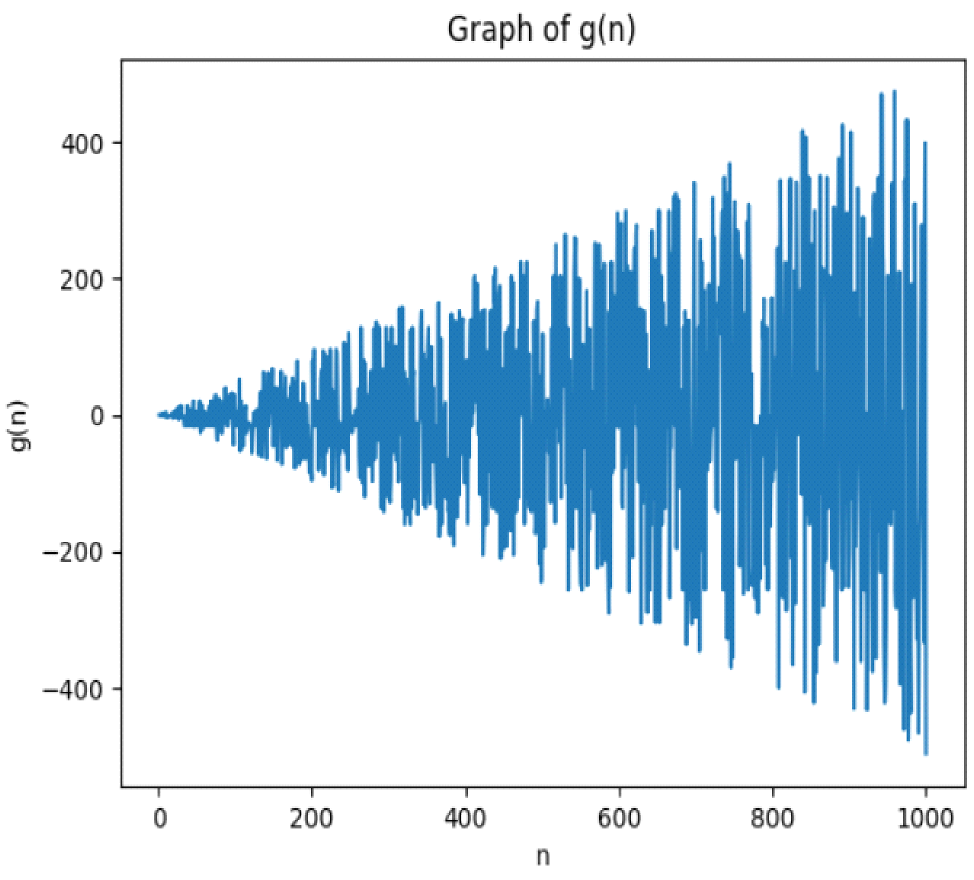
CENG 371 - Scientific Computing
Fall 2022
Homework 1

Şengül, Sena Nur
e2310498@ceng.metu.edu.tr

October 24, 2022

Question 1 :

a)

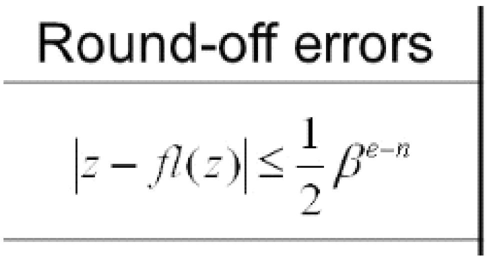


b) g(1) g(2) g(4) g(8) g(16) g(32) g(64) g(128) g(256) g(512) gives output as a 0.0. {1,2,4,8,16,32,128,256} satisfies g(n)=0.

c) Because of roundoff error, result is different than 0. Rounding error means difference between rounded-off numerical value and actual value. Reason of rounding errors is mistake in the representation of real numbers and the arithmetic operations done with them. For example , in Python:

4.9-4.845 == 0.055 returns false. Because result of this operation is 0.055000000000000604. Floating number can not be represented by the exact number, it is just approximation, computers is inefficient to represent some numbers correctly. When it is used in arithmetic, small error is occured.

d) This algorithm is unstable algorithm. Unstable algorithm is says that small changes produce large changes in the output . Also, According to formula, error increases with increasing values of the argument n . For instance g(1) =0.0 and g(1000)=-496.0


Round-off errors

$$|z - fl(z)| \le \frac{1}{2}\beta^{e-n}$$

Question 2 :

a) ∑(nums) = 1.01+1.00999999+ 1.00999998……………..= 1.005.000,005

10^6+10^6(10^-2+10^-8)-((10^6+1)*10^6)/2 *10^8 with calculator , gives this result

b) Pairwise summation is similar to divide-conquer algorithm. That is recursively divide the array into two pairs , summing each pair and adding the two sums.

c) **Double Precision**:

Naive sum= 1005000.0049999995

Compensated(Kahan) sum=1005000.005

Pairwise sum=1005000.0049999999

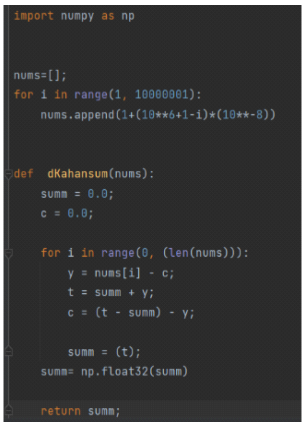
Single Precision:

Naive sum=1002466.8

Compensated (Kahan) sum =1005000.0

Pairwise sum=1005000.0

d)



```
import numpy as np


nums=[];
for i in range(1, 10000001):
    nums.append(1+(10**6+1-i)*(10**-8))


def  dKahansum(nums):
    summ = 0.0;
    c = 0.0;

    for i in range(0, (len(nums))):
        y = nums[i] - c;
        t = summ + y;
        c = (t - summ) - y;

        summ = (t);
    summ= np.float32(summ)

    return summ;
```

 In Kahan summation algorithm code, c is referred to error and its value is 4.552136445568067e-12 and Kahan summation has the least round-off error in those ways of summation . In addition Pairwise summation reduces roundoff error compared to naive sum.

