Coding Assignment 1

Wanghei Lo

September 23, 2022

1. Executive Summary

This report will discuss how to use algorithms to convert real numbers to floating point numbers, and how to add the converted floating point numbers. In the process of these operations, we will discuss the idea and operation method of the algorithm, as well as the error calculation of each step.

2. Statement of the Problem

The first part of the problem is to implement the algorithms to perform translation of a real number (x) into a floating point number fl(x) , then proceed round to nearest to fl(x). Upon obtaining 2 floating point number fl(x1), fl(x2), perform fl(x1) ⊞ fl(x2) which = (fl(x1) + fl(x2))(1+δ).

The second part is to generate a large set of floating point number then compute the relative error of each fl(xi) with xi, and also the relative error of fl(xi) ⊞ fl(xj) with fl(xi) + fl(xj). In order to called a correct result, all errors must be less than $u_n = \frac{\beta^{1-t}}{2} = 5 * 10^{-5}$.

The last part of the problem is to compute the actual error of $\hat{s} = s_n + \sum_{k=2}^{n} s_k \delta_k$ with $s_n = \xi_{1:n}$, whether it is less than or equal to $(n-1)(\zeta_{1:n})u_n$. Next, we will have to compute the relative condition number for a large amount of iteration, obtain the set's max value which is approximately equal to the exact relative condition number. Then, show whether the exact sum error is less than or equal to the approximated condition number.

3. Description of the Algorithms and Implementation

All computation in this report are performed using the Python 3 language. Please use Google collab or Jupyter notebook to open the code document.

For routine 1, 2, 3, the bounds are $t = 5, e_{min} = -9, e_{max} = 0$

In routine 1, we have to round a given number to it's nearest based on it's remainder. The algorithm takes 3 data input: remainder, mantissa, and exponent. The algorithm first perform an if statement to check whether the remainder is greater than or equal to 5. If the if statement is true, than it will proceed the rounding process: We add 1

for the mantissa since the remainder is greater than or equal to 5. Moreover, in order to prevent the scenario of rounding a number that has a remainder ≥ 5 and $m = 99999$, we divide m by 10 and add 1 for e. However, if the if statement for remainder ≥ 5 is false, it means the mantissa does not need to be round to nearest. The algorithm will generate an output as a rounded m and it's e.

In routine 2, we have to translate a given number into it's floating point number. The algorithm take 1 data input, $x \in R$. It first obtain the sign of the number, then compute the absolute value of x and called it a. We also have to initialize the variable m, e, and remainder. The first step of the algorithm is to use a if statement to check whether a is greater than 10^5 or smaller than or equal to 10^5.

If $a > 10^5$, then we proceed an integer division for a by 1 which will get rid of all it's decimal places. Then we can call a as m. In order to proceed m to rounding it's nearest we first have to obtain it's remainder, m, and e. However, we have to perform a while loop to trigger while $m \geq 10^5$ to make sure m has the correct t value: Each time when we enter the while loop we get it's remainder by doing $m \, mod \, 10$, then integer divide m by 10, then add 1 to e. Once m is no longer $\geq 10^5$, we can proceed the number into round to nearest. After getting the output from round to nearest, we then multiply s to m to make sure it has the correct sign since we perform absolute value of x in the beginning. Then it will return the output of the algorithm.

If $a \leq 10^5$, we then proceed a while loop trigger while $a < 10^4$ to make sure a has the correct t value. We then multiply a by 10 and add 1 to it's corresponding e. After the while loop is over, we obtain it's remainder by first getting it's decimal place by doing $\llcorner 10 * (a - \llcorner a \lrcorner) \lrcorner$. We then obtain it's m by doing $\llcorner a \lrcorner$. From the data we have we can proceed to the round to nearest function. After rounding we also have to multiply m with s to make sure it has the correct sign.

The output of this algorithm will be the floating point number of x.

