Sorbonne Université

Advanced Numerical Algorithms

Practical 4 : - Floating-point summation algorithms

Anatole Vercelloni

Teacher : Stef Graillat

avril 2023

Table des matières

1 Exercise 2

We want to compare the accuracy of some compensated method for the computation of a sum.

A sum is hard to compute with a good accuracy when it condition number is high. We saw in tutorials that the condition number of a sum can be written as the following :

$$cond(s) = \frac{\sum |x_i|}{|\sum x_i|}$$

We have to measure the relative forward error to estimate the accuracy of a sum. This error is define as :

$$e = \frac{|s - res}{|s|}$$

Where sis the exact summation and res is the sum that we computed

In this practical we will study four different summation method : The naive one, the sCompSum, the dCompSum and the CompSum

These algorithms are based on the compensated methods that compute the rounding error and add it to the result.

2 The naive sum

For the first one, the naive one, we just compute the sum without any improvement.

```
function res = Sum(p)
    si = 0;
    n = length(p);
    for i = 1: n
        si = (si + p(i));
    end
   res = si;
end
```

FIGURE 1 – naive sum

3 SCompSum

Then, the SCompSum is build with the compensated method FastTwoSum which allowedus to be more accurate each time we proceed a + b. However, we have to assume that $a \leq b$ so basically that the array is sorted.

```
function res = SCompSum(p)
    n = length(p);
    si = 0;
    e = 0;
    for i = 1 : n
        y = p(i) + e;
        [si, e] = FastTwoSum(si, y);
    end
res = si;
```

FIGURE 2 – SCompSum

4 DCompSum

After that, we have the DCompSum which is a bit more elaborated. Indeed, this time we still use theFastTwoSum method but not only once. The goal is to minimize the error due to the classic + operation. Here we used the + only for summing two errors (So we just not taking into account the error of the error)

```
function res = DCompSum(p)

    n = length(p);
    s = 0;
    c = 0;
    for i = 1 : n
        [y, u] = FastTwoSum(c, p(i));
        [t, v] = FastTwoSum(y, s);
        z = u + v;
        [s,c] = FastTwoSum(t, z);
    end
res = s;
```

FIGURE 3 – DCompSum

5 CompSum

For the last algorithm, the difference is that we change of compensated method. Here we are using the TwoSum method which do not need any assumption on the ordering of the array.

```
function res = CompSum(p)
    n = length(p);
    pi = 1:n;
    q = 1:n;
    s = 1:n;
    pi(1) = p(1);
    s(1) = 0;
    for i = 2 : n
        [pi(i), q(i)] = TwoSum(pi(i-1), p(i));
        s(i) = (s(i-1) + q(i));
    end
res = (pi(n) + s(n));
```

FIGURE 4 – CompSum

6 Analysis

So as said in the introduction, we will compare the accuracy of all these methods. For that we measured the relative forward error depending on the condition number of the sum. For the exact sum, we took matlab operation sum.

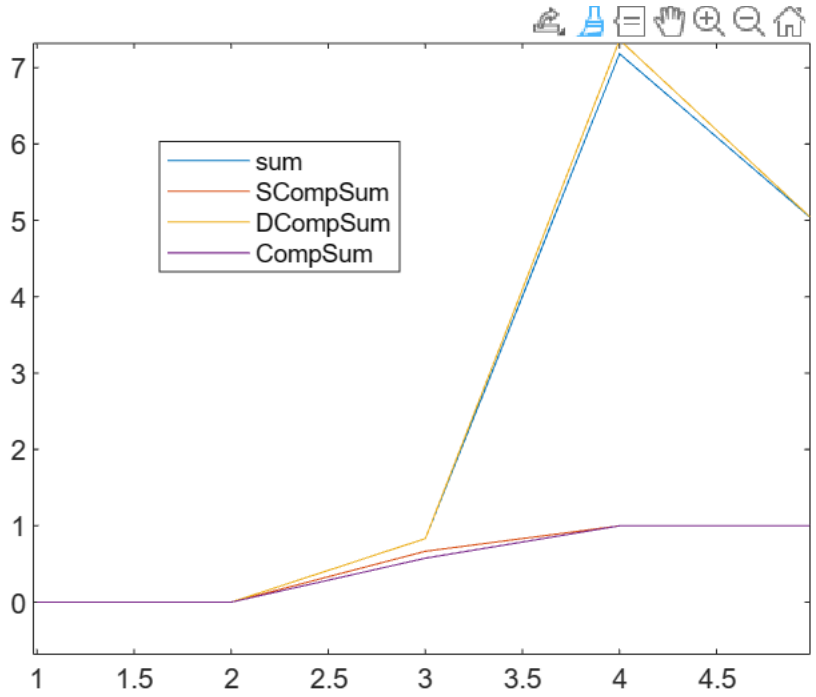

FIGURE 5 – relative forward depending of the condition number(power of ten)

We can observe in the figure that, as expected, the value of the error is increasing when the condition number is increasing. The naive sum has the larger error (with DCompSum which is weird, I may have a mistake somewhere).SCOmpSum and CompSum are well better.

7 Conclusion

We can conclude with this practical that, even with something which is really easy like a sum, we can have a completely wrong result. So when we are manipulating floating point number, it is important to take care abound rounding error