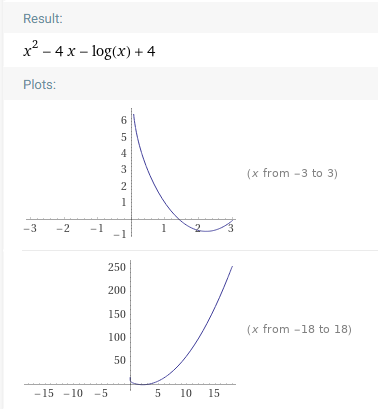
**고소실 4주차 보고서 – 2반 20171617 김소연**

**실습 1-1-1.**

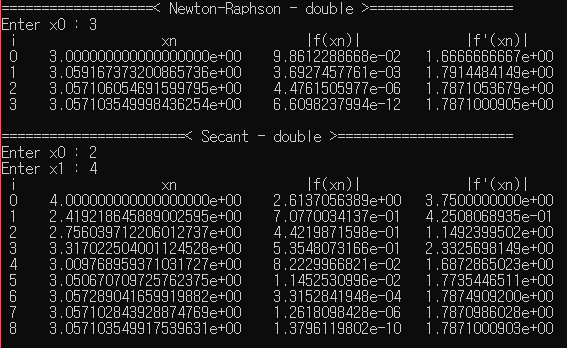
**<근 산출>**



위의 수식을 그래프로 그려보면 아래와 같다.



약 3의 값을 갖는 근이 있음을 확인할 수 있다. 이 수식에 대해 Newton-Raphson method, Secant method를 통해, 을 만족하는 근을 산출했을 때 아래와 같은 과정을 확인할 수 있다.



충분히 0에 가까워 0과 같다고 간주할 수 있는 delta 값을 0.00001로 설정하였을 때, 의 결과가 delta보다 작았기 때문에 이는 곧 과 같다고 간주할 수 있으며, 이는 그래프와 모순되지 않기에 산출해낸 x값(약 3.057)을 근으로 간주할 수 있다.

**<수렴속도>**

각 반복문에서의 오차 라고 했을 때, 이를 정리해보면 아래와 같다

|  |  |  |  |
| --- | --- | --- | --- |
| Newton-Raphson Method ( | | | |
| i |  |  |  |
| 0 | 3.000000000000000000e+00 | 0.057103549998436254 | 0.6329 |
| 1 | 3.059167373200865736e+00 | 0.002063823202429482 | 0.5880 |
| 2 | 3.057106054691599795e+00 | 0.000002504693163541 |  |
| 3 | 3.057103549998436254e+00 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Newton-Raphson Method ( | | | |
| i |  |  |  |
| 0 | 2.500000000000000000e+00 | 0.55710355028098002 | 1.7830 |
| 1 | 3.610484553123591844e+00 | 0.553381002842611824 | 0.3542 |
| 2 | 3.165573068979050220e+00 | 0.1084695186980702 | 0.5215 |
| 3 | 3.063239996571998081e+00 | 0.006136446291018061 | 0.5851 |
| 4 | 3.057125585855885230e+00 | 0.00002203557490521 |  |
| 5 | 3.057103550280980020e+00 |  |  |



반복문의 단계가 깊지 않아 비교가 어려운 면이 있으나, 둘 다 대체로 위의 값으로 산출한 c의 값이 일정 값과 근사한 것을 알 수 있다. (을 적용한 수식에서 초기 값에 대한 c의 값의 오차가 심한데, 이는 함수의 형태에 따라 특정한 근삿값에 대해 발생하는 오차인 것으로 추측된다.

