FCM 1 HW 1 Problem 1.4

Ryan Bausback

September 20 2020

1 Executive Summary

The stability and conditioning of three different summation algorithms was evaluated using a variety of test sums designed to push the algorithms to their limits. The results confirmed the theoretical bounds for stability and conditioning. The single precision binary fan-in tree algorithm performed overall better than the single precision recursion algorithm, which was predicted by the theoretical bounding being lower.

2 Statement of Problem

Floating-point arithmetic on a computer is not associative due to the finite nature of representation on the machine, potentially leading to differing sums when numbers are added in differing orders.[1] How much the computed sum differs from the exact sum is a function both of the algorithm chosen to perform the summation (the stability) and the numbers being summed themselves (the conditioning).

Our goal is to first code three algorithms: a recursive single-precision accumulation algorithm, a binary fan-in tree algorithm, and recursive double-precision accumulation algorithm to prove their effectiveness. Then, a few different potentially problematic sums will be fed through them in order to confirm the theory about which algorithm is the most stable and well conditioned, as well as that the theoretical bounds for stability and conditioning hold.

3 Description of Mathematics

When addition is performed on floating point, error can be introduced as a result of rounding that occurs when the number of digits in the sum exceeds the number allowed in the mantissa. This addition is often denoted as:

$$g(d) = (\xi_1 + \xi_2)(1 + \epsilon_1)$$

where ϵ_1 is the error associated with adding the two floating point numbers and $f(d) = \xi_1 + \xi_2$ is the true sum.[2] This error will continue to propagate after each

subsequent operation, albeit in distinct ways depending on the algorithm. One of the ways it is often revealed is through cancellation. If digits were chopped during operations, the final result will not be able to retain them even if its magnitude is small enough where the mantissa would otherwise have room for them.

One way to evaluate how likely error is to occur is to examine how sensitive a particular problem is to changes in the parameters. This is done through the condition number, which for summation is calculated as:

$$\kappa_{rel} = \frac{|x_0|+...+|x_N|}{|x_0+...+x_N|}$$

The bound for addressing the conditioning of each algorithm that we would like to confirm, in general, is:

$$\frac{|f(d+e)-f(d)|}{|f(d)|} \leq \kappa \frac{|e|}{|d|}$$

where e is the relative perturbation of the true summation $f(d)$ and d is the input.[2] For the case were the input d is a vector, $|d|$ is the L^1 norm of the vector and e is the L^1 norm of the vector of relative perturbations.[3]

The bound for addressing the stability of each algorithm that we would like to confirm, in general, is:

$$\frac{|g(d)-f(d)|}{|f(d)|} \leq \kappa_{rel} p(n) u$$

Here, as derived in set 3 of the FCM 1 notes, $p(n) = n$ for the single-precision accumulation algorithm, where n is the number of elements that the algorithm will sum.[2] As we found in problem 1.3 of HW 1, for the binary fan-in tree, $p(n) = i$ where i is exponent on 2^i, with 2^i being the number of elements that the algorithm will sum. Therefore, since $n = 2^i > i$, we can predict that the error (the left half of the stability bound) should be less for the binary fan-in algorithm than for the single-precision accumulation algorithm, as well as within the bound computed with the condition number, $p(n)$, and unit round-off.

For the double-precision accumulation algorithm, we expect that since we are performing all operations in *double* and then typecasting into $float$ at the end of the algorithm, there will be additional round-off error, potentially leading to a less accurate result and a looser stability bound. Therefore, $p(n)$ should be some function greater than n in this case.

4 Description of Algorithm and Implementation

The Single Precision Accumulation Algorithm utilizes a for loop to continuously update the value in SSP until all the values in $XISP$ have been added to the summation. The Double Precision Accumulation Algorithm functions identical to Single Precision, except the final sum SDP is type-cast back into a float before being returned.

The Binary Fan-In Tree Algorithm, as its name implies, functions as a tree and therefore is most easily generalized for 2^n elements in the initial array

$XISP$. However,$XISP$ is not limited to only having a number of elements of the form 2^n, hence a blank array of 0s (denoted $SSPprelim$) is created having 2^n elements that is at least as large as $XISP$. Then, the elements of $XISP$ are fed into $SSPprelim$. This done does not change the actual number of arithmetic operations performed on the data and therefore will not impact the overall error.

