고소실 4주차 과제

20191571 김세영

1. **프로그램의 구동 방법 및 간략한 소개**

차례로 problem 1-1, 1-2,1-4, homework 1, homework 2 순이다.

실습 1-1 에서 f1의 root를 구하기 위해 Newton-Raphson Method X0 , Secant Method의 X0 X1을 입력한다.

1-1에서 f2의 root를 구하기 위해 Newton-Raphson Method X0 , Secant Method의 X0 X1을 입력한다.

1-2에서 f3의 root를 구하기 위해 Newton-Raphson Method X0 3개를 입력한다. 밑의 예시에서는 주어진 구간의 중점을 입력하였다.

1-4에서 f1(x)=lnx-1=0의 근(e)을 구하기 위해 Newton-Raphson Method X0, single precision으로 작성한 Newton-Raphson Method X0을 입력한다.

숙제 1에서 f1, f2, f3의 root를 Bisection Method로 구하기 위해 a0 b0을 3번 입력한다.

f1,f2,f3의 root를 newton-Raphson Method로 구하기 위해 X0 3번 입력한다.

f1,f2,f3의 root를 Secant Method로 구하기 위해 X0 X1를 3번 입력한다.

숙제 2에서 주어진 식의 root를 구하기 위해 Newton-Raphson Method X0을 입력한다.

입력 후 콘솔창에서 구한 근을 알 수 있고, result.txt에서는 반복문 수행 과정이 저장되어있다.

1. **실습**

**1)1-1**

i)

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

ii) **수행결과:**

f1

Newton-Raphson Method: root= 3.057103549998436e+00

Secant Method: root= 3.057103549917540e+00

f2

Newton-Raphson Method: root= 6.819748188604164e-01

Secant Method: root= 6.819748117285146e-01

근이 맞는지 확인하는 방법 : 최종적으로 구한 값을 식에 대입하여 결과값의 절대값이 초기값을 대입했을 때의 절대값보다 0에 근접하게 나오는지 확인한다.

|f(xn1)|

f1: Newton-Raphson Method:

초기값: 9.861228866810978e-02 결과: 6.609823799408332e-12

Secant Method:

초기값: 2.613705638880109e+00 결과 :1.379611980212303e-10

f2: Newton-Raphson Method:

초기값: 3.357864376269049e-01 결과: 4.453119828440322e-08

Secant Method:

초기값: 2.000000000000000e+00 결과 : 1.315406050750312e-08

모두 초기값을 대입했을 때의 결과보다 0에 근접하고, 결과 자체도 충분히 0에 근접하기 때문에 근이 맞다고 할 수 있다.

(iii) f1: Newton-Raphson Method는 n=3, Secant Method는 n=8일 때 수렴하기 때문에 Secant Method의 수렴 속도가 더 느리다.

f2: Newton-Raphson Method는 n=3, Secant method는 n=6일때 수렴하기 때문에 Secant Method의 수렴속도가 더 느리다.

Newton-Raphson은 차수가 2, Secant는 차수가 1.62로 수렴하기 때문에 이론적으로도 Secant의 수렴속도가 더 느리고, 이것을 실제 수행결과에서도 확인하였다.

(iv) 주어진 식의 해가 여러 곳에 존재할 가능성이 있다.

f1과 f2에 각각 newton-Raphson, secant순으로 임의의 초기값

10.0

1.0 50.0

10.0

0.0 5.0

을 대입했을 때,

Newton-Raphson Method: root= 3.057103549995379e+00

Secant Method: root= 1.412391159024317e+00

Newton-Raphson Method: root= 7.007143762206204e+01

Secant Method: root= -1.000000000000000e+00

로 f1은 secant method, f2는 newton Raphson method와 secant method가 다른 값에 수렴하였다.

f1의 secant method에서 구한 근의 |f(xn1)|는 충분히 작고, n=7이기 때문에 다른 근을 찾은 것으로 볼 수 있다. 초기값을 60.0 80.0으로 바꾸었을 때는 n=13일 때 수렴하여, 수렴속도가 더 느려졌다.