|  |  |  |  |
| --- | --- | --- | --- |
| Secant Method ( | | | |
| i |  |  |  |
| 0 | 4.000000000000000000e+00 | 0.942896450082460369 | 0.7016 |
| 1 | 2.419218645889002595e+00 | 0.637884904028537036 | 0.6237 |
| 2 | 2.756039712206012737e+00 | 0.301063837711526894 | 1.8172 |
| 3 | 3.317022504001124528e+00 | 0.259918954083584897 | 0.4198 |
| 4 | 3.009768959371031727e+00 | 0.047334590546507904 | 0.9007 |
| 5 | 3.050670709725762375e+00 | 0.006432840191777256 | 0.6587 |
| 6 | 3.057289041659919882e+00 | 0.000185491742380251 | 0.7836 |
| 7 | 3.057102843928874769e+00 | 0.000000705988664862 |  |
| 8 | 3.057103549917539631e+00 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Secant Method ( | | | |
| i |  |  |  |
| 0 | 3.500000000000000000e+00 | 0.442896613505282755 | 0.5857 |
| 1 | 2.900528771771655911e+00 | 0.156574614723061334 | 0.7037 |
| 2 | 3.022198360725741217e+00 | 0.034905025768976028 | 0.8350 |
| 3 | 3.060744174275685570e+00 | 0.003640787780968325 | 0.6803 |
| 4 | 3.057027204195450931e+00 | 0.000076182299266314 |  |
| 5 | 3.057103386494717245e+00 |  |  |



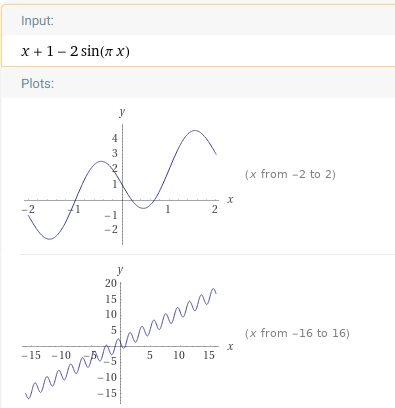
위 수식을 통해 산출해낸 c를 산출했다. (의 경우 c의 값이 대체로 특정 값에 근접함을 알 수 있다. 이를 통해 이론에서의 Secant Method의 수렴 속도와 대체로 일치하는 것을 알 수 있다. 하지만, (을 초기값으로 주었을 때에는 c의 오차가 심한 것을 확인할 수 있는데, 이는 함수의 형태에 따라, 특정 초기값에 대해서 발생하는 오차인 것으로 추측할 수 있다.

**실습 1-1-2.**

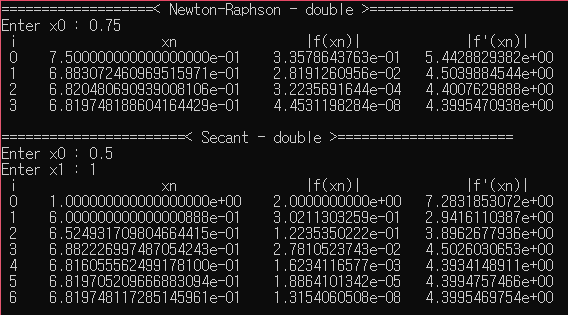
**<근 산출>**



위의 수식을 그래프로 그려보면 아래와 같다.



약 0.7의 값을 갖는 근이 있음을 확인할 수 있다. 이 수식에 대해 Newton-Raphson method, Secant method를 통해, 을 만족하는 근을 산출했을 때 아래와 같은 과정을 확인할 수 있다.



마찬가지로 delta값을 0.00001로 설정하였을 때, 의 결과가 delta보다 작았기 때문에 이는 곧 이라고 간주할 수 있으며, 이는 그래프와 모순되지 않는다. 그렇기에 산출해낸 x값을(약 6.8197) 근으로 간주할 수 있다.

**<수렴 속도>**

|  |  |  |  |
| --- | --- | --- | --- |
| Newton-Raphson Method ( | | | |
| i |  |  |  |
| 0 | 7.500000000000000000e-01 | 0.0680251811395835571 | 1.3684 |
| 1 | 6.883072460969515971e-01 | 0.0063324272365351542 | 1.8267 |
| 2 | 6.820480690939008106e-01 | 0.0000732502334843677 |  |
| 3 | 6.819748188604164429e-01 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Newton-Raphson Method ( | | | |
| i |  |  |  |
| 0 | 5.999999999999999778e-01 | 0.0819748087405525183 | 3.0846 |
| 1 | 7.027032563493527118e-01 | 0.0207284476088002157 | 1.7017 |
| 2 | 6.827059843413129103e-01 | 0.0007311756007604142 | 1.8795 |
| 3 | 6.819758135866519355e-01 | 0.0000010048460994394 |  |
| 4 | 6.819748087405524961e-01 |  |  |



