連続系アルゴリズム
レポート課題4

連絡先：mail@myuuuuun.com

平成27年11月12日

レポート内で使用したプログラムは GitHub にアップロードしています。 `https://github.com/myuuuuun/various/tree/master/ContinuousAlgorithm/HW4/`

問題 1

区間 $I = [1, 2]$ で $\frac{1}{x}$ を最適に近似する 1 次式 $p(x) = ax + b$ を求めよ。ただし、近似式は以下の条件をみたす。

- 誤差関数 $E(x) = \frac{1}{x} - p(x)$ を最小にする
- $E(x)$ は I 上の 3 点で極値をもつ。そのうち 2 つは両端点。その 3 箇所の絶対値は等しく、符号は交互になっている

端点の関数値が一致するので、

$$
\begin{aligned}
1 - a - b &= \frac{1}{2} - 2a - b \\
\therefore a &= -\frac{1}{2}
\end{aligned}
$$

このとき、

$$
E(1) = E(2) = \frac{3}{2} - b
$$

となる。

E(x) の一階条件から、

$$
\begin{aligned}
-\frac{1}{x^2} - a &= 0 \\
ax^2 &= -1 \\
\therefore x &= \sqrt{2}
\end{aligned}
$$

よって、

$$
\begin{aligned}
E(\sqrt{2}) = \frac{1}{\sqrt{2}} + \frac{\sqrt{2}}{2} - b &= b - \frac{3}{2} \\
\therefore b &= \frac{3 + 2\sqrt{2}}{4}
\end{aligned}
$$

問題2

$x = 0.5$ から 1.0 まで 0.1 刻みで次の関数の値を計算し、配列に格納するプログラムを書け。

- $1 + x + x^2 + \ldots + x^n$（n=2 から 7 まで、Horner 法で）
- $exp(x)$ および $log(x)$

上記の値を参照し、ネヴィルの算法で多項式補間を行って、$x = 0.75$ における値を求めよ。また、もとの関数値と補間値との差を求めよ。

結果:

- Horner(n=2): 近似値 2.3125, 真値との差 0
- Horner(n=3): 近似値 2.73437, 真値との差 4.44089e-16
- Horner(n=4): 近似値 3.05078, 真値との差 4.44089e-16
- Horner(n=5): 近似値 3.28809, 真値との差 8.88178e-16
- Horner(n=6): 近似値 3.46607, 真値との差 3.51563e-06
- Horner(n=7): 近似値 3.59957, 真値との差 2.19727e-05
- exp x: 近似値 2.117, 真値との差 1.03531e-08
- log x: 近似値-0.287686, 真値との差 3.71795e-06