In routine 3, we will proceed a floating point addition denoted as $\boxplus$ for 2 floating point numbers. The input of this algorithm will be 2 sets of floating point number in their (m, e) form. We first start with computing the shift of the two floating point numbers by doing $e_x - e_y$. If shift > 0, we then multiply m_x by 10^{shift} and call it a, we then call m_y as b, and e_y will be e. If shift ≤ 0, we then multiply m_y by 10^{-shift} and call it a, we then call m_x as b and e_x be e. We then assign $c = a + b$, and $s = sign(c)$, then we take the absolute value of c and reassign it to itself. We then have to check the bound for t by checking whether is $c \geq 10^5$, if this is true, we then perform a while loop trigger while

$c \geq 10^5$. In this while loop we obtain c's remainder by doing $c\ mod\ 10$, then we perform reassign c by $\llcorner c \div 10 \lrcorner$, and assign $e = e + 1$. On the other hand, while $c < 10^4$, we reassign c by $c * 10$, and assign $e = e - 1$. There will not be a remainder here and we just assign it's remainder as 0. Now we are left with a data of remainder, c, e, which is now ready to proceed into the round to nearest algorithm. After getting the output from it we then multiply c with s to make sure it has the correct sign. We now have the output from routine 3 which is $x \boxplus y$ in the $(m,\ e)$ form.

For the last algorithm: Accumulation. The input is 100 floating point numbers and the goal is to perform $\boxplus$ for each of the floating point numbers. We used algorithm 3 for $\boxplus$ calculation and generated 100 $fl(x_1)$ and 100 $fl(x_2)$ in the bound of $[-\ 999, 999]$ for the data. We first compute $fl(x_1) \boxplus fl(x_2)$ and insert the result into an array in position 0. By doing that we make sure everytime we call array[0] will be able to obtain the previous $\boxplus$ result. Then we proceed $\boxplus$ until we reach $fl(x_{100})$. The result of the algorithm will be $\widehat{s_{100}}$.

4. Description of the Experimental Design and Results

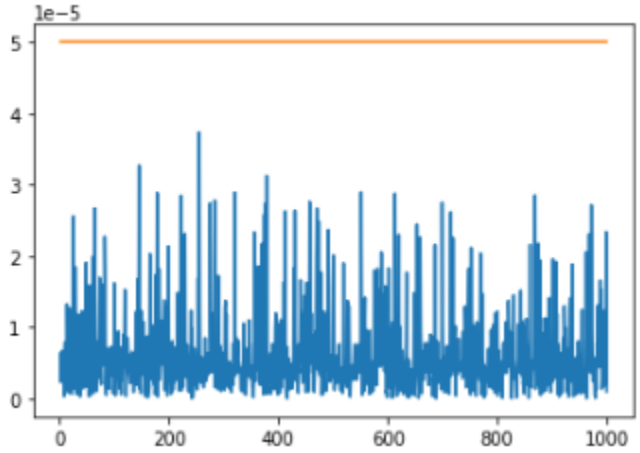
The result of task 1, for algorithm 1 the input is remainder = 8.0, m = 99837.0, e = 0. The output is m = 99838.0, e = 0

For algorithm 2 the input is x = 99837.83616282244, the output fl(x) = 99838.0 and e = 0

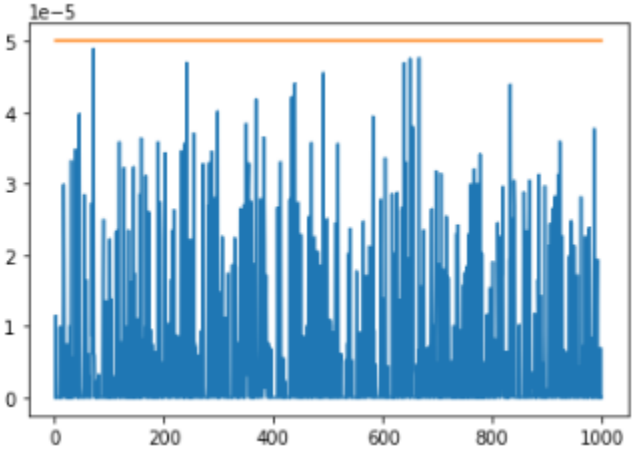
For algorithm 3 the input is $fl(x_m, x_e) = (99838,\ 0)$, $fl(y_m, y_e) = (-\ 18825,\ 0)$. The output is $fl(x) \boxplus fl(y)_m, e = (81013,\ 0)$

```
---------------------- Start of Task 1 --------------------------
Algorithm 1
Data: 8.0, 99837.0, 0
Result: 99838.0, 0

Algorithm 2
Data: 99837.83616282244
Result: 99838.0, 0

Algorithm 3
Data: (99838.0, 0), (-18825.0, 0)
Result: 81013.0, 0

---------------------- End of Task 1 --------------------------
```

For the correctness test, we generated 2000 floating point numbers and 1000 $fl(x) \boxplus fl(y)$ result. The max relative error for x is $3.85293 * 10^{-5}$ which is smaller than u_n. The following figure illustrate the comparison of error of x and u_n, the orange line indicate u_n and each vertical blue line indicate relative error of x.