반복문의 단계가 깊지 않아 비교가 어려운 면이 있으나, 둘 다 대체로 위의 값으로 산출한 c의 값이 일정 값과 근사한 것을 알 수 있다. (을 적용한 수식에서 초기 값에 대한 c의 값의 오차가 심한데, 이는 함수의 형태에 따라 특정한 근삿값에 대해 발생하는 오차인 것으로 추측된다.

|  |  |  |  |
| --- | --- | --- | --- |
| Secant Method ( | | | |
| i |  |  |  |
| 0 | 1.000000000000000000e+00 | 0.3180251882714854039 | 0.5244 |
| 1 | 6.000000000000000888e-01 | 0.0819748117285145073 | 1.6958 |
| 2 | 6.524931709804664415e-01 | 0.0294816407480481546 | 1.8839 |
| 3 | 6.882226997487054243e-01 | 0.0062478880201908282 | 1.3747 |
| 4 | 6.816055562499178100e-01 | 0.0003692554785967861 | 1.5612 |
| 5 | 6.819705209666883094e-01 | 0.0000042907618262867 |  |
| 6 | 6.819748117285145961e-01 |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Secant Method ( | | | |
| i |  |  |  |
| 0 | 9.000000000000000222e-01 | 0.21802519390976971 | 0.2919 |
| 1 | 6.572155222490418947e-01 | 0.0247592838411884175 | 2.6326 |
| 2 | 6.753945306025679329e-01 | 0.0065802754876623793 | 1.1424 |
| 3 | 6.823085291030577260e-01 | 0.0003337230128274138 | 1.8035 |
| 4 | 6.819705986744702964e-01 | 0.0000042074157600158 |  |
| 5 | 6.819748060902303122e-01 |  |  |

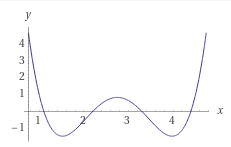


위 수식을 통해 산출해낸 c를 산출했다. (의 경우 c의 값이 대체로 특정 값에 근접함을 알 수 있다. 이를 통해 이론에서의 Secant Method의 수렴 속도와 대체로 일치하는 것을 알 수 있다. 하지만, (을 초기값으로 주었을 때에는 c의 오차가 심한 것을 확인할 수 있는데, 이는 함수의 형태에 따라, 특정 초기값에 대해서 발생하는 오차인 것으로 추측할 수 있다.

**(부가설명)**

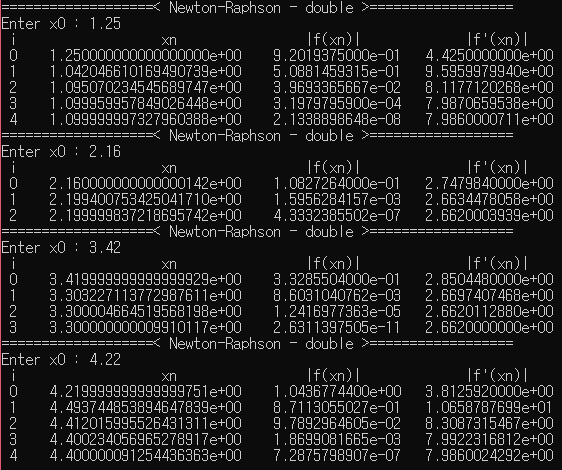
Newton-Raphson method가 Secant method보다 대체로 빠른 속도로 근에 수렴한다.

