수치해석

HW01

담당교수 : 심동규 교수님

학 번 : 2021202058

성 명 : 송채영

1. Introduction

이번 과제는 구간 [1,2]에서 $f(x) = x^3 + 4x^2 - 10$ 의 방정식의 해를 찾는다. 수치근을 근사하기 위해 Bisection Method 를 활용하며, Floating-point(부동소수점)와 Fixed-point(고정소수점)를 통해 solution 을 구한다. Fixed-point 는 16 bit 에서 구현해야 하며 parameter 에 2 의 거듭제곱을 곱해주어 연산을 수행할 수 있다. 이를 통해 Floating-point 와 Fixed-point 의 Bisection Method 를 비교한 후 각각의 정확도와 연산적인 면에서의 효율성을 알아본다.

2. Bisection Method

Bisection Method(이분법)은 주어진 구간에서 함수의 근사치를 찾는 수치해석적 방법 중 하나이다. 주어진 구간 [a, b]에서 f(x)가 연속이고 함수 값의 부호가 바뀔 때, 즉 f(a)>0, f(b)<0 or f(a)<0, f(b)>0 일 때 사용된다.

알고리즘의 동작 순서는 다음과 같다.

1. 초기화

- 주어진 구간 [a, b]와 허용오차를 설정한다.

2. 반복

- 초기 구간 [a, b]에서 중간 점 c 를 계산한다. ($c = \frac{a+b}{2}$)

3. 근사값과 실제 근 확인

- 함수 f(c)가 0 에 충분히 가깝다면 (|f(c)| < tolerance), c 는 근사 해 이므로 알고리즘을 종료하고 c 를 최종 근사값으로 사용한다.

- 그렇지 않으면, f(c)의 부호를 확인한다.

4. 구간 축소

- f(c)가 양수라면 c 를 새로운 구간의 오른쪽 끝으로 대체한다. (b = c)

- f(c)가 음수라면 c 를 새로운 구간의 왼쪽 끝으로 대체한다. (a = c)

- 현재 구간이 점점 좁아지고, 다음 iteration 에서는 좁아진 구간에서 중간 점을 다시 계산한다.

5. 종료 조건

- 원하는 정확도에 도달하거나, 반복 횟수에 도달하면 알고리즘을 종료한다.

6. 결과

- 알고리즘이 종료됐을 때 구간 [a, b]는 근을 포함하는 구간이 되며 c는 해당 근사값이 되며 이를 반환한다.

Bisection method 는 구간을 반으로 나누면서 근을 찾아 나가는 방식으로 동작하며, 구간을 축소하면서 근을 점차 정확하게 찾아간다.

3. Experiments

우선 이번 과제는 Bisection method 를 구간 [a, b]에서 $f(x) = x^3 + 4x^2 - 10$ 의 근을 구하는 것이다. Floating-point 방식으로 구현하기 위해서 float type (32 bit)을 사용했으며, Fixed-point 방식으로 구현하기 위해서 과제 조건에 따라 short type (16bit)를 사용하였다.

각각의 방식에 대해 설명하겠다.

- **Floating-point**

우선 floating-point 방식을 구현하면서 32bit 인 float type 을 사용하였다. Floating-point 방식과 16 bit 로 고정된 Fixed-point 방식을 비교할 때 64bit 의 double type 을 사용할 경우 결과값이 16bit 의 Fixed-point 방식과 큰 차이를 보여 비교 분석이 어려울 것이라 생각했다. Floating-point 의 높은 정밀도와 범위는 결과값을 보다 상세하게 나타낼 수 있어 불필요한 과정이 생길 수 있어 일관성 있는 비교와 정밀도 측면에서 더 적절하다고 판단된 32bit type 인 float 자료형을 선택하였다.

구현한 방식에 대해 설명하겠다.

- f1(x) 함수는 실수 x 에 대한 함수로 $x^3 + 4x^2 - 10$ 에 대한 값을 반환하여 함수의 근을 찾는다.

