고급소프트웨어

4주차 과제

20170101

이은지

1. Bisection 방법 구현 및 다른 방법과의 비교

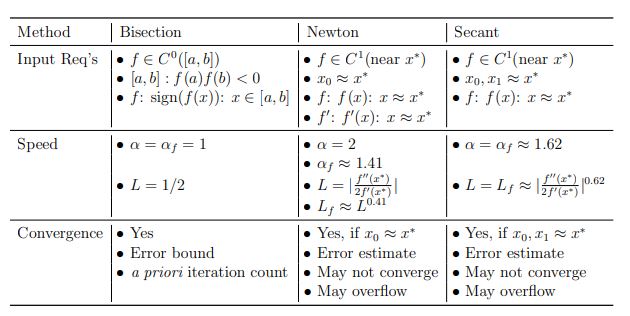
근을 포함하는 구간을 임의로 정하고, 그 구간을 반씩 줄여가며 근을 구할 수 있다.

예를 들어 구간을 ( , )으로 설정하면 방정식에 적용하여 f() f() <0 이면 해당 구간 사이에 적어도 하나 이상의 근이 존재한다. 이후 새로운 근을 추정하며 구간을 줄여나가 근을 찾아낼 수 있다.

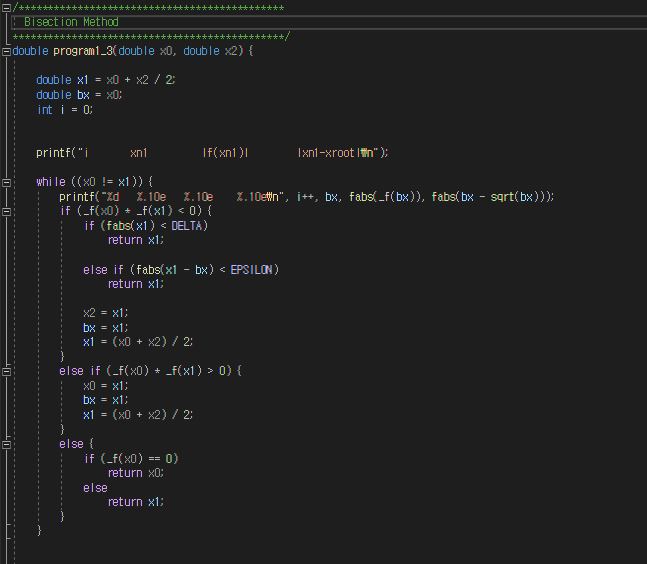
|  |
| --- |
| , ← two initial guesses;  while( && There’s no root yet)  = / 2;  if (f() f() < 0)  오차범위 내 근 발견 시 함수 종료  =  =  else if (f() f() > 0  =  =  else  Find out the root |

이는 직관적이고 단순하지만, 다른 방식에 비해 자칫 구간을 잘못 설정할 경우 지나친 반복문으로 비효율적인 계산이 이루어진다.

아래의 표를 참고하여 더욱 정밀한 비교가 가능하다.



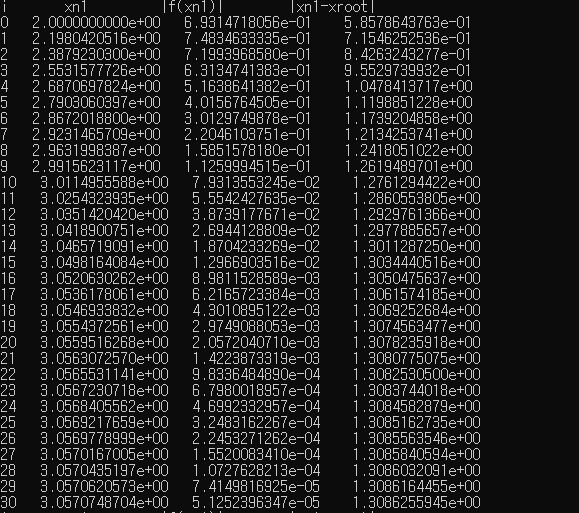
**적용**

****

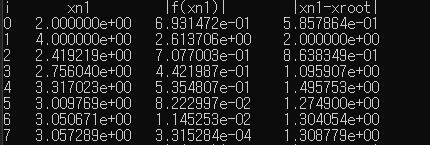
= 2.0, = 4.0으로 가정하고 f(x) = 로 설정하였다.

