CENG 371 - Scientific Computing
Fall 2022
Homework 1

Ağış, Fırat
e2236867@ceng.metu.edu.tr

October 30, 2022

---

1. (a) If we plot $g(n)$ using the ϵ value provided by numpy, using matplotlib, the result becomes,

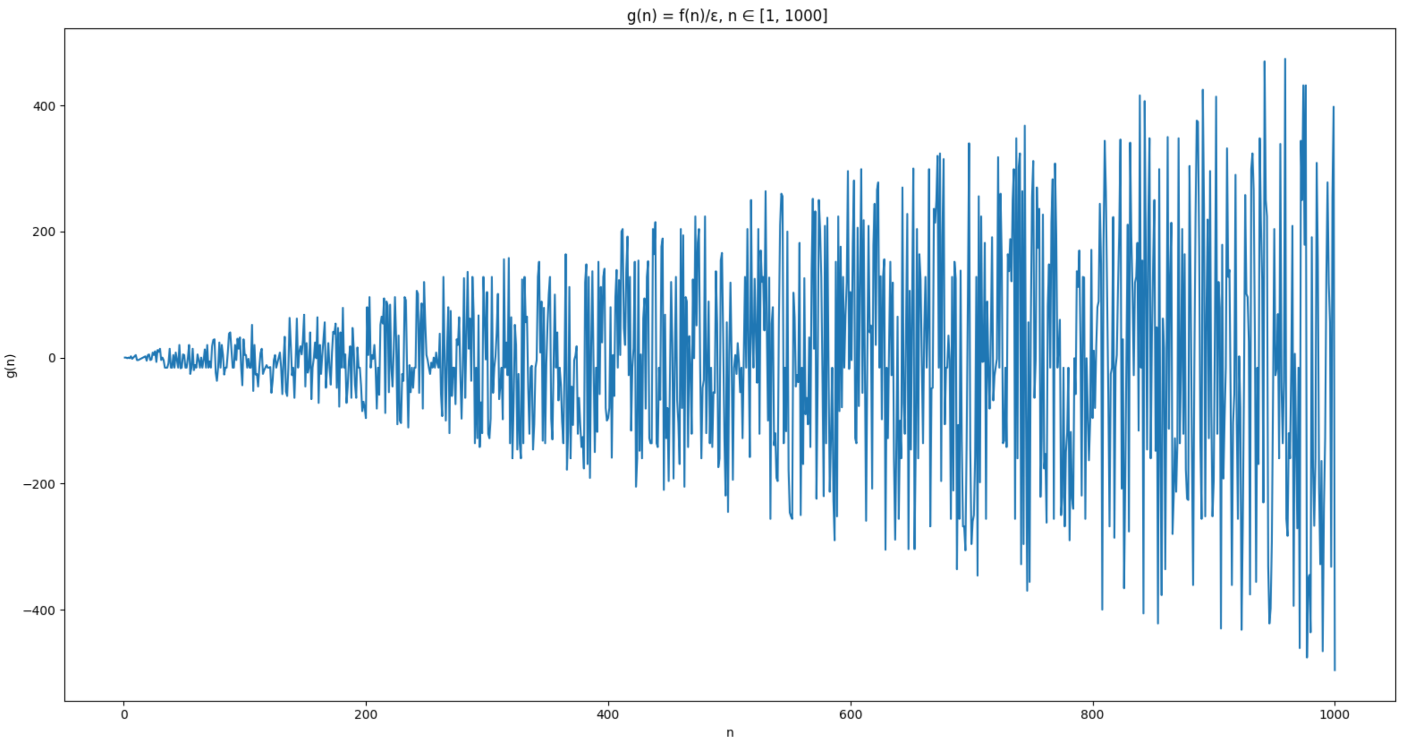

Figure 1: $g(n)$, where $f(n) = n\left(\frac{n+1}{n} - 1\right) - 1$, for the values of $n \in [1, 1000]$.

(b) $g(n) = 0$ when $n \in \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512\}$, which are the powers of 2 in $[1, 1000]$.

(c) Every operation in $f(n)$ other than $\frac{n+1}{n}$ is between relatively small whole numbers, which are all machine numbers. This makes the operation $\frac{n+1}{n}$ biggest source of error, as operations between whole numbers other than division tend to produces other whole numbers, all of which is machine numbers, ignoring the case for overflowing. If we solve

$$
\begin{aligned}
f(n) &= n\left(\frac{n+1}{n} - 1\right) - 1 \\
&= \left(n * \frac{n+1}{n} - n * 1\right) - 1 \\
&= (n + 1 - n) - 1 \\
&= 1 - 1 \\
&= 0
\end{aligned}
$$

for all values of $n \neq 0$. This means $g(n) \neq 0 \leftrightarrow f(n) \neq 0$ only happens because of errors introduces during operations. Because

$$\frac{x}{n}, n = 2^y, y \in \Re$$

is a simple bit-shift operation that only modifies the value of the exponent. As long as exponent does not overflow, we can guarantee that the operation also produces a machine number. This means when n is a power of 2, $\frac{n+1}{n}$ produces a machine number for the scales we are working with.

