MAT3008 수치해석 Homework#2 report

컴퓨터소프트웨어학부
2020048868 오수아

Test environment:

Apple M3, Sonoma 14.5
Apple clang 15.0.0

01. Method1 – using Bessel function

Result:

```
(base) ➜  NA cc -o run ./Homework#2/code/test.c ./NRs/ansi/recipes/bessj0.c ./NRs/ansi/recipes/bessj1.c ./NRs/ansi/recipes/rtbis.c ./NRs/ansi/recipes/rtflsp.c
./NRs/ansi/recipes/rtsec.c ./NRs/ansi/recipes/rtnewt.c ./NRs/ansi/recipes/rtsafe.c ./NRs/ansi/recipes/zbrak.c ./NRs/ansi/other/nrutil.c

(base) ➜  NA ./run
*** Range [2.350000, 2.440000] ***

Bisection: 2.404826

Linear Interpolation: 2.404826

Secant: 2.404825

Newton-Raphson: 2.404825

Newton with bracketing: 2.404825

*** Range [5.500001, 5.590001] ***

Bisection: 5.520078

Linear Interpolation: 5.520078

Secant: 5.520078

Newton-Raphson: 5.520078

Newton with bracketing: 5.520078

*** Range [8.650006, 8.740006] ***

Bisection: 8.653728

Linear Interpolation: 8.653728

Secant: 8.653728

Newton-Raphson: 8.653728

Newton with bracketing: 8.653728

**********
```

Implement: 대체로 NR에 포함되어있는 methods를 그대로 사용. Newton-Raphson과 Newton with bracketing의 경우 Bessel의 도함수가 필요했기 때문에 bessj1.c를 참조하고 bessj0_를 구현함.

```c
void bessj0_(float x, float *f, float *df){ //derivative
    *f = bessj0(x);
    *df = -bessj1(x);
}
```

Muller method는 강의자료를 참고해서 알고리즘대로 설계함. sng함수도 간단하게 만들어놓음.

Muller method(II)

- Algorithm

Given initial approximations p_0, p_1, and p_2, generate

$$p_3 = p_2 - \frac{2c}{b + \operatorname{sgn}(b)\sqrt{b^2 - 4ac}},$$

where

$$c = f(p_2),$$

$$b = \frac{(p_0 - p_2)^2[f(p_1) - f(p_2)] - (p_1 - p_2)^2[f(p_0) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)},$$

and

$$a = \frac{(p_1 - p_2)[f(p_0) - f(p_2)] - (p_0 - p_2)[f(p_1) - f(p_2)]}{(p_0 - p_2)(p_1 - p_2)(p_0 - p_1)}.$$

Then continue the iteration, with p_1, p_2, and p_3 replacing p_0, p_1, and p_2.

```c
float muller(float (*func)(float), float x1, float x2, float xacc) {
    float p1 = x1, p2 = x2, p3;
    float p0 = (p1 + p2) / 2; //median
    float f0, f1, f2, h1, h2, a, b, c, rad, den;
    int iter, max_iter = 100;

    for (iter = 0; iter < max_iter; iter++) {
        f0 = func(p0);
        f1 = func(p1);
        f2 = func(p2);
        //get a,b,c
        c = f2;
        b = ((p0 - p2) * (p0 - p2) * (f1 - f2) - (p1 - p2) * (p1 - p2) * (f0 - f2)) /
            ((p0 - p2) * (p1 - p2) * (p0 - p1));
        a = ((p1 - p2) * (f0 - f2) - (p0 - p2) * (f1 - f2)) /
            ((p0 - p2) * (p1 - p2) * (p0 - p1));
        //Discriminant, root formulas
        rad = sqrt(b * b - 4 * a * c);
        den = b + sgn(b) * rad;
        p3 = p2 - (2 * c) / den;
        //end of iteration
        if (fabs(p3 - p2) < xacc) {
            return p3;
        }
        p0 = p1;
        p1 = p2;
        p2 = p3;
    }

    //error and escape
    printf("Muller method failed to converge\n");
    return p3;

}
```

Trouble shooting: 특기사항 없음

02. Comparing each methods

수렴속도가 가장 빠른 것은 O(n^2)에 가까운 속도로 수렴하는 Newton-Raphson Method와 도함수를 사용하지 않는 경우 O(1.618n)로 Secant Method이다. 두 방법 모두 접선의 기울기를 구하면서 접근하기때문에 빠른 것으로 예상된다. 다만 초기값을 잘못설정할 경우 발산하면서 결과를 얻지 못하기에 안정성이 떨어진다. Newton with Bracketing은 bracket의 범위와 초기값이 적절히 설정되어있을 경우 Newton-Raphson Method와 거의 비슷한 성능을 안정적으로 보여줄 것으로 생각된다. 다만 조건이 조건이다보니 최적의 수단으로 선정되기에는 한계가 있어 보인다. 이분법과 선형보간법은 모두 계산이 단순해 결과물이 안정적이나 반복 횟수가 많기 때문에 수렴 속도가 비교적 많이 느리다.

03.Method2 – using original nonlinear function

Result:

```
(base) ➜  NA cc -o run2 ./Homework#2/code/nonlin.c ./NRs/ansi/recipes/rtbis.c ./NRs/ansi/recipes/rtflsp.c ./NRs/ansi/recipes/rtsec.c ./NRs/ansi/recipes/rtnewt.c ./NRs/ansi/recipes/rtsafe.c ./NRs/ansi/recipes/zbrak.c ./NRs/ansi/other/nrutil.c

(base) ➜  NA ./run2
*** Range [1.540000, 1.630000] ***

Bisection method: 1.609902

Linear Interpolation method: 1.609902

Secant method: 1.609902

Newton-Raphson method: 1.609902

Newton with bracketing: 1.609902

Muller Method: 1.609902

*** Range [2.260000, 2.350000] ***

Bisection method: 2.276429

Linear Interpolation method: 2.276429

Secant method: 2.276429

Newton-Raphson method: 2.276429

Newton with bracketing: 2.276429

Muller Method: 2.276429

*** Range [2.620000, 2.710000] ***

Bisection method: 2.632177

Linear Interpolation method: 2.632177

Secant method: 2.632177

Newton-Raphson method: 2.632177

Newton with bracketing: 2.632177

Muller Method: 2.632177

**********
```

Implement: transcendental equation sin(x)–0.5x=0을 채택. 삼각함수의 주기성이 더해져서 더 다양한 구간에서 해를 찾아볼 수 있지 않을까 라는 발상에 기인함. 이전 01에서 만든 구조를 그대로 반영. Bessel function이 들어가는 부분만 새로운 함수로 대체하는 수정 과정을 거침. 막상 실행을 해보니 근이 나오는 구간이 단 한개밖에 도출되지 않음. 샘플 수와 구간을 조정해봐도 결과에 변화가 없음. 그래서 기준이 되는 수식을 sin(x)–0.5x+cos(5x)로 수정.

```c
float nonlin(float x){
    return sin(x) - 0.5 * x + cos(5 * x);
}
```

```c
void nonlin_(float x, float *f, float *df){ //derivative
    *f = sin(x) - 0.5 * x + cos(5 * x);
    *df = cos(x) - 0.5 - 5 * sin(5 * x);
}
```

Trouble shooting: 특기사항 없음