プログラム演習 (1) 二次方程式の解を求める

プログラム言語は大きく二つに分類することが出来る。一つは**システム言語**であり、もう一つは**スクリプト言語**である。

システム言語としては、FORTRAN、C++などが挙げられる。これらを用いる場合、プログラマーが記述したプログラムをコンピュータが実行できる形式に変換する作業が必要であり、その作業を**コンパイル**と呼ぶ。システム言語は、プログラム中に用いる変数の型（整数、実数、文字、等）を明示する必要がある、処理速度が速い、などの特徴を持つ。

一方、スクリプト言語としては、Perl、Python などが挙げられる。これらはコンパイルの必要が無く、プログラマーが記述したプログラムをコンピュータが直接読んで処理を行う。スクリプト言語は、変数の扱いがシステム言語と比べると融通がきく、処理が遅い、などの特徴を持つ。

科学技術計算では一般にシステム言語が用いられるが、処理速度が重要ではない場合や、数値データが書き込まれたファイルの読み書きと加工などを行う場合にはスクリプト言語も用いられるようである。また、計算エンジンには処理の速いシステム言語を用い、計算する人とのインターフェースには融通の効くスクリプト言語を用いるという階層的な使い方もある[1]。

科学技術計算に用いるシステム言語は、一昔前であれば FORTRAN 全盛であったが、炉物理計算に限って言うと、最近開発されているプログラム（コード）は C++や Java 等で記述されている。プログラムの拡張性、メンテナンス等の観点からは、FORTRAN よりも C++等のほうが優れていると、個人的には考えている。ただし、世の中には長い間使われてきた由緒正しきコードが多々あり、それらの多くは FORTRAN で記述されている。それらを相手にする場合には FORTRAN のプログラミング技術が必須となる。

本演習では、二次方程式 $ax^2 + bx + c = 0$ の解を求めるプログラムを C++で作成する。

はじめに、実数を定義する際の、**単精度**と**倍精度**の違いについて説明する。

科学技術計算では整数や実数のパラメータを扱う必要があり、C++ではそれらパラメータに対応する変数に対して「型」を明示する必要がある。整数のための変数に対しては `int` 型を用いるが、実数のための変数に対しては `float` 型と `double` 型を用いることが出来る。これら `float` 型と `double` 型の違いは数値の有効桁数にあり、`float` 型の変数は 7 桁程度、`double` 型の変数は 16 桁程度が有効となる。この実数の有効桁の違いは、2 つの実数の差を計算する際に大きく現れる場合がある。例として、以下のプログラムについて考えよう（このプログラムを打ち込んで、`prog0.cxx` というファイル名で保存すること）。

