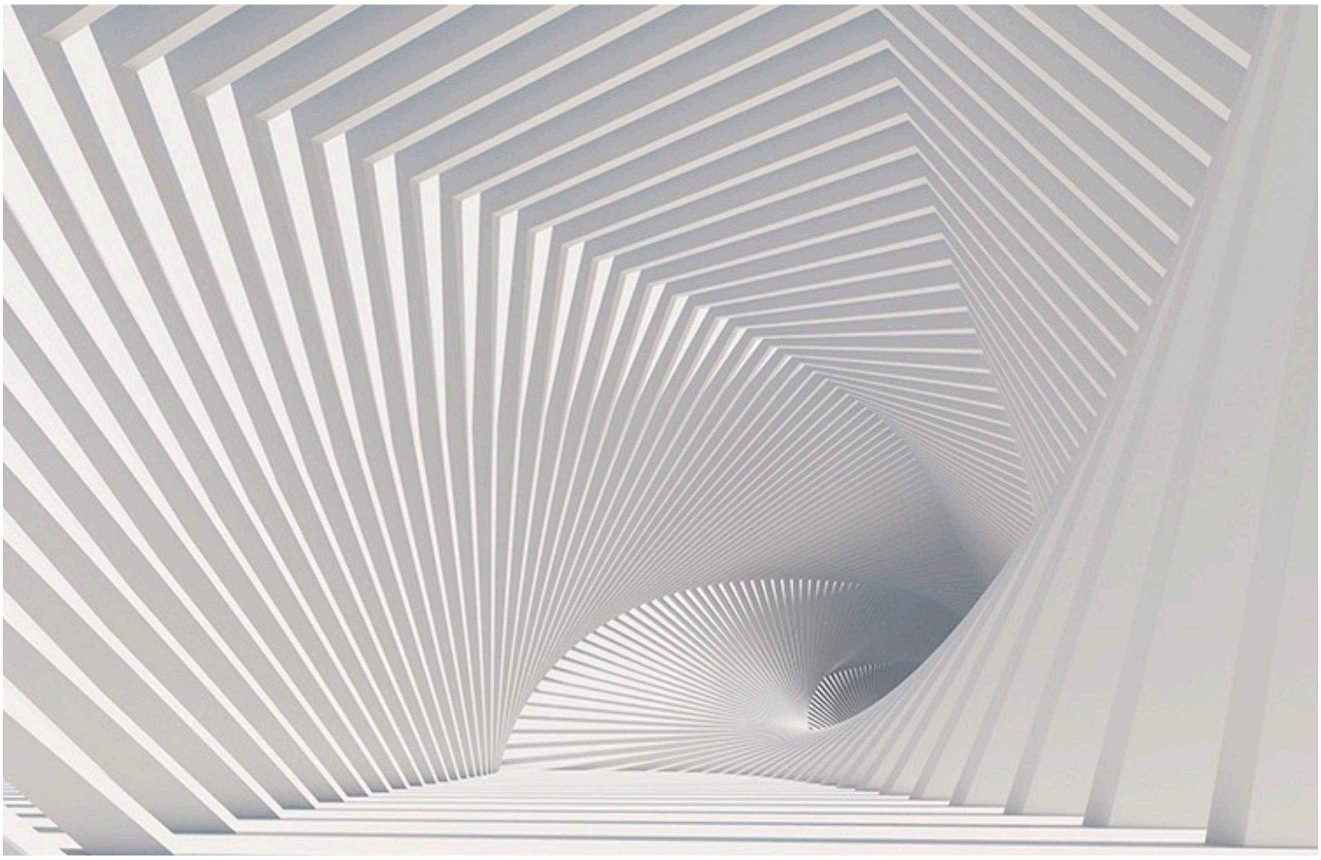
FLOATING-POINT ERRORS AND COMPENSATION ALGORITHM



Ao XIE

Ingénierie des modèles et Simulation

Institut Supérieur d'Informatique, de Modélisation et de leurs Applications

Introduction

As the foundation of modern computing, floating point numbers play an important role in scientific computing, financial modeling and engineering. However, due to the limitations of binary representation, floating-point operations inherently introduce errors and may accumulate over multiple operations.

