

非線形方程式の課題

学籍番号 2120029, 氏名 政野玄空

2023 年 7 月 14 日

1 二分法

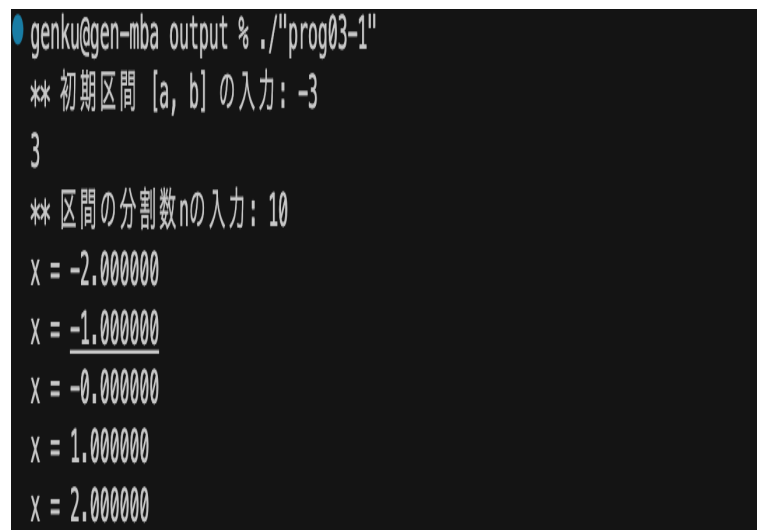
1.1 1

穴埋め箇所 1

- 穴埋め箇所 1: $(b-a)/n$
- 穴埋め箇所 2: $x-h$
- 穴埋め箇所 3: $f(a)*f(c)$
- 穴埋め箇所 4: $(a+b)/2.0$
- 穴埋め箇所 5: $\text{pow}(x,5)-(5*\text{pow}(x,3))+(4*x)$

1.2 2

1 で穴埋めした出力結果をスクリーンショットを示す.



```
genku@gen-mba output % ./"prog03-1"
** 初期区間 [a, b] の入力: -3
3
** 区間の分割数nの入力: 10
x = -2.000000
x = -1.000000
x = -0.000000
x = 1.000000
x = 2.000000
```

図 1: prog03-1.c を 1 の通りに埋めて初期区間 a,b を -3,3, 区間の分割数 n の入力 10 を入力した結果

-2.0,-1.0,0.0,1.0,2.0 の結果が得られた.

2 ニュートン法

2.1 1

$x^3 - 3x^2 - x + 3 = 0$ をニュートン法を用いて解くプログラムを作成した.

ソースコード 1: newton.c

```
1  #include <stdio.h>
2  #include <math.h>
3  #include <stdlib.h>
4
5  double newton(double x, double m, double p);
6  double f(double x);
7  double df(double x);
8
9  int main(void)
10 {
11     double x, max = 100000 ,eps = pow(2.0, -30.0);
12
13     printf("** 初期値の入力: ");
14     scanf("%lf", &x);
15
16     printf("x = %lf\n", newton(x, max, eps));
17
18     return (0);
19 }
20 /* newton */
21 double newton(double x, double m, double p)
22 {
23     double n = 0;
24     double d;
25     do{
26         d= -f(x)/df(x);
27         x = x + d;
28         n++;
29     } while (fabs(d) > p && n < m);
30     if (n == m) {
31         printf("failed");
32         exit(0);
33     } else {
34         return x;
35     }
36 }
37
38 /* 関数の定義 */
39 double f(double x)
40 {
41     return pow(x,3)+(-3*pow(x,2))+(-1*x)+3;
42 }
43 /* 関数の微分 */
```

```

44 double df(double x)
45 {
46     return (3*pow(x,2))+(-6*x)+(-1);
47 }

```

2.2 2

1 のコードの初期値を-2,1.5,5.0 にした結果のスクリーンショットを示す.

```

● genku@gen-mba output % ./"newton"
  ** 初期値の入力: -2
  x = -1.000000
● genku@gen-mba output % ./"newton"
  ** 初期値の入力: 1.5
  x = 1.000000
● genku@gen-mba output % ./"newton"
  ** 初期値の入力: 5.0
  x = 3.000000
○ genku@gen-mba output % 

```

図 2: newton.c の初期値を-2,1.5,5.0 にした結果

$x^3 - 3x^2 - x + 3 = 0$ をニュートン法を用いて初期値を-2,1.5,5.0 にして解いた結果は-1.0,1.0,3.0 となることを確認した.