- Bisection_float 함수는 Bisection method 방식으로 floating-point 를 구현한 부분이다. 주어진 알고리즘에 맞게 구현하였으며 이 함수는 주어진 구간 [a, b]에서 f1(x)의 근을 찾아 반환한다.

- a 와 b 의 중간 점 c 를 계산하고, 이 지점에서 f1(c)의 값을 계산한다. 이후 fabs 함수를 사용해서 f(c)의 절대값을 계산한다. 이때 fabs 함수는 부동 소수점 숫자의 절대값을 계산하는 역할을 한다. 만약 이 값이 tolerance 보다 작으면 중간 값 c 를 찾았으므로 c 를 반환한다. 작지 않다면 f1(a)와 f1(c)의 값을 비교해 다음 반복의 구간

[a,b]를 구한다. 이때 부호가 다른 경우 b 를 c 로, 부호가 같다면 a 를 c 로 업데이트하며 최대 반복 횟수만큼 반복한다.

- main 함수에서는 초기 구간 [a_float, b_float]와 허용 오차 tol_float 를 각각 1.0, 2.0, 0.0001 로 설정하였다. Toleration 은 근사적으로 찾으려는 근과 실제 근 사이의 허용 가능한 최대 오차를 나타내며, 알고리즘이 찾은 근과 실제 근 사이의 차이가 0.0001 보다 작을 때까지 알고리즘이 반복된다.

- **Fixed-point**

우선 Fixed-point 를 구현할 때 16bit 로 제한되어 있어 short type 을 사용하였다. 또한 정확도를 최대한 확보하기 위해 소수부분에 중점을 두어 scale 을 하려고 하였다. 16bit 내에서 연산을 해야 하므로 정수부와 소수부의 합은 최대 8bit 로 두었으며 정수부 2bit, 소수부는 6bi 가 되도록 설정하였다.

구현한 방식에 대해 설명하겠다.

- FIXED_POINT_BITS 와 FIXED_POINT_SCALE 매크로를 사용하여 고정 소수점 형식을 설정하였다. FIXED_POINT_BITS 는 전체 비트 수를 나타내며 8bit 로 설정하였고, FIXED_POINT_SCALE 은 소수 부분의 크기를 나타내며 2^6 이다.

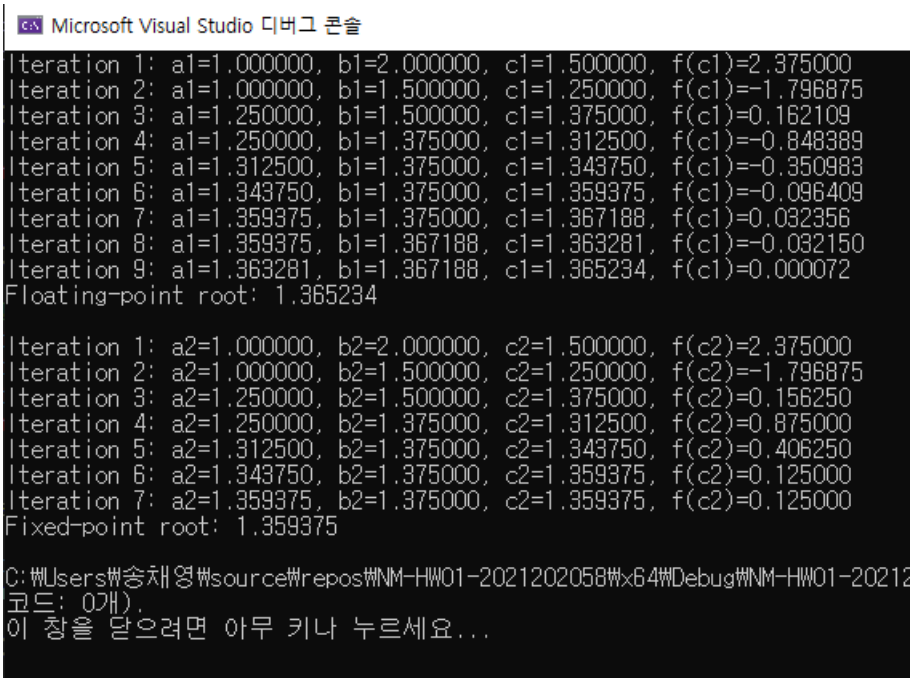
- f(2)x 함수는 실수 x 에 대한 함수로 $x^3 + 4x^2 - 10$ 에 대한 값을 반환하여 함수의 근을 찾는다. x^2 과 x^3 을 계산하고 result 변수에 $x^3 + 4x^2 - 10$의 결과를 저장한 후 반환하였다. 각각의 단계에서는 FIXED_POINT_SCALE 로 나누어 주었다.

