**고급소프트웨어실습1 - 10주차 과제**

코드 최적화 기법/부동 소수점 연산에 대한 소개

|  |  |  |
| --- | --- | --- |
| **분반** | **:** | 3 |
| **학번** | **:** | 20121635 |
| **이름** | **:** | 장종석 |

1. **숙제 1**
2. double 타입의 사용

float 의 경우 32bit를 사용하고, double 의 경우 64bit를 사용하기 때문에 double 을 사용할 시 표현 범위가 더 넓다. 즉, float에 비하여 정확한 결과를 얻을 수 있기 때문에 double를 사용하였다.

1. 어떤 방법이 더 정확한 것으로 판단되는가?

샘플 데이터 는 랜덤으로 생성한 난수 수열이다. 그러므로 의 분산은 0에 가까워야 한다. 샘플 데이터의 개수를 의미하는 과 방법에 따른 분산 값은 다음과 같다.

방법 1 :

|  |  |  |
| --- | --- | --- |
| n |  |  |
| 100 | 0.083249 | 0.083249 |
| 1,000 | 0.085769 | 0.085769 |
| 10,000 | 0.082074 | 0.082074 |
| 100,000 | 0.589562 | 0.083127 |
| 1,000,000 | -114.514448 | 0.083410 |
| 10,000,000 | 31284.148151 | 0.083350 |
| 100,000,000 | 469484.303493 | 0.083330 |

방법 2 : 의 평균)

위의 결과를 보게 되면, n의 값이 커짐에 따라 방법1과 방법2의 결과가 상이한 결과를 알 수 있는데, 방법 1을 사용하게 되면, n의 값이 커짐에 따라 오차가 급격히 커지는 반면, 방법 2를 사용한 경우 n 값에 상관없이 안정된 값을 도출 하는 것을 알 수 있다. 즉, 방법 2를 통해 분산을 구하는 경우 신뢰도가 높다.

1. 충분히 큰 n에 대하여 두 방법 중 어떤 방법이 더 빠르게 계산하는가?

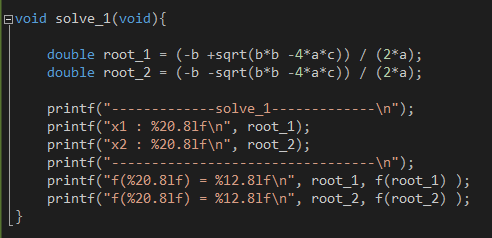
다음은 n값과 방법에 따른 연산 시간(s)이다.

|  |  |  |
| --- | --- | --- |
| N |  |  |
| 100 | 0.000000 | 0.000001 |
| 1,000 | 0.000003 | 0.000005 |
| 10,000 | 0.000026 | 0.000050 |
| 100,000 | 0.000306 | 0.000503 |
| 1,000,000 | 0.002604 | 0.005240 |
| 10,000,000 | 0.026561 | 0.051557 |
| 100,000,000 | 0.259508 | 0.499208 |

위의 결과를 보게 되면, 방법 1의 경우 방법 2에 비하여 상당히 빠르게 결과를 도출 하는 것을 알 수 있다. 그 이유는 연속적인 합을 계산할 때, 방법 1의 경우 x\_bar 를 계산하는 과정에서 한번의 반복문이 더 실행이 되기 때문에, 방법 2보다 더 긴 시간이 소요됨을 알 수 있다.

1. 다음과 같은 이차방정식 f(x) = ax2 +bx+c = 0(a ̸= 0) 의 근을 구하는 문제를 생각하자.
2. 콘솔 윈도우에서 임의의 a,b 그리고 c값을 읽어들여 중학교에서 배운 근의 공식을 사용하여 두 실근을 구하여 출력하는 프로그램을 작성하라.(편의상 두 개의 실근이 존재하는 경우만 고려)

근의 공식을 이용하여 근을 출력하는 프로그램의 경우, 다음과 같이 작성을 하였다.



