プログラム演習 (1) 二次方程式の解を求める

千葉豪

平成 29 年 8 月 31 日

プログラム言語は大きくふた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。また、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。また、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、ファイルの読み書き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、それらはその歴史が古いということで、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作成する。C++、FORTRAN それぞれ分けて解説するので、必要な方を参照すること。なお、最後に問題を示しておく。

[1] 最近では Python における NumPy のように、スクリプト言語に数値計算を効率的に行なうための機能が整備されつつある。

1 C++の場合

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（このプログラムを打ち込んで、「prog1.cxx」というファイル名で保存すること）。

Listing 1: Program1

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

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。

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 $x=2$、$x=-1$ となる。

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++ prog1.cxx -o prog1.lm」と打ち込む（その後エンターキー）[2]。ここで、「prog1.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」「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。

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、「./prog1.lm」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。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）。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」の表示がでるであろう。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

[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）。

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 2: Program2

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

例題 1 ：$x^2+2x+1=0$ 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、プログラムに判別式がゼロとなる場合の条件分岐を加えれば良い。例えば、以下のように書けるであろう。

Listing 3: Program2_2

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    float a=1.;
9    float b=2.;
10   float c=1.;
11
12   float bac=b*b-4*a*c;
13   if(bac<0.){
14     cout<<"There is not real solution.\n";
15   }else if(bac==0.){
16     float sol=-b/(2*a);
17     cout<<"solution : "<<sol<<"\n";
18   }else{
19     float sol1=(-b+sqrt(bac))/(2*a);
20     float 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 4: Program3

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    // For the first problem
9
10   float a=1.;
11   float b=-1.;
12   float c=-2.;
13
14   float 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     float sol=-b/(2*a);
21     cout<<"solution : "<<sol<<"\n";
22   }else{
23     float sol1=(-b+sqrt(bac))/(2*a);
24     float 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     float sol=-b/(2*a);
43     cout<<"solution : "<<sol<<"\n";
44   }else{
45     float sol1=(-b+sqrt(bac))/(2*a);
46     float 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 5: Program4

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  void cal(float a,float b,float c)
7  {
8    float 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     float sol=-b/(2*a);
14     cout<<"solution : "<<sol<<"\n";
15   }else{
16     float sol1=(-b+sqrt(bac))/(2*a);
17     float 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   float a=1.;
29   float b=-1.;
30   float 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 6: Program4_2

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

using namespace std;

int cal(float a,float b,float c)
{
  float bac=b*b-4*a*c;
  int num;

  if(bac<0.){
    cout<<"There is not real solution.\n";
    num=0;
  }else if(bac==0.){
    float sol=-b/(2*a);
    cout<<"solution : "<<sol<<"\n";
    num=1;
  }else{
    float sol1=(-b+sqrt(bac))/(2*a);
    float 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

  float a=1.;
  float b=-1.;
  float 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;
};
```

2 FORTRAN の場合

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、二次方程式 $ax^2 + bx + c = 0$ の解を求めるプログラムは以下のように書けるであろう（このプログラムを打ち込んで、「prog1.f」というファイル名で保存すること）。

Listing 7: Program1

```
1       a=1.0
2       b=-1.0
3       c=-2.0
4  c
5       bac=b*b-4*a*c
6  c
7       sol1=(-b+sqrt(bac))/(2*a)
8       sol2=(-b-sqrt(bac))/(2*a)
9  c
10      write(6,*)'solution 1 : ',sol1
11      write(6,*)'solution 2 : ',sol2
12 c
13      stop
14      end
```

FORTRAN77 では各行の頭に 6 個以上の空白を空ける必要がある。また最初の欄に「c」が付されている行はコメント行として扱われ、処理上は無視される（プログラムの見栄えを良くしたり、そこで行われている処理を書いたりすることに使う）。また、この計算では a、b、c には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 $x = 2$、$x = -1$ となる。

さて、それでは、このプログラムを走らす前にコンパイルを行う。ターミナル（端末）上で、「gfortran prog1.f -o prog1.lm」と打ち込む（その後エンターキー）[3]。ここで、「prog1.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」「実行形式」と呼ばれるものであり、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は「a.out」という名前のファイルが生成される。

それでは生成されたロードモジュールを用いて計算を実行する。この場合、「./prog1.lm」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。計算結果の出力はサンプル中の 10、11 行目で行われる。「write(6,*)」の「6」が、出力を画面（端末）上に行うことを意味し、「*」は出力形式を特に指定しないことを意味する。

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）。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」の表示がでるであろう。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 8: Program2

```
1       a=1.0
2       b=-1.0
3       c=+2.0
4  c
5       bac=b*b-4*a*c
6       if(bac<0.)then
```

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

```
7           write(6,*)'There is no real solution.'
8           stop
9         endif
10  c
11        sol1=(-b+sqrt(bac))/(2*a)
12        sol2=(-b-sqrt(bac))/(2*a)
13  c
14        write(6,*)'solution 1 : ',sol1
15        write(6,*)'solution 2 : ',sol2
16  c
17        stop
18        end
```

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 8 行目の「stop」文で行っている。

例題 1 ：$x^2+2x+1=0$ 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、プログラムに判別式がゼロとなる場合の条件分岐を加えれば良い。例えば、以下のように書けるであろう。

Listing 9: Program2_2

