CS 数値計算 第 4 回 演習レポート

学籍番号 2210342, 鈴木謙太郎

2023 年 11 月 10 日

1　課題 1

変数 x の値を 1.29 に設定した. これは棗いつきさんという私が個人的に好きなシンガーの方の名前の語呂合わせである. いい曲をたくさん出されているのでぜひ一度聴いてみて欲しい. 他の方がどのくらいエピソードを書いているかわからないが, 私は本当にただの語呂合わせなのでこれだけしか書けない. 本題に移る.

課題を始める前に IEEE754 で規定されている倍精度浮動小数点の数式を式 (1) に示しておく.
s は符号部, e は指数部, f は仮数部である.

$$(-1)^s \times 2^{e-1023} \times 1.f \tag{1}$$

では 1.29 を IEEE754 で表現するとどうなるかを考える. まず符号部は,1.29 が正の値であるから $s = 0$ である. 次に指数部は,$2^0 \leq 1.29 < 2^1$ であるから $e - 1023 = 0$ である. よって $e = 1023$ であり, これを 11 桁の 2 進数で表すと 01111111111 となる. 最後に仮数部は,0.29 を 2 進数に変換することで表すことができる. これを手計算[*1]したところ, 式 (2) のようになった. ここから,$64 - 1 - 11 = 52$ 桁を取り出したものが仮数部であるはずである.

$$0.29_{10} = 0.01\dot{0}01010000111101011100\dot{0}\cdots_2 \tag{2}$$

ここで, 実際に演習資料として配布された bit.c を用いて 1.29 の浮動小数点を出力した. 変数に代入する値を変更しただけであるためソースコードは省略する. この出力は下のようになった. なお, 循環小数部分がわかりやすいように縦線を挿入しているため出力そのままではない.

```
> ./bit
0 01111111111 01|0010100011110101100|0010100011110101100|0010100100
```

これを見ると, 上 2 桁とそこから 2 回の循環部分までは手計算した仮数部と一致しているが, 3 回目の循環の仮数部が一致していないことがわかる. これは, 浮動小数点では循環小数を有限桁で表現することができないためである. これにより, 上位 52 桁を取り出す際に丸めが発生し, 誤差が生じていると考えられる.

[*1] きちんと手計算しました. 必要なら電子ノートを提出できます.

2　課題 2

「せいぜい十分小さな値」h を用いて数値微分を行う際の近似式は式 (3) のようになる.

$$\frac{df}{dx} \approx \frac{f(x+h/2) - f(x-h/2)}{h} \tag{3}$$

「せいぜい十分小さな値」を 10^0 から 10^{-14} まで 0.1 刻みで小さくした際の微分値や相対誤差を調べる. 念のため使用したソースコードをコード 1 に示す.

Code 1: 課題 2 で使用したソースコード

```c
#include <math.h>
#include <stdio.h>

double f(double x) { return sin(x); }

void print_delta(double h) {
  double x = M_PI / 4.0;
  double dfdx = (f(x + h / 2.0) - f(x - h / 2.0)) / h;
  double relerr = fabs(dfdx - cos(x)) / fabs(cos(x));
  printf("%1.14f %1.14f %1.14f\n", h, dfdx, relerr);
}

int main(void) {
  double h_array[] = {1e0,  1e-1, 1e-2,  1e-3,  1e-4,  1e-5,  1e-6, 1e-7,
                      1e-8, 1e-9, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14};
  for (int i = 0; i < 15; i++) {
    print_delta(h_array[i]);
  }
}
```

これによって得られた出力を基に表 1 を作成した. 表より, 以下のことがわかる.

まず, 微分値および誤差は $h = 10^0$ のときを除くと 0.01 刻みで見た際にあまり大きな差は見られない. $h = 10^0$ のときは単純に h が大きすぎるために式 (3) でいう分母が広くなり, 大きな誤差が生じていると考えられる.

次に残りの誤差などについて考察していく. 最も誤差が小さかったのは $h = 10^{-5}$ のときであった. 誤差は最も小さくなるのは h が最も小さくなるときであると考えていたが実際には異なった. これは,h が小さくなりすぎると, それを有限桁の倍精度浮動小数点数で表現する際に発生する丸め誤差が自身に対して無視できない大きさになるため, かえって誤差が大きくなっているものだと考えられる. つまり,$h = 10^{-5}$ 程度が, 倍精度浮動小数点数で誤差を最大限抑えながら表現できる下限であり, より小さい h を用いたければ, より高精度な浮動小数点数を用いる必要があるとも考えられる.

