第 1 周作业

李子龙 518070910095

2020 年 9 月 14 日

目录

1 公共库：多项式类 `Polynomial`

任何一个实函数F都可以通过插值多项式的方法得到一个拟合的还不错的多项式，所以在这里为了简化我们的问题，就直接使用多项式函数求解问题。

当然，如果需要使用真正的超越方程，可以使用类似于下面的代码创建内联函数，提高调用效率：

```
    inline double f(double x){ return x*exp(x)-1; }
```

对于一个多项式，用户依次输入幂次、各项系数，就可以构建一个 `Polynomial` 多项式类，并由此来创建新的头文件库 LIB polynomial.h。

```
#ifndef POLY
#define POLY

//   檫簣檫瞉
class Polynomial {
private:
    int         n;          // 鬖鬖
    double*     coeff;      // 瞉鬖
public:
    // constructor
    Polynomial(){
        std::cout << "Please␣input␣the␣power␣of␣polynomial:";
        std::cin >> n;
        coeff = new double[n + 1];
        std::cout << "Please␣input␣the␣coefficients␣of␣polynomial(start␣
           from␣the␣constant):";
        for(int i = 0; i <= n; ++i)
            std::cin >> coeff[i];
    }

    //    檫檫檫鬖檫
```

```
    double getValue(double x){
        double value = 0;
        double item = 1;
        for (int i = 0; i<= n; ++i){
            value += coeff[i] * item;
            item *= x;
        }
        return value;
    }

    //  檫檫檫鬘檫
    double getDerivValue(double x){
        double derivValue = 0;
        double item = 1;
        for (int i = 1; i<= n; ++i){
            derivValue += i * coeff[i] * item;
            item *= x;
        }
        return derivValue;
    }

    // distructor
    ~Polynomial(){
        delete[] coeff;
    }
};


#endif
```

2 问题 1: Romberg 积分法

对于给定的函数f与区间$[a, b]$，为了计算积分$\int_a^b f(x)\mathrm{d}x$，根据 Wikipedia 中关于 Romberg 积分法的描述，可以得到关于该积分法的以下递推公式：

$$h_n = \frac{1}{2^n}(b - a)$$

$$
\begin{aligned}
R(0,0) &= h_1(f(a)+f(b)) \\
R(n,0) &= \frac{1}{2}R(n-1,0) + h_n \sum_{k=1}^{2^{n-1}} f(a+(2k-1)h_n) \\
R(n,m) &= \frac{1}{4^m-1}\left(R(n,m-1) - R(n-1,m-1)\right) \\
&\text{where } n \geq m \text{ and } m \geq 1
\end{aligned}
$$

定义了一个 `Romberg` 类，使用动态规划的方式存储之前的数据，最后输出 $R(n,n)$ 作为结果。

用户输入两个数据，程序会使用 Romberg 积分法输出。

由于计算多项式的值速度会比较慢，所以增加一个进度显示。

```
// Week 1 - Problem 1:
// Romberg's Integral Method

#include "../std_lib_facilities.h"
#include "polynomial.h"

//    鬘叿毜叿毜檫
void showProcess(double process){
    cout << "\rCalculating␣Process:" << setw(4) << process << "%";
    fflush(stdout);
}

/// Romberg  叿檫叿
class Romberg{
private:
    int         n;                          //   鬘檫叿檫鬘
    double*     h;                          //   叿檫簣毜叿簣
    double      a;                          //  檫簣毜檫
    double      b;                          //  檫簣叿鬘
    double**    R;                          //  叿檫叿簣
    int*        BIN_P;                      //2  鬘檫毜
    int*        QUA_P;                      //4  鬘檫毜
    Polynomial* poly;                       // 檫簣檫

    // calculate parameter list h
    void calcList_h(){
        h[0] = b - a;
        for (int i = 1; i <= n; ++i)
            h[i] = h[i-1] / 2;
    }

```

```
    void calcListBIN_P(){
        BIN_P[0] = 1;
        for (int i = 1; i < n; ++i)
            BIN_P[i] = BIN_P[i-1] * 2;
    }

    void calcListQUA_P(){
        QUA_P[0] = 1;
        for (int i = 1; i <= n; ++i)
            QUA_P[i] = QUA_P[i-1] * 4;
    }

    double integral(){
        float process;
        // First, calculate R(n,0)
        R[0][0] = h[1] * (poly->getValue(a) + poly->getValue(b));
        for(int i = 1; i <= n; ++i){
            double sum = 0;
            for(int k = 1; k<= BIN_P[i-1]; ++k)
                sum += poly->getValue(a + (2*k - 1)*h[i]);
            R[i][0] = 0.5 * R[i-1][0] + h[i] * sum;
            showProcess(50.0 * i / n);
        }

        // Then, calculate R(n,m)
        for(int i = 1; i<= n; ++i){
            for(int j = 1; j <= i; ++j)
                R[i][j] = R[i][j-1] + 1.0 / (QUA_P[j] - 1) * (R[i][j-1] -
                    R[i-1][j-1]);
            showProcess(50 + 50.0 * i / n);
        }
        putchar('\n');

        return R[n][n];
    }

public:

    // constructor
    Romberg(Polynomial* polynomial, int maxDivsionNum){
        n = maxDivsionNum;
        h = new double[n+1];

        BIN_P = new int[n];
```

```
        calcListBIN_P();

        QUA_P = new int[n+1];
        calcListQUA_P();

        R = new double*[n+1];
        for(int i = 0; i <= n; ++i)
            R[i] = new double[n+1];

        poly = polynomial;
    }

    // distructor
    ~Romberg(){
        delete[] h;
        for(int i = 0; i <= n; ++i)
            delete[] R[i];
        delete[] R;
        delete[] BIN_P;
        delete[] QUA_P;
    }

    double Integral(int intervalStart, int intervalEnd){
        a = intervalStart;
        b = intervalEnd;
        calcList_h();
        return integral();
    }
};

int main(){
    Polynomial* poly = new Polynomial();

    int n;
    cout << "Please␣input␣the␣fineness␣of␣integral:";
    cin >> n;

    Romberg r1(poly,n);

    double a,b;
    cout << "Please␣input␣the␣start␣and␣end␣of␣intergral␣interval:";
    cin >> a >> b;

    cout << r1.Integral(a,b);
```

```
    return 0;
}
```