CENG 371 - Scientific Computing
Fall 2022-2023
Homework 1

KILIÇ, DEKAN
dekan.kilic@metu.edu.tr

October 30, 2022

---

1. (a) This is the plot of g(n) = f(n) / eps.The code that generates this graph is in q1.m file.

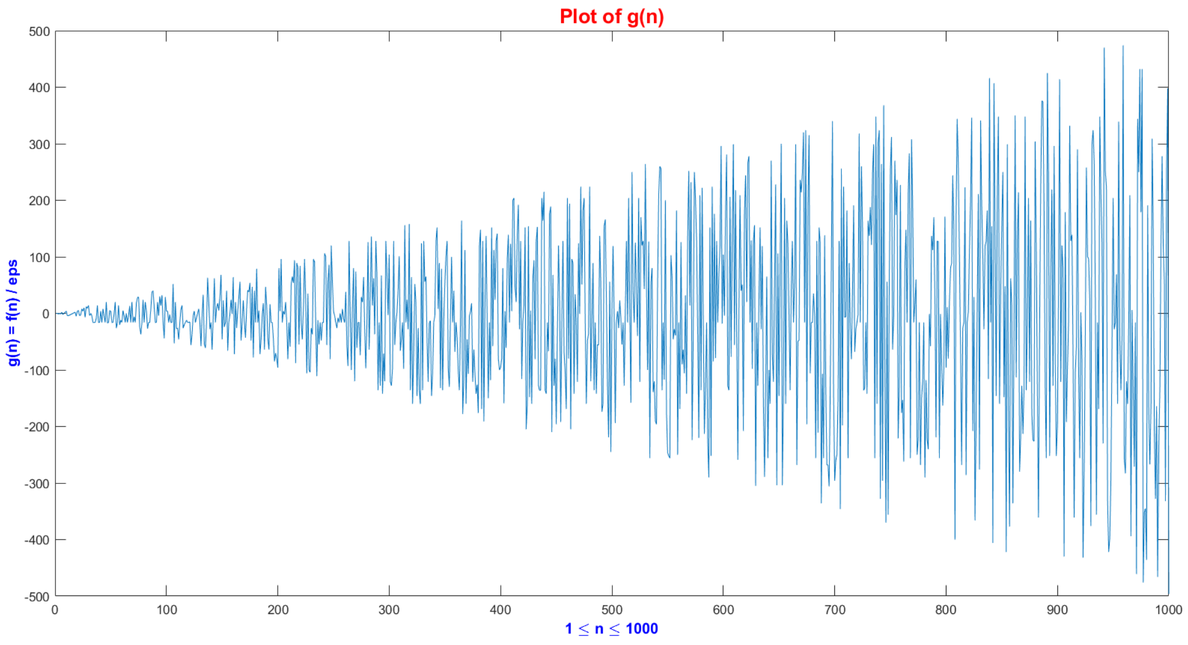

Figure 1: Plot of g(n)

   (b) For the values of n that are the power of 2 in the given range, g(n) = 0.That is when n is equal to 1,2,4,8,16,32,64,128, 256,512, the function g(n) becomes zero.Also, in q1.m file, I am printing these values that leads g(n) to be zero, and you can also check them there.

   (c) Since most of the decimal values of f(n) are not exactly representable as a binary fraction, the calculation of f(n) does not give zero when n is different from the power of 2, but rather reveals the quantity eps which is equal to 2.2204e-16 for the double precision.For example, for n = 5,

$$5\left(\frac{6}{5}-1\right)-1$$

   is not equal to 0 because the decimal value $\frac{6}{5}$ is not exactly representable as binary on computers.

   (d) Since the result of f(n) returns a nonzero value that is eps(n), and eps(n) depends on the value of n, so as n gets larger in size, so does eps(n).Also, Since the values of f(n) is getting large in size when n increases except for the values of n that are equal to power of 2,the values of g(n) is also getting large because we are dividing it with epsilon, and both of them are very small numbers, so the result of the divisions returns large numbers when we consider its absolute value.

2. (a) I have both written the code that calculates the result of this summation naively in part c and calculated it with summation formula here.

$$\sum_{i=1}^{n} 1+(10^6+1-i)\times 10^{-8} = \sum_{i=1}^{n} 1+\sum_{i=1}^{n} 10^{-2}+\sum_{i=1}^{n} 10^{-8}-10^{-8}\times\sum_{i=1}^{n} i$$

$$n + 10^{-2}n + 10^{-8}n - 10^{-8}\left[\frac{n(n+1)}{2}\right]$$

$$n\left(1 + 10^{-2} + 10^{-8}\left[1 - \frac{n+1}{2}\right]\right)$$

where $n = 10^6$.
Thus, the result is 1005000.005 = 1.005000005 $\times 10^6$

(b) In numerical analysis, pairwise summation, also called cascade summation, is a technique to sum a sequence of finite-precision floating-point numbers that substantially reduces the accumulated round-off error compared naively accumulating the sum in sequence.It works by recursively breaking the sequence of numbers into halves summing each half, and adding the two sums, so it is a divide-and-conquer algorithm.

