**고급소프트웨어실습1 [4주차과제]**

비선형 방정식의 풀이 기초

학과 : 컴퓨터공학과

분반 : 2

학번 : 20181593

이름: 계인혜

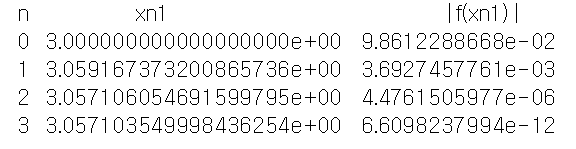
1. 프로그램의 구동 방법 및 간략한 소개

이 프로그램은 함수 포인터와 도함수 포인터를 이용하여 사용자가 입력한 함수의 근을 찾아주는 것으로, function.cpp에 자신이 근을 구하고 싶은 함수와 도함수를 입력한다. 해당 프로그램에서는 근을 구하는 방법으로 Newton-Rapson method, Secant method, Bisection method를 이용한다. 각각의 방법은 program1\_1, program1\_2, program1\_3에 구현되어 있다. 또한 double 타입 뿐만 아니라 float 타입의 계산을 수행하기 위하여 Newton-Rapson method, Secant method를 사용하여 근을 구하는 함수가 있다. 이는 각각 sp\_program1\_1, sp\_program1\_2이다.

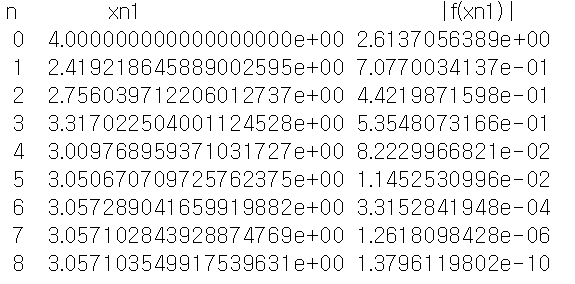
1. 실습문제 내용 기술

(ii) 두 방법에 대한 초기값을 각각 x0 = 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 확인할 수 있을까? 논리적으로 타당한 방법으로 분석한 내용을 보고서에 기술하라.

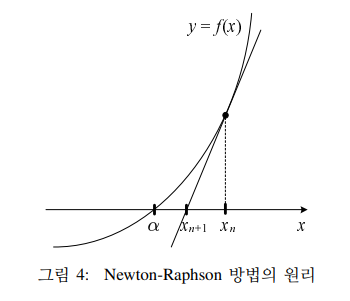
<Newton-Rapson>



<Secant>



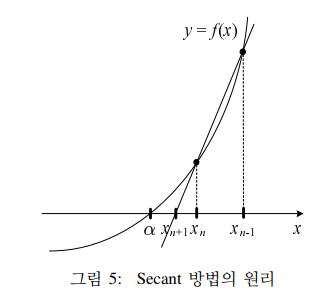
두 방법 모두 3.0571035499에 가까운 근을 출력하는 것을 확인할 수 있다.

<Newton-Rapson>

주어진 초기값 x0에 대해 간단한 반복문을 통하여 근에 수렴하는 (근 에 수렴하기 희망하는) 수열 값 xn (n = 1,2,3,···)을 구하게 된다. 다음 그림을 참고하여 자세히 살펴보도록 하자. 에서 그은 접선의 방정식은 y = f()+ f ′ ()(x−)이다. 이때 접선의 방정식의 x절편을 새로운 추정 값()으로 설정한다. 이 과정을 반복하여 비선형 방정식의 근을 찾는다.

<Secant>

Newton-Raphson 방법은 널이 쓰이는 우수한 방법이나 몇 가지 문제가 존재한다. 그 중 하나는 매번 반복문이 수행될 때마다 임의의 지점 xn에서 미분값 f ′ (xn)을 계산해주어야 한다는 점이다. 실제로 다양한 응용 문제를 수치적으로 풀고자 할 때 미분값을 계산하는 과정은 비용이 많이 들거나, 또는 매우 어려운 경우가 종종 발생한다. 따라서 Secant 방법은 미분 값 대신 평균변화율로 이를 근사하여 근을 찾는다.