表 1: 課題 2 の出力から得られた値

小さな値	微分値	相対誤差
1.00000000000000	0.67801009884209	0.04114892279159
0.10000000000000	0.70681219018734	0.00041661458643
0.01000000000000	0.70710383491198	0.00000416666145
0.00100000000000	0.70710675172370	0.00000004166676
0.00010000000000	0.70710678089170	0.00000000041698
0.00001000000000	0.70710678118369	0.00000000000405
0.00000100000000	0.70710678112818	0.00000000008255
0.00000010000000	0.70710678090613	0.00000000039657
0.00000001000000	0.70710677313457	0.00000001138722
0.00000000100000	0.70710681754349	0.00000005141648
0.00000000010000	0.70710659549889	0.00000026260201
0.00000000001000	0.70711214661401	0.00000758786028
0.00000000000100	0.70710104438376	0.00000811306430
0.00000000000010	0.70721206668622	0.00014889618156
0.00000000000001	0.69944050551385	0.01084175102922

3 課題 3

実数解を持つ適当な非線形方程式を選び, ニュートン法を用いて解を求める. 今回は式 (4) を用いる. 課題を解くのに用いたソースコードをコード 2 に示す.

$$f(x) = e^x + \sin(x) - 3x^2 + 2x + 5 = 0 \tag{4}$$

なお, 事前に初期値 0 を用いて解を計算しており,-0.94 付近であることを確認しており, 比較に使用する解として -0.9460532773247446 を用いた. コード 2 では初期値として -17.0, -5.0, -3.0, -1.0, 1.0, 15.0 の 6 つを用いた. このプログラムの出力から表 2 を作成した.

表 2: 課題 3 のプログラムの出力から得られた値

初期値	ステップ数	近似解	相対誤差
-17.0	8	-0.9460532773247448	0.0000000000000001
-5.0	6	-0.9460532773247449	0.0000000000000002
-3.0	6	-0.9460532773247448	0.0000000000000001
-1.0	4	-0.9460532773247448	0.0000000000000001
1.0	19	-0.9460532773247446	0.0000000000000000
15.0	60	-0.9460532773247446	0.0000000000000000

試したどの初期値でも相対誤差は 10^{-16} 程度であり, 精度のよい近似解が得られていることがわかる. ステップ数については, 解の絶対値から同じ絶対値離れると同じステップ数増加すると予想していた. 実際には初期値が正の方向に大きくなるほどステップ数は大きく増大するが, 負の方向に大きくなってもステップ数が大きく増大することはなかった.

Code 2: 課題 3 で使用したソースコード

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#define Epsilon (1e-14)
//  f(x) = e^x + \sin(x) - 3x^2 + 2x + 5
double f(double x) { return exp(x) + sin(x) - 3.0 * x * x + 2.0 * x + 5.0; }

// f'(x) = e^x + \cos(x) - 6x + 2
double dfdx(double x) { return exp(x) + cos(x) - 6.0 * x + 2.0; }

double newton_method(double (*f)(double), double (*dfdx)(double),
                     double initial_value, double epsilon) {
  double x = initial_value;

  int i = 0;
  while (fabs(f(x)) > epsilon) {
    fprintf(stderr, "%d %1.16f\n", i, x);
    x = x - f(x) / dfdx(x);
    i++;
  }

  printf("iteration = %d\n", i);

  return x;
}

int main(void) {
  double ans = -0.9460532773247446;
  double initial_value[] = {-17.0, -5.0, -3.0, -1.0, 1.0, 15.0};

  printf("ans = %1.16f\n\n", ans);

  for (int i = 0; i < 6; i++) {
    double value = initial_value[i];
    printf("initial value = %1.16f\n", value);
```

