Numerical Methods
HW01(Bisection Method)

담당교수 : 심동규 교수님

학 번 : 2019202050

성 명 : 이강현

Introduction

이번 과제에서는 Bisection Method를 floating point, fixed point 방식으로 각각 구현하여 해를 비교해본다. 해를 비교해보면서 구현 방식에 따라 달라지는 오차를 확인하고 그러한 결과가 발생하는 이유와 각각의 구현방식이 가지는 장단점과 특징을 생각해본다.

Bisection Method

Bisection Method는 주어진 방정식의 근을 근이 확실하게 존재하는 어떠한 폐구간을 반복적으로 나누면서 대략적으로 찾는 알고리즘이다. 구간을 a와 b, 연속함수를 f라고 할 때 대략적인 순서는 아래와 같다.

1. 구간의 중간점인 c를 구한다. c=(a+b)/2
2. f(c)를 계산한다.
3. f(c)의 부호를 f(a) 혹은 f(b)와 비교하고 a,b 둘 중 하나와 c를 대체하여 영점교차가 있는 새로운 구간을 형성한다.
4. 1,2,3을 c-a,혹은 f(c)가 충분히 작을 때까지 반복한다.

매우 간단하고 직관적이지만 상대적으로 속도가 많이 느리다는 특징이 있다.

Experiments

$f(x) = x^3 + 4x^2 - 10$

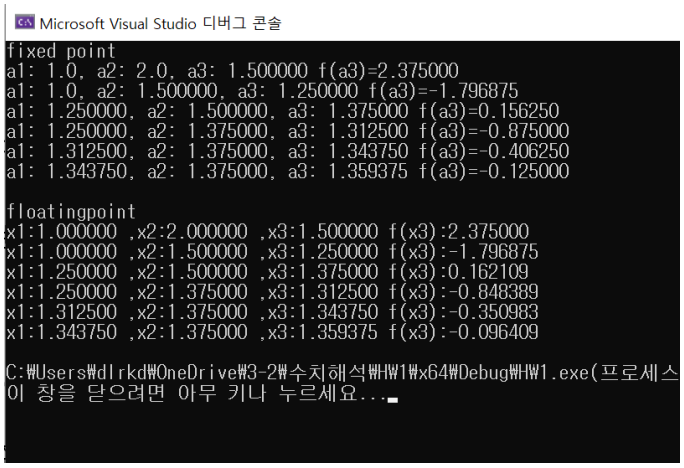
근을 구하려는 방정식은 위와 같고 이에 따른 floating point 방식과 fixed point방식의 코드를 c언어로 구현하였다. Floating point 방식은 float변수를 사용하여 구현하였고 fixed point는 과제 조건에 맞게 16bit크기인 short 자료형을 통해 구현하였다. Floating point 구현에 float변수를 사용한 이유는 float변수는 대략 6~9만큼의 정밀도를 가지고 있기에 소수부 6자리를 사용하게 구현한 fixed point와의 비교가 잘 맞을 것 같다는 생각을 했기 때문이다.

Floating point의 구현은 주어진 알고리즘에 맞게 구현을 하였고 tolerance value는 fixed point 구현과 마찬가지로 소수점 6자리만큼의 차이보다 작게 되면 반복을 멈추도록 구현했다. Fixed point 구현에는 floating point 구현과 다르게 소수부를 정수부처럼 저장하기 때문에 스케일링 작업이 필요했다. Fixed point 구현의 소수부를 6자리만큼으로 설정한 이유는 f(x)의 형태와 관련이 있다. 제곱과 세제곱항을 가진 f(x)를 연산해야 했기에 정수부의 손실이 생기지 않게 16bit내의 저장공간에서 연산을 진행하기 위해서는 정수부와 소수부의 합이 8bit을 넘지 않아야 했다. 따라서 소수부를 7비트로 정수부를 1비트로 설정하여 하였다. 정수부의 범위는 1과 2사이의 값을 연산하는 것

이라 초기에 2를 연산하지 않고 1을 연산한 값과 1.5를 연산한 값을 서로 비교하면 되었고 1.5를 제곱한다고 하더라도 제곱연산 후의 정수부는 2비트이므로 2.25를 충분히 표현할 수 있었다. 하지만 4라는 값을 연산해야 하는 것이 문제였다. 4를 곱하거나 혹은 식을 변형해서 $x^2(x+4) - 10$ 으로 수식을 바꾸더라도 정수부가 100인 4라는 값을 연산하기 위해서는 정수부가 최소 3bit이상은 되어야 했다. 또한 첫번째 연산의 중간값 1.5를 계산해 보면 -10을 제외한 식의 값이 12.375로 정수부가 4비트나 필요해진다. 따라서 정수부 2비트, 소수부 6비트로 설정하는 것이 f(x)를 계산하기 위한 최적의 스케일링이라고 생각하였다.

위와 같이 설정한 후에는 연산을 진행하면서 덧셈은 그대로 진행하고 곱셈은 스케일링을 진행한 만큼으로 우로 쉬프트 연산해주면 되었다. 이와 같이 전체적인 bisection method 알고리즘을 16bit short형으로 연산하는 것을 구현하였다.

추가적으로는 2진수로 표현되어 있는 소수부를 10진수의 형태로 출력하는 것을 진행하였다. 소수부 6비트를 AND 연산을 통해 뽑아낸 후 각 비트의 값이 0인지 1인지에 따라 10진수 값을 더해주게끔 구현하였다.



Conclusion

결과적으로는 정확도가 floating point가 더 좋았다. 입력값을 선정하는 방식이 2로 나눠가는 방식이므로 비트로 표현하기 용이했고 f(x) 수식 자체도 스스로와의 곱셈, 2^2과의 곱셈이므로 floating point 방식에게 치명적인 rounding error가 거의 발생하지 않는다는 점과 정수부가 소수부에 비해 연산이 진행됨에 있어서 거의 사용되지 않기에 정수부가 사용되지 않아도 그 자리를 유지한 fixed point와는 달리 정수부가 사용되지 않아 모든 비트를 소수부에게 할당할 수 있다는 점이 floating point가 더 정확한 값을 뽑아낸 이유라고 생각했다. 절대적으로는 floating point는 32bit이고 구현한 fixed point는 16bit라는 점이 가장 큰 이유라고 생각하였다. 하지만 함수값에서의 약간의 오차는 있었으나 올바르지 않은 입력값을 고르진 않았기에 상황에 따라 trade-off를 고려한다면 오히려 더 적은 bit수로 값을 구해낼 수 있기에 fixed point 방식 또한 고려할만한 가치가 있다고 판단하였다.