```
1         a=1.0
2         b=2.0
3         c=1.0
4   c
5         bac=b*b-4*a*c
6         if(bac.lt.0.)then
7           write(6,*)'There is no real solution.'
8           stop
9         else if(bac.eq.0.)then
10          sol=-b/(2*a)
11          write(6,*)'solution : ',sol
12        else
13          sol1=(-b+sqrt(bac))/(2*a)
14          sol2=(-b-sqrt(bac))/(2*a)
15  c
16          write(6,*)'solution 1 : ',sol1
17          write(6,*)'solution 2 : ',sol2
18        endif
19  c
20        stop
21        end
```

「if A then B else if C then D else E endif」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。

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 10: Program3

```
1         a=1.0
2         b=-1.0
3         c=-2.0
4   c
5         bac=b*b-4*a*c
6   c
7         sol1=(-b+sqrt(bac))/(2*a)
```

```
8         sol2=(-b-sqrt(bac))/(2*a)
9   c
10        write(6,*)'solution 1 : ',sol1
11        write(6,*)'solution 2 : ',sol2
12  c
13        a=1.0
14        b=-2.0
15        c=-3.0
16  c
17        bac=b*b-4*a*c
18  c
19        sol1=(-b+sqrt(bac))/(2*a)
20        sol2=(-b-sqrt(bac))/(2*a)
21  c
22        write(6,*)'solution 1 : ',sol1
23        write(6,*)'solution 2 : ',sol2
24  c
25        stop
26        end
```

これ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**サブルーチン**を用いる。

Listing 11: Program4

```
1         a=1.0
2         b=-1.0
3         c=-2.0
4   c
5         call cal(a,b,c,sol1,sol2)
6   c
7         write(6,*)'solution 1 : ',sol1
8         write(6,*)'solution 2 : ',sol2
9   c
10        a=1.0
11        b=-2.0
12        c=-3.0
13  c
14        call cal(a,b,c,sol1,sol2)
15  c
16        write(6,*)'solution 1 : ',sol1
17        write(6,*)'solution 2 : ',sol2
18  c
19        stop
20        end
21  c
22  c-------------------------------------------------
23  c
24        subroutine cal(a,b,c,sol1,sol2)
25  c
26        bac=b*b-4*a*c
27  c
28        sol1=(-b+sqrt(bac))/(2*a)
29        sol2=(-b-sqrt(bac))/(2*a)
30  c
31        return
32        end
```

サブルーチン cal は 24 行目以降で定義されており、サブルーチンを呼び出すときは 5、14 行目にあるように「call」を用いる。サブルーチン cal における「a,b,c,sol1,sol2」は「引数」と呼ばれるものである。また、サブルーチン内の「return」文（この例では 31 行目）で元のプログラムに戻る。

腕試し：計算結果を出力させている部分も冗長である。それらもサブルーチンで記述せよ。

3 問題

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 12: Program5（C++の場合）

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

using namespace std;

int main(){

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

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

  return 0;
};
```

Listing 13: Program5（FORTRAN の場合）

```
      a=1.0
      b=-1.0
      c=-2.0
c
      x=-50.
      do i = 1 , 100
        fx=a*x*x+b*x+c
        write(6,*)x,fx
        x=x+1.
      enddo
c
      stop
      end
```

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 14: Program6（C++の場合）

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

```

[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。

```
4  using namespace std;
5
6  int main(){
7
8    float a=1.;
9    float b=-1.;
10   float c=-2.;
11
12   float x=-50.;
13   for(int i=1;i<=100;i++){
14     float fx=a*x*x+b*x+c;
15     if(fx==0.){
16       cout<<x<<"\n";
17     };
18     x+=1.;
19   };
20
21   return 0;
22 };
```

Listing 15: Program6（FORTRAN の場合）

```
1        a=1.0
2        b=-1.0
3        c=-2.0
4  c
5        x=-50.
6        do i = 1 , 100
7          fx=a*x*x+b*x+c
8          if(fx.eq.0)then
9            write(6,*)x
10         endif
11         x=x+1.
12       enddo
13 c
14       stop
15       end
```

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 16: Program7（C++の場合）

```
1  #include<cmath>
2  #include<iostream>
3
4  using namespace std;
5
6  int main(){
7
8    float a=1.;
9    float b=-1.;
10   float c=-1.9;
11
12   float x=-50.;
13   for(int i=1;i<=100;i++){
14     float fx=a*x*x+b*x+c;
15     if(abs(fx)<1.){
16       cout<<x<<"\n";
17     };
18     x+=1.;
19   };
20
21   return 0;
22 };
```

Listing 17: Program7（FORTRAN の場合）

```
      a=1.0
      b=-1.0
      c=-1.9
c
      x=-50.
      do i = 1 , 100
        fx=a*x*x+b*x+c
        if(abs(fx).lt.1.)then
          write(6,*)x
        endif
        x=x+1.
      enddo
c
      stop
      end
```

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。

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。