連続系アルゴリズム
レポート課題1

経済学部金融学科3年
07-152042　松下 旦
連絡先：mail@myuuuuun.com

平成27年10月26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/HW1/

問題 1

実数 x に対して $y = cos(x)$ を浮動小数点数で計算した時に、y に含まれる誤差を見積もれ。

※ x を浮動小数点で表現する時の誤差と $cos(x)$ を浮動小数点数で表現する時の誤差だけを考える。

x に δ 程度の誤差が含まれている時、$y = cos(x)$ に伝播する誤差を見積もる。2 乗の項まで 0 周りで Taylor 展開すると、

$$
\begin{aligned}
|cos(x+\delta) - cos(x)| &= \left| \left\{ 1 - sin(\delta)(x+\delta) - cos(\delta)\frac{(x+\delta)^2}{2} + O(x^3) \right\} - \left\{ 1 - \frac{x^2}{2} + O(x^3) \right\} \right| \\
&\approx |x\delta| + \frac{\delta^2}{2}
\end{aligned}
$$

となる。ただし $\delta \simeq 0$ を想定しているので、$sin(\delta) \approx 0$, $cos(\delta) \approx 1$ となることを用いた。

浮動小数点形式の表現誤差（相対誤差）を ϵ とおくと、ある浮動小数点数 r に含まれる絶対誤差は $\epsilon|r|$ 程度になるので、y に含まれる誤差は結局、

$$
\begin{aligned}
\Big| cos(x + \epsilon|x|) + \epsilon \Big| cos(x + \epsilon|x|) \Big| - cos(x) \Big| &= \Big| cos(x + \epsilon|x|) - cos(x) \Big| + \epsilon \Big| cos(x + \epsilon|x|) \Big| \\
&\simeq \epsilon x^2 + \frac{\epsilon^2 x^2}{2} + \epsilon \Big| cos(x + \epsilon|x|) \Big| \\
&\simeq \epsilon x^2 + \epsilon \Big| cos(x + \epsilon|x|) \Big| \\
&\leq \epsilon x^2 + \epsilon
\end{aligned}
$$

と見積もることが出来る。ただし、ϵ^2 が十分に小さいことを用いた。

$|x|$ が大きい時、$cos(x)$ の精度が下がるのはなぜか。

倍精度の場合、$\epsilon = 2^{-52} \simeq 10^{-16}$ 程度であるので、$|x|$ がの整数部が 6 桁程でも計算結果に大きく影響することが予想できる。この誤差を解消するためには

$$
x \quad \leftarrow \quad x \ \ mod\ 2\pi
$$

として（← は代入）、先に 2π の定数倍を引いてから計算すれば良い。

$|x|$ が 0 に近い時、$1 - cos(x)$ の相対精度が下がるのはなぜか。

$|x| \simeq 0$ の時、$cos(x) \simeq 1$ となるので、$1 - cos(x)$ を計算する際に桁落ちが生じる。これを防ぐためには、

$$1 - cos(x) = 2sin^2\left(\frac{x}{2}\right)$$

として、右辺を計算すれば良い。

問題 2

半径 r の 2 円が等速直線運動をしている時、衝突時刻を求めるプログラムを書け。

半径 r, 2 円の初期位置（x 座標, y 座標）, 2 円の移動速度（x 方向, y 方向）を与え、衝突時間を求めるプログラムが以下。

```
#include <iostream>
#include <array>
#include <cmath>
#include <limits>
using namespace std;


// ax^2 + bx + c = 0 の形の二次方程式を解く
template <class T>
array<T, 2> quadratic(T a, T b, T c){
  T x1, x2;
  T d;
  array<T, 2> rst;

  // 判別式D < なら0を返すnan
  d = pow(b, 2) - 4 * a * c;
  if(d < 0){
    rst[0] = numeric_limits<T>::quiet_NaN();
    rst[1] = numeric_limits<T>::quiet_NaN();
    return rst;
  }

  // 桁落ち防止
  if(b < 0){
    x1 = (-1 * b + sqrt(d)) / (2 * a);
  }
  else{
    x1 = (-1 * b - sqrt(d)) / (2 * a);
  }

  x2 = c / (a * x1);
  rst[0] = x1;
```

```
  rst[1] = x2;

  return rst;
}


int main(){
  double r, x1, y1, x2, y2, vx1, vy1, vx2, vy2;
  double x, y, vx, vy;

  r = 1.0;

  x1 = 10;
  y1 = 30;
  vx1 = 1.0;
  vy1 = 1.0;

  x2 = 10;
  y2 = 0;
  vx2 = -0.1;
  vy2 = 0;

  x = x1 - x2;
  y = y1 - y2;
  vx = vx1 - vx2;
  vy = vy1 - vy2;

  array<float, 2> clash_time_f
        = quadratic((float)(vx*vx+vy*vy),
                (float)(2*(x*vx+y*vy)), (float)(x*x+y*y-4*r*r));
  array<double, 2> clash_time_d
        = quadratic(vx*vx+vy*vy, 2*(x*vx+y*vy), x*x+y*y-4*r*r);

  cout << "x1, y1, vx1, vy1: " << x1 <<
        ", " << y1 << ", " << vx1 << ", " << vy1 << endl;
  cout << "x2, y2, vx2, vy2: " << x2 <<
        ", " << y2 << ", " << vx2 << ", " << vy2 << endl;

  if(isnan(clash_time_d[0])){
    cout << "clash time: " << "NaN" << endl;
  }
  else if(clash_time_d[0] < 0 && clash_time_d[1] < 0){
    cout << "clash time: " << "NaN" << endl;
  }
  else{
    if(clash_time_d[0] < 0 || clash_time_d[1] < clash_time_d[0]){
      cout << "clash time(double): " << clash_time_d[1]
        << "clash time(float): " << clash_time_f[1] << endl;
    }
    else{
      cout << "clash time(double): " << clash_time_d[0]
        << "clash time(float): " << clash_time_f[0] << endl;
    }
  }

  return 0;
}
```