連続系アルゴリズム
レポート課題2

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11月2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/HW2/

問題 1

a を実数定数として、次の $f(x) = 0$ を解くニュートン法の式を導け。

- $f(x) = \frac{1}{x} - a$
- $f(x) = \frac{1}{x^2} - a$

(1) $x := 2x - ax^2$

(2) $x := \frac{3}{2}x - \frac{a}{2}x^3$

となる。

$x^3 - 1 = 0$ の根を求めるプログラムを書け。$-1 \leq real \leq 1$, $-1 \leq imag \leq 1$ をみたす初期値に対して、反復回数を求めよ。

ソースコード（主要部のみ）: hw2-1.cpp

```
#include <iostream>
#include <complex>
#include <array>
#include <vector>
#include <cmath>
#include <limits>
#include <fstream>
const double EPS = 1.0e-12;
const double PI = 3.1415926535;
using namespace std;

// 自動微分
template <class Type> class Dual
{

public:
  Type value, diff;
  Dual();
  Dual(Type v);
  Dual(Type v, Type d);
  Type get_value();
  Type get_diff();
  void set_value(Type v);
  void set_diff(Type d);

  template <class SType> Dual<Type>& operator=(SType x);
  Dual<Type>& operator=(const Dual<Type> &d1);
```

```
  template <class SType> bool operator==(const SType &x);
  bool operator==(const Dual<Type> &d1);
  template <class SType> bool operator!=(const SType &x);
  bool operator!=(const Dual<Type> &d1);

  template <class SType> Dual<Type> operator+(const SType &x);
  template <class SType> Dual<Type> operator-(const SType &x);
  template <class SType> Dual<Type> operator*(const SType &x);
  template <class SType> Dual<Type> operator/(const SType &x);
  template <class SType> Dual<Type>& operator+=(const SType &x);
  template <class SType> Dual<Type>& operator-=(const SType &x);
  template <class SType> Dual<Type>& operator*=(const SType &x);
  template <class SType> Dual<Type>& operator/=(const SType &x);

  Dual<Type> operator+(const Dual<Type> &d1);
  Dual<Type> operator-(const Dual<Type> &d1);
  Dual<Type> operator*(const Dual<Type> &d1);
  Dual<Type> operator/(const Dual<Type> &d1);
  Dual<Type>& operator+=(const Dual<Type> &d1);
  Dual<Type>& operator-=(const Dual<Type> &d1);
  Dual<Type>& operator*=(const Dual<Type> &d1);
  Dual<Type>& operator/=(const Dual<Type> &d1);
};

// コンストラクタ
template <class Type> Dual<Type>::Dual(){value = 0, diff = 0;}
template <class Type> Dual<Type>::Dual(Type v){value = v, diff = 0;}
template <class Type> Dual<Type>::Dual(Type v, Type d){value = v, diff = d;}

template <class Type1, class Type2>
Dual<Type1> pow(const Dual<Type1> d1, const Dual<Type2> d2){
  Dual<Type1> d3;
  d3.value = pow(d1.value, d2.value);
  d3.diff = d3.value * (d2.diff * log(d1.value) + d2.value * d1.diff / d1.value);
  return d3;
}

template <class Type>
Dual<Type> pow(const Dual<Type> d1, const double n){
  Dual<Type> d2;
  d2.value = pow(d1.value, n);
  d2.diff = n * pow(d1.value, n-1) * d1.diff;
  return d2;
}

template <class Type>
Dual<Type> pow(const double n, const Dual<Type> d1){
  Dual<Type> d2;
  d2.value = pow(n, d1.value);
  d2.diff = log(n) * d1.diff * d2.value;
  return d2;
}

template <class Type>
Dual<Type> sqrt(const Dual<Type> d1){
  return pow(d1, 0.5);
}
```

```
template <class Type>
Dual<Type> log(const Dual<Type> d1){
  Dual<Type> d2;
  d2.value = log(d1.value);
  d2.diff = d1.diff / d2.value;
  return d2;
}


using compd = complex<double>;
using vector2d = vector< vector<double> >;

template <class Type>
array<Type, 2> newton(Dual<Type> (*func)(Dual<Type>),
        Type initial_v, const double epsilon = EPS){
  Type new_v;
  Dual<Type> func_d, initial_d;

  // newton main loop
  int count = 0;
  while(true){
    count++;
    initial_d.value = initial_v;
    initial_d.diff = 1;
    func_d = func(initial_d);

    new_v = initial_v - func_d.value / func_d.diff;
    // isinf の判定に絶対値を入れて複素数に対応
    if(abs(initial_v - new_v) < epsilon || isinf(abs(new_v))){
      break;
    };
    initial_v = new_v;
  }
  array<Type, 2> result={new_v, 0};
  result[1] = compd(count, 0);

  return result;
}

inline Dual<compd> newton_f(const Dual<compd> x){
  compd c1(1, 0);
  return pow(x, 3) - c1;
}

// 浮動小数点数num の小数第place 位未満を切り捨てる
template <class Type>
Type dec_floor(Type num, int place){
  num *= pow(10, place);
  num = floor(num);
  num /= pow(10, place);
  return num;
}

int main(int argc, char *argv[])
{
  const int row = 1000, col = 1000;
  double real, imag;
```

```
  vector2d result((row+1)*(col+1), vector<double>(4));
  compd value;
  array<compd, 2> values;
  int current;
  cout.precision(5);

  // 適当な初期値に対して法を実行Newton
  for(int i=0; i<=row; i++){
    for(int j=0; j<=col; j++){
      Dual<compd> rst;
      real = (i - row/2) * 0.002;
      imag = (j - col/2) * 0.002;
      compd initial(real, imag);

      values = newton(newton_f, initial, EPS);
      value = values[0];

      current = i * (col+1) + j;

      result[current][0] = real;
      result[current][1] = imag;

      if(abs(value.imag()) < 0.1){
        result[current][2] = 1;
      }
      else if(0.8 < value.imag() && value.imag() < 0.9){
        result[current][2] = 2;
      }
      else if(-0.9 < value.imag() && value.imag() < -0.8){
        result[current][2] = 3;
      }
      else{
        result[current][2] = 0;
      }
      result[current][3] = values[1].real();
    }
  }

  // 出力CSV
  ofstream ofs("./rst.csv");
  for(int i=0; i<row+1; i++){
    for(int j=0; j<col+1; j++){
      int current = i * (col+1) + j;
      ofs << result[current][0] << ",";
      ofs << result[current][1] << ",";
      ofs << result[current][2] << ",";
      ofs << result[current][3] << endl;
    }
  }
  ofs.close();
  return 0;
}
```