Listing 1: Program 0

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    float a=1.000000000000000;
9    float b=0.999999999987654;
10   float c=a-b;
11   cout<<c<<"\n";
12
13   double ad=1.000000000000000;
14   double bd=0.999999999987654;
15   double cd=ad-bd;
16   cout<<cd<<"\n";
17
18   return 0;
19 };
```

C++には標準ライブラリという様々な便利な機能を有するライブラリが存在する。このプログラムでは `cmath`、`iostream` というライブラリを利用するので、1、2 行目にあるようにそれらのライブラリを「include」する必要がある。4 行目はとりあえず「おまじない」として理解してもらいたい。

さて、それでは、このプログラムを走らすためにコンパイルを行おう。ターミナル（端末）上で、`g++ prog0.cxx -o prog0.lm` と打ち込む（その後エンターキー）[2]。ここで、`prog0.lm` なるファイルが作成されるであろう。このファイルはプログラムの**ロードモジュール**（もしくは**実行形式**）と呼ばれるものであり、コンパイル時にオプション`-o` のあとでその名前を指定することが出来る。なお、`-o` オプションを省略した場合（すなわち `g++ prog0.cxx` とだけ打ち込んだ場合）は、ロードモジュールとして `a.out` という名前のファイルが作成される。

[1] 最近では、Python における NumPy のように、スクリプト言語でも数値計算を効率的に行なうための機能が整備されており、このような説明は「時代遅れ」となりつつある。

[2] なお、`g++: command not found` と表示された場合には、g++コンパイラがインストールされていないので、`sudo apt-get install g++`と打ち込んでインストールすること（ただしインターネットに繋がっている必要あり）。

それでは作成されたロードモジュールを用いて計算を実行しよう。この場合、`./prog0.1m` と打ち込めばよい。すると計算結果が出力されるであろう。`float` 型を用いた場合、有効桁数が小さいため、適切な結果を得ることが出来ないが、`double` 型を用いることにより、この問題が解消することが分かるであろう。数値計算においては、`float` 型で適切か（事足りるか）どうかを事前に判断することは難しいため、一般的には `double` 型を用いるべきと言える。この演習シリーズのサンプルプログラムでも `float` 型が用いられているが、**以降は必ず `double` 型を使うようにしてもらいたい**[3]。

さて、それでは本題に戻ろう。二次方程式 $ax^2 + bx + c = 0$ の解を求めるプログラムは、解の公式から $x = (-b \pm \sqrt{b^2 - 4ac})/2a$ となることを利用すると、以下のように書けるであろう

Listing 2: Program 1

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    double a=1.;
9    double b=-1.;
10   double c=-2.;
11
12   double bac=b*b-4*a*c;
13
14   double sol1=(-b+sqrt(bac))/(2*a);
15   double sol2=(-b-sqrt(bac))/(2*a);
16
17   cout<<"solution 1 : "<<sol1<<"\n";
18   cout<<"solution 2 : "<<sol2<<"\n";
19
20   return 0;
21 };
```

この計算では、解くべき二次方程式の係数に対応する float 型（実数型）の変数 a、b、c に適当な値を入れており、解くべき方程式は $x^2 - x - 2 = (x-2)(x+1) = 0$ となっている。従って解は $x = 2$、-1 となる。

計算結果の出力はサンプルプログラム中の 17、18 行目で行われる。これらの最後の部分の `\n` は改行を行うことを意味する（試しにこれらを消してコンパイル、実行してみるとよい）。

次に、$x^2 - x + 2 = 0$ を解かせてみよう。先程の例とは「c」の値が異なるので、プログラムを変更し、コンパイルを行い、実行すると、おそらく `nan` の表示が出るであろう。これは「not a number」の略であり、正常な計算が行われていないことを意味する。これは、二次方程式の判別式 $(b^2 - 4ac)$ が負となり、`sqrt` 演算（平方根をとる演算）が正常に行われないことによる。そこで、判別式が負となるような場合には計算を行わないことにするため、プログラムを以下のように修正する。

Listing 3: Program 2

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    double a=1.;
9    double b=-1.;
10   double c=2.;
11
12   double bac=b*b-4*a*c;
13   if(bac<0.){
14     cout<<"There is not real solution.\n";
15     return 0;
16   };
17
18   double sol1=(-b+sqrt(bac))/(2*a);
19   double sol2=(-b-sqrt(bac))/(2*a);
20
21   cout<<"solution 1 : "<<sol1<<"\n";
22   cout<<"solution 2 : "<<sol2<<"\n";
23
24   return 0;
25 };
```

このプログラムでは、$(b^2 - 4ac)$ が負の場合にはメッセージを出力して処理を終了する。このような処理を**例外処理**と呼ぶ。ちなみに、プログラムの強制終了は 15 行目の `return 0` で行っている。

[3] `float` 型を使い続けた場合、いずれかのタイミングで有効桁数不足による問題が発生する可能性がある。そのような場合には、なかなか問題を特定する（問題が有効桁不足に起因することに気付く）ことが難しいことから、今から `double` 型を使うようにして欲しい。

例題：$x^2+2x+1=0$ の解は重根となるため、解がひとつしかない。このような場合にも対応できるよう、プログラムを修正せよ。

解が重根となるのは判別式がゼロとなる場合である。従って、判別式がゼロとなる場合の条件分岐をプログラムに加えれば良い。例えば、以下のように書けるであろう。

Listing 4: Program 2_2

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    double a=1.;
9    double b=2.;
10   double c=1.;
11
12   double bac=b*b-4*a*c;
13   if(bac<0.){
14     cout<<"There is not real solution.\n";
15   }else if(bac==0.){
16     double sol=-b/(2*a);
17     cout<<"solution : "<<sol<<"\n";
18   }else{
19     double sol1=(-b+sqrt(bac))/(2*a);
20     double sol2=(-b-sqrt(bac))/(2*a);
21     cout<<"solution 1 : "<<sol1<<"\n";
22     cout<<"solution 2 : "<<sol2<<"\n";
23   };
24
25   return 0;
26 };
```