3 考察課題

3.1 ニュートン法で初期値を 2.1547,-0.1547 としたとき, 解が求まらない理由

自身で設定した Max の値では求まってしまうので数値を 100,50 と下げて確認してみると 50 で失敗した. ヒント通り $x^3 - 3x^2 - x + 3 = 0$ の増減表を書き出してみる. 解の公式より x は $\frac{-2\sqrt{3}+3}{3}, \frac{2\sqrt{3}+3}{3}$ となる. $\sqrt{3}$ は 1.73205 で計算してみると-0.1547,2.1547 になる.

x	...	-0.1547	...	2.1547	...
$f'(x)$	+	0	-	0	+
$f(x)$		↗		↘	↗

このことより-0.1547,2.1547 を初期値として入力すると算出したい x から最も離れた位置になってしまい, 試行回数によっては収束しなくなると考えられる.

3.2 proc03-1.c の main 関数内の for 文を `for (x=a+h; x != b; x+=h)` とするとプログラムが停止しなくなる理由

まずこのようにプログラムを変更して `h` の値と変化する `x` の値, `x` が `b` を上回った地点をデバックしてみる. また `for` 文の中に `x==b` のときのデバックも仕込んでみる.

```
genku@gen-mba output % ./"prog03-1"
```

```
** 初期区間 [a, b] の入力: -3
```

```
3
```

```
** 区間の分割数 n の入力: 10
```

```
h = 0.600000
```

```
x = -2.400000
```

```
x = -1.800000
```

```
x = -2.000000
```

```
x = -1.200000
```

```
x = -0.600000
```

```
x = -1.000000
```

```
x = 0.000000
```

```
x = -0.000000
```

```
x = 0.600000
```

```
x = 1.200000
```

```
x = 1.000000
```

```
x = 1.800000
```

```
x = 2.400000
```

```
x = 2.000000
```

```
x > b 3.000000
```

```
x = 3.000000
```

```
x > b 3.600000
```

```
x = 3.600000
```

```
^C
```

見た限りでは `x` は `3.000000` のときがありループから抜けられそうではあるが実際は題意の通りそうはならない. `x==b` のときは `for` 文を抜けるので当然デバックもされない. 適当に `double` の数値を入力して比較するプログラムを作ってみる.

ソースコード 2: sample1.c

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(void)
5 {
6     double a, b;
7     scanf("%lf %lf", &a, &b);
8     if (a == b) {
9         printf("a==b");
10    }
```

```
11 return (0);
12 }
```

```
genku@gen-mba output % ./"test"
3
3
a==b%
```

この結果から double の計算のなかでなんらかの誤差が発生しているのではないかと考えられる。プログラムのなかの計算だけを抜き出してみるとこのようになる。

ソースコード 3: sample2.c

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main(void)
5 {
6     double a= -3,b= 3,n= 10,x,h;
7     h= (b-a)/n;
8     x = a;
9     for(int i = 0; i < 10; i++){
10         x+=h;
11     }
12     printf("x = %lf b = %lf",x,b);
13     if(b==x) {
14         printf("success");
15     }
16     return (0);
17 }
```

実行結果はこのようになり同値だが”success”の表記はされない。

```
genku@gen-mba output % ./"test"
x = 3.000000 b = 3.000000%
```

ここで h を計算せず 0.6 を代入してみる。しかし結果は変わらなかった。0.6 という数字になんらかの原因があると思われる。ここで 0.6 を 2 進数表記で表してみると 0.100110011.. と無限に続いていくことがわかる。double 型は 32bit なのでその中に収めなければいけなくどこかで丸める必要が出てくる。コンピュータで計算するには 2 進数にする必要があるので 0.6 という数字はそのままではおいておくことのできない数字ということがわかる。その丸められた無視で切るような誤差が 10 回繰り返して足し算をするうちに無視できない誤差になり同値のように見えるがプログラム上ではイコールにならないと考察できる。

試しに変数に代入した 0.6 と 0.6 を比較してみる。double より精度の低い float で試してみる。

ソースコード 4: sample3.c

```
1 #include <stdio.h>
2 #include <math.h>
3
```

```
4 int main(void)
5 {
6     float h = 0.6;
7     if (h == 0.6) {
8         printf("success");
9     }
10    printf("%lf",h);
11    return (0);
12 }
```

実行結果はこうになる. 代入した途端から数値に誤差が発生してしまっている.

```
genku@gen-mba output % ./"test"
0.600000%
```

よって `x == b` にならない理由は 0.6 を double 型におさめるときに丸めた誤差が繰り返しの処理によって無視できなくなるものになっているからと言える.