이에 따라 차례대로 Newton\_Raphson, Secant, Bisection을 적용한다.

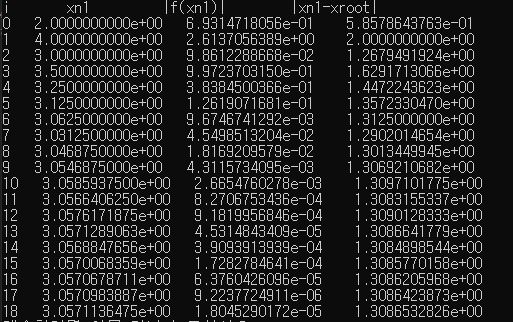
**Newton\_Raphson**



**Secant**

****

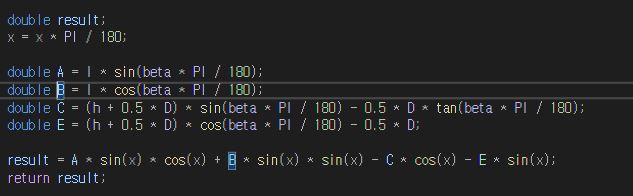
**Bisection**

****

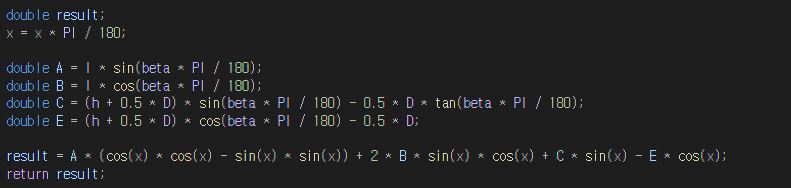
1. Newton-Raphson 방법으로 방정식의 해를 풀이

****

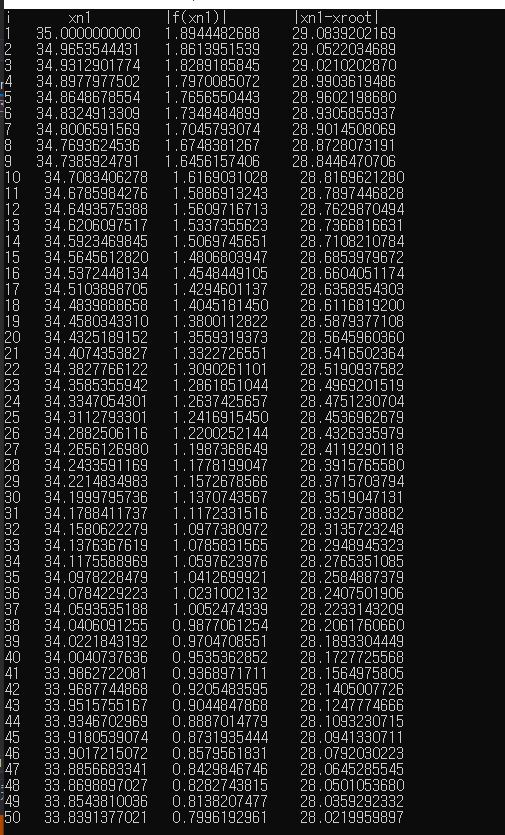
이를 구현한 코드이다. Degree와 라디안을 적절히 전환하였다.



아래는 위 방정식에 대한 도함수이다. 삼각함수의 미분을 활용하였다.



