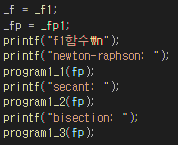
[고소실\_4주차과제]2반\_20161595\_배성현

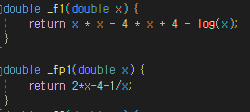
**1. 프로그램의 구동 방법 및 간략한 소개.**

프로그램은 어떤 함수 f(x)=0의 실근의 근사값을 구할 수 있도록 하여준다. 이를 구하기 위한 방법에는 Newton-Raphson 방법과, Secant방법, Bisection 방법이 있는데 Newton-Raphson방법을 사용하기 위해서는 program1\_1(fp);를, Secant방법을 사용하기 위해서는 program1\_2(fp2);를, Bisection방법을 사용하기 위해서는 program1\_3(fp);를 main함수에서 각각 아래와 같이 호출하면 된다.



(main의 일부)

이 때, 구하고자 하는 함수의 원형과, 함수의 도함수를 function.cpp(double형)나, sp\_function.cpp (float형)에 구현하고 이를 my\_solver.h의 헤더파일에 선언해주어야 한다. 또 각각의 방법을 실행하기 전에, 위와 그림과 같이 구하고자 하는 함수를 main에 있는 함수의 포인터(\_f, \_fp)에 assign하여 주어야 한다.



(function.cpp 에 정의된 함수 \_f1과 해당함수의 도함수 \_fp1)



(my\_solver.h의 함수 선언부)

아무것도 고치지 않은 상태에서 처음 프로그램을 수행하게 되면, 실습의 f1함수에 대한 근을 구하기 위해 Newton-Raphson방법, Secant방법, Bisection방법을 순서대로 수행하게 되고, 그 다음으로 f2함수에 대해 Newton-Raphson방법, Secant방법, Bisection방법을 순서대로 수행하게 된다. 또 이어서 f3함수에 대한 근을 구하기 위한 Newton-Raphson방법, Secant방법, Bisection방법을 순서대로 수행하게 되는데 f3함수는 근을 4개 가지고 있기 때문에 각각의 근에 대하여 총 4번씩 구하게 된다. 이어서 실습 1-4의 함수에 대한 근을 Newton-Raphson방법으로 double precision, single precision의 방법으로 각각 구하게 되고, 과제 2의 함수에 대한 근을 Newton-Raphson방법으로 구하게 된다.

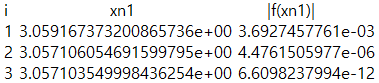
**2. 실습 문제 1-1, 1-2, 1-4에 대한 내용 기술**

**1-1 (ii). 두 방법에 대한 초기값을 각각 x0는 3.0과 x0=2.0, x1=4.0으로 설정하여 자신이 작성한 프로그램을 수행시켜 자신의 프로그램이 원하는 근을 정확히 찾고 있는지 분석하라. 과연 근이 맞는지 어떻게 확인 할 수 있을까? 논리적으로 타당한 방법으로 분석한 내용을 기술하라.**

Newton-Raphson방법과 Secant의 두 방법에 대해 초기값을 각각 x0=3.0과 x0=2.0, x1=4.0으로 설정하여 의 근을 찾았을 때에 Newton-Raphson 방법에서는 n=3, x1=3.057103549998436254e+00, f1(x1)=6.6098237994e-12의 값을 얻을 수 있었고, Secant방법에서는 n=8, x2=3.057103549917539631e+00, f1(x2)=1.3796119802e-10의 값을 얻을 수 있었다. 두 방법 모두 약 3.0571035499의 근을 도출하였고, 이에 대한 함수 값을 계산 하였을 때 Newton-Raphson의 경우 6.6098237994e-12의값, Secant의 경우 1.3796119802e-10의 값을 나타내어 이는 y=0과 비교하였을 때 아주 근소한 차이(임의로 설정한 충분히 작은 값 δ보다 작은 차이)를 보이기 때문에 f1(x)=0의 실제 근에 근사한 값이라고 할 수 있다. 또한 프로그램의 또 다른 종료조건을 보았을 때에, 반복문을 아주 많이 실행한 경우나, x1-x0이 굉장히 작아 근이 의미 있는 변화를 가지지 못하는 경우에 종료되게 되므로 이 또한 구한 근이 정확한 근을 구한 것은 아니라는 것을 의미한다. 따라서 프로그램을 수행시켰을 때에 완전히 정확한 근의 값은 아니지만, 실제 근에 근사한 값을 구할 수 있게 된다.

