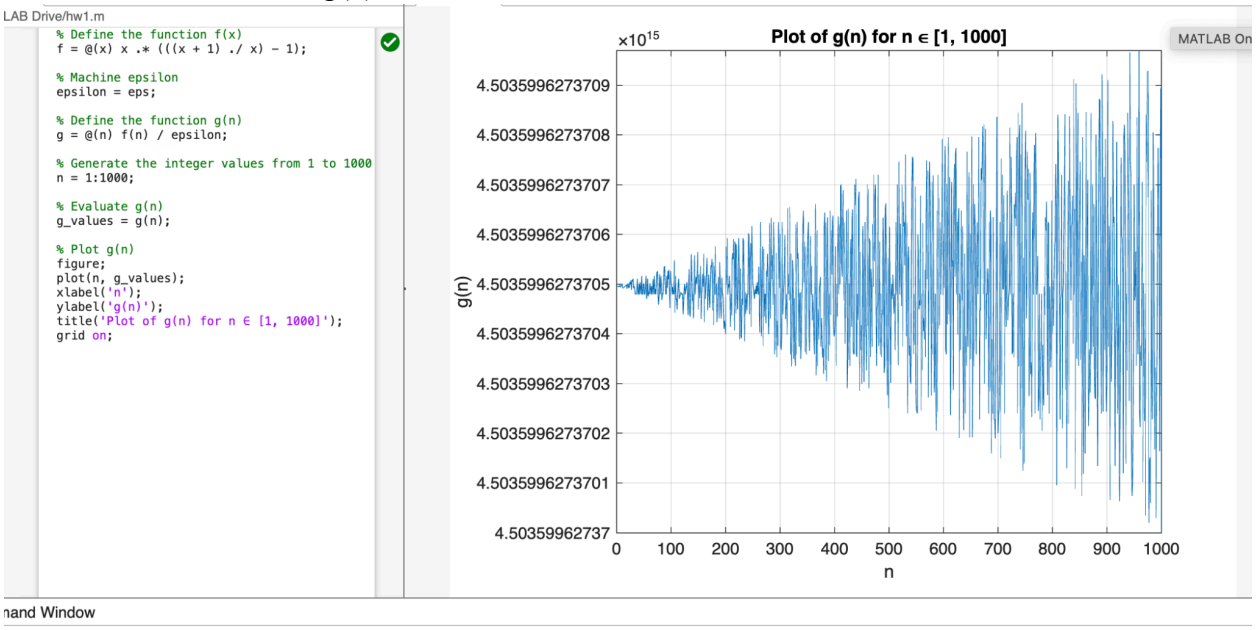
CENG 371 HW1
Aneliya Abdimalik

Question 1

1. Matlab Code and Plot of g(n)



2. Only power of 2 in range [1,1000] satisfies g(n)=0 because the calculations in MatLab(or any other programming scope) are done with numbers in finite binary representations(double precision floating point IEEE 754 standard). We can represent evenly/exactly only powers of 2, which are less prone for rounding errors while calculating. So MatLab returns exact zeros g(n)=0 only when n is equal to power of 2. Even Though theoretically/analytically, the answer must be for all values of n.

```
% Find values of n where g(n) is 0
zero_indices = find(g_values == 0);

% Display the results
if isempty(zero_indices)
    disp('No values of n satisfy g(n) = 0 within the range [1, 1000].');
else
    disp('Values of n that satisfy g(n) = 0:');
    disp(n(zero_indices));
end
```

```
Values of n that satisfy g(n) = 0:
     1     2     4     8    16    32    64   128   256   512
```

3. The problematic part causing errors in the g(n) is (n+1)/n because others are relatively whole numbers and operations that are resulting in whole numbers(machine numbers) whereas division may cause floating point. So g(n)!=0 caused by errors introduced from floating point operations

4. When we are dividing (x+1)/x in f(x), floating point errors are present due to precision limitation. So these errors are amplified when we divide by such a tiny number as epsilon. Therefore, when n goes to 1000, it introduces larger relative errors, causing g(n) to scale up.

Question 2

1. Let's define an array of k as knums of size N. Knums is linear series, we can use formula for theoretical sum as (knums(1)+knums(N)/2)*N which results in 1005000.005
2. Pairwise summation is an algorithm that improves the numerical stability of summing large arrays by recursively splitting the array into pairs and summing them, then combining those partial sums. This method reduces the error associated with floating-point arithmetic by minimizing the propagation of rounding errors. When using floating point numbers, 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 number of numbers.
3.

```
>> untitled
Naive sum (single precision): 1004999.8750000000
Naive sum (double precision): 1005000.0049999999
Kahan sum (single precision): 1005000.0000000000
Kahan sum (double precision): 1005000.0050000000
Pairwise sum (single precision): 1005000.0000000000
Pairwise sum (double precision): 1005000.0049999999
>>
```

4.

```
The theoretical sum is: 1005000.0000000000
Naive Summation - Single Precision:
Sum: 1004999.8750000000, Absolute Error: 1.2500000000e-01, Relative Error: 1.2437810426e-07, Runtime: 0.00018 s
Naive Summation - Double Precision:
Sum: 1005000.0049999999, Absolute Error: 4.9999998882e-03, Relative Error: 4.9751243125e-09, Runtime: 0.00020 s
Kahan Summation - Single Precision:
Sum: 1005000.0000000000, Absolute Error: 0.0000000000e+00, Relative Error: 0.0000000000e+00, Runtime: 0.00933 s
Kahan Summation - Double Precision:
Sum: 1005000.0050000000, Absolute Error: 4.9999998882e-03, Relative Error: 4.9751243125e-09, Runtime: 0.00848 s
Pairwise Summation - Single Precision:
Sum: 1005000.0000000000, Absolute Error: 0.0000000000e+00, Relative Error: 0.0000000000e+00, Runtime: 0.28933 s
Pairwise Summation - Double Precision:
Sum: 1005000.0049999999, Absolute Error: 4.9999998882e-03, Relative Error: 4.9751243125e-09, Runtime: 0.46854 s
>>
```

As we can see from from the implementation of

a) Naive summation runs exactly N times, size of array, causing O(N)
b) Kahan summation also runs O(N) times but it reduces rounding error by using additional correction steps for each element. The number of additional steps as a coefficient in BigO can be ignored.

c) Pairwise summation divides the array in half repeatedly until reaching individual elements, performing logN recursive summation steps, with each step involving O(N) operations. Therefore, the overall complexity is O(NlogN).

5. As demonstrated, computations using single precision floating-point numbers are indeed more prone to errors compared to double precision due to the larger spacing between representable machine numbers. This spacing leads to higher rounding errors and reduced accuracy. I applied three types of summation methods—naive, Kahan (compensated), and pairwise—to investigate the errors introduced during the summation process.

The results highlighted an interesting behavior: using compensated summation (Kahan) in single precision did not significantly improve the outcome compared to pairwise summation. This can be attributed to the inherent magnitude of errors introduced when calculating the array elements. When the precision of individual array elements is already compromised, enhancing the accuracy of the summation itself becomes less impactful. In such cases, even if the summation method eliminates summation errors effectively, the total error remains high due to inaccuracies present in the array elements themselves.

Therefore, while methods like Kahan and pairwise summation are effective in minimizing errors during summation, their benefits may be limited when working with data in single precision where significant precision loss has already occurred during element computation. This underlines the importance of understanding the limits of single precision and opting for double precision when higher accuracy is required, especially when both array element calculation and summation precision are crucial.