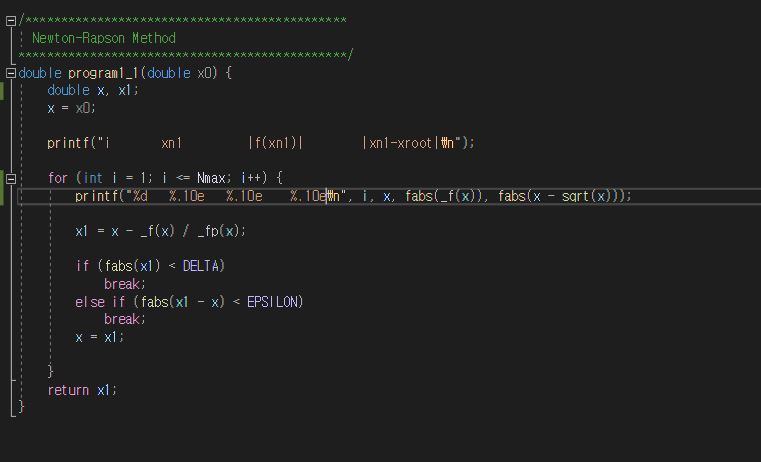
그리고 위를 적용한 program 1\_1에 인자로 35를 넘겨주어 아래와 같은 유사근을 얻을 수 있었다.



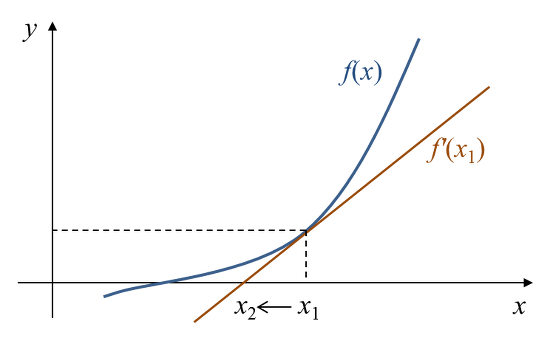
1. 1-1,1-2,1-4 과제 1,2 부분에 대한 보고서

**실습 1-1.**

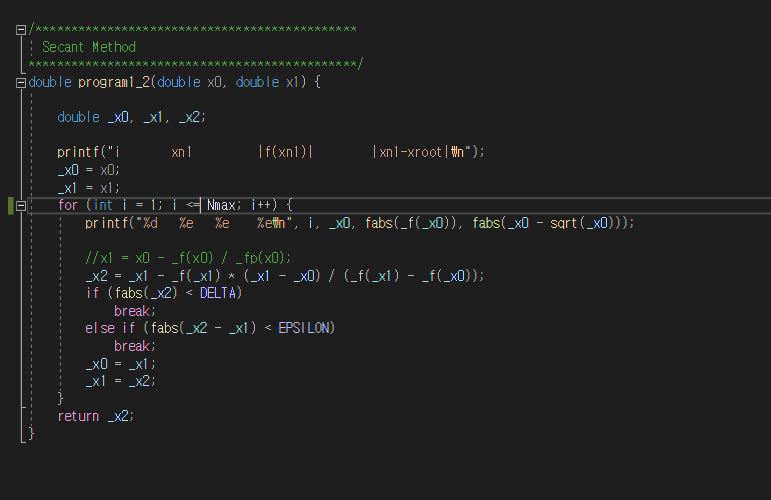
**프로그램 구현**



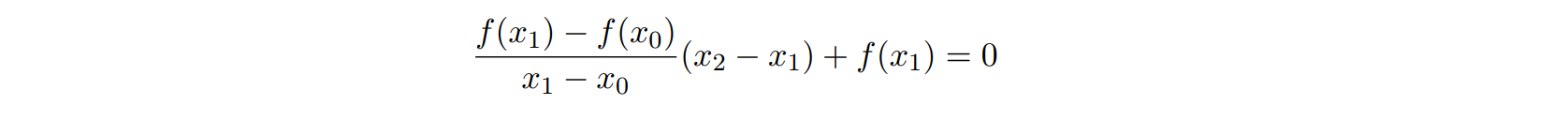
임의의 x를 입력받아 오차를 줄여가며 근에 가장 근사한 값에 도달한다.

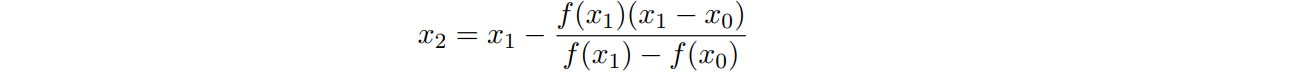


이와 같이 x2 = f(x1)/ f’(x1) 식으로 다가가다 보면 x의 값을 구할 수 있으나 지나친 계산량을 방지하고자 함수 값이 충분히 작거나, 50회 정도 충분히 계산을 했을 경우, 이전 x값보다 의미있는 진전을 하지 않는 경우에는 계산을 중단하고 예상 근을 도출한다.



