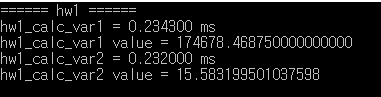
고급소프트웨어 실습 10주차 과제

20171666 이예은

1. 두 분산 값 계산 방법의 결과가 상당히 차이가 나게 해주는 샘플 데이터를 생성한 후, 계산 결과를 비교 분석하라. 분산을 어떻게 계산한 것이 더 정확한 것으로 판단되는가?

과제 1번의 실행 결과는 다음과 같다.



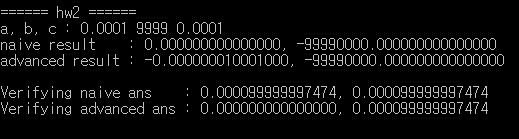
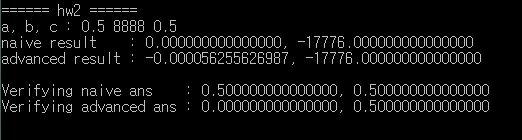
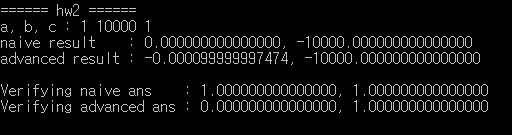
이는 난수가 아닌 실험에 적절한 샘플 데이터를 이용한 값이며, 일반적인 평균을 이용한 분산 식이 값을 더 정확하게 구한다는 것을 확인할 수 있다.

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

N을 100000로 충분히 크게 하였을 때 결과가 위와 같으므로 일반적인 평균을 이용한 분산 식이 더 빠르게 분산 값을 계산하는 것을 확인할 수 있다.

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 확임함으로서 심각한 문제가 발생하였다는 것을 증명하라.

심각한 문제를 발생시키는 상황은 b가 a\*c보다 매우 커 근의 공식에서 비슷한 수의 뺄셈이 일어나는 경우이다. 이를 위해 각각 b가 매우 큰 케이스를 3개 넣어 결과를 확인한다.

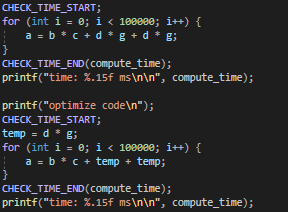


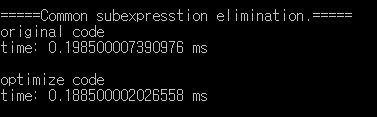
위의 naïve 결과가 일반적인 근의 공식을 사용했을 경우로 verify가 0이 아니므로 올바른 값을 구하지 못했다는 것을 알 수 있다.

1. 다음 그러한 문제를 완화시킬 수 있는 방법을 사용하여 위의 2차 방정식을 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.

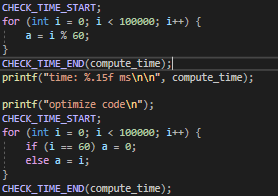
Advanced는 이를 완화시키는 방법으로 만약 비슷한 수의 뺄셈이 일어나는 경우 를 로 바꾸어 계산한다. B가 0보다 클 때, 는 비슷한 수의 뺄셈이 아니므로 이전 근의 공식과 똑 같은 식으로 계산한다. 만약 b가 0보다 작다면 이 방식을 반대로 적용하면 된다. 이를 통해 구한 값을 verify 했을 때, 0으로 제대로 된 값을 구한 것을 확인할 수 있다. 하지만 현재 결과에서는 근의 공식을 사용했을 때 정확한 값이 잘 나오지 않는데, 이는 sqrt 함수를 사용해 root의 값을 구할 때, float형의 범위 한계로 인한 것으로 보인다.

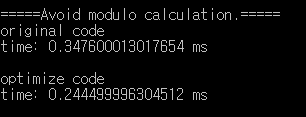
2. 첫번째 방법은 같은 연산을 미리 계산하는 것이다. 코드는 다음과 같다.

 위는 d\*g를 계속 새로 계산하지만 아래는 미리 temp에 d\*g 계산한 값을 넣어 한번 만 계산하게 하였다. 이를 여러 번 연산해 시간 측정해보면 다음과 같이 optimize 기법이 더 적은 시간이 걸리는 것을 확인할 수 있다.

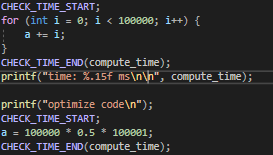


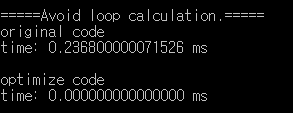
1. 두번째 기법은 modulo연산을 최대한 피하는 것이다. Modulo 연산이나 나눗셈은 비용이 매우 큰 연산이기 때문에 차라리 if 문을 넣는 것이 시간이 더 적게 걸릴 수 있다.



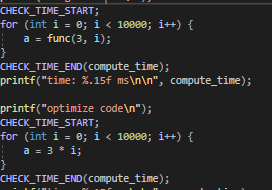


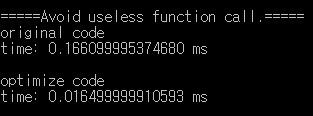
1. 세번째 기법은 loop를 이용한 연산을 최대한 줄이는 것이다. 예를 들어 1부터 100까지 더하는 연산의 경우, loop를 이용하는 것보다 계산식을 이용하는 것이 훨씬 빠르다.





1. 네번째 방법은 함수 호출을 줄이는 것이다. 함수를 호출할 때는 이를 호출 스택에 넣고 다시 빼는 과정이 포함되어 있기 때문에 간단한 연산이라면 함수를 호출하지 않고 바로 계산하는 것이 훨씬 빠르다.





1. 다섯번째 방법은 구조체를 parameter로 넘길 때, 포인터로 넘기는 것이다. 포인터로 넘기지 않으면 함수에서 받을 때 struct 내부의 모든 변수를 할당하여 다시 복사하는데 시간이 걸리지만, 포인터로 받으면 해당 주소 값만 받아 바로 사용할 수 있기 때문에 빠르다.