The representation of floating-point numbers follows the IEEE 754 standard[1], which defines a finite-precision binary floating-point format. For integers, an exact representation is possible based on this definition. However, for decimals, exact representation is not possible. For example, the binary representation of 0.1 is an infinite loop decimal. This limitation requires truncation of decimals, which introduces rounding errors[2].

In addition, floating-point endings have a limited number of bits, which limits the precision of floating-point numbers. When computing values with large magnitude differences, loss of valid digits may occur, known as catastrophic elimination[1].

In this context, this experiment will validate the precision error of floating point numbers themselves and use Kahan Algorithm[3] and its improved algorithm, the Neumaier [4], to try to reduce this error.

Experimental Data Set

In total, three different forms of datasets were set up in this experiment to try to reproduce the problems that arise during the computation of floating point numbers.

1 Random Dataset

The use of random numbers between 0 and 1 is used to simulate regular numerical distributions. This data is used to simulate scenarios of random number distributions that occur widely in real-world applications.

2 Uniform Dataset

Using the same decimals is used to validate that the algorithm handles the accumulation of uniformly distributed data. This data is used to simulate the errors generated during simple accumulation and also for the cases in the course.

3 Extreme Dataset

The use of alternating extremely large and extremely small values was used to test the stability of the algorithm when dealing with large spans of values.

Algorithms

A total of three different algorithms were used in this experiment to calculate the different errors they produce. These are the use of the Native method as the baseline model and the other two compensation algorithms.

1 Native Summation Algorithm

In this experiment, the most primitive cumulative approach was used as the baseline model for the experiment. The role of the baseline model is to verify the errors that can occur in floating-point numbers when computed without correction and whether the correction algorithm can correct these errors to make a reference.

2 Kahan Summation Algorithm

Kahan summation algorithm is a computational method proposed by William Kahan in 1965[3] for improving the accuracy of numerical calculations.

The core idea of this algorithm is to record the errors rounded off in each addition operation by introducing a compensation variable, and gradually compensate these errors back in subsequent calculations. The algorithm is shown in Algorithm 1.

Algorithm 1: Kahan Summation

Input: Array arr of size n
Output: Sum of the array with error compensation

Initialize $sum \leftarrow 0.0$, $c \leftarrow 0.0$ `// Compensation for lost low-order bits`
for $i \leftarrow 1$ *to* n **do**
  $y \leftarrow \mathrm{arr}[i] - c$ `// Correct the current value with compensation`
  $t \leftarrow sum + y$
  $c \leftarrow (t - sum) - y$ `// Update the compensation value`
  $sum \leftarrow t$
end
return sum

This algorithm is effective in compensating rounding errors without introducing a lot of extra operations and has a simple structure for easy completion.

3 Neumaier Summation Algorithm

An improved version of the Kahan summation algorithm was proposed by L. Neumaier in 1974[4] to address the stability and robustness of the Kahan summation algorithm in the face of summing over large spans of data.

The core idea of the Neumaier summation algorithm is to complete the compensation in the Kahan summation algorithm by means of dynamic compensation. Specifically, its by determining the relationship between the absolute size of the current sum and the cumulative value, and adjusted to the appropriate way of error compensation. Its specific implementation is shown in Algorithm 2.

Algorithm 2: Neumaier Summation

Input: Array arr of size n

Output: Sum of the array with enhanced error compensation

```
Initialize sum ← 0.0, c ← 0.0          // Enhanced compensation for
  rounding errors
for i ← 1 to n do
    y ← arr[i] − c
    t ← sum + y
    if |sum| ≥ |y| then
        c ← c + (sum − t) + y
    else
        c ← c + (y − t) + sum
    end
    sum ← t
end
return sum + c
```

This algorithm presents stronger robustness error compensation with a slight increase in computational complexity.