(c) i. I calculated the result of the naive summation in q2.m file by using MATLAB.I have printed the results of both double and single precision to command window, and also I calculated the time needed for the execution of the naiveSumDouble and naiveSumSingle functions.They are named as timeNaiveDouble and timeNaiveSingle as shown on workspace.

ii. I calculated the result of the compensated summation in q22.m file by using MATLAB.The same applies here.

iii. I calculated the result of the pairwise summation in q222.m file by using MATLAB.The same applies here.

(d) In the naive summation, when we calculate the result of the summation in double precision, it is equal to 1005000.0049999995389953.Whereas, it is equal to 1002466.6875000000000000 in single precision.The result of single precision naive summation is approximately 2500 less than the double precision.The run time of the naive summation is approximately 0.0012 seconds in double precision, and it is equal to 0.0010 seconds in single precision.In single precision, the run time is a little bit less than in double precision because single precision numbers such as float use less memory, so they can be transferred to registers much faster.The relative error of the naive summation is $\leq (n-1)\,\epsilon\left(\frac{\sum|x_i|}{|\sum x_i|}\right)$.

In the compensated summation, when we calculate the result of the summation in double precision, it is equal to 1005000.0050000000046566.Likewise, it is equal to 1005000.0000000000000000 = 1005000 in single precision as well.The run time of the compensated summation is approximately 0.0040 seconds in double precision, and it is equal to 0.0038 in single precision because of the same reason that I have already mentioned in the naive summation above.The relative error of the compensated summation is $\leq 2\epsilon + O(n\epsilon^2)\left(\frac{\sum|x_i|}{|\sum x_i|}\right)$.

In the pairwise summation, when we calculate the result in double precision, it is equal to 1005000.0049999998882413. Likewise, it is equal to 1005000.0000000000000000 = 1005000 in single precision as well.The run time of the pairwise summation is approximately 1.3557 seconds in double precision, and it is equal to 1.3293 seconds in single precision.The same applies here as well..The relative error of the pairwise summation is $\leq \frac{\epsilon\log_2 n}{1-\epsilon\log_2 n}\left(\frac{\sum|x_i|}{|\sum x_i|}\right)$

In conclusion, the single precision naive summation has more roundoff errors due to its calculation algorithm.The single precision compensated and pairwise summation have the same result.Both of these(compensated and pairwise) algorithms do not have any advantages over each other in single precision.But, in the double precision case, they have some differences.The algorithm that gives the best result is compensated summation in double precision case because its error is effectively $O(\epsilon)$.That is it is independent of n.

(e) In the naive summation, relative error is $\leq (n-1)\,\epsilon\left(\frac{\sum|x_i|}{|\sum x_i|}\right)$ which means that it could be large if n is large, also if $|\sum x_i|$ goes to 0, the relative error goes to infinity.Thus, it is fully dependent on n.In our example, n is equal to 10^6, so it is quite large,which means that the naive summation has more errors than other algorithms.The reason why the single precision naive summation has much larger errors is due to the epsilon ϵ because epsilon ϵ is much larger in single precision than in double precision.It is equal to 2^{-t} where t = 24 in single precision, t = 53 in double precision.
In the compensated summation, relative error is $\leq 2\epsilon + O(n\epsilon^2)\left(\frac{\sum|x_i|}{|\sum x_i|}\right)$, which means that it is actually independent of n because the term $O(n\epsilon^2)$ goes to 0 when n gets larger(also, the ϵ^2 is very small number comperated to n).Thus, the relative error is small for the compensated summation.That is it gives the best result in double precision.However, the only drawback of the compensated summation is that it requires more arithmetic operations that naive summation.Approximately, it requires 4 times arithmetic operations than naive summation, so it has more latency than naive summation.
In the pairwise summation, relative error is $\leq \frac{\epsilon\log_2 n}{1-\epsilon\log_2 n}\left(\frac{\sum|x_i|}{|\sum x_i|}\right)$, which means that its relative error is less than naive summation's relative error, and larger than compensated summation.The compensated summation has lower roundoff error, but pairwise summation has lower arithmetic operations, and it is same as naive summation.

References

[1] This is the link that I get some help for compensated summation : Kahan summation algorithm. *Wikipedia*,

[2] This is the link that I get some help for pairwise summation : Pairwise summation algorithm. *Wikipedia*,

[3] This is the link that I get some help for the run-times of the methods : TIMEIT. *MATLAB*,

[4] This is the link that I get some help for floating-point numbers in Matlab : Floating-Point. *MATLAB*,