**실습 1-2.**



**(Newton-Raphson Method)**

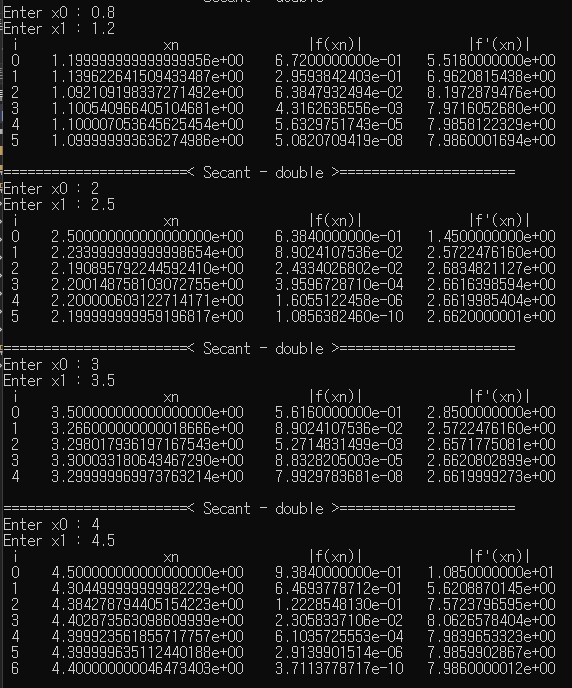
위의 수식의 근을 얻기 위해, 4개의 실근이 존재하는 구간의 중간값을 Newton-Raphson Method의 초기값으로 입력하여 4개의 실근을 얻었다.



이는 그래프와 모순되지 않는 근임을 알 수 있다.

**(Secant Method)**

4개의 실근을 얻기 위해, 각각의 근이 위치하는 범위 내의 일정 값을 입력해주었다.

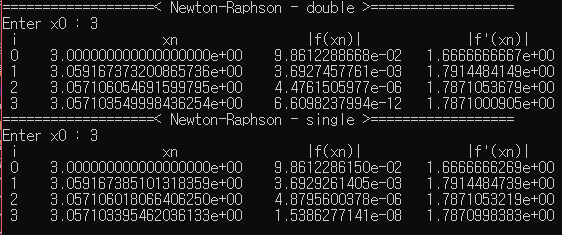


**실습 1-4.**

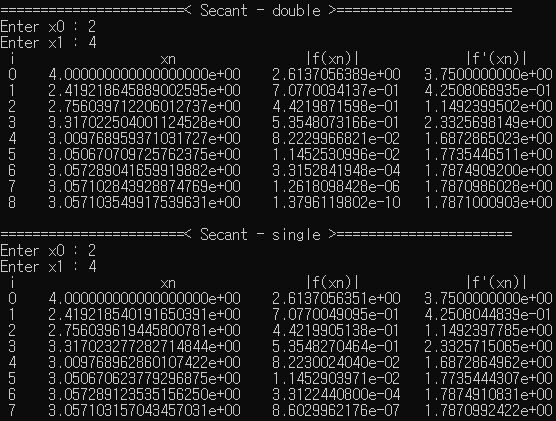
실습 1-1에서 사용한 두 수식과, 새로운 수식 하나에 대해 Newton-Raphson Method와 Secant Method를 Single-precision과 double-precision으로 적용하여 두 방법에 따른 차이를 확인했다.



(Newton-Raphson method)

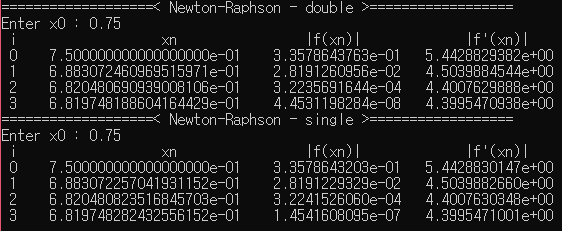


(Secant Mathod)

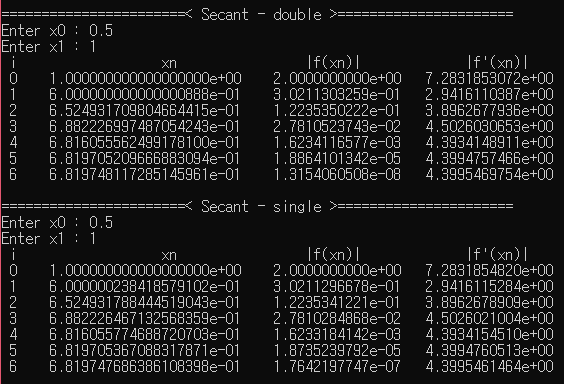




(Newton-Raphson Mathod)