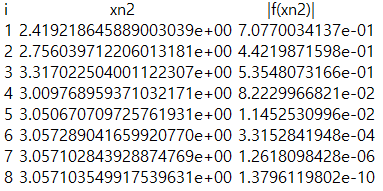
**1-1 (iii). 위에서 산출한 결과를 볼 때, 각 방법의 근의 수렴 속도가 과연 앞에서 설명한 속도로 수렴하는지 비교 분석한 후 그 결과를 보고서에 기술하라.**

의 근을 산출한 결과를 보았을 때에,



(Newton-Raphson)

Newton-Raphson방법의 경우 3번만에 값을 도출해 내었으며



(Secant)

Secant 방법의 경우 8번만에 값을 도출해 내었다는 것을 확인 할 수 있다. 또 위의 결과를 가지고 절대 오차를 알아보게 되면 아래와 같다.

|  |  |  |
| --- | --- | --- |
| Newton-Raphson | | |
| X1 | 절대오차 |  |
| 3.059E+00 | 2.063E-03 | 0.5883 |
| 3.057E+00 | 2.504E-06 | 0 |
| 3.057E+00 | 0.000E+00 |  |

|  |  |  |
| --- | --- | --- |
| Secant | | |
| X1 | 절대오차 |  |
| 2.419E+00 | 6.378E-01 | 0.6237 |
| 2.756E+00 | 3.010E-01 | 1.8177 |
| 3.317E+00 | 2.599E-01 | 0.41990 |
| 3.009E+00 | 4.733E-02 | 0.90079 |
| 3.050E+00 | 6.432E-03 | 0.65855 |
| 3.057E+00 | 1.854E-04 | 0.78416 |
| 3.057E+00 | 7.059E-07 | 0 |
| 3.057E+00 | 0.000E+00 |  |

위의 경우에 있어서는 n이 굉장히 작기 때문에, 앞에서 설명한 속도인 Newton-Raphson의 수렴속도인 과, Secant의 수렴속도인를 따르지는 않는다. 하지만 Rapson이 Secant에 비해 더 빠르게 수렴한다는 것을 확인할 수 있다.

**1-1 (iv).** **위에서 제시한 초기값 외에 임의의 초기값들을 사용하여 자신이 작성한 프로그램을 수행한 후, 이 두 방법이 항상 임의의 초기값에 대해 빠르게 수렴하는지 기술하라.**

위에서 구하였을 때에 의 정확하진 않지만 근사한 근으로 3.0571035499를 구할 수 있었는데, 임의의 초기값에 대해서도 빠르게 수렴하는지 알기 위해 아래와 같이 근사한 근에 가까운 값부터 근사한 근에서 먼 값까지 다양한 값들을 넣어 프로그램을 실행시켜 보았다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Newton-Raphson 값 | Secant x0값 | Secant x1값 | **Newton-Raphson 반복횟수** | **Secant 반복횟수** |
| 5 | 3 | 7 | **5** | **5** |
| 25 | 12 | 38 | **8** | **11** |
| 100 | 80 | 120 | **10** | **14** |
| 250 | 200 | 300 | **12** | **16** |
| 500 | 250 | 750 | **13** | **17** |
| 1000 | 500 | 1500 | **14** | **19** |

위의 다양한 임의의 초기값들을 넣어보았을 때 두 방법 모두 임의의 초기값에 대하여 빠르게 근으로 수렴하는 것을 확인 할 수 있다. 그리고 수렴 속도에 있어서, Newton-Raphson이 더 빠르다는 것 역시 확인 할 수 있다.

**1-4 (ii). 부동 소수점 연산의 정밀도가 다른 두 방법(double precision, single precision)이 구한 근의 값이 정확한 근 e와 비교하여 어떤 차이가 있는지 자신이 알아낸 사실을 기술하라.**



(double precision 버전)



(single precision 버전)

위와 같이 f1(x)=lnx-1=0의 근을 구하기 위해 double precision과 single precision으로 나누어 Newton-Raphson방법을 이용하였을 때 실제 근인 2.718281828459045235360287471352…와의 절대 오차를 구하여 보면 다음과 같다.

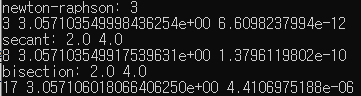
Double precision 절대 오차: 1.46103366E-08

Single precision 절대 오차: 8.25484000E-08

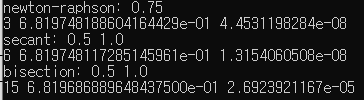
위의 절대 오차를 보았을 때 Double precision을 사용하였을 때, 더 많은 소수점 아래 자리 숫자들이 연산에 참여하기 때문에, Single precision에 비해 절대오차가 작아 실제 근과 더 가까운 근사값을 가지게 되는 것을 확인 할 수 있다.