1. 위 프로그램이 심각한 문제를 야기하는 상황을 세 가지 발생시켜라. 즉 그런 문제를 일으킬 a,b 그리고 c값을 적절히 설정한 후, 위에서 구한 두 근을 다시 f(x)에 대입하여 0이 나오는지 확인 함으로서 심각한 문제가 발생하였다는 것을 증명하라.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| a | b | c | X1 | X2 | f(x1) | f(x2) |
| 2 | 20000 | 1 | -0.00005 | -9999.999995 | -0.00000001 | -0.00000003 |
| 1 | 100000 | 1 | -0.00001 | -99999.99999 | -0.00000034 | 0.00000000 |
| 4 | 100000 | 1 | -0.00001 | -24999.99999 | 0.00000003 | 0.00000000 |
| 5 | 55555555 | 1 | -0.00000002 | -11111110.99999998 | 0.00658926 | 0.00000000 |

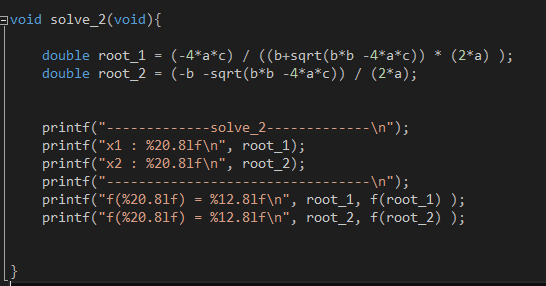
위의 실험 결과를 확인하면, 오차가 발생하는 것을 확인 할 수 있다. 이는 심각한 문제가 발생될 수 있는 반례이다.

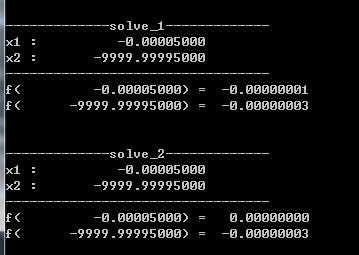
1. 다음 그러한 문제를 완화시킬 수 있는 방법을 사용하여 위의 2차방정식을 풀어주는 함수를 새롭게 구현한 후, 위의 문제와 동일한 과정을 거쳐(즉 자신이 구한 근에 대해 f(x) 함수 값을 구하여), 위에서 심각한 문제를 야기한 세 경우 각각에 대해 자신의 두 번째 함수가 안정적으로 근을 구했음을 밝혀라.

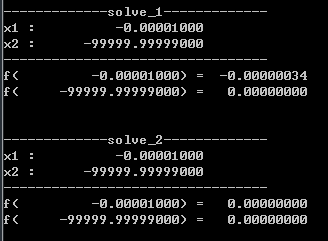
(2)에서 발생한 문제의 경우, 부동소수점 표기에 의한 문제점이다. 이는 강의자료에도 잘 표현되어 있듯, 비슷한 숫자끼리의 뺄셈을 할 때 발생하는 것으로,

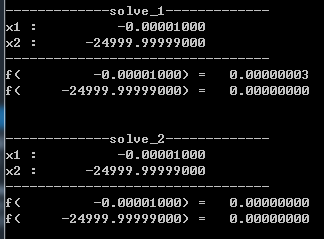
의 식에서 b^2의 값이 4ac 에 비하여 너무 클 때, 의 값이 너무나 작다. 이 경우, 부동소수점 계산시 정규화 해주는 과정에서 왼쪽으로의 쉬프트가 발생하면서, 오른쪽에 아무 의미 없는 0이 삽입되어 값에 문제가 발생하는 경우이다. 이를 해결하기 위하여, 중학교때 배웠던 분모의 유리화를 이용하여, 분자와 분모에 분자의 켤레를 곱하여 두 수의 뺄셈을 다음과 같이 수정하였다.

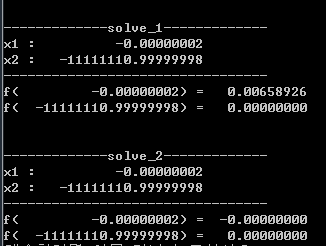
이렇게 수정을 하게 되면, 비슷한 숫자끼리의 뺄셈에서 야기하는 문제를 해결하기 때문에, (2) 에서 발생한 문제를 해결 할 수 있다. 구현의 경우 아래와 같이 진행하였으며, (2)에서 발생한 문제가 새롭게 작성한 함수에서는 발생하지 않는 것을 확인 할 수 있다.



 (a = 2, b = 20000, c = 1)

 (a = 1, b = 100000, c = 1)

 (a = 4, b = 100000, c = 1)

 (a = 5, b = 55555555, c = 1)