Experimental Record

The main content of this section is to convert the above algorithms into C code and present the final results of the experiments in the form of tables. The experimental results will also be analyzed and summarized to compare the performance, accuracy and applicability of different algorithms in an intuitive way, so as to provide support and reference for the application of the algorithms in practical scenarios.

In addition to the baseline model, the experiments introduced the MPFR library[5] as a high-precision reference standard. In this case, the library is utilized to obtain theoretical results for the ideal case, which can be used as a reference standard for practical calculations.

1 Random Dataset

During the computation of random data, each result is computed using 1000 randomly generated random numbers between 0 and 1 and accumulated. The final results are shown in Table 1.

Table 1 - Cumulative test for random data

	MPFR	**Native Result**	**Native Error**	**Kahan Result**	**Kahan Error**	**Neumaier Result**	**Neumaier Error**
1	498.544783810407295732	498.544783810407579949	0.000000000000284217	498.544783810407295732	0.000000000000000000	498.544783810496539900	0.000000000089244168
2	479.816517785106100291	479.816517785105872917	0.000000000000227374	479.816517785106100291	0.000000000000000000	479.816517784974223559	0.000000000131876732
3	513.348933232179319930	513.348933232178978869	0.000000000000341061	513.348933232179319930	0.000000000000000000	513.348933232182730535	0.000000000003410605
4	505.267555140549120551	505.267555140549120551	0.000000000000000000	505.267555140549120551	0.000000000000000000	505.267555140800936897	0.000000000251816346
5	498.147556402789234653	498.147556402789064123	0.000000000000170530	498.147556402789234653	0.000000000000000000	498.147556402619215987	0.000000000170018666
6	506.182040249547924304	506.182040249547753774	0.000000000000170530	506.182040249547924304	0.000000000000000000	506.182040249561623568	0.000000000013699264
7	487.139282804047354603	487.139282804047297759	0.000000000000056843	487.139282804047354603	0.000000000000000000	487.139282804100275825	0.000000000052921223
8	494.903422127898522831	494.903422127898238614	0.000000000000284217	494.903422127898522831	0.000000000000000000	494.903422127788530815	0.000000000109992015
9	475.860664682397953129	475.860664682398294190	0.000000000000341061	475.860664682397953129	0.000000000000000000	475.860664682248511781	0.000000000149441348
10	485.702045298508380711	485.702045298508608084	0.000000000000227374	485.702045298508380711	0.000000000000000000	485.702045298365760573	0.000000000142620138

In fact, during the experiments in this section we try to use the MT random number generator[6] to get better random numbers for our experiments. But the generator was not used in the end due to the fact that the high resolution MT random numbers are close to uniformly distributed in the interval, a property that largely reduces the accumulation of rounding errors.

Based on the information in the table, it is easy to see that for random numbers between zero and one, the computational error of the baseline model is less than that of Neumaier algorithm when MPFR is used as the reference standard, but it shows less stability and accuracy relative to Kahan algorithm.

2 Uniform Dataset

For the uniform data test, since the results were the same for each experiment, data from 0.1 to 1.0 directly cumulative in order of 0.1 were used as the test data. The final test results are shown in Table 2.