이 방법은 다음 그림과 같이 접선의 기울기 대신 이전의 추정 값 두개를 사용하여 다음 추정 값을 찾는 방식으로 앞에서의 접선의 기울기를 두 점사이의 기울기로 근사하여 생각한다. 따라서 새로운 추정 값()은 과 를 이은 직선의 x절편이 된다. 이 과정을 반복하여 비선형 방정식의 근을 찾는다.

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

Newton-Rapson과 Secant 방법은 각각 , 으로 오차가 감소한다. 실제로 출력된 결과를 살펴보았을 때도 Newton-Rapson 방법이 Secant 방법보다 근에 빠르게 접근하는 것을 확인할 수 있었다.

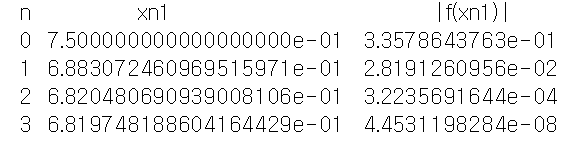
(iv) 위에서 제시한 초기값 외에 임의의 초기값들을 사용하여 자신이 작성한 프로그램을 수행한 후, 이 두 방법이 항상 임의의 초기값에 대해 빠르게 수렴하는지 보고서에 기술하라.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Newton-Rapson** | | **Secant** | | |
| **초기값** | **연산 횟수** | **초기값** | **초기값2** | **연산 횟수** |
| 30 | 8 | 29 | 30 | 12 |
| 100 | 10 | 99 | 100 | 14 |
| 9999999 | 27 | 9999999 | 10000000 | 38 |

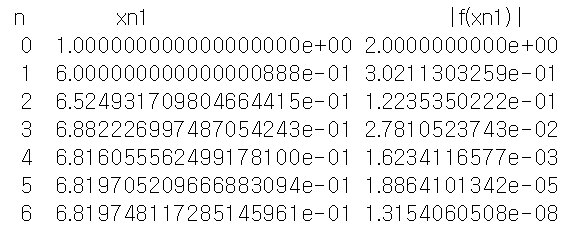
직접 프로그램에 표와 같이 초기값을 입력해봤을 때, 두 방법 모두 초기값이 실제 근에 가까울수록 연산 횟수가 줄어든다는 것을 확인할 수 있었다. 그러나 분모가 0이 되는 경우에는 해를 찾을 수 없었다. Newton-Rapson의 경우에는 접선의 기울기가 0이 되는 경우, Secant의 경우에는 두 점을 이은 직선의 기울기가 0이 되는 경우가 그렇다.

(ii) 두 방법에 대한 초기값을 각각 x0 = 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 확인할 수 있을까? 논리적으로 타당한 방법으로 분석한 내용을 보고서에 기술하라.

<Newton-Rapson>



<Secant>



두 방법 모두 6.8197481에 근사한 값을 근으로 출력하는 것을 확인할 수 있다.

정확한 근을 찾는 분석은 2.1과 동일하다.

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

2.1과 동일하다.

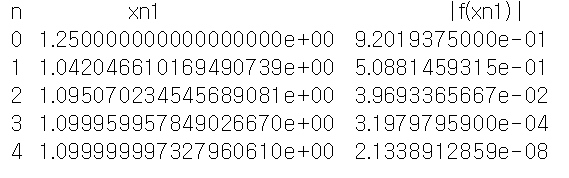
(iv) 위에서 제시한 초기값 외에 임의의 초기값들을 사용하여 자신이 작성한 프로그램을 수행한 후, 이 두 방법이 항상 임의의 초기값에 대해 빠르게 수렴하는지 보고서에 기술하라.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Newton-Rapson** | | **Secant** | | |
| **초기값** | **연산 횟수** | **초기값** | **초기값2** | **연산 횟수** |
| 1 | 4 | 0.8 | 1 | 5 |
| 1.25 | 5 | 1 | 1.5 | 6 |
| 0.75 | 3 | 0.5 | 1 | 6 |

