第 1 周作业

LogCreative

2020 年 12 月 29 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

// 多项式类
class Polynomial {
private:
    int        n;         //次数
    double*    coeff;     //系数
public:
    // constructor
    Polynomial(){
        std::cout << "Please input the power of polynomial:";
        std::cin >> n;
```

```
14          coeff = new double[n + 1];
15          std::cout << "Please input the coefficients of polynomial(start
                from the constant):";
16          for(int i = 0; i <= n; ++i)
17              std::cin >> coeff[i];
18      }
19
20      // 得到函数值
21      double getValue(double x){
22          double value = 0;
23          double item = 1;
24          for (int i = 0; i<= n; ++i){
25              value += coeff[i] * item;
26              item *= x;
27          }
28          return value;
29      }
30
31      // 得到导数值
32      double getDerivValue(double x){
33          double derivValue = 0;
34          double item = 1;
35          for (int i = 1; i<= n; ++i){
36              derivValue += i * coeff[i] * item;
37              item *= x;
38          }
39          return derivValue;
40      }
41
42      // distructor
43      ~Polynomial(){
44          delete[] coeff;
45      }
46  };
47
48
49  #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：

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

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

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"

// 显示计算进度
void showProcess(double process){
    cout << "\rCalculating Process:" << setw(4) << process << "%";
    fflush(stdout);
}

/// Romberg 积分类
class Romberg{
private:
    int        n;                    //最大细分数
    double*    h;                    //细分间距矩阵
    double     a;                    //区间起始
    double     b;                    //区间终止
    double**   R;                    //积分矩阵
    int*       BIN_P;                //2次幂表
    int*       QUA_P;                //4次幂表
    Polynomial* poly;                //多项式

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

```

```
32      void calcListBIN_P(){
33          BIN_P[0] = 1;
34          for (int i = 1; i < n; ++i)
35              BIN_P[i] = BIN_P[i-1] * 2;
36      }
37
38      void calcListQUA_P(){
39          QUA_P[0] = 1;
40          for (int i = 1; i <= n; ++i)
41              QUA_P[i] = QUA_P[i-1] * 4;
42      }
43
44      double integral(){
45          float process;
46          // First, calculate R(n,0)
47          R[0][0] = h[1] * (poly->getValue(a) + poly->getValue(b));
48          for(int i = 1; i <= n; ++i){
49              double sum = 0;
50              for(int k = 1; k<= BIN_P[i-1]; ++k)
51                  sum += poly->getValue(a + (2*k - 1)*h[i]);
52              R[i][0] = 0.5 * R[i-1][0] + h[i] * sum;
53              showProcess(50.0 * i / n);
54          }
55
56          // Then, calculate R(n,m)
57          for(int i = 1; i<= n; ++i){
58              for(int j = 1; j <= i; ++j)
59                  R[i][j] = R[i][j-1] + 1.0 / (QUA_P[j] - 1) * (R[i][j-1] - R
                        [i-1][j-1]);
60              showProcess(50 + 50.0 * i / n);
61          }
62          putchar('\n');
63
64          return R[n][n];
65      }
66
67  public:
68
69      // constructor
70      Romberg(Polynomial* polynomial, int maxDivsionNum){
71          n = maxDivsionNum;
72          h = new double[n+1];
73
74          BIN_P = new int[n];
```

```
75          calcListBIN_P();
76
77          QUA_P = new int[n+1];
78          calcListQUA_P();
79
80          R = new double*[n+1];
81          for(int i = 0; i <= n; ++i)
82              R[i] = new double[n+1];
83
84          poly = polynomial;
85      }
86
87      // distructor
88      ~Romberg(){
89          delete[] h;
90          for(int i = 0; i <= n; ++i)
91              delete[] R[i];
92          delete[] R;
93          delete[] BIN_P;
94          delete[] QUA_P;
95      }
96
97      double Integral(int intervalStart, int intervalEnd){
98          a = intervalStart;
99          b = intervalEnd;
100         calcList_h();
101         return integral();
102     }
103 };
104
105 int main(){
106     Polynomial* poly = new Polynomial();
107
108     int n;
109     cout << "Please input the fineness of integral:";
110     cin >> n;
111
112     Romberg r1(poly,n);
113
114     double a,b;
115     cout << "Please input the start and end of intergral interval:";
116     cin >> a >> b;
117
118     cout << r1.Integral(a,b);
```