Table 2 - Cumulative test for uniform data

	Data	MPFR	Native Result	Native Error	Kahan Result	Kahan Error	Meumaier Result	Meumaier Error
1	0.1	100.000000000000000000	99.9999999999998593125	0.000000000001406875	100.000000000000000000	0.000000000000000000	100.000000000006153300	0.000000000006153300
2	0.2	200.000000000000000000	199.9999999999997186251	0.000000000002813749	200.000000000000000000	0.000000000000000000	200.000000000012306600	0.000000000012306600
3	0.3	300.000000000000000000	300.000000000000005627498	0.000000000005627498	300.000000000000000000	0.000000000000000000	300.000000000000170530	0.000000000000170530
4	0.4	400.0000000000000000000	399.999999999994372502	0.000000000005627498	400.000000000000000000	0.000000000000000000	400.000000000024613200	0.000000000024613200
5	0.5	500.000000000000000000	500.000000000000000000	0.000000000000000000	500.000000000000000000	0.000000000000000000	500.000000000000000000	0.000000000000000000
6	0.6	600.0000000000000000000	600.000000000011254997	0.0000000000011254997	600.000000000000000000	0.000000000000000000	600.000000000000341061	0.000000000000341061
7	0.7	700.000000000000000000	700.00000000000006366463	0.000000000006366463	700.000000000000000000	0.000000000000000000	700.000000000032059688	0.000000000032059688

	Data	MPFR	Native Result	Native Error	Kahan Result	Kahan Error	Meumaier Result	Meumaier Error
8	0.8	800.000000000000000000	799.999999999988745003	0.0000000000112 54997	800.000000000000000000	0.000000000000000000	800.000000000049226401	0.000000000049226401
9	0.9	900.0000000000000000000	899.999999999984879651	0.0000000000151 20349	900.000000000000000000	0.000000000000000000	900.000000000028762770	0.000000000028762770
10	1.0	1000.0000000000000000000	1000.0000000000000000000	0.000000000000000000	1000.0000000000000000000	0.000000000000000000	1000.0000000000000000000	0.000000000000000000

According to the data in the table, it is still found that Kahan algorithm performs better for the baseline and Nermaie algorithm. And for decimal numbers that are easy to represent in binary (e.g., 0.5 and 1.0), no rounding error occurs.

3 Extreme Dataset

Based on the data obtained earlier, it can be seen that the results obtained by the Neumaier algorithm are not satisfactory, and this result does not have a significant impact in the above conclusions due to the fact that, according to its proposed conception, this algorithm is proposed in order to improve the accuracy when the data variance is large. Therefore in this subsection, an attempt will be made to correct this algorithm. Namely, the algorithm will include a comparison of the size for 1e6 and then two types of compensation will be added. For the experimental results after correcting this algorithm are shown in Table 3.

Table 3 - Cumulative test for extreme data

	MPFR	Native Result	Native Error	Kahan Result	Kahan Error	Neumaier Result	Neumaier Error
1	2944773162291.724121093750000000	2944773162291.742675781250000000	0.018554687500000000	2944773162291.724121093750000000	0.000000000000000000	2944773162291.670410156250000000	0.053710937500000000
2	4699571328440.020507812500000000	4699571328439.992187500000000000	0.028320312500000000	4699571328440.020507812500000000	0.000000000000000000	4699571328440.223632812500000000	0.203125000000000000

	MPFR	Native Result	Native Error	Kahan Result	Kahan Error	Neumaier Result	Neumaier Error
3	4208044288774.875000000000000	4208044288774.892578125000000000	0.0175781250000000000	4208044288774.874511718750000000	0.000488281250000000	4208044288775.008300781250000000	0.133300781250000000
4	2667364545043.2148437500000000000	2667364545043.2128906250000000000	0.001953125000000000	2667364545043.214843750000000000	0.000000000000000000	2667364545043.182617187500000000	0.032226562500000000
5	4274142913011.098144531250000000	4274142913011.048828125000000000	0.049316406250000000	4274142913011.098144531250000000	0.000000000000000000	4274142913011.165527343750000000	0.067382812500000000
6	2487699649446.864257812500000000	2487699649446.823730468750000000	0.0405273437500000000	2487699649446.864257812500000000	0.000000000000000000	2487699649446.992675781250000000	0.128417968750000000
7	4094478017414.747558593750000000	4094478017414.668457031250000000	0.079101562500000000	4094478017414.747558593750000000	0.000000000000000000	4094478017414.663085937500000000	0.084472656250000000
8	2717640620071.469238281250000000	2717640620071.461914062500000000	0.0073242187500000000	2717640620071.469238281250000000	0.000000000000000000	2717640620071.567871093750000000	0.098632812500000000
9	373571692605.866943359375000000	373571692605.865966796875000000	0.000976562500000000	373571692605.866943359375000000	0.000000000000000000	373571692605.880493164062500000	0.0135498046875000000
10	4963965825884.912109375000000000	4963965825884.9316406250000000000	0.019531250000000000	4963965825884.911132812500000000	0.000976562500000000	4963965825885.091796875000000000	0.179687500000000000