The actual fan-in proceeds by adding the first element to the last element and saving it in the first element, adding the second element to the second-to-last element and saving it in the second element, and so forth, for each level of the tree (as illustrated below). In this way, the final sum will always be located in the first position of the array. An middle indicator (denoted *indicator*) is used to denote the stopping point for each level of additions for the interior for-loop, while a regular for-loop counts upwards toward n (since there are n-levels of addition for a 2^n tree).

```
PS C:\Users\Xaalt\Documents\FCM 1> javac hw1.java
PS C:\Users\Xaalt\Documents\FCM 1> java hw1
The numbers to be added are:
1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000, 1.000000,
The number of additions for this level is 4
1.0
2.0
2.0
2.0
1.0
1.0
1.0
0.0

The number of additions for this level is 2
3.0
4.0
2.0
2.0
1.0
1.0
1.0
0.0

The number of additions for this level is 1
7.0
4.0
2.0
2.0
1.0
1.0
1.0
0.0

The sum is: 7.000000 using Single precision fan-in.
```

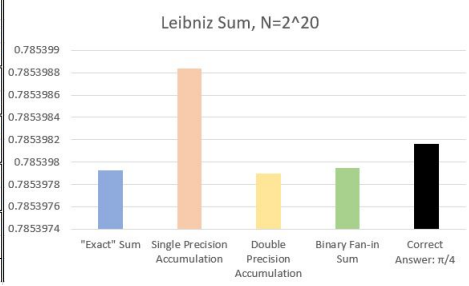
5 Experimental Design and Results

The first experiment was to confirm that the algorithms were able to add a known sum with some level of accuracy. For this purpose, the Leibniz sum was chosen.[4]

$$\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = \frac{\pi}{4} = 0.7853981633974483$$

This sum was chosen not only because it has a known total, but also because it shares many of its characteristics with subsequent summations tested, such as alternating signs and decreasing rational terms. Since the sum converges to exact value at $i = \infty$, only a large value of N, 2^{20} was used to confirm.

Correct Answer: π/4	~ 0.7853981633974483
	Leibniz Sum; N=2^20
"Exact" Sum	0.785397925
Condition Number	10.07543637
Single Accum. Stablity Bound	0.629665702
Fan-In Stability Bound	1.20E-05
Single Precision Accumulation	0.78539884
Relative Error	1.17E-06
Double Precision Accumulation	0.7853979
Relative Error	4.81E-08
Binary Fan-in Sum	0.78539795
Relative Error	2.78E-08
Conditioning Error	1.00E-06
Conditioning Bound	1.01E-05
Relative Error From π/4	3.04E-07



The next three summations were designed to take advantage of the problems associated with adding floating point numbers, potentially leading to contradictions of the theory. Each sum was then run for $N = 1,000$, $N = 10,000$, and $N = 2^{20}$. $N = 2^{20}$ was chosen to be a large number of the form 2^K so that the binary fan-in algorithm would not waste time adding hundreds of 0's.

The first test sum to be used was:

$$\sum_{i=1}^{N} i \sin[\frac{1}{i} + \frac{i}{2}]$$

This sum was chosen because the sin function produces elements of similar magnitudes with differing signs, providing ample opportunities for round-off error with simple accumulation since it adds things sequentially, as illustrated below.

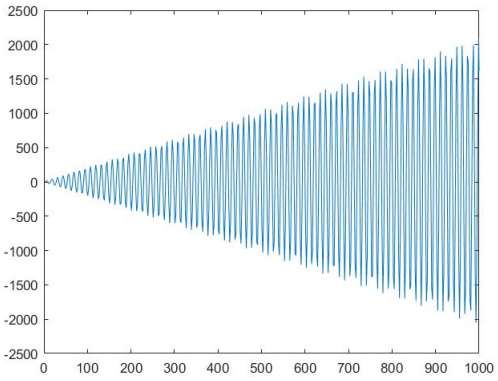
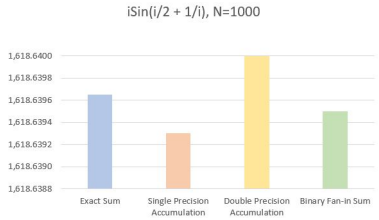
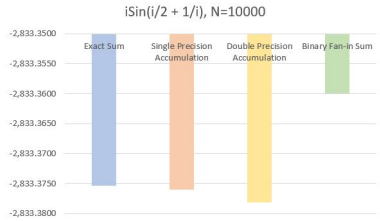