13 行目から 23 行目の `if (A){B}else if(C){D}else{E}` の構文であるが、これは「条件 A が真（成り立つ）ならば B の処理を行い、そうでない（偽）ならば、さらにもし C が真ならば D の処理を行い、そうでないならば E の処理を行う」という意味となる。なお、変数 `bac` がゼロであるかどうかの条件であるが、`bac==0.` と記述する。ここで、**等号が二つ連続することに注意が必要である**（等号が一つだけだと、変数 `bac` の値をゼロにする、という意味となる）。

さて、次に、二つの二次方程式 $x^2-x-2=0$ と $x^2-2x-3=0$ の解をそれぞれ計算したいとする。この場合、例えば次のようなプログラムが書けるであろう。

Listing 5: Program 3

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    // For the first problem
9
10   double a=1.;
11   double b=-1.;
12   double c=-2.;
13
14   double bac=b*b-4*a*c;
15
16   cout<<"For the first problem\n";
17   if(bac<0.){
18     cout<<"There is not real solution.\n";
19   }else if(bac==0.){
20     double sol=-b/(2*a);
21     cout<<"solution : "<<sol<<"\n";
22   }else{
23     double sol1=(-b+sqrt(bac))/(2*a);
24     double sol2=(-b-sqrt(bac))/(2*a);
25     cout<<"solution 1 : "<<sol1<<"\n";
26     cout<<"solution 2 : "<<sol2<<"\n";
27   };
28   cout<<"\n";
29
30   // For the second problem
31
32   a=1.;
33   b=-2.;
34   c=-3.;
35
36   bac=b*b-4*a*c;
37
38   cout<<"For the second problem\n";
39   if(bac<0.){
40     cout<<"There is not real solution.\n";
```

```
41    }else if(bac==0.){
42      double sol=-b/(2*a);
43      cout<<"solution : "<<sol<<"\n";
44    }else{
45      double sol1=(-b+sqrt(bac))/(2*a);
46      double sol2=(-b-sqrt(bac))/(2*a);
47      cout<<"solution 1 : "<<sol1<<"\n";
48      cout<<"solution 2 : "<<sol2<<"\n";
49    };
50    cout<<"\n";
51
52    return 0;
53  };
```

8、30 行目は//から始まっているが、これにより、この行はコメント行として扱われ、計算処理には全く影響しないことになる。また、32 から 36 行目では、10 から 14 行目と異なり、頭に `float` の型宣言が付されていないが、これは、これらの変数は 10 から 14 行目ですでに定義されていることによる。このプログラムをコンパイルして実行すれば、ふたつの解のセットが得られるであろう。ただし、このプログラムを眺めれば、同一の処理を繰り返し記述しており、明らかに冗長であることが分かるだろう。そこで次のプログラムのように二次方程式の解を算出し出力する**関数**を作成する。

Listing 6: Program 4