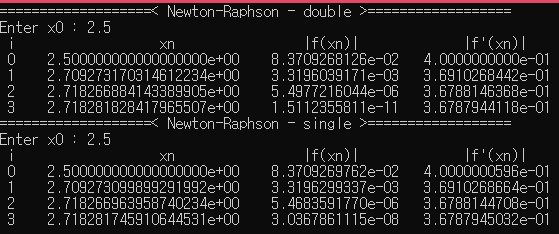


(Secant Mathod)

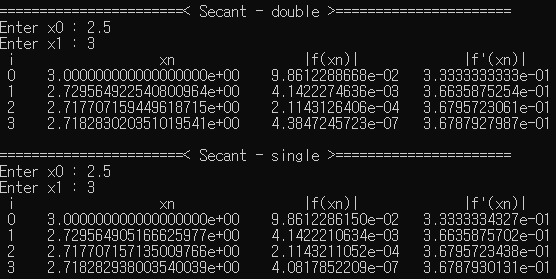




(Newton-Raphson Mathod)



(Secant Mathod)

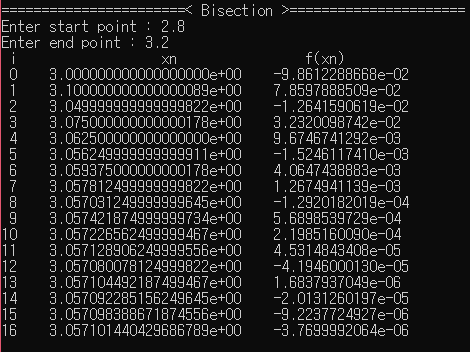


Single Precision과 Double Precision에 대하여, 값이 조금씩 차이가 나는 것을 확인할 수 있다. 둘 중 Double Precision 측의 값이 좀 더 정확한데, 이는 Double precision이 표현할 수 있는 소수의 정확도가 조금 더 높기 때문이다.

**과제 1**

**(근 산출)**





f(x)의 값이 충분히 작기 때문에, 0으로 간주할 수 있을 정도의 값을 가지는 x값을 찾아냈고, 찾아낸 값은 실습 1-1-1에서 제시했던 그래프와 모순되지 않기 때문에 이 근은 의미있는 근으로 해석할 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| Secant Method ( | | | |
| i |  |  |  |
| 0 | 3.000000000000000000e+00 | 0.057101440429686789 | 0.04289 |
| 1 | 3.100000000000000089e+00 | 0.0428985595703133 | 6.0408 |
| 2 | 3.049999999999999822e+00 | 0.007101440429686967 | 0.3967 |
| 3 | 3.075000000000000178e+00 | 0.017898559570313389 | 3.3154 |
| 4 | 3.062500000000000000e+00 | 0.005398559570313211 | 6.3405 |
| 5 | 3.056249999999999911e+00 | 0.000851440429686878 | 0.3744 |
| 6 | 3.059375000000000178e+00 | 0.002273559570313389 | 3.1974 |
| 7 | 3.057812499999999822e+00 | 0.000711059570313033 | 10.1304 |
| 8 | 3.057031249999999645e+00 | 0.000070190429687144 | 0.2190 |
| 9 | 3.057421874999999734e+00 | 0.000320434570312945 | 2.5609 |
| 10 | 3.057226562499999467e+00 | 0.000125122070312678 | 4.5555 |
| 11 | 3.057128906249999556e+00 | 0.000027465820312767 | 1.2857 |
| 12 | 3.057080078124999822e+00 | 0.000021362304686967 | 6.9999 |
| 13 | 3.057104492187499467e+00 | 0.000003051757812678 | 0.3333 |
| 14 | 3.057092285156249645e+00 | 0.000009155273437144 | 3.0000 |
| 15 | 3.057098388671874556e+00 | 0.000003051757812233 |  |
| 16 | 3.057101440429686789e+00 |  |  |