According to the experimental results, despite the attempts to change the Neumaier algorithm, the performance results were still unsatisfactory and even performed worse than the baseline model. However, in this experiment, Kahan algorithm still gave relatively good results, and despite some small observable errors in some cases, the overall results were still better than the baseline model and Neumaier algorithm.

Conclusion

The purpose of this experiment is to compare the results obtained by the native summation method, Kahan summation algorithm and Neumaier summation algorithm. And to use the results obtained from MPFR library as a reference ideal result for the data.

Based on the experimental results, it can be clearly concluded that the traditional method (i.e., the baseline model) produces a certain rounding error for the vast majority of fractional arithmetic operations from 0 to 1, and therefore this method has absolute limitations. At the same time, it can be seen that Kahan algorithm gets better performance in all three cases listed in the experiment. A small amount of error occurs only when there are large fluctuations in the size of the data.

The result of Neumaier algorithm in the experiment is not a normal result, and according to its proposed purpose, a third experiment was designed to verify its effect. However, for all three experiments, this algorithm did not give an excellent result in terms of error, and even in some experimental cases the error obtained was larger than the baseline model. This phenomenon occurs both before and after the algorithm correction, i.e., the possible problem is in the implementation of the algorithm.

Kahan algorithm continues to have a profound impact in currently popular fields, such as Amro Eldebiky et al. who proposed a technique for error compensation in simulation computation using a similar idea to reduce the impact of hardware noise on neural network training and inference[7], and Yifei Cheng et al. who proposed a technique to reduce communication overheads using a similar approach[8]. Therefore research based on this algorithm as well as its ideas is still of research value at the present time.

Bibliography

[1] "754-2019 - IEEE Standard for Floating-Point Arithmetic | IEEE Standard | IEEE Xplore," doi: 10.1109/IEEESTD.2019.8766229.

[2] GoldbergDavid, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys (CSUR),* vol. 23, no. 1, 1991-03-01, doi: 10.1145/103162.103163.

[3] KahanW., "Pracniques: further remarks on reducing truncation errors," *Communications of the ACM,* vol. 8, no. 1, 1965-01-01, doi: 10.1145/363707.363723.

[4] A. Neumaier, "Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen," *Zamm-zeitschrift Fur Angewandte Mathematik Und Mechanik,* 1974, doi: 10.1002/ZAMM.19740540106">10.1002/ZAMM.19740540106.

[5] FousseLaurent, HanrotGuillaume, LefèvreVincent, PélissierPatrick, and ZimmermannPaul, "MPFR," *ACM Transactions on Mathematical Software (TOMS),* vol. 33, no. 2, 2007-06-01, doi: 10.1145/1236463.1236468.

[6] MatsumotoMakoto and NishimuraTakuji, "Mersenne twister," *ACM Transactions on Modeling and Computer Simulation (TOMACS),* vol. 8, no. 1, 1998-01-01, doi: 10.1145/272991.272995.

[7] A. Eldebiky, G. L. Zhang, G. Boecherer, B. Li, and U. Schlichtmann, "CorrectNet: Robustness Enhancement of Analog In-Memory Computing for Neural Networks by Error Suppression and Compensation," 2022/11/27, doi: 10.48550/arXiv.2211.14917.

[8] Y. Cheng *et al.*, "Communication-Efficient Distributed Learning with Local Immediate Error Compensation," 2024/02/19, doi: 10.48550/arXiv.2402.11857.