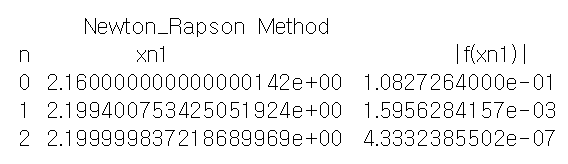
지금 어떤 근˙ 분˙ 리˙ (root separation) 방법을 사용하여, 이 방정식은 네 개의 서로 다른 실근을 가지며, 각 실근은 [1.02,1.48], [1.95,2.37], [3.11,3.73], [3.83,4.61] 구간에 존재한다는 사실을 밝혀냈다. 이 사실을 근거로 Newton-Raphson 방법을 사용하여 모든 실근을 구하라.

<Newton\_Rapson>

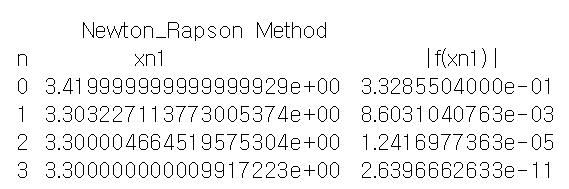
(x1)



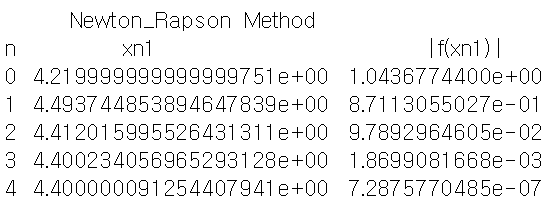
(x2)



(x3)

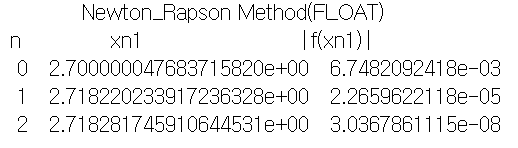


(x4)

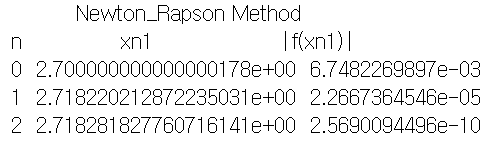


다음 f1(x) = lnx − 1 = 0과 같이 근을 알고 있는 비선형 방정식을 고려하자 (잘 알다시피 근은 e = 2.718281828459045235360287471352···임). 이 방 정식의 근을 Newton-Raphson 방법을 사용하여 구하려 하는데, 적절한 초기 값 x0에 대해 자신이 작성한 double-precision 버전과 single-precision 버전 각각을 사용하여 근을 구하여 보자. 이때 부동 소수점 연산의 정밀도가 다른 두 방법이 구한 근의 값이 정확한 근 e와 비교하여 어떤 차이가 있는지, 자신이 알아낸 사실을 보고서에 상세히 기술하라.

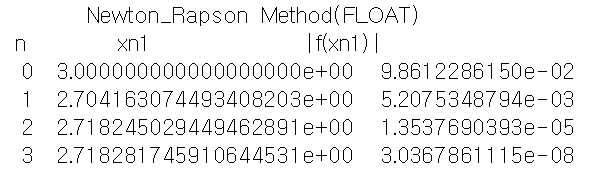
**(single precision)**



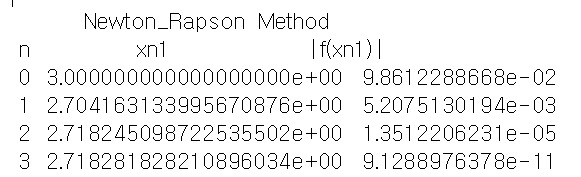
**(double precision)**



**(single precision)**

****

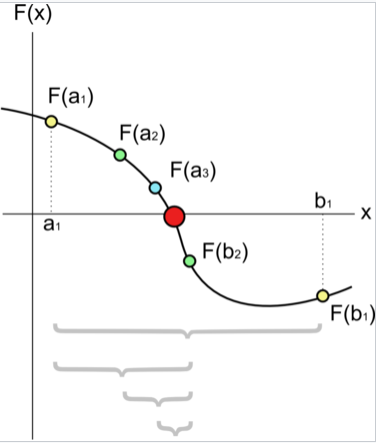
**(double precision)**

****

초기값으로 각각 2.7과 3.0을 주었을 때 정확도 면에서 double precision이 우수했다. 또한 구한 근을 대입하면 함수 값이 0이 되어야 하는데, 함수 값 측면에서도 double precision이 0에 더 가까운 것을 확인할 수 있었다. 마지막으로 두 자료형에 대해서 공통적으로 실제 근과 멀어질수록 연산 횟수가 증가한다는 것을 확인하였다.