Run time in Pycharm Python compiler:

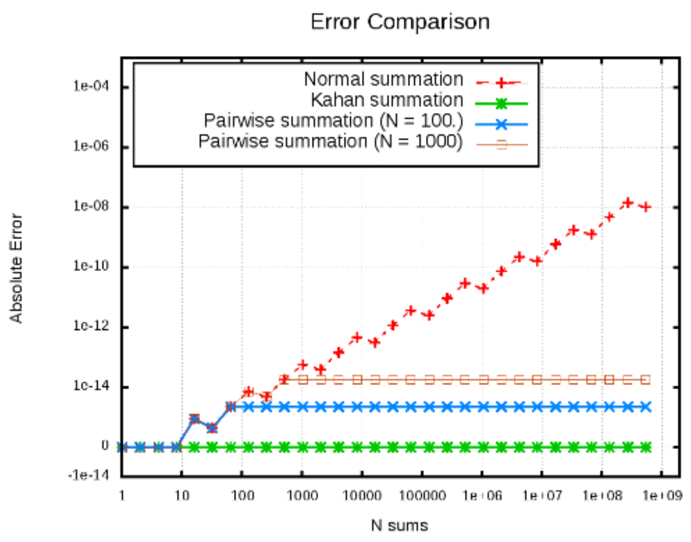
--- 1.4787838459014893 seconds ---(naive sum)

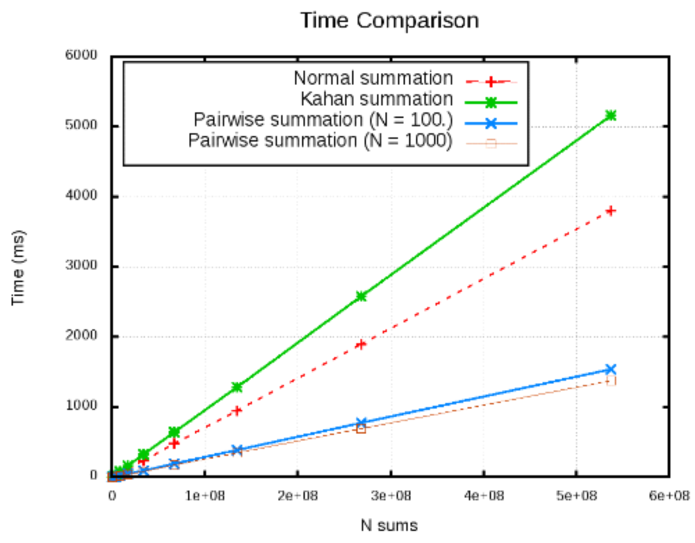
--- 1.5551159381866455 seconds ---(pairwise)

--- 1.5027167797088623 seconds ------(compensated)

Because of pairwise summation has recursion , it is slower than compensated summation.Also , compensated summation has more floating point operations. Naive is fast but it tends to more numerical error.

e) As a result, naive summation is fast, but it tends to accrue numerical error due to finite precision. The compansated summation uses an extra variable to follow and restore lost precision, but because of more floating point operation, has more significant cost. Finally, Pairwise summation is simply recollect the addition operations to get better steadiness than naive summation without floating point operations. Pairwise summation is implemented with simple recursion and slower than Kahan summation.





Improvements:

Increasing block size for pairwise summation is numerically identical and gets better and better.

For Kahan summation, there are some way for improvements such as “improved Kahan–Babuška algorithm", "iterative Kahan–Babuška algorithm".

```
function KahanBabushkaNeumaierSum(input)
    var sum = 0.0
    var c = 0.0                          // A running compensation for lost low-order bits.

    for i = 1 to input.length do
        var t = sum + input[i]
        if |sum| >= |input[i]| then
            c += (sum - t) + input[i] // If sum is bigger, low-order digits of input[i] are lost
        else
            c += (input[i] - t) + sum // Else low-order digits of sum are lost.
        endif
        sum = t
    next i

    return sum + c                       // Correction only applied once in the very end.
```

This function has more case than Compensated sum ,which is when the next term is bigger than running sum. For instance, [2.0,+ 15^100,2.0,-15^100] gives 0.0 in Kahan’s algorithm , however, improved Kahan–Babuška algorithm gives 4.0 (correct).

```
function KahanBabushkaKleinSum(input)
    var sum = 0.0
    var cs  = 0.0
    var ccs = 0.0
    var c   = 0.0
    var cc  = 0.0

    for i = 1 to input.length do
        var t = sum + input[i]
        if |sum| >= |input[i]| then
            c = (sum - t) + input[i]
        else
            c = (input[i] - t) + sum
        endif
        sum = t
        t = cs + c
        if |cs| >= |c| then
            cc = (cs - t) + c
        else
            cc = (c - t) + cs
        endif
        cs = t
        ccs = ccs + cc
    end loop

    return sum + cs + ccs
```

Also,this function gives better result.