[CSE 4152] 고급 소프트웨어 실습 I

코드 최적화 기법/부동 소수점 연산에 대한 소개

20140938 임다은

숙제 1

(ii) 두 분산 값 계산 방법의 결과가 상당히 차이가 나게 해주는 샘플 데이터를 생성한 후, 계산 결과를 비교분석하라. 분산을 어떻게 계산한 것이 더 정확한 것으로 판단되는가?

1) 일반적인 랜덤 값을 넣었을 경우 첫번째 방법과 두번째 방법이 비슷한 값을 계산해낸다. 다음의 결과가 이를 증명한다. 그림은 순서대로 값들의 예시, 계산 결과, 랜덤 값 형성 코드를 보여준다.

```
data[i]: 793.157288
data[i]: 593.933044
data[i]: 232.565613
data[i]: 730.236023
data[i]: 76.385864
data[i]: 817.178650
data[i]: 321.840149
```

```
avg: 492.671692, var1: 78666.578125, var2: 78666.476562
```

```
    data[i] = ((float)rand())/RAND_MAX*1000;
```

2) 값이 아주 비슷한 랜덤 값들을 넣었을 경우 첫번째 방법과 두번째 방법의 차이가 크다. 다음의 결과가 이를 증명한다. 그림은 순서대로 값들의 예시, 계산 결과, 랜덤 값 형성 코드를 보여준다.

```
data[i]: 999.227966
data[i]: 1000.768311
data[i]: 999.405518
data[i]: 1000.273071
data[i]: 999.509827
data[i]: 1000.924500
data[i]: 999.307739
data[i]: 999.666504
data[i]: 1000.692627
data[i]: 1000.431396
```

```
avg: 999.835754, var1: 0.517172, var2: 0.350001
```

```
data[i] = ST - (float)rand()/RAND_MAX * pow(-1,(int)rand());
```

또한 1000 개의 수를 바탕으로 검사한 결과 첫번째 방법은 음수가 나오기도 한다.

```
avg: 999.994324, var1: -1.929573, var2: 0.329486
```

3) 분석 결과

$\sigma_2^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i-\bar{x})^2$ 방법이 더 위험할 것으로 판단된다. 만약 평균값과 랜덤 x 값 하나의 차이가 거의 없다면 비슷한 값의 뺄셈이 되므로 값에 오류가 생길 것이기 대문이다. 따라서 변형된 방법인 $\sigma_1^2 = \frac{1}{n(n-1)}\left(n\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2\right)$ 으로 계산하는 것이 옳다.

(iii) 충분히 큰 n 에 대하여 두 방법 중 어떤 방법이 더 빠르게 분산 값을 계산하는가?

두가지 랜덤데이터의 형태 모두 올바른 계산을 하는 $\sigma_1^2 = \frac{1}{n(n-1)}\left(n\sum_{i=1}^{n}x_i^2-(\sum_{i=1}^{n}x_i)^2\right)$ 가 더 느렸다.

< 완전 랜덤데이터 계산 시간 >

```
makeVar1: 0.000073 milli sec
makeVar2: 0.000044 milli sec
```

< 비슷한 값을 갖는 랜덤데이터 계산 시간 >

```
makeVar1: 0.000062 milli sec
makeVar2: 0.000037 milli sec
```

숙제 2

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

문제가 발생할 수 있는 경우는 -b + sqrt(b^2 - 4*a*C)에서 -b 와 sqrt(b^2 - 4*a*C)의 값이 비슷해질 때이다. 이 둘이 비슷한 수가 될 경우, 즉 4*a*c 가 b^2 에 비해 아주 작은 수가 될 경우 비슷한 숫자들끼리의 뺄셈이 되어 문제가 발생할 것이다.

1) a = 1, b =32423, c = 1

```
DEY:week10_hw yimdaeun$ ./3.out
scanf numbers
1 32423 1
Formula: 1.000000x^2 + 32423.000000x + 1.000000

solved with formula 1
ans1: -3.0842304113e-05, ans2: -3.2422999969e+04
check value with ans1: -2.5314959240e-08
check value with ans2: 0.0000000000e+00
```

첫번째 근에서 같은 값을 뺄셈하는 문제가 발생하여 나온 근을 대입했을 때 결과가 0 이 나오지 않는다.

2) a = 1, b = -32423, c = 1 (위와 같은데, b 가 음수인 경우)

```
DEY:week10_hw yimdaeun$ ./3.out
scanf numbers
1 -32423 1
Formula: 1.000000x^2 + -32423.000000x + 1.000000

solved with formula 1
ans1: 3.2422999969e+04, ans2: 3.0842304113e-05
check value with ans1: 0.0000000000e+00
check value with ans2: -2.5314959240e-08
```

2 번째 루트에서 같은 값을 뺄셈하는 문제가 발생하여 나온 근을 대입했을 때 결과가 0 이 나오지 않는다.

3) a = -1, b = 32423, c= 1 (a 가 음수인 경우)

```
DEY:week10_hw yimdaeun$ ./3.out
scanf numbers
-1 32423 1
Formula: -1.000000x^2 + 32423.000000x + 1.000000

solved with formula 1
ans1: -3.0842304113e-05, ans2: 3.2423000031e+04
check value with ans1: -2.7217454734e-08
check value with ans2: 0.0000000000e+00
```

나온 근을 대입했을 때 결과가 0 이 나오지 않는다.

(iii) 다음 그러한 문제를 완화시킬 수 있는 방법을 사용하여 위의 2 차 방정식을 풀어주는 함수를 새롭게 구현한 후, 위의 문제와 동일한 과정를 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.

비슷한 숫자끼리의 뺄셈을 막아주어야 한다.

- $b > 0$이고 $b^2 \gg |ac|$일 때, 식 $-b+\sqrt{b^2-4ac}$를 식 $\frac{-4ac}{b+\sqrt{b^2-4ac}}$으로 대치함.

을 이용하여 다시 코드를 작성하였다. 다음은 위의 식을 대입했을 때의 근과 확인 결과다.
또한 b<0 일때는 위와 다르게 분자를 b - sqrt(b^2-4*a*c)로 해줘야 한다. 따라서 수정된 코드에서는 b 가 0 보다 클 때와 작을 때를 나눠서 처리해주는 것이 중요하다.

1)

```
solved with formula 2
ans1: -3.0842303333e-05, ans2: -3.2422999969e+04

check value with ans1: 0.0000000000e+00
check value with ans2: 0.0000000000e+00
```

첫번째 근에서 문제가 없어졌다. 나온 근을 대입했을 때 결과가 0 이 나온다.

2)

```
solved with formula 2
ans1: 3.2422999969e+04, ans2: 3.0842303333e-05

check value with ans1: 0.0000000000e+00
check value with ans2: 0.0000000000e+00
```

두번째 근에서 문제가 없어졌다. 나온 근을 대입했을 때 결과가 0 이 나온다.

3)

```
solved with formula 2
ans1: -3.0842303274e-05, ans2: 3.2423000031e+04

check value with ans1: 1.1102230246e-16
check value with ans2: 0.0000000000e+00
```

완전한 근을 구하지는 못했지만 오차가 소수점 16 째자리까지 줄어든 것으로 나온다. 나온 근을 대입했을 때 결과가 0 이 나온다.