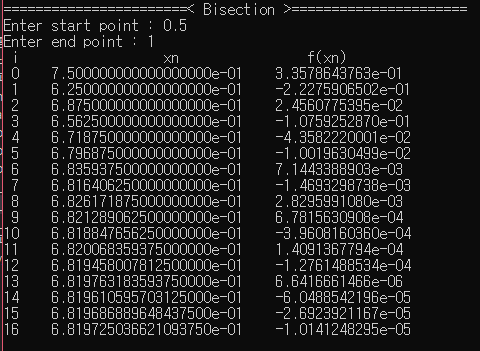
**(수렴 속도)**

이론상으로 Besection Method의 수렴 속도는



에 수렴한다. 따라서 의 값은 2에 수렴하여야 하지만, 결과를 보면 2와 차이가 많이 나는 것을 확인할 수 있다. 이는 다음 산출한 범위에 포함되는 근이 범위의 경계에 있는 경우 오차가 심해지는 것으로 추측할 수 있다. 오차는 작은 값들이기 때문에 작은 차이에도 그 값이 크게 차이나는 것으로 볼 수 있다.





f(x)의 값이 충분히 작기 때문에, 0으로 간주할 수 있을 정도의 값을 가지는 x값을 찾아냈고, 찾아낸 값은 실습 1-1-1에서 제시했던 그래프와 모순되지 않기 때문에 이 근은 의미있는 근으로 해석할 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| Secant Method ( | | | |
| i |  |  |  |
| 0 | 7.500000000000000000e-01 | 0.068027496337890625 | 1.1940 |
| 1 | 6.250000000000000000e-01 | 0.056972503662109375 | 10.3071 |
| 2 | 6.875000000000000000e-01 | 0.005527496337890625 | 0.2148 |
| 3 | 6.562500000000000000e-01 | 0.025722503662109375 | 2.5474 |
| 4 | 6.718750000000000000e-01 | 0.010097503662109375 | 4.4190 |
| 5 | 6.796875000000000000e-01 | 0.002285003662109375 | 1.4094 |
| 6 | 6.835937500000000000e-01 | 0.001621246337890625 | 4.8850 |
| 7 | 6.816406250000000000e-01 | 0.000331878662109375 | 0.5147 |
| 8 | 6.826171875000000000e-01 | 0.000644683837890625 | 4.1219 |
| 9 | 6.821289062500000000e-01 | 0.000156402587890625 | 1.7826 |
| 10 | 6.818847656250000000e-01 | 0.000087738037109375 | 2.5555 |
| 11 | 6.820068359375000000e-01 | 0.000034332275390625 | 1.2857 |
| 12 | 6.819458007812500000e-01 | 0.000026702880859375 | 7 |
| 13 | 6.819763183593750000e-01 | 0.000003814697265625 | 0.3333 |
| 14 | 6.819610595703125000e-01 | 0.000011444091796875 | 3 |
| 15 | 6.819686889648437500e-01 | 0.000003814697265625 |  |
| 16 | 6.819725036621093750e-01 |  |  |