1. 과제 내용 기술
   1. Bisection Method

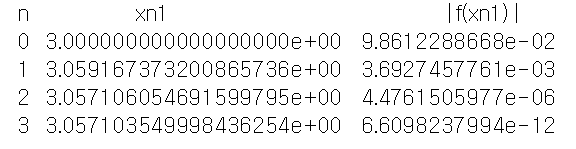
(iii) 프로그램이 완성되면, 실습 시간에 사용한 세 개의 함수 f1(x), f2(x), 그리고 f3(x)에 대해 적절한 초기 구간을 사용하여 올바르게 근에 수렴하는지에 대한 분석내용을 보고서에 기술하라.

이 방법은 다음 그림과 같이 구간 [a,b]에서 연속인 함수 f에 대하여 *f*(*a*)*f*(*b*) < 0,인 폐구간 [*a*,*b*]에 대해서 계속해서 c = (a+b)/2{\displaystyle {\frac {\left|b+a\right|}{2}}}을 하여 나오는 또다른 수 c를 하나의 폐구간 끝점으로 잡은 새로운 폐구간을 만든다. 이를 반복하면 점차 근에 가까워진다. 만약 f(a)\*f(c) < 0 이면, 실제 해는 [a,c] 구간 사이에 있으며 f(a)\*f(c) > 0 이면, 실제 해는 [c,b] 구간 사이에 있다.

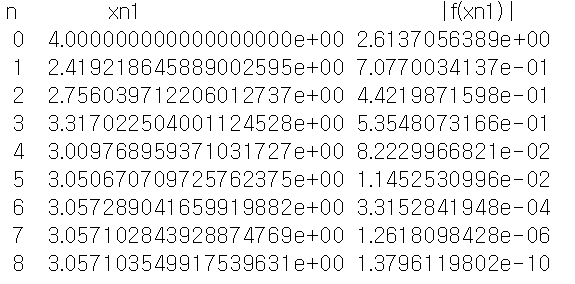
이 방법은 함수가 축에 수직이거나, x축을 지나지 않을 경우에는 해를 구할 수가 없다. 그러나 이외의 모든 함수에서는 항상 근을 구할 수 있다. 또한 다른 방법들에 비해 수렴 속도가 느리다는 특징이 있다.