```
37      double x = newton_method(f, dfdx, value,
38                               Epsilon);  // find x ~= -0.94... from initial_value
39      printf("%1.16f %1.16f\n\n", x, fabs(x - ans));
40    }
41  }
```

4　課題 4

課題 3 と同様に実数解を持つ適当な非線形方程式を選び, ニュートン法を用いて解を求める. 今回は式 (5) を 2 つの方程式を用いる. 実行結果は表 3 のようになった.

$$\begin{aligned} f(x) &= x^2 - 2x + 1 \\ g(x) &= x^2 - 3x + 2 \end{aligned} \tag{5}$$

表 3: 課題 4 のプログラムの出力から得られた値

関数	ステップ数	近似解	相対誤差
f(x)	20	1.0000000954936352	0.0000000954936352
g(x)	4	0.9999999999999994	0.0000000000000006

$f(x)$ と $g(x)$ で必要なステップ数が大きく違う原因を考察する. $f(X)$ と $g(X)$ は, 次数は同じであるが,$f(x)$ は 2 重解を持っているという違いがあり, これが大きく差が出た原因と考えられる. 詳細な考察は 5 節で行う.

5　課題 5

4 節で扱った 2 つの方程式について, ニュートン法における各ステップの相対誤差を計算し, 表 4 および表 5 に示した. これらから,$f(x)$ は 1 次収束しており,$g(x)$ は 2 次収束していることがわかる.

ニュートン法の原理から, 重解の場合に収束が遅い理由を考察する. ニュートン法は, 式 (6) のような反復式で表される試行を繰り返すことで解を求める.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{6}$$

一般的に, これは 2 次収束する (誤差が 2 乗に比例して小さくなる) ことが知られている. しかし重解の場合, 式 (6) の分母の部分が 0 に近似できるため, 収束性が悪くなり, 具体的には 1 次収束する (誤差が 1 乗に比例して小さくなる) ことが知られている. このため, 収束が大幅に遅くなり, また誤差が変化する速度も小さくなるので, 収束条件ぎりぎりの値が最終的な近似解になり, 相対誤差が大きくなったと考えられる.

表 4: 課題 4 の $f(x)$ のステップごとの相対誤差

ステップ数	相対誤差
1	0.0500000000000000
2	0.0250000000000006
3	0.0124999999999988
4	0.0062499999999950
5	0.0031249999999954
6	0.0015624999999946
7	0.0007812499999749
8	0.0003906250000247
9	0.0001953125001395
10	0.0000976562501183
11	0.0000488281251319
12	0.0000244140632435
13	0.0000122070303643
14	0.0000061035110224
15	0.0000030517538190
16	0.0000015258743711
17	0.0000007629151879
18	0.0000003814973588
19	0.0000001908810769
20	0.0000000954936352

表 5: 課題 4 の $g(x)$ のステップごとの相対誤差

ステップ数	相対誤差
1	0.0124999999999997
2	0.0001524390243903
3	0.0000000232305734
4	0.0000000000000006

6 課題 6

複素解を持つ式 (7) をニュートン法を用いて解く.

$$\begin{aligned} &z = x + iy \text{ としたとき,} \\ &f(z) = z^2 + 1 = 0 \end{aligned} \tag{7}$$

これは, 実部と虚部の式を分けて式 (8) および式 (9) のように表すことができる.

$$f_{real}(x,y) = x^2 - y^2 + 1 = 0 \tag{8}$$
$$f_{imag}(x,y) = 2xy = 0 \tag{9}$$

これを, ニュートン法を多変数関数に適用する際の式に起こしていく. $J(x)$ を $f(x)$ のヤコビ行列として $f(x)$ を $x^{(k)}$ のまわりでテイラー展開すると, 式 (10) のようになる.

$$f(x) = f(x^{(k)}) + J(x^{(k)})(x - x^{(k)}) + O(\|x - x^{(k)}\|^2) \tag{10}$$

ここで,$f(x) = 0$ の代わりに $\tilde{f}(x) \simeq f(x^{(k)}) + J(x^{(k)})(x - x^{(k)}) = 0$ を解いてそれを $x^{(k+1)}$ とすると, 式 (11) のようになる.

$$x^{(k+1)} = x^{(k)} - J(x^{(k)})^{-1} f(x^{(k)}) \tag{11}$$

これを式 (8) および式 (9) に適用する. まず, ヤコビ行列 $J(x)$ を求める. これは式 (12) のようになる.

$$\begin{aligned} J(x) &= \begin{pmatrix} \frac{\partial f_{real}}{\partial x} & \frac{\partial f_{real}}{\partial y} \\ \frac{\partial f_{imag}}{\partial x} & \frac{\partial f_{imag}}{\partial y} \end{pmatrix} \\ &= \begin{pmatrix} 2x & -2y \\ 2y & 2x \end{pmatrix} \end{aligned} \tag{12}$$

よって逆行列 $J(x)^{-1}$ は式 (13) のようになる.

$$J(x)^{-1} = \frac{1}{4x^2 + 4y^2} \begin{pmatrix} 2x & 2y \\ -2y & 2x \end{pmatrix} \tag{13}$$

式 (11) に $J(x)^{-1}$ を代入して, この方程式における反復式は式 (14) のようになる.

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \frac{1}{4x^2 + 4y^2} \begin{pmatrix} 2x & 2y \\ -2y & 2x \end{pmatrix} \begin{pmatrix} x^2 - y^2 + 1 \\ 2xy \end{pmatrix} \\ &= x^{(k)} - \frac{1}{4x^2 + 4y^2} \begin{pmatrix} 2x(x^2 - y^2 + 1) + 2y(2xy) \\ -2y(x^2 - y^2 + 1) + 2x(2xy) \end{pmatrix} \\ &= x^{(k)} - \frac{1}{4x^2 + 4y^2} \begin{pmatrix} 2x^3 + 2xy^2 + 2x \\ 2x^2 y + 2y^3 - 2y \end{pmatrix} \end{aligned} \tag{14}$$

式 (14) をもとに実際に解を計算するプログラムを C 言語で実装したものがコード 3 である. これを実行した出力結果は下の通りである.

