Modern computing with distributed high-speed math units has allowed us to exceed the accuracy of the basic operations provided by the computational units. This is a particularly common problem for long addition sequences. That is, given a sufficiently large sequence of finite precision numbers, when added with finite precision math, will yield an exceeding inaccurate result. *This can be really bad!*

Much modern analysis depends on being able to add long sequences of numbers correctly. For example, sums of powers of ***X*** are frequently computed in statistics. Many “weighting” strategies used in Finance and Data Science require correctly computed sums of long sequences. Even computing monetary totals can be at risk. A large retailer could incorrectly total sales amounts for sufficiently large sales volume.

We will explore the background of the problem, offer mitigations for the problem, and showcase concrete examples of the addition problem in many computer systems.

## Background

Let’s begin with a motivating example. Suppose we have a three significant digit calculator (e.g., similar to a Slide Rule from the ancients). We need to sum this sequence of four decimal values:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Add in** | | | | | | |  | **Accumulator - mantissa** | | | | **Accumulator - exponent** |
|  |  |  |  |  |  |  |  |  | 0 | 0 | 0 | 0 |
| 1 | 2 | 3 | . |  |  |  |  | . | 1 | 2 | 3 | 10\*\*3 |
|  | 2 | 3 | . | 4 |  |  |  | . | 1 | 4 | 6 | 10\*\*3 |
|  |  | 3 | . | 4 | 5 |  |  | . | 1 | 4 | 9 | 10\*\*3 |
|  |  |  | . | 4 | 5 | 6 |  | . | 1 | 4 | 9 | 10\*\*3 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 5 | 0 | . | 3 | 0 | 6 |  |  | 1 | 4 | 9 | ***<-Result*** |

The true sum of these four numbers is 150.306, but our finite precision calculator has computed 149.0 as the estimated sum. This example resulted in an absolute error of -1.306 (149.0 – 150.3), with an associated relative error of or -0.00869 (-1.306 / 150.3). The sum is 0.9% low.

Now imagine that we have *many* more numbers to add up in than in this example, but the remaining numbers are all less than one. The sum will never increase beyond 149. We can make the absolute error very large, and worse yet, we can make the relative error -100% (see note ***33*** below.)

This may not seem so bad, but imagine we are dealing with Diamonds, and these numbers represent carat weights of diamonds at a major Jeweler. Diamonds average about $1000 a carat for small ones. A company would be upset with a $1,306 of under counted diamonds.

We suffer from two sources of error when operate on quantities in Finite Precision Arithmetic: representational and operational errors. Our numbers are approximations of the true quantities when represented in finite precision arithmetic.

Here is a classic example of the *representation* problem shown using Python:

|  |  |
| --- | --- |
| **Code** | **Output** |
| **x = float(1.0) / float(10.0)**  **print('Add x = {:.18f} 10 times'.format(x))**  **wrong\_sum = float(0.0)**  **for i in range(10):**  **wrong\_sum += x**  **print('Sum is: {:.18f}'.format(wrong\_sum))** | Add x = 0.100000000000000006 10 times  Sum is: 0.999999999999999889 |

Operational error, as seen in the first example, is a property of the computing system in conjunction with finite precision representation. A good Finite Precision Math Tutorial is found at <https://www.cs.purdue.edu/homes/skeel/CS515/2.pdf>.

Now that you have gotten this far, here is a metaphor for the problem mitigation, expressed in the form of an old joke:

A man goes to a doctor, holding his hand behind his back, and says: “Doc, it hurts when I do this”. The doctor replies: “well then, don’t do that!”

Now let’s get a more detailed view of the problem and how it manifests itself in my software systems.