- FIXED_POINT_SCALE 로 나누는 이유는 고정 소수점 형식의(정수 부분과 소수 부분으로 나누어)값으로 결과를 표현하기 위해서이다.

- Bisection_fixed 함수는 Bisection method 방식으로 fixed-point 를 구현한 부분이다. 주어진 알고리즘에 맞게 구현하였으며 이 함수는 주어진 구간 [a, b]에서 f2(x)의 근을 찾아 반환한다.

- a 와 b 의 중간 점 c 를 계산하고, 이 지점에서 f2(c)의 값을 계산한다. 이후 abs 함수를 사용해서 f(c)의 절대값을 계산한다. 이때 abs 함수는 절대값을 계산하는 역할을 한다. 만약 이 값이 tolerance 보다 작으면 중간 값 c 를 찾았으므로 c 를 반환한다. 작지 않다면 f1(a)와 f1(c)의 값을 비교해 다음 반복의 구간 [a,b]를 구한다. 이때 부호가 다른 경우 b 를 c 로, 부호가 같다면 a 를 c 로 업데이트하며 최대 반복 횟수만큼 반복한다.

- main 함수에서는 초기 구간 [a_fixed, b_flxed]와 허용 오차 tol_fixed 를 각각 1*64 , 2*64, 1 로 설정하였다. Toleration 은 근사적으로 찾으려는 근과 실제 근 사이의 허용 가능한 최대 오차를 나타내며, 알고리즘이 찾은 근과 실제 근 사이의 차이가 1 보다 작을 때까지 알고리즘이 반복된다. 근을 찾은 후에는 정수 부분과 소수 부분을 계산하여 출력해주었다.



Floating-point 는 9 번의 반복 후 솔루션이 1.365234 가 나왔으며 해당 값을 f(x)에 넣었을 때 0.000072 가 나왔다. Fixed-point 는 7 번의 반복 후 솔루션이 1.359375 가 나왔으며 해당 값을 f(x)에 넣었을 때 0.125000 이 나왔다.

4. Conclusion

위의 결과를 바탕으로 결론을 내리면 Floating-point 가 더 좋은 연산 결과를 도출했다. Fixed-point 의 결과가 더 좋게 나올 것이라고 예상했지만 결과는 달랐다. 우선 가장 큰 이유는 Floating-point 는 32bit 자료형인 float 를 사용하였으며, Fixed-point 는 16bit 자료형인 short 를 사용했기 때문이라고 생각한다. Float type 은 높은 정밀도를 제공하며, 더 넓은 범위의 값을 표현할 수 있지만, short type 은 보다 작은 범위와 정밀도를 갖기 때문에 결과가 floating-point 보다 제한적일 수밖에 없다. 더 나아가 Fixed-point 에서는 16bit 를 또 정수부와 소수부로 나누어 사용했기 때문에 오차를 계산할 수 있는 bit 가 상대적으로 부족하다. 하지만 fixed-point 가 더 적은 iteration 을 가지며 연산 속도에서는 보다 빠르다고 생각한다. 즉 floating-point 는 높은 정밀도와 넓은 범위를 가지지만, fixed-point 가 더 빠른 연산이 가능하다는 점에서 상황에 따른 형식을 사용하면 좋을 것 같다.