```
    return 0;
}
```

```
Please input the power of polynomial:2
Please input the coefficients of polynomial(start from the
↪  constant):-1 2 5
Please input the fineness of integral:1000
Please input the start and end of intergral interval:1 5
Calculating Process: 100%
226.667
```

3 问题 2: 求解函数零点

3.1 二分法

函数使用公共的头文件输入。

定义了一个函数 FzeroSolver 用于寻找函数零点，为了简化问题，我们只对输入区间两端函数值为异号的情况进行计算，否则会抛出异常 SameSymbolException 。(Matlab 的 fzero 函数也是这么做的。)

当超过迭代次数的时候，将会抛出异常 OutOfIterationLimitException，并显示超出迭代次数。

3.2 牛顿法

也使用了牛顿法对零点进行计算，使用端点值作为迭代初值用于计算。

```
// Week 1 - Problem 2:
// Solving zero of the function

// By using Binary Division
// The polynomial interpretation

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

#define ITER_THES 2000      // 迭代次数阈值
#define ACCURACY 1e-6       // 精确度

// 区间两端值相等
class SameSymbolException {
public:
    SameSymbolException():
        message("The symbol of the function values of each side should be different.\
    const char* what() const {return message;}
```

```
private:
    const char* message;
};

// 超出迭代次数上限
class OutOfIterationLimitException {
public:
    OutOfIterationLimitException():
        message("Out of iteration limit.\n") {}
    const char* what() const {return message;}
private:
    const char* message;
};

double BinaryZeroSolver(Polynomial* poly,double a, double b){
    double left = a, right = b, mid;
    double fleft = poly->getValue(left),
           fright = poly->getValue(right),
           fmid;
    try{
        if(fleft * fright>0)
            throw SameSymbolException();

        int iter = 0;
        do {
            mid = left + (right - left) / 2;

            fleft = poly->getValue(left);
            fmid = poly->getValue(mid);
            fright = poly->getValue(right);

            if(fmid == 0) return mid;
            if(fleft * fmid < 0) right = mid;
            else left = mid;

            if(++iter>ITER_THES)
                throw OutOfIterationLimitException();
        } while(right-left>ACCURACY);
        return left + (right - left) / 2;
    }
    catch(SameSymbolException ex){
        cout << ex.what();
    }
    catch(OutOfIterationLimitException ex){
```

```
        cout << ex.what();
    }
}

// Newton's method:
// x[k+1] = x[k] - f(x[k]) / fprime(x[k])
double NewtonZeroSolver(Polynomial *poly, double initialValue){
    double xprev, xfwd  = initialValue;
    int iteration = 0;
    try{
        do{
            xprev = xfwd;
            xfwd = xprev - poly->getValue(xprev) / poly->getDerivValue(xprev);
            if (++iteration>ITER_THES)
                throw OutOfIterationLimitException();
        } while(abs(xfwd - xprev)>2e-5);
        return xfwd;
    }
    catch(OutOfIterationLimitException ex){
        cout << ex.what();
    }
}

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

    double intStart, intEnd;
    cout << "Please input the solving interval:";
    cin >> intStart >> intEnd;
    cout << "The zero of function(Binary Method):";
    cout << BinaryZeroSolver(poly, intStart, intEnd) << endl;
    cout << "The zero of function(Newton's Method):";
    cout << NewtonZeroSolver(poly, intStart) << endl;
    return 0;
}
```

```
Please input the power of polynomial:2
Please input the coefficients of polynomial(start from the
↪  constant):-11 1 5
Please input the solving interval:1 5
The zero of function(Binary Method):1.38661
The zero of function(Newton's Method):1.38661
```

```
Please input the power of polynomial:2
Please input the coefficients of polynomial(start from the
↪  constant):1 1 5
Please input the solving interval:1 5
The zero of function(Binary Method):The symbol of the function values
↪  of each side should be different.
nan
The zero of function(Newton's Method):Out of iteration limit.
nan
```