f2의 secant는 n=1일 때 반복문을 종료하였지만, newton-Raphson method에서는 n=49까지 반복문을 수행하고 반복문의 최대값에 도달하여 종료하였다. 이때도 |f(xn1)|=7.062634031329169e+01으로 값을 찾지 못하였다. 따라서 항상 빠르게 수렴하는 것은 아니고, 근과 유사한 값을 초기값으로 한 경우에는 근을 잘 찾는다고 할 수 있다.

(v) (i)~(iv)의 내용에 포함.

**2)1-2**

주어진 구간의 중점 1.25 2.16 3.42 4.22 를 Newton-Raphson Method의 초기값으로 사용하여 4개의 실근을 구했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**3)1-4**

e=2.7182… 이기 때문에 이와 유사한 2.7을 초기값으로 잡았다.

텍스트이(가) 표시된 사진

자동 생성된 설명

double precision으로 구한 e= 2.718281827760716e+00,

single precision으로 구한 e= 2.718281745910645e+00

실제 e = 2.718281828459045235360287471352···이므로 double precision의 e가 더 유사하기 때문에 double precision의 정밀도가 single precision보다 더 높다고 할 수 있다.

single precision의 경우 소수 6째짜리까지 실제 e와 일치했는데, 실제 single precision의 유효숫자는 6~7개이므로 위의 결과와 일치한다고 볼 수 있다. double precision의 경우는 소수 8째짜리까지 일치했다.

1. **과제**
2. **숙제 1**

(iii) f1, f2, f3의 초기값은 기존 실습의 secant method에서 사용했던 초기값을 그대로 사용하였으며, f2는 실습에서 주어진 실근의 구간 중 [1.95,2.37]를 사용하였다.

텍스트이(가) 표시된 사진

자동 생성된 설명

위의 실습에서 구했던 근

3.057103549917540e+00

6.819748117285146e-01

2.199996643066406e+00 과 유사하므로 올바르게 근에 수렴한다고 할 수 있다. 3가지 모두 |f(xn1)| 가 EPSILON보다 작거나 유사한 정도로, 매우 작은 값이다.

(iv) Bisection 수렴 속도의 차수는 1이므로 Newton-Raphson, Secant보다 느리다.

이론적으로 수렴 속도는 Newton-Raphson>Secant>Bisection 이어야한다.

f1: Newton-Raphson은 n=3, Secant는 n=8, Bisection은 n=17일 때 수렴하므로 이론적인 수렴 속도와 일치한다.

f2: Newton-Raphson은 n=3, Secant는 n=6, Bisection은 n=15일 때 수렴하므로 이론적인 수렴 속도와 일치한다.

f3: Newton-Raphson은 n=2, Secant는 n=4, Bisection은 n=15일 때 수렴하므로 이론적인 수렴 속도와 일치한다.

1. **숙제2**

Newton-Raphson을 사용하기 위해 program1-1을 호출하였다. 초기값 x0을 먼저 출력하고, n=1부터 nMax까지 반복문을 이용하여 x1 = x0 - f / p 으로 함수의 미분값을 이용하여 구하고, fabs(x1 - x0) < EPSILON || fabs(\_f(x1)) < DELTA는 반복문 탈출 조건이므로 break로 반복문을 빠져나온다. 그렇지 않으면 x0=x1로 값을 업데이트 하고, 식에 x0을 대입한 값과 식을 미분한 식에 x0을 대입한 값을 업데이트한다.

방정식에 x를 대입하여 값을 return 하는 함수 \_f와 미분값을 return하는 함수 \_fp는 다음과 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

주어진 각도는 라디안 값이 아니므로 M\_PI를 이용하여 라디안 값으로 변환한다.

텍스트이(가) 표시된 사진

자동 생성된 설명