i 的系数。具体输出结果参考 `/tmp/result.txt`，以下仅展示处理后的输出结果。

2.1 问题 B

函数 $f(x) = \frac{1}{1+x^2}$，在 $[-5, 5]$ 区间上的均匀插值。其中 $x_i = -5 + 10\frac{i}{n}, n = 2, 4, 6, 8$。

对于该问题，使用 Newton 插值方法求解得到的函数如下：

$$p_2(x) = -0.0384615x^2 + 1$$

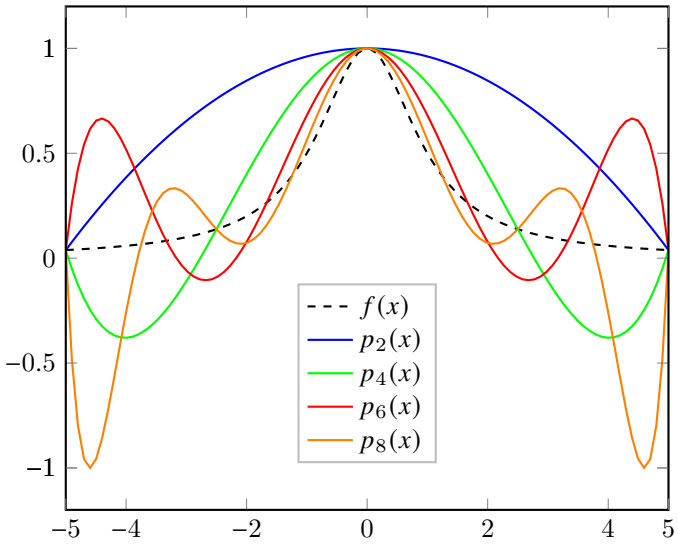
$$p_4(x) = 0.00530504x^4 - 0.171088x^2 + 1$$

$$p_6(x) = -0.000840633x^6 + 8.67362\mathrm{e}{-19}x^5 + 0.0335319x^4 + 2.08167\mathrm{e}{-17}x^3 - 0.351364x^2 + 2.77556\mathrm{e}{-17}x + 1$$

$$p_8(x) = 0.000137445x^8 - 0.00658016x^6 + 0.0981875x^4 - 6.93889\mathrm{e}{-17}x^3 - 0.528121x^2 + 2.77556\mathrm{e}{-17}x + 1$$

将 $f(x)$ 以及 Newton 插值的拟合结果 $p_2(x), p_4(x), p_6(x), p_8(x)$ 绘制图像如下：

Results of Problem B

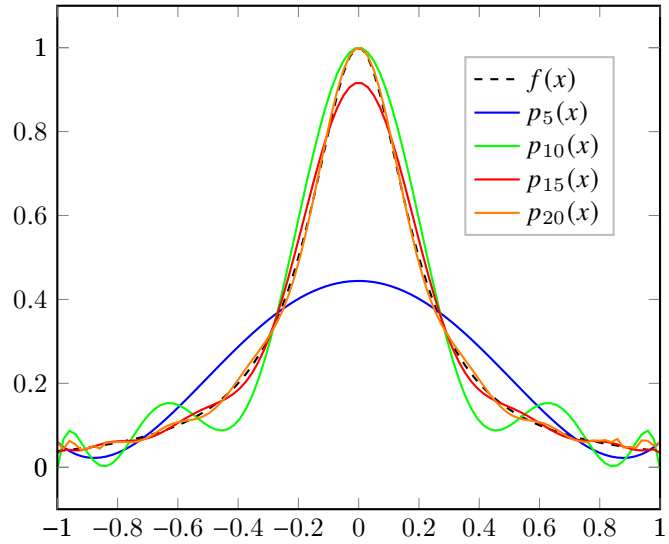


观察图像可以发现，随着 n 的增大，插值多项式结果在 $[-2, 2]$ 区间上的拟合效果越来越好，然而在 $|x| \geq 2$ 位置的振荡明显严重，产生 Runge 现象。

2.2 问题 C

函数 $f(x) = \dfrac{1}{1+25x^2}$，在 $[-1, 1]$ 区间上的切比雪夫插值。其中 $x_i = -1 + 2\dfrac{i}{n}, n = 5, 10, 15, 20$。
对于该问题，使用 Newton 插值方法求解得到的函数见 `/tmp/result.txt`。
将 $f(x)$ 以及 Newton 插值的拟合结果 $p_5(x), p_{10}(x), p_{15}(x), p_{20}(x)$ 绘制图像如下：