解の収束先を色分け（回数別の色分けはまたいずれ）

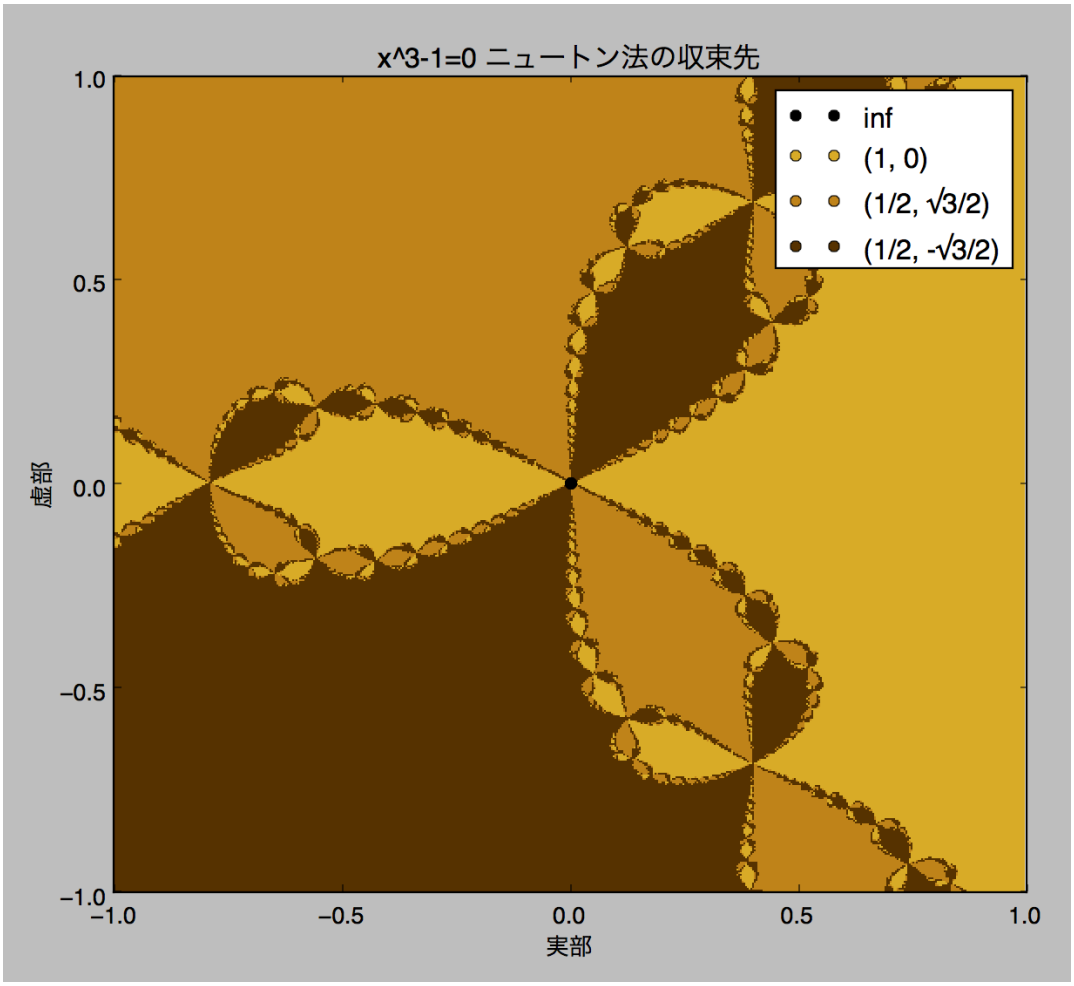
x^3-1=0 ニュートン法の収束先
1.0
0.5
0.0
−0.5
−1.0
虚部
−1.0
−0.5
0.0
0.5
1.0
実部
inf
(1, 0)
(1/2, √3/2)
(1/2, -√3/2)