```
1   #include<cmath>
2   #include<iostream>
3
4   using namespace std;
5
6   void cal(double a,double b,double c)
7   {
8     double bac=b*b-4*a*c;
9
10    if(bac<0.){
11      cout<<"There is not real solution.\n";
12    }else if(bac==0.){
13      double sol=-b/(2*a);
14      cout<<"solution : "<<sol<<"\n";
15    }else{
16      double sol1=(-b+sqrt(bac))/(2*a);
17      double sol2=(-b-sqrt(bac))/(2*a);
18      cout<<"solution 1 : "<<sol1<<"\n";
19      cout<<"solution 2 : "<<sol2<<"\n";
20    };
21    cout<<"\n";
22  };
23
24  int main(){
25
26    // For the first problem
27
28    double a=1.;
29    double b=-1.;
30    double c=-2.;
31
32    cout<<"For the first problem\n";
33    cal(a,b,c);
34
35    // For the second problem
36
37    a=1.;
38    b=-2.;
39    c=-3.;
40
41    cout<<"For the second problem\n";
42    cal(a,b,c);
43
44    return 0;
45  };
```

二次方程式の解を算出する関数は 6 から 22 行目で定義されており、この関数を呼び出すときは 33、42 行目にあるように `cal();` を用いる。6 行目で関数の頭が `void` で始まっているのは、この関数に返り値が無いことを意味している。

なお、解の個数を cal 関数の返り値とさせるならば、以下のようなプログラムとなるであろう。

Listing 7: Program 4_2

```
1   #include<cmath>
2   #include<iostream>
3
4   using namespace std;
5
6   int cal(double a,double b,double c)
7   {
8     double bac=b*b-4*a*c;
9     int num;
10
11    if(bac<0.){
12      cout<<"There is not real solution.\n";
13      num=0;
```

```
  }else if(bac==0.){
    double sol=-b/(2*a);
    cout<<"solution : "<<sol<<"\n";
    num=1;
  }else{
    double sol1=(-b+sqrt(bac))/(2*a);
    double sol2=(-b-sqrt(bac))/(2*a);
    cout<<"solution 1 : "<<sol1<<"\n";
    cout<<"solution 2 : "<<sol2<<"\n";
    num=2;
  };

  return num;
};

int main(){

  // For the first problem

  double a=1.;
  double b=-1.;
  double c=-2.;

  cout<<"For the first problem\n";
  int num_prob1=cal(a,b,c);
  cout<<"    number of answers : "<<num_prob1<<"\n";
  cout<<"\n";

  // For the second problem

  a=1.;
  b=-2.;
  c=-3.;

  cout<<"For the second problem\n";
  int num_prob2=cal(a,b,c);
  cout<<"    number of answers : "<<num_prob2<<"\n";
  cout<<"\n";

  return 0;
};
```

さて、これまでの例では解が解析的に得られることが分かっているため、解析解を公式に基づいて得たが、解析解が得られない場合は「数値的に」解を得なければならない。二次方程式 $ax^2+bx+c=0$ の解を数値的に求める方法としては、様々な x に対して $f(x)=ax^2+bx+c$ を計算し、$f(x)$ がゼロとなる x を探す、というものが考えられるであろう。例えば次のようなプログラムを書いてみる。

Listing 8: Program 5

```
#include<cmath>
#include<iostream>

using namespace std;

int main(){

  double a=1.;
  double b=-1.;
  double c=-2.;

  double x=-50.;
  for(int i=1;i<=100;i++){
    double fx=a*x*x+b*x+c;
    cout<<x<<" "<<fx<<"\n";
    x+=1.;
  };

  return 0;
};
```

13 から 17 行目のループは、その条件が 13 行目で指定されており、「始めに変数 i を 1 としてループを開始し（int i=1 に対応）、ループの最後に i を一つ増加させる（i++に対応）。その後、i が 100 以下であるならば（i<=100 に対応）、再度繰り返し処理を行う」ということを意味している。従って、このループは i が 1 から 100 まで合計で 100 回の繰り返し計算が行われることになる。

これをコンパイルして走らせると、x と $f(x)$ のペアが出力されるであろう。この結果から、$f(x)$ がゼロとなるところを自分で探すことで解が得られる[4]。なお、プログラム中で変数 x の初期値を-50 としているのは、「解はこれよりも大きい値になるであろう」という想定に基づくものであり、理由があってのものではない。

自分で解を探すのは野暮なので、コンピュータにやらせるならば、以下のようなプログラムになるであろう。

Listing 9: Program 6

[4]出力が長すぎる場合は、./prog5.lm > output 等として、output という名前のファイルに出力することができる。