**()**

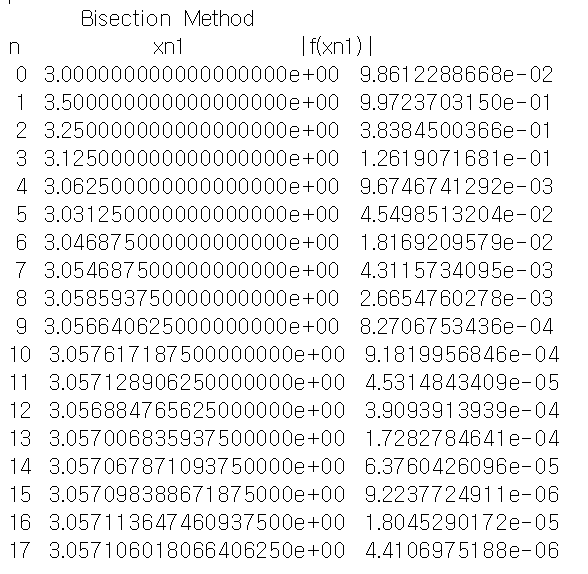
<Newton-Rapson>



<Secant>

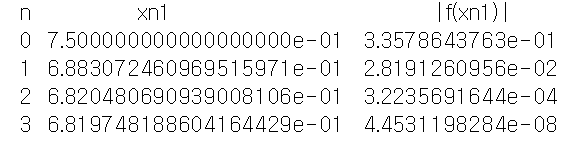


<Bisection>

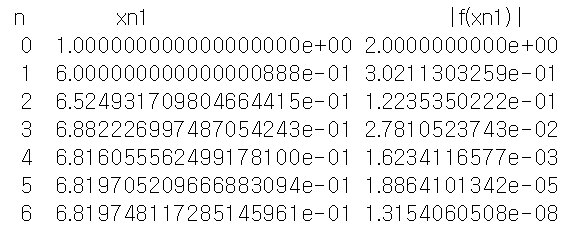


**()**

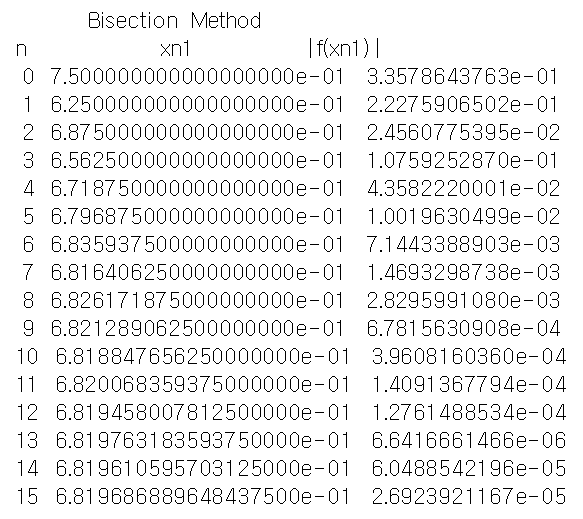
<Newton-Rapson>



<Secant>



<Bisection>

****

(iv) Bisection 방법의 수렴 속도는 선형적인, 즉 형태를 보인다. 위의 세 함수에 대한 방정식을 대상으로 Newtown-Raphson 방법, Secant 방법, 그리고 Bisection 방법을 적용해보고, 과연 각 방법이 이론적인 수렴 속도를 보이는지를 분석하고 그 내용을 보고서에 기술하라.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Newton-Rapson | Secant | | Bisection | |
|  | 초기값 | 초기값1 | 초기값2 | 초기값1 | 초기값2 |
|  | 3 | 2 | 4 | 2 | 4 |
| 연산횟수 | 3 | 6 | | 15 | |

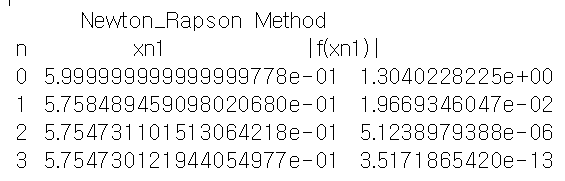
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Newton-Rapson | Secant | | Bisection | |
|  | 초기값 | 초기값1 | 초기값2 | 초기값1 | 초기값2 |
|  | 0.75 | 0.5 | 1 | 0.5 | 1 |
| 연산횟수 | 3 | 8 | | 17 | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Newton-Rapson | Secant | | Bisection | |
|  | 초기값 | 초기값1 | 초기값2 | 초기값1 | 초기값2 |
|  | 4 | 3.83 | 4.61 | 3.83 | 4.61 |
| 연산횟수 | 4 | 8 | | 16 | |

각각의 방법이 모두 수렴속도에 부합하는 연산횟수를 보였다. Newton-Rapson 방법은 , Secant 방법의 수렴 속도는 , Bisection 방법은 의 수렴속도를 갖는데, 이론적으로 보면 연산횟수는 Bisection > secant > Newton-rapson 순서로 나타나야 한다. 실험 결과 역시 이론과 일치하는 것을 확인할 수 있었다.

3.2

<Newton-Rapson method>



이렇게 나온 근을 60분법으로 바꾸면 33도와 유사한 32.97217500537520366도가 나오는 것을 확인할 수 있다.