```
> ./newton-cplx
x = -0.0000000000000000, y = 1.0000000000000000
iterations: 6
```

式 (7) の解は $z = i$ であるから, 正しい解を求められていることがわかる.

Code 3: 課題 6 で使用したソースコード

```c
#include <math.h>
#include <stdio.h>

#define Epsilon (1e-14)

double f_re(double x, double y) { return x * x - y * y + 1.0; }

double f_im(double x, double y) { return 2.0 * x * y; }

void newton_step(double *x, double *y) {
  double tx = *x;
  double ty = *y;

  double dx = -1 / (4 * tx * tx + 4 * ty * ty) *
              (2 * tx * tx * tx + 2 * tx * ty * ty + 2 * tx);
  double dy = -1 / (4 * tx * tx + 4 * ty * ty) *
              (2 * ty * ty * ty + 2 * tx * tx * ty - 2 * ty);

  *x = tx + dx;
  *y = ty + dy;
}

int main() {
  double x = 1.0;
  double y = 2.0;

  int iterations = 0;

  while (fabs(f_re(x, y)) > Epsilon || fabs(f_im(x, y)) > Epsilon) {
    newton_step(&x, &y);
    iterations++;
  }

  printf("x = %1.16f, y = %1.16f\n", x, y);
  printf("iterations: %d\n", iterations);

  return 0;
}
```

7　課題 7

3 節で用いた式 (4) を割線法で解いてみる．初期値として -5.0 を使用し，またこの際最初にニュートン法が計算した値は -2.5523890241644089 である．これら 2 つの値から割線法で解を求める C 言語プログラムをコード 4 のように作成した．

このプログラムの出力は以下の通りである．

```
> ./secant
ans = -0.9460532773247446

secant method
0 -2.5523890241644089
1 -1.7161278400922084
2 -1.1555462337167537
3 -0.9810892323798662
4 -0.9479554431757200
5 -0.9460715750647819
6 -0.9460532869781404
7 -0.9460532773247937
iteration = 8
-0.9460532773247448 0.0000000000000001
```

以上からわかるように，割線法では 7 ステップで解が収束した．なお，同じ条件でニュートン法で解くと 6 ステップで収束する．このため，割線法はニュートン法よりも収束が遅いことがわかる．実際，割線法は $(1+\sqrt{5})/2 \simeq 1.6$ 次で収束することが知られており，ニュートン法の 2 次よりは遅く収束する．

精度については，少なくとも今回の例ではニュートン法とほぼ同じくらいのものが出ているので，実用上問題はないだろう．しかし，今回はある程度解がわかっている状態で始めたが，そうでない場合はある程度解に近いであろう値を 2 つ推測しないと使えないため可能であればニュートン法を用いたほうがよいだろう．

Code 4: 課題 7 で使用したソースコード

```
1  #include <math.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  #define Epsilon (1e-14)
6  //  f(x) = e^x + \sin(x) - 3x^2 + 2x + 5
7  double f(double x) { return exp(x) + sin(x) - 3.0 * x * x + 2.0 * x + 5.0; }
8
9  double secant_method(double (*f)(double), double initial_value1,
10                       double initial_value2, double epsilon) {
11   double x1 = initial_value1;
```

```
12    double x2 = initial_value2;
13
14    int i = 0;
15    while (fabs(f(x2)) > epsilon) {
16      fprintf(stderr, "%d %1.16f\n", i, x2);
17      double x3 = x2 - f(x2) * (x2 - x1) / (f(x2) - f(x1));
18      x1 = x2;
19      x2 = x3;
20      i++;
21    }
22
23    printf("iteration = %d\n", i);
24
25    return x2;
26  }
27
28  int main(void) {
29    double ans = -0.9460532773247446;
30    double initial_value_1 = -3.0;
31    double initial_value_2 = -1.5261471295549704;
32
33    printf("ans = %1.16f\n\n", ans);
34    printf("secant method\n");
35    double x = secant_method(f, initial_value_1, initial_value_2,
36                             Epsilon);  // find x ~= -0.94... from initial_value
37    printf("%1.16f %1.16f\n\n", x, fabs(x - ans));
38  }
```