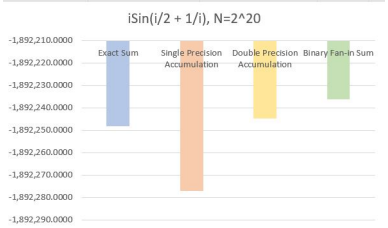
Figure 1: Progression of Single Precision Accumulation, $i \sin[\frac{1}{i} + \frac{i}{2}]$, N = 1000

This sum also is problematic for binary fan-in, as the sin function is multiplied by i. Since the implementation of binary fan-in being used adds the first element to the last element, the differing magnitudes will lead to cancellation on the second level of additions, potentially leading to error close to the theoretical bound.[5]

	iSin(i/2 +1/i); n=1,000	iSin(i/2+1/i); n=10,000	iSin(i/2+1/i); n=2^20
Exact Sum	1618.639653	-2833.375377	-1892248.275
Condition Number	196.7553904	11234.86105	184957.6664
Single Accum. Stablity Bound	0.011726621	6.695977186	11558.95333
Fan-In Stability Bound	1.17E-04	0.009374368	0.220469538
Single Precision Accumulation	1618.6393	-2833.376	-1892277.2
Relative Error	2.29E-07	2.12E-07	1.53E-05
Double Precision Accumulation	1618.64	-2833.3782	-1892244.8
Relative Error	2.23E-07	9.87E-07	1.86E-06
Binary Fan-in Sum	1618.6395	-2833.3599	-1892236.2
Relative Error	7.84E-08	5.48E-06	6.35E-06
Conditioning Error	1.00E-06	2.98E-08	1.00E-06
Conditioning Bound	1.97E-04	3.35E-04	1.85E-01







The second sum was chosen for similar reasons to the above sin summation, although it manifests itself in the opposite way. This sum was:

$$\sum_{i=1}^{N}(-1)^i\frac{1}{i^2}$$

Once again, the alternating sign of successive elements of similar magnitude creates opportunities for cancellation with the simple accumulation algorithms, although this time with each additional sum getting progressively smaller. This

also can be the case with the binary fan-in, since an even N will result in the first and last of be of opposite signs, leading to cancellation.[5] An additional concern with this sum is that if N is large enough, terms with i close to N may be less unit round-off once added to the sum or to terms with significantly larger magnitude, resulting in no change to the overall sum and creating further errors. This is illustrated below for single precision accumulation:

