プログラム演習 (4) 三次方程式の解を二分法とニュートン法を用いて求める

プログラム演習 (1) では二次方程式の解を簡単なアルゴリズムで求めたが、今回はもう少しだけ賢いアルゴリズムで求めることとする。今回採り上げるのは、二分法とニュートン法である。なお、本演習では、解くべき問題を「$f(x) = 0$ を満足する x の値を求める（ただし $f(x)$ は連続関数）」というものとする。

はじめに二分法について説明する。

ある２つの x について、その関数値 $f(x)$ の符号が互いに異なるようなものを探し、これを x_1 と x_2 とする。このようなときには、$f(x) = 0$ の解のひとつは必ず区間 $[x_1, x_2]$ に存在するということが言える。そこで、x_1 と x_2 の中点を x_3 とし、そこでの関数値 $f(x_3)$ を計算する。下の図のように、$f(x_3)$ と $f(x_2)$ の符号が同一である場合には、$f(x) = 0$ の解のひとつは $[x_1, x_3]$ に存在することが言え、解の存在（推定）範囲を 1/2 に狭めることが出来る。逆に $f(x_3)$ と $f(x_2)$ の符号が異なる（$f(x_3)$ と $f(x_1)$ の符号が同一である）場合には、$f(x) = 0$ の解のひとつは $[x_3, x_2]$ に存在することが言える。この操作を繰り返すことによって解の存在範囲を小さくすること、つまり精度の高い解の推定が行えることになる。これが二分法の考え方である。

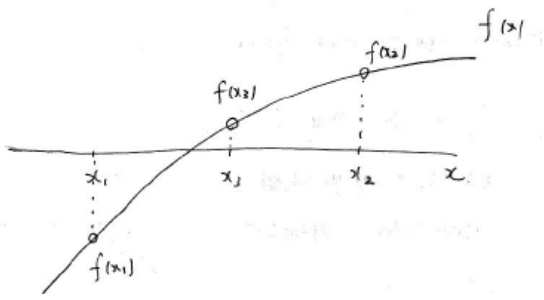

Fig. 1: 二分法の考え方

問題１：二分法に基づいて $x^3 - 4x^2 - x + 4 = 0$ の解を求めるプログラムを作成し、解を数値的に求め、解析解と比較せよ。

ゼロから自分でプログラムを作成するのが難しい場合は、以下のサンプルプログラムを参考にして作成するとよい（そのままコピーしてはダメです）。

Listing 1: 二分法のサンプルプログラム

```
#include <stdlib.h>
#include <cmath>
#include <iostream>

using namespace std;

int main()
{
  float a=1.;
  float b=-4.;
  float c=-1;
  float d=4;

  float x_init=-3.;
  float dx_init=10.;
  int iter_max=20;
  float threshold=1e-6;

  float x1=x_init;
  float fx1=a*(x*x*x)+b*(x*x)+c*x+d;

  float x2=x+dx_init;
  float fx2=a*(x2*x2*x2)+b*(x2*x2)+c*x2+d;

  if(fx1*fx2>0.){
    cout<<"# The signs of fx1 and fx2 are indentical.\n";
    exit(0);
  };

  if(abs(fx1)<threshold){
    cout<<x1<<"\n";
    exit(0);
```

```
33    };
34    if(abs(fx2)<threshold){
35      cout<<x2<<"\n";
36      exit(0);
37    };
38
39    float x3,fx3;
40    for(int i=0;i<iter_max;i++){
41      x3=(x1+x2)*0.5;
42      fx3=a*(x3*x3*x3)+b*(x3*x3)+c*x3+d;
43      if(abs(fx3)<threshold)break;
44      if(fx*fx3<0.){
45        x2=x3;
46        fx2=fx3;
47      }else{
48        x1=x3;
49        fx1=fx3;
50      };
51    };
52
53    cout<<x3<<"\n";
54
55    return 0;
56  };
```

このプログラムは、$ax^3+bx^2+cx+d=0$ の解を求めるためのものである。

x の最初の推定値を x_init とし、その推定値における関数値と、推定値に dx_init を加えた値における関数値の符号が異なる場合に、二分法により解を探索する。この条件が満たされない場合には強制終了となるため、x_init と dx_init を適切な値に変更する必要がある。また、二分法により求まる解もこれらの値に依存するため、複数の解を探索する場合は、これらの値を変更して対応することになる。二分法の繰り返し回数は変数 iter_max により設定している。なお、計算途中に関数値がゼロに近い値になった場合（具体的には、変数 threshold よりも絶対値が小さくなった場合）、繰り返し計算を終了し、解を出力させている。絶対値を計算するために abs 関数を使っているが、環境によっては返り値が整数型になってしまう場合がある。このような場合は解析解を再現することが出来ないため、千葉まで相談すること。

次に、二分法よりも効率的に解を探索することができるニュートン法について説明する。

下図のように、ある値 x_1 とその値における関数値 $f(x_1)$ を考える。ここで、x_1 において $f(x)$ の接線を引き、その接線が $f(x)=0$ と交わる x を解の推定値とするのがニュートン法である。この解の推定値 x_2 は、図に記載されているように、以下の式で求めることができる。

$$x_2 = x_1 - \Delta x = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{1}$$

解の推定値が真値から大きく離れている場合、すなわち $f(x)$ がゼロから大きく離れている場合は、一次微係数に基づく推定は精度が悪いため、推定された解と真値にはまだ差が残る。そのため、この操作を繰り返し行う必要がある。繰り返しの過程で推定値が真値に近づくにつれて推定の精度は向上し、最終的に真値へ収束していくことが期待される。繰り返し計算は、前回の推定値と新しい推定値の相対差を調べ、その絶対値がある値未満になった場合に終了させるのが一般的である。なお、初期推定値によっては繰り返しによって推定値が発散する場合もあるため、繰り返しの最大回数を設定しておくとよいだろう（このような設定をしておかないと、いつまで経っても計算が終わらないという状況に陥る）。

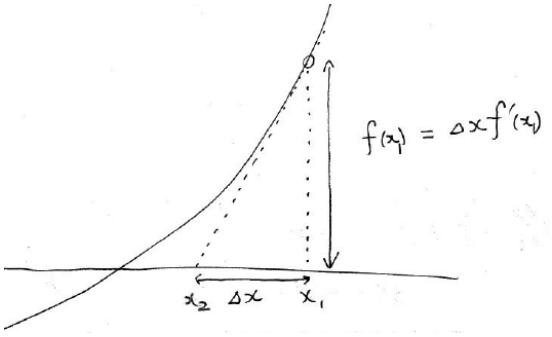

Fig. 2: ニュートン法の考え方

二分法では、初期推定値として 2 点を設定する必要があったが、ニュートン法では 1 点のみ設定すればよいので、使い勝手が良いと言えるだろう。また、一般的に、真値への収束もニュートン法のほうが速い。

問題 2：問題 1 の解をニュートン法により求め、解析解と比較せよ。