**3. 과제 1, 2에 대한 내용기술**

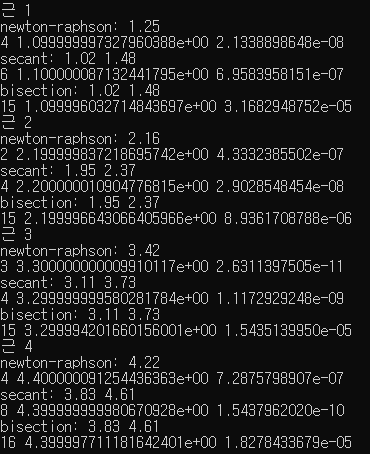
**숙제1 (iii). 프로그램이 완성되면, 실습 시간에 사용한 세 개의 함수 f1(x), f2(x), 그리고 f3(x)에 대해 적절한 초기 구간을 사용하여 올바르게 근에 수렴하는지에 대한 분석 내용을 보고서에 기술하라.**



(f1함수의 Newton-Raphson, Secant, Bisection에 따른 각각의 n, x1, |f(x1)|)



(f2함수의 Newton-Raphson, Secant, Bisection에 따른 각각의 n, x1, |f(x1)|)

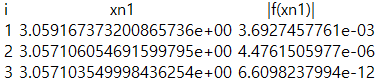


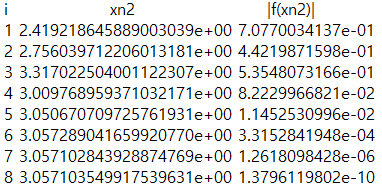
(f3함수의 Newton-Raphson, Secant, Bisection에 따른 4개의 근의 각각의 n, x1, |f(x1)|)

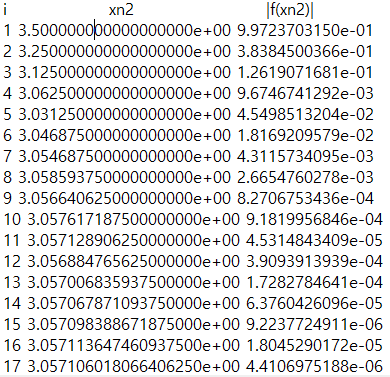
위의 사진들은 f1(x)=0, f2(x)=0, f3(x)=0의 근을 bisection 방법으로 구한 것으로써 실습시간에 하였던 Newton-Raphson, Secant 방법으로 구한 값들과 비교하였을 때 충분히 작은 값의 차이를 보이는 것을 확인 할 수 있다. 따라서 Bisection의 방법 역시 올바르게 근에 수렴한다는 것을 알 수 있다.

**숙제1 (iv). Bisection 방법의 수렴 속도는 선형적인, 즉 형태를 보인다. 위의 세 함수에 대한 방정식을 대상으로 Newton-Raphson방법, Secant 방법, 그리고 Bisection 방법을 적용하여 보고, 과연 각 방법이 이론적인 수렴 속도를 보이는지를 분석하고 그 내용을 기술하라.**

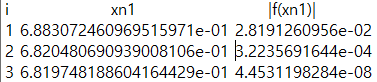
f1(x)의 수렴 속도:

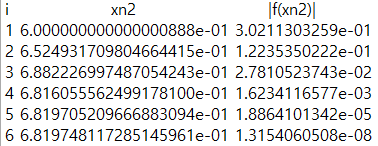
(Newton-Raphson 방법)

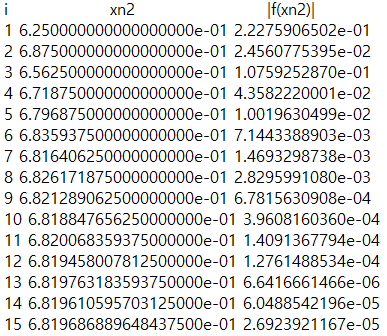
(Secant 방법)

(Bisection 방법)

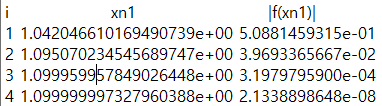
f2(x)의 수렴 속도:

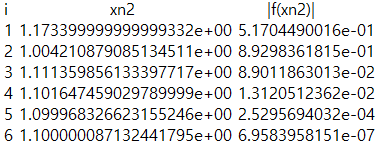
(Newton-Raphson 방법)

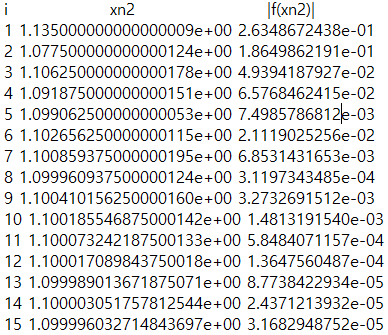
(Secant 방법)