```
#include<cmath>
#include<iostream>

using namespace std;

int main(){

  double a=1.;
  double b=-1.;
  double c=-2.;

  double x=-50.;
  for(int i=1;i<=100;i++){
    double fx=a*x*x+b*x+c;
    if(fx==0.){
      cout<<x<<"\n";
    };
    x+=1.;
  };

  return 0;
};
```

このプログラムでは、$f(x)=0$ の解が画面上に出力されるであろう。

さて、ここで、$c=-1.9$ として同じプログラムを走らせてみよう。何も出力されない筈である。これは、1.0 刻みで x を増やしているため、$f(x)=0$ となる x で計算が行われていないことによる。そこで、$f(x)=0$ の条件を緩和した次のようなプログラムを書くこととする。

Listing 10: Program 7

```
#include<cmath>
#include<iostream>

using namespace std;

int main(){

  double a=1.;
  double b=-1.;
  double c=-1.9;

  double x=-50.;
  for(int i=1;i<=100;i++){
    double fx=a*x*x+b*x+c;
    if(abs(fx)<1.){
      cout<<x<<"\n";
    };
    x+=1.;
  };

  return 0;
};
```

ここでは、$|f(x)|<1$ となる x を出力させている。このプログラムを実行させると、$x=2$、-1 が出力されるであろう。勿論、これは近似解に過ぎない。それは、x を 1.0 ずつ増加させているためである。

さて、以上のプログラムは $f(x)=0$ を満足する x を探索させていたが、ここで仮に我々が $x<0$ の範囲のみの解に関心があるとしよう。この場合、$|f(x)|<1$ を満足し、かつ $x<0$ を満足する x のみを出力させればよく、プログラムは以下のように書けるであろう。

Listing 11: Program 7_2

```
#include<cmath>
#include<iostream>

using namespace std;

int main(){

  double a=1.;
  double b=-1.;
  double c=-1.9;

  double x=-50.;
  for(int i=1;i<=100;i++){
    double fx=a*x*x+b*x+c;
    if(abs(fx)<1.&&x<0.){
      cout<<x<<"\n";
    };
    x+=1.;
  };

  return 0;
};
```

15 行目のみが Program 7 と異なっているが、ここでは条件として `abs(fx)<1.&&x<0` が用いられている。この`&&`であるが、これは `and` を意味しており、これを挟む 2 つの条件がいずれも真の場合に、これ全体が真となる。一方、`or` の場合は`||`を用いればよく、この場合はこれを挟む 2 つの条件のいずれかが真の場合に、これ全体が真となる[5]。

問題 1：解の推定精度を高めるためにプログラムを書き換え、$x^2 - x - 1.9 = 0$ の解を求めよ（x の刻み幅を小さくすればよい）。

なお、このプログラムでは、$|f(x)|$ がある値よりも小さいものを検出するため、想定される以上の解（の候補）が検出される可能性がある。このときは $|f(x)|$ に対する条件を厳しくしてやればよい。

$f(x) = 0$ の解を求める際、$f(x)$ の絶対値がある値以下となる条件を満たす x を探索するこれまでの方法では、少々手間がかかることが分かったと思う。そこで、別な条件を考えよう。

問題 2：x を Δx 毎に増加させていった場合、$f(x_1)$ と $f(x_1+\Delta x)$ の符号が異なったとき、$f(x) = 0$ の解が $[x_1, x_1+\Delta x]$ の区間にあるということが言える。解の推定アルゴリズムをこのように変更し、$x^2 - x - 1.9 = 0$ の解を求めよ。なお、可能な限り冗長な計算を排除するよう、留意すること。

問題 3：$x^2 - x - 1.9 = 0$ の解析解を手計算により求め、Δx を小さくすることによって数値解が解析解に漸近していくことを示しなさい。

[5]例えば条件が 3 つあって全てが真である場合を条件とするならば、`A&&B&&C` と記述される。また、「条件 A と条件 B がいずれも真、もしくは条件 C と条件 D がいずれも真」というような場合は、括弧を使って、`(A&&B)||(C&&D)` と記述すればよい。