図 1:

問題 2

半径 r の球が等加速度で面 $z = cos(\sqrt{(}x^2 + y^2))$ に落下する時、衝突時間を求めるプログラムを書け。

ソースコード: hw2-2.cpp

```
#include <iostream>
#include <array>
#include <cmath>
#include <limits>
#define PI 3.1415926535
using namespace std;

template <class T> array<T, 2> quadratic(T a, T b, T c);
template <class T>
  T compute_clash_time(array<T, 3> circle, T radius, T gravity);

int main(){
  double radius = 1.0;
  double clash_time;
  array<double, 3> circle={0, 0, 6};
  cout.precision(numeric_limits<double>::max_digits10);
  clash_time = compute_clash_time(circle, -1.0, 1.0);
  cout << clash_time << endl;

  return 0;
}


// ax^2 + bx + c = 0 の形の2 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 = sqrt(pow(b, 2) - 4 * a * c);

  if(isnan(d)){
    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 + d) / (2 * a);
  }
  else{
    x1 = (-1 * b - d) / (2 * a);
  }

  // 解と係数の関係
```

```
  x2 = c / (a * x1);
  rst[0] = x1;
  rst[1] = x2;

  return rst;
}


// 重力は下に落ちる場合、マイナスに設定する
template <class T>
T compute_clash_time(array<T, 3> circle, T radius, T gravity){
  T x, y, z, c;
  T clash_time;

  x = circle[0];
  y = circle[1];
  z = circle[2];
  c = cos(sqrt(x*x+y*y));

  // 初期衝突判定
  if(x*x+y*y+pow(z-c, 2) < radius * radius){
    return numeric_limits<T>::quiet_NaN();
  }

  array<T, 2> time_list =
    quadratic(gravity*gravity, -2*gravity*(z-c), x*x+y*y+pow(z-c, 2)-radius*radius);

  if( isnan(time_list[0]) || (time_list[0] < 0 && time_list[1] < 0) ){
    clash_time = numeric_limits<T>::quiet_NaN();
  }
  else{
    if(time_list[0] < 0 || time_list[1] < time_list[0]){
      clash_time = time_list[1];
    }
    else{
      clash_time = time_list[0];
    }
  }

  return clash_time;
}
```

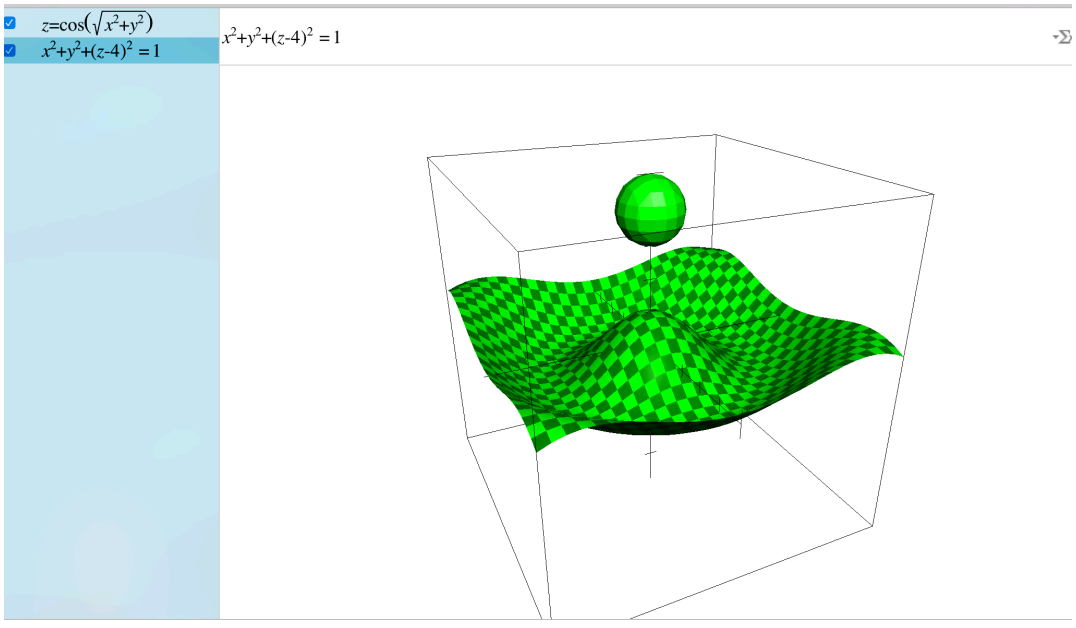

図 2: イメージ