임의의 x0, x1의 차를 통해서 점차 근에 가까운 수를 구한다.

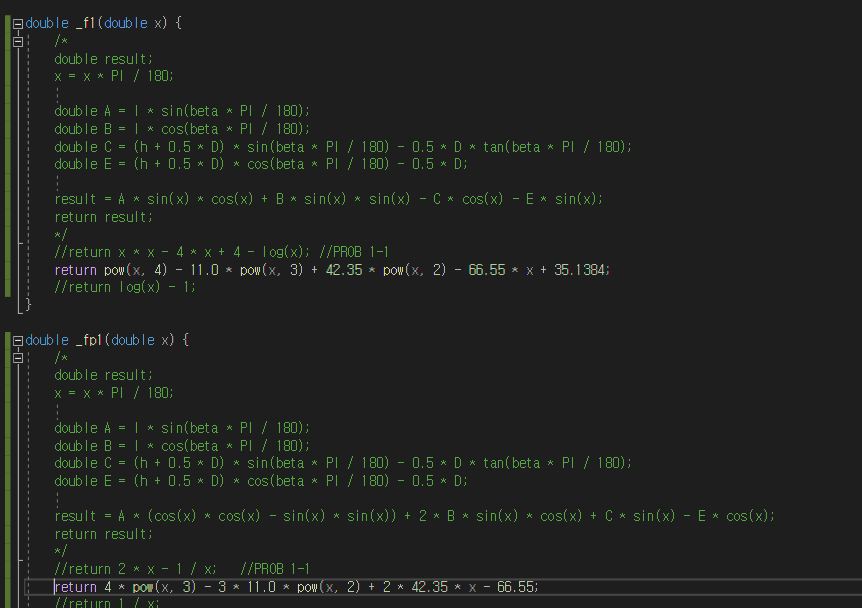




이러한 과정을 반복하다가 지나친 계산량을 방지하고자 함수 값이 충분히 작거나, 50회 정도 충분히 계산을 했을 경우, 이전 x값보다 의미있는 진전을 하지 않는 경우에는 계산을 중단하고 예상 근을 도출한다.

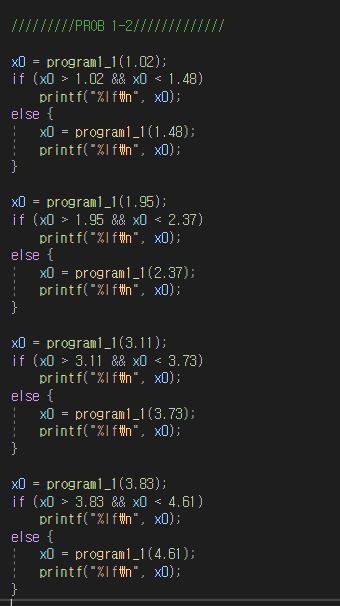
**실습 1-2.**

**특정함수의 모든 실근 계산**



주어진 함수를 위와 같이 구현하였다.

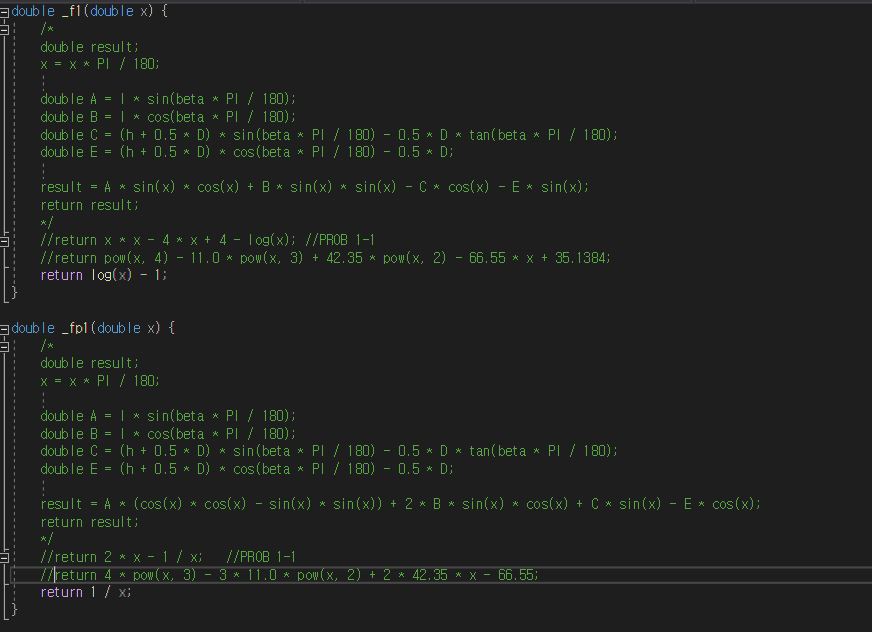
그리고 main 함수에서 만약 주어진 구간이 (a,b)라면 a를 통해 근을 추적하였고, 주어진 구간에 속한다면 그 근이 예상했던 값이므로 그대로 출력하고, 만약 주어진 구간에 속하지 않는다면 b를 통하여 다시 근을 추적하여 이를 출력한다.