(Bisection 방법)

f3(x)의 수렴 속도:

(Newton-Raphson 방법)

(Secant 방법)

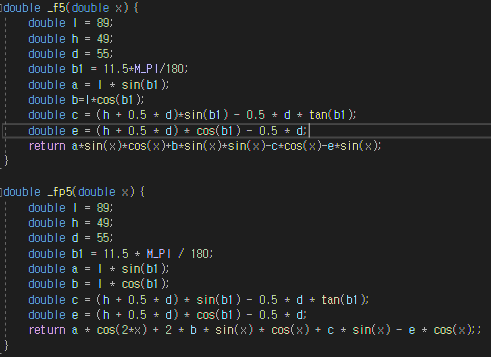
(Bisection 방법)

위의 실습 보고서 항목에 적었듯이, Newton-Raphson과, Secant의 경우에 n이 굉장히 작기 때문에, 앞에서 설명한 속도인 Newton-Raphson의 수렴속도인 과, Secant의 수렴속도인를 따르는 것을 쉽게 확인하기 어려웠다. Bisection의 경우 역시 이론적 수렴 속도인 을 따르는 것을 쉽게 확인하기는 어려웠는데 이는 실제 근을 가지고 절대오차를 계산하는 것이 아닌 실제 근에 근사한 값을 가지고 절대오차를 계산을 하기 때문에 쉽게 이론상의 공식과 맞게 답이 나오지는 않는 것 같다.

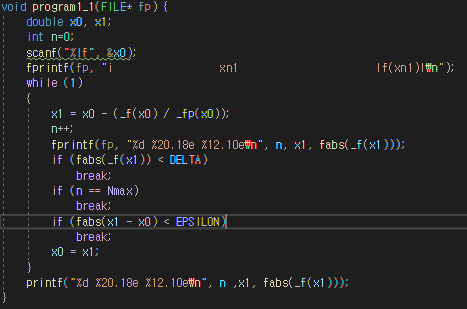
|  |  |  |  |
| --- | --- | --- | --- |
| f1(1) Bisection | | | |
| N | X1 | 절대오차 |  |
| 11 | 1.1001E+00 | 7.7209E-5 | 0.273 |
| 12 | 1.1000E+00 | 2.1057E-5 | 0.333 |
| 13 | 1.1000E+00 | 7.0190E-6 | 0.999 |
| 14 | 1.0999E+00 | 7.0190E-6 | 0 |
| 15 | 1.0999E+00 | 0 |  |

하지만 위의 실험 결과를 바탕으로 보았을 때에 3개의 함수 모두에서 Newton-Raphson이 실행 횟수가 가장 적고, 다음으로 Secant가 적고, 다음으로 Bisection인 것을 확인 할 수 있었는데 따라서 수렴속도는 Newton-Raphson방법 > Secant방법 > Bisection 방법 순이라는 것을 알 수 있다.

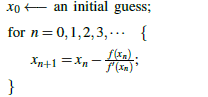
**숙제2 [참고3] 이를 위하여 어떠한 반복문을 사용하였는지 보고서에 관련 수식을 설명한 후 자신이 구한 답에 수렴해가는 반복결과와 함께 제출하라.**



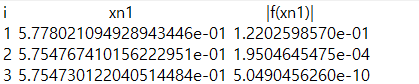
먼저 함수와, 해당 함수에 대한 도함수는 위와 같이 정의 하였다.



그리고 이에 대한 답을 구하기 위해 위와 그림과 같이 Newton-Raphson을 구현하는 반복문을 사용하였다. 즉 Newton-Raphson방법인 아래의 식을 구현한 것으로 Newton-Raphson 방법은 접선의 방정식의 성질을 이용하여 xn+1값이 점점 근에 수렴할 수 있도록 한 것이다.



또한 이를 적절한 시기에 종료하기 위해서 함수의 값이 충분히 작을 때, 또는 충분히 많은 횟수만큼 반복문이 수행 되었을 때, 또는 현재 구한 xn+1이 직전에 구한 xn에 비해 의미 있는 전진을 하지 않았을 때 반복문을 탈출할 수 있도록 하는 종료 조건을 만들어 주었다.



(반복결과)

x의 초기값으로 30도에 해당하는 값을 라디안으로 표현한 값(30\*PI/180)인 0.523598…을 입력으로 주었으며, 이를 바탕으로 최종적으로 나오게 된 n=3, x=5.754730122040514484e-01을 (x\*180/PI)하여 60분법으로 표현하게 되면 약 33도가 나오게 되는 것을 확인할 수 있다.