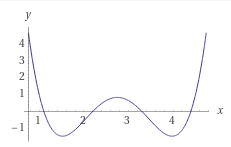
**(수렴 속도)**

이론상으로 Besection Method의 수렴 속도는



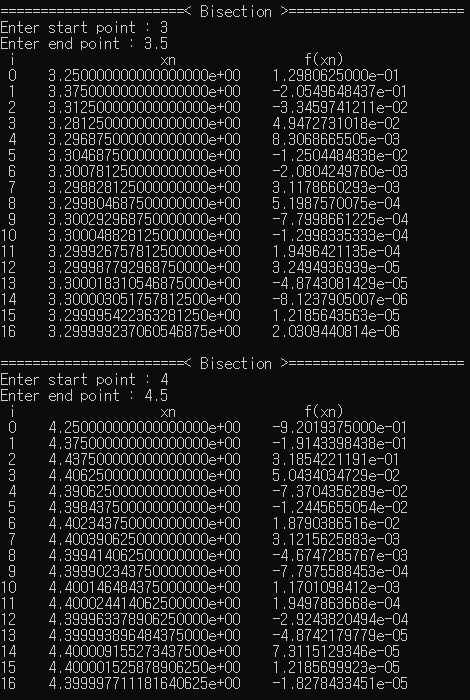
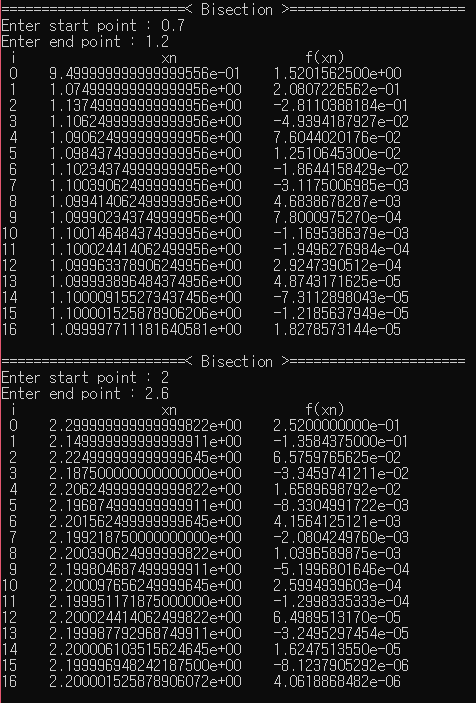
에 수렴한다. 따라서 의 값은 2에 수렴하여야 하지만, 결과를 보면 2와 차이가 많이 나는 것을 확인할 수 있다. 이는 다음 산출한 범위에 포함되는 근이 범위의 경계에 있는 경우 오차가 심해지는 것으로 추측할 수 있다. 오차는 작은 값들이기 때문에 작은 차이에도 그 값이 크게 차이나는 것으로 볼 수 있다.





(Newton-Raphson method와 secant method는 실습 1-2항목에 존재)

실근 4개를 구하기 위해, 근이 있어야하는 범위 내에서 부호가 변화하는 범위를 값으로 입력해주었다.

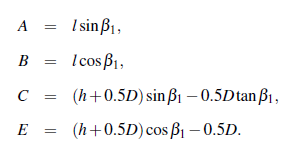


그래프와 결과가 모순되지 않는다. 더불어, Secant와 Newton-Raphson method의 결과와도 크게 다르지 않은 결과가 나온 것을 확인할 수 있다. 수렴속도는 범위를 어떻게 정해주느냐에 따라 크게 달라졌는데, 초기값으로 입력해준 범위의 중간 값이 근에 근사할수록 수렴속도가 빨라졌다.

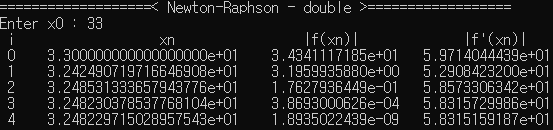
**과제 2**

이번 문제를 해결하기 위한 f(x), f’(x)에 해당하는 함수를 ‘function.cpp’에  
\_f\_a(double x)와 \_fp\_a(double x)로 정의하였다.





l = 89, h = 49, D = 55, Beta = 11.5일 때, 33의 값을 초기값으로 입력하면 다음과 같은 결과를 얻을 수 있다.



이를 통해 약 32.4822가 알파에 해당하는 값임을 알 수 있다.

이 문제를 해결하기 위한 반복문으로는 Newton-Raphson Method를 구현하며 사용한 반복문이 그대로 사용되었는다. 기본적으로 x의 값을 계속 바꾸어가며 계산을 수행하는데, 다음 x는 현재 x의 접선이 y = 0과 만나는 부분의 x값을 넣어준다.

이때, 이 반복문을 종료해주는 조건은 다음과 같다.

1. 반복문이 충분히 많이 실행됨 (>50)
2. 지금 x값에 대한 f(x)의 값이 충분히 작아 0으로 간수할 수 있음
3. 지금 x값과 다음 x값에 대한 차가 의미가 없을만큼 작음

해당 조건으로 반복문을 빠져나오면, 지금 x에 저장되어있는 값이 이 식의 근이라고 간주한다.