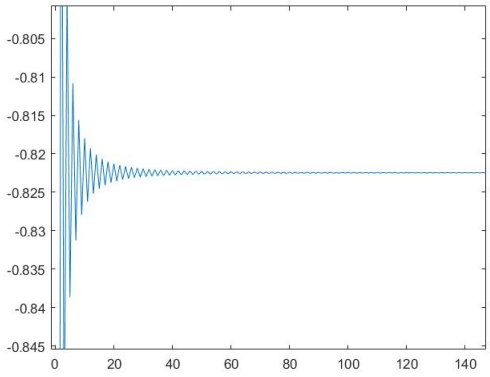


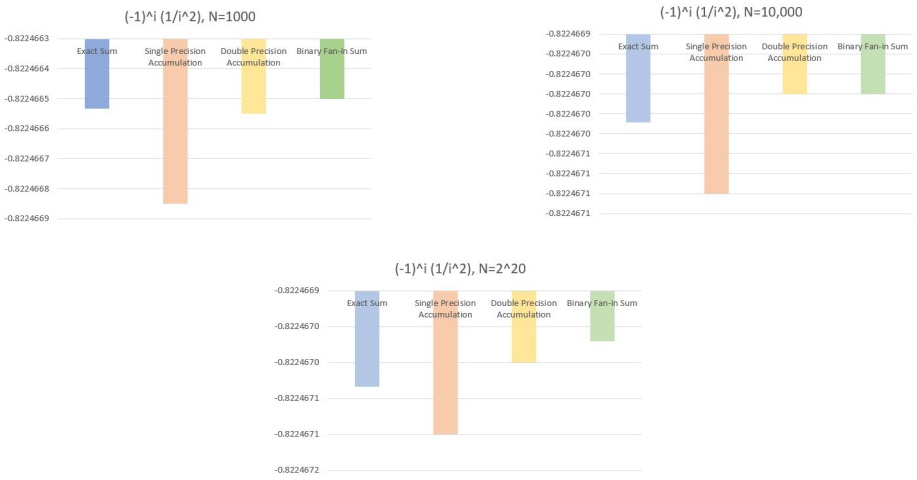
Figure 2: Progression of Single Precision Accumulation, $\sum_{i=1}^{N}(-1)^i \frac{1}{i^2}$, N=1000

	(-1)^i (1/i^2); n=1000	(-1)^i (1/i^2); n=10000	(-1)^i (1/i^2); n=2^20
Exact Sum	-0.822466534	-0.822467028	-0.822467033
Condition Number	1.998785967	1.999878433	1.99999884
Single Accum. Stablity Bound	1.19E-04	0.001191928	0.124990187
Fan-In Stability Bound	1.19E-06	1.67E-06	2.38E-06
Single Precision Accumulation	-0.82246685	-0.8224671	-0.8224671
Relative Error	3.85E-07	7.33E-08	6.72E-08
Double Precision Accumulation	-0.82246655	-0.822467	-0.822467
Relative Error	2.23E-08	8.15E-10	5.26E-09
Binary Fan-in Sum	-0.8224665	-0.822467	-0.82246697
Relative Error	5.02E-08	8.15E-10	7.77E-08
Conditioning Error	1.00E-06	1.00E-06	1.00E-06
Conditioning Bound	2.00E-06	2.00E-06	2.00E-06

The final sum tested was:

$$\sum_{i=1}^{N} \omega_i, \quad \omega_i \in [-3, 3]$$

This sum was chosen because it potentially could have terms of similar magnitudes with alternating signs, leading to cancellations after round-off.[5] This is especially true for the binary fan-in algorithm, since terms do not get progressively larger as $i \to N$. This sum is therefore expected to perform better for the simple accumulation that for binary fan-in, since the chance of getting directly



opposite terms in first few slots of the array many times in a row is relatively small.

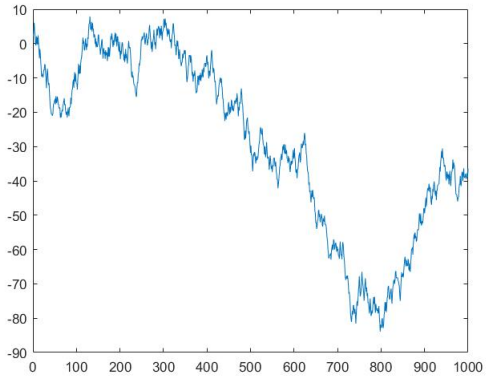
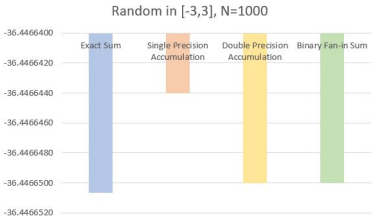


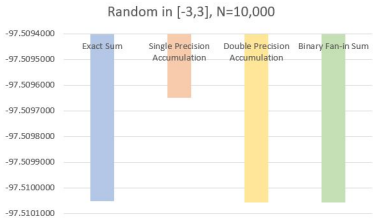
Figure 3: Progression of Single Precision Accumulation, $\sum_{i=1}^{N} \omega_i, \omega_i \in [-3, 3]$, N=1000