Results of Problem C



观察图像可以发现，随着 n 的增大，插值多项式的拟合效果越来越好，并且振荡幅度越来越微弱，Runge 现象被削弱。

2.3 问题 D

根据题意可知，待插值函数满足：

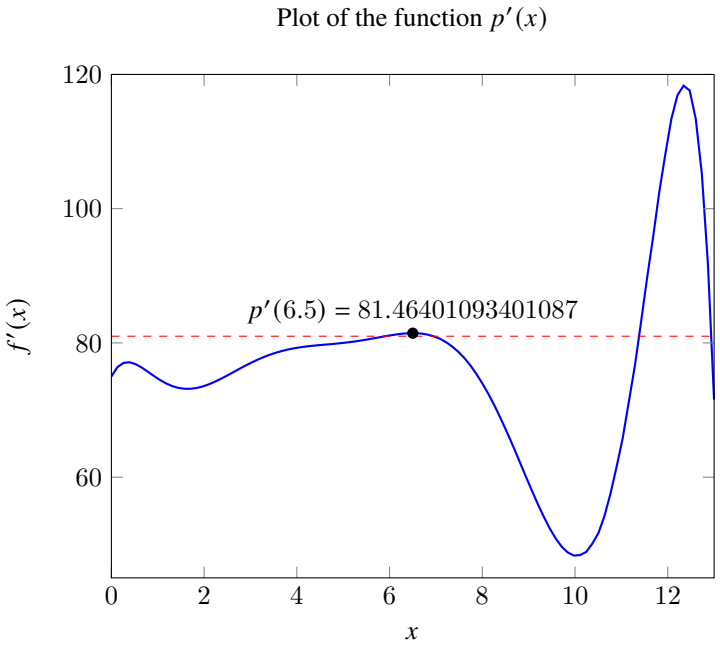
x	0	3	5	8	13
$f(x)$	0	225	383	623	993
$f'(x)$	75	77	80	74	72

将 $x = 0, 3, 5, 8, 13$ 处的函数值和导数值代入 Hermite 插值，得到结果为：

$$p(x) = -2.02236\text{e}{-05}x^9+0.00104059x^8-0.0218757x^7+0.243041x^6-1.5383x^5+5.50812x^4-10.0953x^3+7.16191x^2+75x$$

$$p'(x) = -0.000182013x^8+0.00832472x^7-0.15313x^6+1.45825x^5-7.69148x^4+22.0325x^3-30.2859x^2+14.3238x+75$$

1. 根据插值结果，$p(10) = 742.503, p'(10) = 48.3817$，分别为时间为 10 时位移与速度的估计值。
2. $p'(x)$ 的图像如下：



根据图像，$p'(x)$ 在 $x = 6.5$ 附近超过了限速 81 ft，而在 $x = 12$ 附近速度突增再突减，峰值远远超过了 81 ft。这显然不太可能与实际情况相符，可能是由于高次多项式的振荡。

这说明插值方法在个别点处的拟合结果的相对误差可能会很大。要进一步提高拟合精度，可能需要更多的点或者采用分段插值。

2.4 问题 E

根据已知信息，

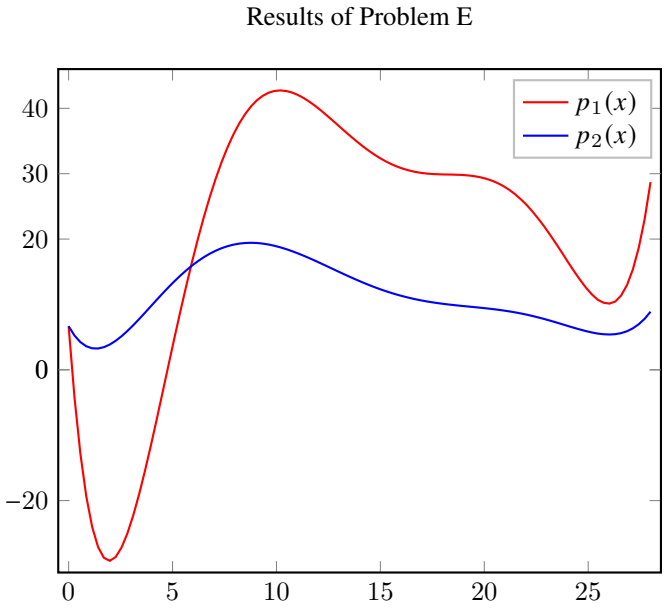
x	0	6	10	13	17	20	28
$f_1(x)$	6.67	17.3	42.7	37.3	30.1	29.3	28.7
$f_2(x)$	6.67	16.1	18.9	15.0	10.6	9.44	8.89

使用 Newton 插值，得到结果为：

$$p_1(x) = 4.1477\text{e}{-05}x^6 - 0.00371557x^5 + 0.128281x^4 - 2.11512x^3 + 16.2855x^2 - 43.0127x + 6.67$$

$$p_2(x) = 8.6768\text{e}{-06}x^6 - 0.000777473x^5 + 0.0265858x^4 - 0.424283x^3 + 2.98227x^2 - 5.85018x + 6.67$$

1. 插值结果 $p_1(x), p_2(x)$ 的图像如下：



   $f_1(x)$ 的插值结果 $p_1(x)$ 在 $x = 2$ 附近存在较大振荡，导致函数值出现在负数区域，与实际情况不符合。可能原因是 $f_1(x)$ 的已知点值的变换幅度过大，要更准确的拟合函数，可能需要更高次的函数（因此需要更多的点）。$f_2(x)$ 的插值结果与实际走势基本吻合。

2. 根据插值结果估计 15 天后的两类幼虫质量为 $p_1(43) = 14640.3, p_2(43) = 2981.48$，这个值远高于实际可能的情况。可以发现两个函数在 $x \geq 25$ 之后都有快速增长的趋势，这是因为提供的用于插值的点都在 $[0, 28]$ 范围内，对插值范围之外的点的估计不可靠。由于两个函数都是一个最高次项系数大于 0 的六次多项式，因此当 x 远离插值范围后，都会快速增长。

参考文献

[1] 张庆海. “Notes on Numerical Analysis and Numerical Methods for Differential Equations”. In: (2023).