For the relative error of $fl(x) \boxplus fl(y)$, the max relative error is $4.8926 * 10^{-5}$ which is smaller than u_n. The following figure illustrate the comparison of error of $fl(x) \boxplus fl(y)$ and u_n, the orange line indicate u_n and each vertical blue line indicate relative error of $fl(x) \boxplus fl(y)$.



```
---------------------- Start of Task 2 ---------------------------

Algorithm performed 1000 times

Overall error for x1, len = 1000
Min_error = 2.9228064304521024e-09, Max_error = 3.734839252673293e-05, Mean_error = 6.2276427138087646e-06

Overall error for x2, len = 1000
Min_error = 1.842075298546923e-09, Max_error = 3.852931525023169e-05, Mean_error = 6.2561776740473166e-06

Overall error for x1 + x2, len = 1000
Min_error = 0.0, Max_error = 4.892607270414404e-05, Mean_error = 6.291407815358132e-06

---------------------- End of Task 2 ---------------------------
```

For the accumulation part, we obtained $\widehat{s} = 5067.7$, based on Python language's number inexact representation, there is an output error for getting the correct t value. We obtain s by perform simple addition (not $\boxplus$) for x_1 to x_{100}. The actual error of $|\widehat{s_{100}} - s_{100}| = 0.52700$, which is smaller than the bound $(n-1) * ||\xi_{1:n}|| * u_n = 272.61$. We can say the bound is not tight at all since $||\xi_{1:n}||$ is the norm of all vectors in $\xi_{1:n}$, which will definitely result a larger number than $|\widehat{s} - s|$.

```
s_hat = 5067.700000000001
s_n = 5067.1730000000025

|s_hat - s_n| = 0.5269999999982247
(n-1)*||ξ 1:n||*u_n = 272.60915715000004
```

For the conditioning analysis part, in order to compute c_{rel} we have to obtain 4 input, $\sum_{i=1}^{n=100} x_i$, $\sum_{i=1}^{n=100} (x_i + p_i)$, $||x_{1:100}||$, $||p_{1:100}||$. To get $||\delta x||$ we do $|\sum_{i=1}^{n=100} x_i - \sum_{i=1}^{n=100} (x_i + p_i)|$, for $||x||$ we do $|\sum_{i=1}^{n=100} x_i|$, for $||\delta d||$ we do $||p_{1:100}||$, and for $||d||$ we have $||x_{1:100}||$. The algorithm goes by following: We first set a for loop that's going to run 1000 time, for each iteration we used another for loop to obtain p with it's corresponding x and obtain the data that's needed for computing c_{rel}. We insert the value into 0 index of the array because we will not get the correct c_{rel} until we obtain all the p. The reason is because the sum of x_i is a fixed number and p will be generated in each iteration, therefore there will be cases that there will be no p for x before the last iteration

which causes $x - p$ be a large number, which will result in an incorrect conditional number.

Upon 1000 iteration we obtain a array with a length of 1000 which every vector is c_{rel} of each iteration. The $max(c_{rel}) \approx K_{rel} = 4.475774$ which indicate this is a well-conditioned problem because it is a small number

.

```
Algorithm performed 1000 times
-------------------------------------------
delta_x =0.0932130370347295
x = 5067.183357061316
delta_d = 1.3910485909567751
d = 55072.55929840072
-------------------------------------------
Length of C_rel array: 1000
Max value of C_rel array: 4.475774456214143
```

Now, to discuss about the consistency between the relative error between the 2 exact sum of x_i and ξ_i with the approximated condition number, we can rearrange the equation of c_{rel}:

$$K_{rel} = \frac{||\delta x||}{||x||} * \frac{||d||}{||\delta d||}$$

$$K_{rel} * \frac{||\delta d||}{||d||} \geq \frac{||\delta x||}{||x||}$$

Where $\frac{||\delta x||}{||x||}$ is the relative error of x_i and ξ_i and to tell whether it is consistent we compare it whether it is smaller than or equal to $K_{rel} * \frac{||\delta d||}{||d||}$.

```
Relative error of x_i and ξ_i = 2.043948399709101e-06
Consistent? 0.00011305121516184578
```

Which is true, thus it is consistent.