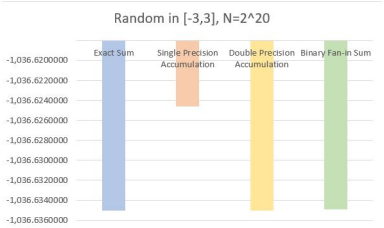
6 Conclusions

For the Leibniz sum, the exact sum algorithm was able accurately produce the correct sum with within six decimal digits of accuracy with a relative error of $3.04 * 10^{-7}$. Considering that the Leibniz sum converges to $\pi/4$ very slowly and requires approximately five billion terms to get within 10 digits of accuracy,

	Random in [-3,3]; n=1000	Random in [-3,3]; n=10,000	Random in [-3, 3]; n=2^20
Exact Sum	-36.44665065	-97.51005136	-1036.635008
Condition Number	41.97463018	153.698791	1516.67332
Single Accum. Stablity Bound	0.002501688	0.091604479	94.78469568
Fan-In Stability Bound	2.50E-05	1.28E-04	0.001807875
Single Precision Accumulation	-36.446644	-97.50965	-1036.6246
Relative Error	1.87E-07	4.10E-06	1.00E-05
Double Precision Accumulation	-36.44665	-97.510056	-1036.635
Relative Error	2.23E-08	4.29E-08	1.96E-09
Binary Fan-in Sum	-36.44665	-97.510056	-1036.6349
Relative Error	2.23E-08	4.29E-08	1.16E-07
Conditioning Error	1.00E-06	1.00E-06	1.00E-06
Conditioning Bound	4.20E-05	1.54E-04	0.001516673







the exact sum algorithm is performing exactly as expected for only adding over one million terms.[6] It is also apparent that the relative forward errors for all three single precision algorithms are well within their respective stability bounds ($0.629... > 1.17 * 10^{-6} > 4.81 * 10^{-8}$ and $1.2 * 10^{-5} > 2.78 * 10^{-8}$ respectively). The backwards stability of all three algorithms was able to be confirmed as well, since the conditioning error (10^{-6}) was less than both stability bounds also. This confirms the predictions of theory and that all of the algorithms are working properly.

None of the experiments listed above were able to disprove the theoretical stability or conditioning bounds either, despite the fact that some of the sums were incredibly ill-conditioned. The random summation and the sin summation condition numbers were greater than 41 and 196 respectively even at the $N = 1000$ level. In the worst case, the condition number for the alternating sin summation, $N = 2^{20}$, was greater than 100,000, but the relative forward error was at most of the order of 10^{-5}. Perhaps the most surprising result was the fact that the sum $\sum_{i=1}^{N}(-1)^i \frac{1}{i^2}$ was the most well-conditioned, despite its alternating

sign. This can be attributed to the fact that each successive terms gets small very quickly, preventing there from being much deviation past the first few.

The binary fan-in tree algorithm performed overall better than the single precision accumulation algorithm in terms of accuracy, as predicted by the stability bound for binary fan-in tree being tighter. This is especially true for the sin summation, which was intentionally used to take advantage of the algorithm's adding first and last terms, leading to major cancellations when the magnitudes differed significantly. For the random summation summations, binary fan-in tree performed better than single precision accumulation and for rational summation before about as well. However, for the sin summation, the binary fan-in performed about as well or worse than the single precision accumulation. This can be attributed to the fact that the sin sum was intentionally used to take advantage of the algorithm's adding first and last terms, leading to major cancellations when the magnitudes differed significantly.

However, a surprising result that contradicts the hypotheses is that double precision accumulation did overall better than either of the two aforementioned algorithms. Being able to update the total sum in double precision and then round off at the end led to less deviation from the exact sum, meaning that $p(n) < n$ for its stability bound. Instead of having to round-off after each successive addition potentially, the double precision algorithm only had to round-off once, indicating that decereasing the number of times round-off occurs will overall increase accuracy.

References

[1] https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set2.pdf

[2] https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set3.pdf

[3] https://mathworld.wolfram.com/L1-Norm.html

[4] http://www2.hawaii.edu/ ngnagata/work/lab01/report01.html

[5] https://www.math.fsu.edu/ gallivan/courses/FCM1new/Locked/Sets/set4.pdf

[6] https://en.wikipedia.org/wiki/Leibniz formula for π