数値計算 Class-2 演習

21T2166D 渡辺大樹

2023 年 4 月 20 日

1　演習内容

Class-2 の演習ではコンピューターで任意の方程式を解くためのアルゴリズム、二分法とニュートン法を C 言語で実装し、実際にいくつかの複雑な方程式について解いていった。

また以下で用いている閉区間や初項は用いる関数をあらかじめ Python でグラフ化しそこから判断している。

1.1　二分法

二分法は、解がある閉区間 $[a,b](a,b$ は異符号$)$ にある時、その区間の中間値 $c=\frac{a+b}{2}$ を求め、a ないし b と異符号かどうかを評価し異符号となった方の値の間にできた閉区間でさらに同じ操作を繰り返すことで解を求めるアルゴリズムである。このアルゴリズムは中間値の定理を利用している。

以下ソースコード 1 が C 言語で二分法を実装し、演習で利用したコードである。

ソースコード 1　bisection.c

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

double f(double x, int i){
	//iの値に応じて適切なf(x)を呼び出す関数
	switch (i){
	case 0:
	case 1:
		return pow(x,4.0) - 3*x + 1;
	case 2:
		return cos(x) - x;
	case 3:
		return pow(M_E, x) - 1/x;
```

```
        default:
                break;
        }
}
#define MAX_ITERATIONS 100
double bisection(double a, double b, double eps, int i) {
        double c;
        int it = 0;
        do {
                c = (a+b)/2.0; //a,bを二分したx座標の値,cを用意
                if (f(a, i)*f(c, i) < 0.0) { //符号が異なるとき
                        b = c;
                }
                else { //符号が一致するとき
                        a = c;
                }
                it++;
                } while(fabs(b-a) >= eps && it <= MAX_ITERATIONS); //a,bの差が2e-30より小さくなるか計算回数が100回を超えるときループを抜け出す
        c = (a+b)/2.0;
        int func[] = {1,1,2,3};
        printf("function_%d it = %d \n", func[i], it);

        return c;
}
int main() {
        double date[4][2] = {{1.0, -1.0},{2.0, 1.0},{1.0, 0.0},{1.0, 0.0}};
        //各関数における解を含む閉区間の配列
        double epsilon = 2e-30;
        for(int i = 0; i < 4; i++){
                //iの値で呼び出す関数と二分法を閉区間を変更できるように実装している
                double root = bisection(date[i][0],date[i][1],epsilon,i);
                printf("a~b閉区間の解はx=%f , f(x) = %g\n", root, f(root,i));
        }

        return 0;
}
```

なおこのコードについては、eALPS 上で示されたものから少し変更しており、すべての方程式の結果を一度に表示することを可能にしている。

1.2　ニュートン法

ニュートン法は、関数 $f(x)$ の $x = x_0$ についてのテイラー展開の一次までの項、すなわち関数の一次近似

$$y = f'(x)(x - x_0) + f(x_0)$$

を利用したアルゴリズムである。この方程式をを $y = 0$ とし x について解くと

$$x = x_0 - \frac{f(x_0)}{f'(x)}$$

となる。この式は

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

と漸化式として近似できる。

こうすることで関数 $f(x)$ とその一階微分となる $f'(x)$、x_0 を指定することで x_n が解に収束していく。精度は $|x_{n+1} - x_n| < \epsilon$ となる ϵ をコード上で指定してやれば任意に決めることができる。

以下ソースコード 2 が C 言語でニュートン法を実装し、演習で利用したコードである。

ソースコード 2　newton.c

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

double f(double x, int i){
	//i の値に応じて適切な f(x)を呼び出す関数
	switch(i){
	case 0:
	case 1:
		return pow(x,4.0) -3*x +1;
	case 2:
		return cos(x) - x;
	case 3:
		return pow(M_E,x) -1/x;
	case 4:
		return (pow(x,3.0) - pow(x,2.0)*3.0 +9.0*x - 8.0);
	default:
		break;
	}
}
```

```

double f_prime(double x, int i){
        //i の値に応じて適切な f_prime(x)を呼び出す関数
        switch (i){
        case 0:
        case 1:
                return 4.0 * pow(x,3.0) - 3;
        case 2:
                return -sin(x) - 1;
        case 3:
                return pow(M_E,x) + 1/pow(x,2);
        case 4:
                return (3.0*pow(x,2.0) - 6.0*x +9.0);
        default:
                break;
        }
}

#define EPSILON 0.000001
double newton(double np, int i){
        int it = 0;
        double xk = np, xk1;
        while(1){
                it++;
                xk1 = xk - f(xk, i)/f_prime(xk, i);
                // 一次近似y=f(a)+f’(a)(x-a)においてy=0とし式を変換し近似して得られた式
                printf("Try %d Solution %.7f \n",it, xk1);
                        if(fabs(xk1 - xk) < EPSILON){ //二数の差が 0.000001以下になるまで繰り返す
                                break;
                        }
                xk = xk1;
                }
        return xk;
        }
int main(){
        int i = 0;
        double initial_array[] = {1.0, -1.0, -1.0, 0.5, -1.0};
        int func[] = {1,1,2,3,4};
        //x_0 となる数の配列、ここが初期地点となる。
        for(i; i < 5; i++){
```

```
            //
                i の値で呼び出す関数とニュートン法を始める初項を変更できるように実装している

            printf("%d_function Final Solution %.6f \n", func[i], newton(
                initial_array[i], i));
        }

        return 0;
}
```

なおこのコードについても、eALPS 上で示されたものから少し変更しており、すべての方程式の結果を一度に表示することを可能にしている。

2 演習結果

以下に演習結果を示す

2.1 二分法

二分法のコードを実行した結果、以下 3 のように出力された。

ソースコード 3 bisection_result

```
function_1 it = 101
a~b 閉区間の解は x=0.337667 , f(x) = 5.44812e-017
function_1 it = 101
a~b 閉区間の解は x=1.307486 , f(x) = -6.32307e-016
function_2 it = 54
a~b 閉区間の解は x=0.739085 , f(x) = 3.33067e-016
function_3 it = 101
a~b 閉区間の解は x=0.567143 , f(x) = 3.27537e-016
```

以上の結果は Python での事前のグラフシミュレートと相違なく、かなりよい精度の数値が出せていると思う。

2.2 ニュートン法

続いて、ニュートン法のコードを実行した結果、以下 4 のように出力された。

ソースコード 4 newton_result

```
Try 1 Solution 2.0000000
~
Try 7 Solution 1.3074861
```

```
1_function Final Solution 1.307486

Try 1 Solution -0.2857143
~
Try 5 Solution 0.3376668
1_function Final Solution 0.337667

Try 1 Solution 8.7162170
~
Try 8 Solution 0.7390851
2_function Final Solution 0.739085

Try 1 Solution 0.5621873
~
Try 4 Solution 0.5671433
3_function Final Solution 0.567143

Try 1 Solution 0.1666667
~
Try 5 Solution 1.1659056
4_function Final Solution 1.165906
```

出力の一部は省略しているがこちらもかなり良い精度の数値が計算できている。また二分法では 100 回計算した方程式もあったがこちらでは最大でも 10 回未満の計算量で済んでいる。

3 考察

上記の結果から、二分法とニュートン法、その両方が 10^{-5} 以上の精度を出して方程式を解くことができる簡易なアルゴリズムであることが理解できた。

またニュートン法は漸化式の性質上 $O(\sqrt{n})$ のオーダーで結果が収束しさらに結果の精度も高いため、二分法より複雑なアルゴリズムでも有用だと感じた。