ソースコード: hw4-1.cpp

```
#include <iostream>
#include <complex>
#include <array>
#include <vector>
#include <cmath>
#include <limits>
const double EPS = 1.0e-12;
using namespace std;
using vector2d = vector< vector<double> >;


// exp(value) を0 周りでtaylor 展開し求める
double taylor_exp(double value){
  double t_sum = 0;

  int sec_max = 100;
  double section_value;
  array<double, 101> sec_val_list;
  sec_val_list[0] = 1.0;
```

```
  for(int i=1; i<=100; i++){
    // exp のtaylor 展開の性質を利用
    section_value = sec_val_list[i-1] * value / i;
    sec_val_list[i] = section_value;
    if(abs(section_value) < EPS
        || section_value == std::numeric_limits<double>::infinity()){
      sec_max = i;
      break;
    }
  }

  for(int i=sec_max; i>=0; --i){
    t_sum += sec_val_list[i];
  }

  return t_sum;
}


// log(value) を1 周りでtaylor 展開し求める
// あまりvalue が1 から離れると精度が無い
double taylor_log(double value){
  double t_sum = 0;

  int sec_max = 100;
  double section_value;
  array<double, 101> sec_val_list;

  // 分子
  double numer = 1;

  for(int i=0; i<=100; i++){
    // log のtaylor 展開の性質を利用
    numer *= value - 1;
    section_value = pow(-1.0, i%2) * numer / (i+1.0);
    sec_val_list[i] = section_value;
    if(abs(section_value) < EPS
        || section_value == std::numeric_limits<double>::infinity()){
      sec_max = i;
      break;
    }
  }

  for(int i=sec_max; i>=0; --i){
    t_sum += sec_val_list[i];
  }

  return t_sum;
}


// Horner 法でべき表現多項式（a_0 + a_1 x + a_2 x^2 + ... ）の値を求める
double horner(double value, vector<double> coefs){
  double sum = 0;
  for(vector<double>::reverse_iterator
        it = coefs.rbegin(); it != coefs.rend(); ++it){
    sum = sum * value + (*it);
```

```
  }

  return sum;
}


// ネヴィルの算法
double neville(double x, vector2d points){
  int size = points[0].size();
  vector2d table(size*size, vector<double>(size));

  for(int i=0; i<size; i++){
    table[i][0] = points[i][1];
    for(int j=1; j<i+1; j++){
      table[i][j] =
        ((points[i][0] - x) * table[i-1][j-1]
                - (points[i-j][0] - x) * table[i][j-1])
        / (points[i][0] - points[i-j][0]);
    }
  }

  return table[size-1][size-1];
}


vector2d make_points_horner(vector<double> x_list, vector<double> coefs){
  int size = x_list.size();
  vector2d points(size*2, vector<double>(size));

  for(int i=0; i<size; i++){
    points[i][0] = x_list[i];
    points[i][1] = horner(x_list[i], coefs);
  }

  return points;
}


int main(){
  vector<double> x_list;
  vector<double> coefs;

  // x_list を定義
  for(int i=0; i<6; i++){
    x_list.push_back(0.5 + 0.1 * i);
  }

  // 1+x+x^2
  for(int i=0; i<3; i++){
    coefs.push_back(1);
  }
  vector2d points = make_points_horner(x_list, coefs);
  cout << 近似値"" << neville(0.75, points)
    << ", 真値との差" << abs(horner(0.75, coefs) - neville(0.75, points)) << endl;

  // +x^3
  coefs.push_back(1);
  points = make_points_horner(x_list, coefs);
```

```
  cout << 近似値"" << neville(0.75, points)
    << ", 真値との差" << abs(horner(0.75, coefs) - neville(0.75, points)) << endl;

  // +x^4
  coefs.push_back(1);
  points = make_points_horner(x_list, coefs);
  cout << 近似値"" << neville(0.75, points)
    << ", 真値との差" << abs(horner(0.75, coefs) - neville(0.75, points)) << endl;

  // +x^5
  coefs.push_back(1);
  points = make_points_horner(x_list, coefs);
  cout << 近似値"" << neville(0.75, points)
    << ", 真値との差" << abs(horner(0.75, coefs) - neville(0.75, points)) << endl;

  // +x^6
  coefs.push_back(1);
  points = make_points_horner(x_list, coefs);
  cout << 近似値"" << neville(0.75, points)
    << ", 真値との差" << abs(horner(0.75, coefs) - neville(0.75, points)) << endl;

  // +x^7
  coefs.push_back(1);
  points = make_points_horner(x_list, coefs);
  cout << 近似値"" << neville(0.75, points)
    << ", 真値との差" << abs(horner(0.75, coefs) - neville(0.75, points)) << endl;

  // e^x
  vector2d points2(6*2, vector<double>(6));
  for(int i=0; i<7; i++){
    points2[i][0] = x_list[i];
    points2[i][1] = taylor_exp(x_list[i]);
  }
  cout << 近似値"" << neville(0.75, points2)
    << ", 真値との差" << abs(exp(0.75) - neville(0.75, points2)) << endl;

  // logx
  for(int i=0; i<7; i++){
    points2[i][0] = x_list[i];
    points2[i][1] = taylor_log(x_list[i]);
  }
  cout << 近似値"" << neville(0.75, points2)
    << ", 真値との差" << abs(log(0.75) - neville(0.75, points2)) << endl;

  return 0;
}
```