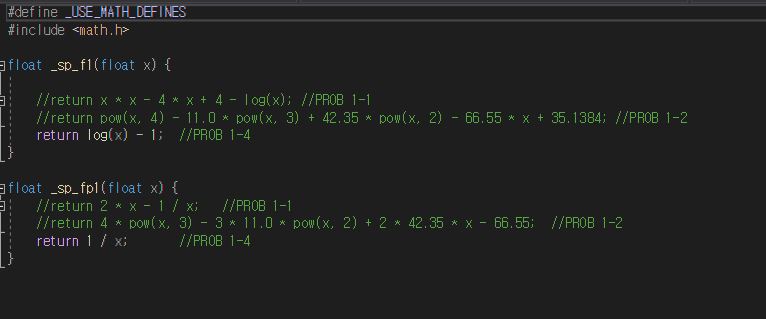
**실습 1-4**

**정밀도와 구한 근의 오차에 대한 상관관계**

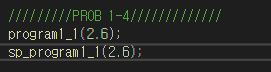
function.cpp



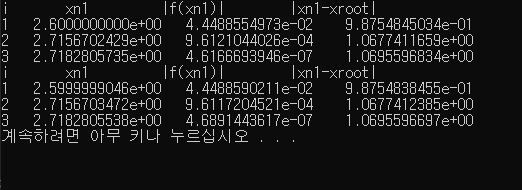
sp\_function.cpp



위와 같이 자료형만 다르고 함수는 동일하게 설정한다.



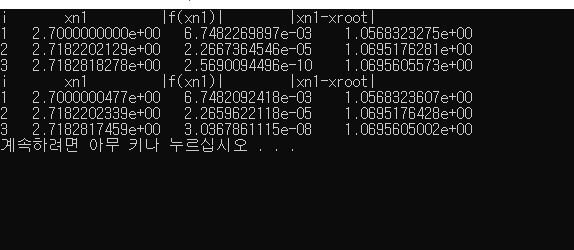
그리고 차례대로 double형과 float형을 호출한다.

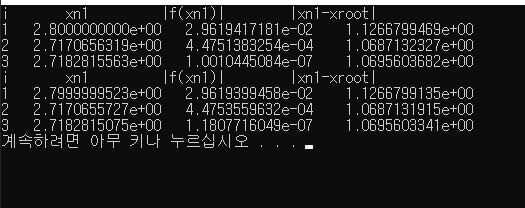


#자꾸 이상하게 2.59….로 출력되지만 중간에 자료형 변화 없이 정확히 2.6을 인자로 받아오는 것을 확인하였습니다.

자연 상수 e = 2.71828182845904523…과 비교하였을 때 두 방식 모두 소수 5번째자리까지 일치하고 6번째부터 차이를 보인다. 그런데 double형이 조금 더 크나 차이가 크게 다르지 않음을 확인할 수 있었다.

그래서 다른 인접수인 2.7과 2.8을 적용하여 접근하였다.





모두 오차가 적지만 확실히 double형이 더 적은 오차를 일관되게 보였다.

이에 의하여 자료형에 따른 오차는 유의미하였다.