(d) The magnitude of $g(n)$ is correlated by the error introduced by $\frac{n+1}{n}$, as absolute error is

$$\begin{aligned}\delta &= \|g(n) - \bar{g}(n)\| \\ &= \|0 - \bar{g}(n)\| \\ &= \|\bar{g}(n)\|\end{aligned}$$

As $n \to 1000$ or $n \to \infty$, the value of $\frac{n+1}{n}$ goes to 1. But because a floating point number is represented by $\pm 1.\mathit{fraction} * 2^{exponent}$, as the value gets closer to 1, more of the fraction will be required to represent how close the value is to 1. As all bits of the fraction is used, the potential of more error will increase as more bits are cut.

2. (a)

$$\begin{aligned}\text{nums}[n] &= 1 + (10^6 + 1 - n) * 10^{-8} \\ &= 1 + \frac{10^6 + 1 - n}{10^8} \\ &= 1 + \frac{1}{10^2} + \frac{1}{10^8} - \frac{n}{10^8} \\ &= 1.01000001 - \frac{n}{10^8}\end{aligned}$$

Because nums$[n]$ is a linearly changing series, we can use the summation formula

$$\sum_{n=1}^{N} \text{nums}[n] = N * \left(\frac{\text{nums}[1] + \text{nums}[N]}{2}\right).$$

If we substitute the values for $n \in [1, 10^6]$,

$$\begin{aligned}\sum_{n=1}^{10^6} \text{nums}[n] &= 10^6 * \left(\frac{\text{nums}[1] + \text{nums}[10^6]}{2}\right) \\ &= 10^6 * \left(\frac{1.01000001 - \frac{1}{10^8} + 1.01000001 - \frac{10^6}{10^8}}{2}\right) \\ &= 10^6 * \left(\frac{1.01000000 + 1.00000001}{2}\right) \\ &= 1005000.005\end{aligned}$$

(b) When using floating point number, every arithmetic operation has a chance of resulting in a non-machine number, causing a rounding, thus introducing errors. Pairwise summation aims to perform the least amount of addition operations using a binary tree like recursion tree, performing only $\log n$ addition operation when summing n number of numbers.

(c) If we compare results of naive, pairwise and compensated summation for both single and double precision floating point number,

Table 1: $\sum \text{nums}[n]$ where $n \in [1, 10^6]$ using multiple methods

Methods	Single Precision	Double Precision
Naive Summation	1002466.6875	1005000.0049999995
Pairwise Summation	1005000.0	1005000.0049999999
Compensated Summation	1005000.0	1005000.005

(d) As formulas for calculating absolute and relative error for a function $f(n)$ is as follows,

$$\text{Absolute Error: } \delta = \|f(n) - \bar{f}(n)\|$$

$$\text{Relative Error: } \delta\Delta = \frac{\delta}{\|f(n)\|}$$

Because we know the correct answer for

$$\sum_{n=1}^{10^6} \text{nums}[n] = 1005000.005,$$

the absolute and relative errors from the data we gathered is as follows.

Table 2: Absolute and relative errors for all methods used

Methods	Single Precision	Double Precision
Naive Summation	$\delta = 2533.3175$	$\delta = 0.0000000005$
	$\delta\Delta = 0.0025207139178073934$	$\delta\Delta = 4.97512435335759 * 10^{-16}$
Pairwise Summation	$\delta = 0.005$	$\delta = 0.0000000001$
	$\delta\Delta = 4.9751243533575904 * 10^{-9}$	$\delta\Delta = 9.950248756218905 * 10^{-17}$
Compensated Summation	$\delta = 0.005$	$\delta = 0$
	$\delta\Delta = 4.9751243533575904 * 10^{-9}$	$\delta\Delta = 0$

Execution time wise, if we use asymptotic analysis on naive, pairwise, and compensated summation,

i. It is easy to see that naive summation has a big o number of $O(n)$, as it a single addition operation for each element of the input.

ii. For compensated summation, it is a binary recursion tree, whose big o number is $O(\log n)$.

iii. Compensated summation is identical to naive summation, other than the fact that it performs constant number of extra operations for each element of the input. Because coefficients are ignored during asymptotic analysis, it also has a big o number of $O(n)$.

(e) As demonstrated in 2.c and 2.d, computations done using single precision floating point numbers are more prone to errors compared to double precision floating point numbers, due to difference between machine numbers being higher. We used a variety of summation methods to decrease the errors introduced in the summation step, but using compensated summation instead of pairwise summation did not improved the result while using single precision floating point numbers unlike double precision floating point numbers.
I hypothesize that that that is due to the magnitude of the errors introduced during the calculation of the elements of nums. As elements of nums themselves becomes less accurate, increasing the accuracy of the summation of those elements becomes less relevant. In that case, It must be so that, even magically eliminating any errors that can be introduced during the summation step completely might not decrease the error present for single precision floating point numbers.